MAR GREGORIOS COLLEGE OF ARTS & SCIENCE

Block No.8, College Road, Mogappair West, Chennai – 37

Affiliated to the University of Madras
Approved by the Government of Tamil Nadu
An ISO 9001:2015 Certified Institution

PG DEPARTMENT OF COMPUTER SCIENCE

SUBJECT NAME: INFORMATION SECURITY

SUBJECT CODE: PSD3B

SEMESTER: III

PREPARED BY: PROF. A. JEROMEROBINSON

Information Security

Title of the Information Security Course/ Paper Core - 11 II Year & Third Semester Credit: 4

Unit 1: Introduction: On 11 February 2013, residents of Great Falls, Montana received the following warning on their televisions [INF13]. The transmission displayed a message anner on the bottom of the screen (as depicted in Figure 1-1).

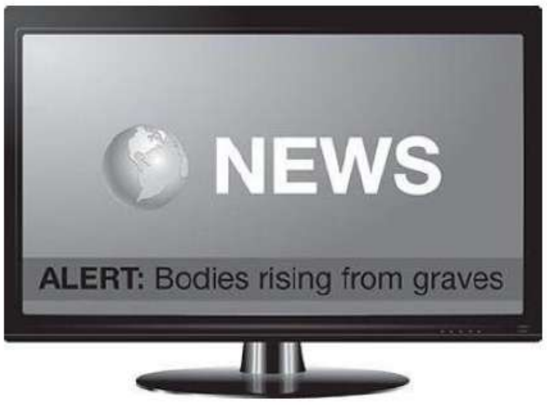


FIGURE 1-1 Emergency Broadcast Warning

And the following alert was broadcast:
[Beep Beep Beep: *the sound pattern of the U.S. government Emergency Alert System. The following text then scrolled across the screen:*] Civil authorities in your area have reported that the bodies of the dead are rising from their graves and attacking the living. Follow the messages on screen that will be updated as information becomes available. Do not attempt to approach or apprehend these bodies as they are considered extremely dangerous. This warning applies to all areas receiving this broadcast. [Beep Beep Beep]

The warning signal sounded authentic; it had the distinctive tone people recognize for warnings of serious emergencies such as hazardous weather or a natural disaster. And the text was displayed across a live broadcast television program. On the other hand, bodies rising from their graves sounds suspicious.

What would you have done?

Only four people contacted police for assurance that the warning was indeed a hoax. As you can well imagine, however, a different message could have caused thousands of people to jam the highways trying to escape. (On 30 October 1938 Orson Welles performed a radio broadcast of the H. G. Wells play *War of the Worlds* that did cause a minor panic of people believing that Martians had landed and were wreaking havoc in New Jersey.)

The perpetrator of this hoax was never caught, nor has it become clear exactly how it was done. Likely someone was able to access the system that feeds emergency broadcasts to local radio and television stations. In other words, a hacker probably broke into a computer system.

You encounter computers daily in countless situations, often in cases in which you are scarcely aware a computer is involved, like the emergency alert system for broadcast media. These computers move money, control airplanes, monitor health, lock doors, play music, heat buildings, regulate hearts, deploy airbags, tally votes, direct communications, regulate traffic, and do hundreds of other things that affect lives, health, finances, and well-being. Most of the time these computers work just as they should. But occasionally they do something horribly wrong, because of either a benign failure or a malicious attack.

This book is about the security of computers, their data, and the devices and objects to which they relate. In this book you will learn some of the ways computers can fail—or be made to fail—and how to protect against those failures. We begin that study in the way any good report does: by answering the basic questions of what, who, why, and how.

Security

Computer security is the protection of the items you value, called the **assets** of a computer or computer system. There are many types of assets, involving hardware, software, data, people, processes, or combinations of these. To determine what to protect, we must first identify what has value and to whom.

A computer device (including hardware, added components, and accessories) is certainly an asset. Because most computer hardware is pretty useless without programs, the software is also an asset. Software includes the operating system, utilities and device handlers; applications such as word processing, media players or email handlers; and even programs that you may have written yourself. Much hardware and software is *off-theshelf,* meaning that it is commercially available (not custom-made for your purpose) and that you can easily get a replacement. The thing that makes your computer unique and important to you is its content: photos, tunes, papers, email messages, projects, calendar information, ebooks (with your annotations), contact information, code you created, and the like. Thus, data items on a computer are assets, too. Unlike most hardware and software, data can be hard—if not impossible—to recreate or replace. These assets are all shown in Figure 1-2.

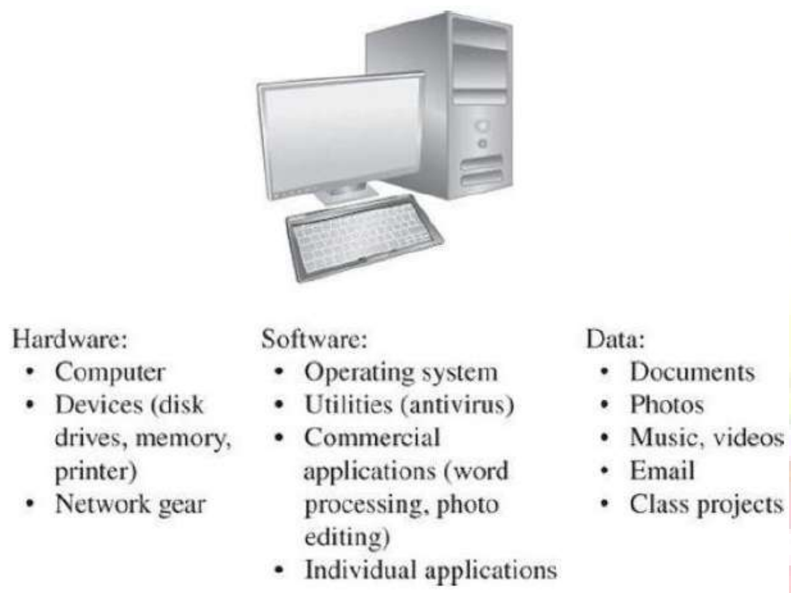


FIGURE 1-2 Computer Objects of Value

These three things—hardware, software, and data—contain or express things like the design for your next new product, the photos from your recent vacation, the chapters of your new book, or the genome sequence resulting from your recent research. All of these things represent intellectual endeavor or property, and they have value that differs from one person or organization to another. It is that value that makes them assets worthy of protection, and they are the elements we want to protect. Other assets—such as access to data, quality of service, processes, human users, and network connectivity—deserve protection, too; they are affected or enabled by the hardware, software, and data. So in most cases, protecting hardware, software, and data covers these other assets as well.

Computer systems—hardware, software, and data—have value and deserve security protection.

In this book, unless we specifically distinguish between hardware, software, and data, we refer to all these assets as the computer system, or sometimes as the computer. And because processors are embedded in so many devices, we also need to think about such variations as mobile phones, implanted pacemakers, heating controllers, and automobiles. Even if the primary purpose of the device is not computing, the device's embedded computer can be involved in security incidents and represents an asset worthy of protection.

Values of Assets

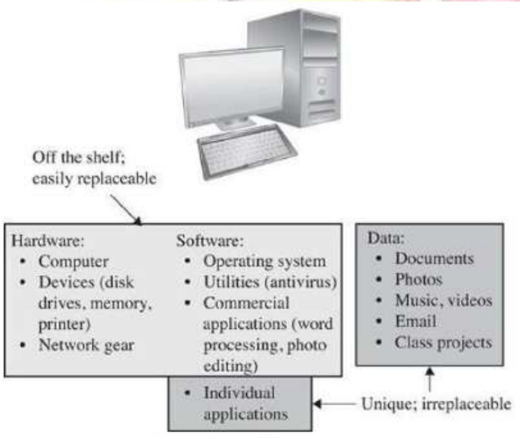


FIGURE 1-3 Values of Assets

Assets' values are personal, time dependent, and often imprecise.

After identifying the assets to protect, we next determine their value. We make value based decisions frequently, even when we are not aware of them. For example, when you go for a swim you can leave a bottle of water and a towel on the beach, but not your wallet or cell phone. The difference relates to the value of the assets. The value of an asset depends on the asset owner's or user's perspective, and it may be independent of monetary cost, as shown in Figure 1-3.

Your photo of your sister, worth only a few cents in terms of paper and ink, may have high value to you and no value to your roommate. Other items' value depends on replacement cost; some computer data are difficult or impossible to replace. For example, that photo of you and your friends at a party may have cost you nothing, but it is invaluable because there is no other copy. On the other hand, the DVD of your favorite film may have cost a significant portion of your take-home pay, but you can buy another one if the DVD is stolen or corrupted. Similarly, timing has bearing on asset value. For example, the value of the plans for a company's new product line is very high, especially to competitors. But once the new product is released, the plans' value drops dramatically.

Assets' values are personal, time dependent, and often imprecise.

The Vulnerability–Threat–Control Paradigm

The goal of computer security is protecting valuable assets. To study different ways of protection, we use a framework that describes how assets may be harmed and how to counter or mitigate that harm.

A **vulnerability** is a weakness in the system, for example, in procedures, design, or implementation, that might be exploited to cause loss or harm. For instance, a particular system may be vulnerable to unauthorized data manipulation because the system does not verify a user's identity before allowing data access.

A vulnerability is a weakness that could be exploited to cause harm.

A **threat** to a computing system is a set of circumstances that has the potential to cause loss or harm. To see the difference between a threat and a vulnerability, consider the illustration in Figure 1-4. Here, a wall is holding water back. The water to the left of the wall is a threat to the man on the right of the wall: The water could rise, overflowing onto the man, or it could stay beneath the height of the wall, causing the wall to collapse. So the threat of harm is the potential for the man to get wet, get hurt, or be drowned. For now, the wall is intact, so the threat to the man is unrealized.

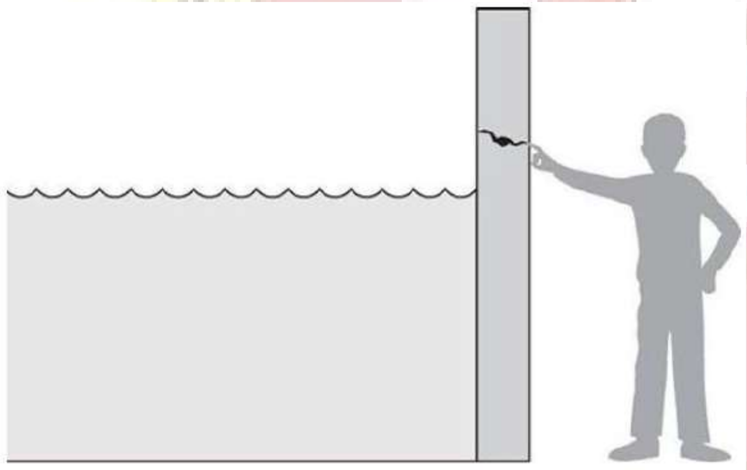


FIGURE 1-4 Threat and Vulnerability

A threat is a set of circumstances that could cause harm.

A threat is a set of circumstances that could cause harm.

However, we can see a small crack in the wall—a vulnerability that threatens the man's security. If the water rises to or beyond the level of the crack, it will exploit the vulnerability and harm the man. There are many threats to a computer system, including human-initiated and computer initiated ones. We have all experienced the results of inadvertent human errors, hardware design flaws, and software failures. But natural disasters are threats, too; they can bring a system down when the computer room is flooded or the data center collapses from an earthquake, for example. A human who exploits a vulnerability perpetrates an **attack** on the system. An attack can also be launched by another system, as when one system sends an overwhelming flood of messages to another, virtually shutting down the second system's ability to function.

Unfortunately, we have seen this type of attack frequently, as denial-of-service attacks deluge servers with more messages than they can handle. (We take a closer look at denial of service in Chapter 6.)

How do we address these problems? We use a **control** or **countermeasure** as protection. That is, a control is an action, device, procedure, or technique that removes or reduces a vulnerability. In Figure 1-4, the man is placing his finger in the hole, controlling the threat of water leaks until he finds a more permanent solution to the problem. In general, we can describe the relationship between threats, controls, and vulnerabilities in this way:

Controls prevent threats from exercising vulnerabilities.

A *threat* is blocked by *control* of a *vulnerability*.

Before we can protect assets, we need to know the kinds of harm we have to protect them against, so now we explore threats to valuable assets.

Attacks

We can consider potential harm to assets in two ways: First, we can look at what bad things can happen to assets, and second, we can look at who or what can cause or allow those bad things to happen. These two perspectives enable us to determine how to protect assets.

Think for a moment about what makes your computer valuable to you. First, you use it as a tool for sending and receiving email, searching the web, writing papers, and performing many other tasks, and you expect it to be available for use when you want it.

Without your computer these tasks would be harder, if not impossible. Second, you rely heavily on your computer's integrity. When you write a paper and save it, you trust that the paper will reload exactly as you saved it. Similarly, you expect that the photo a friend passes you on a flash drive will appear the same when you load it into your computer as when you saw it on your friend's computer. Finally, you expect the "personal" aspect of a personal computer to stay personal, meaning you want it to protect your confidentiality. For example, you want your email messages to be just between you and your listed recipients; you don't want them broadcast to other people. And when you write an essay, you expect that no one can copy it without your permission.

These three aspects, confidentiality, integrity, and availability, make your computer valuable to you. But viewed from another perspective, they are three possible ways to make it less valuable, that is, to cause you harm. If someone steals your computer, scrambles data on your disk, or looks at your private data files, the value of your computer has been diminished or your computer use has been harmed. These characteristics are both basic security properties and the objects of security threats.

We can define these three properties as follows.

• **availability:** the ability of a system to ensure that an asset can be used by any authorized parties

• **integrity:** the ability of a system to ensure that an asset is modified only by authorized parties

• **confidentiality:** the ability of a system to ensure that an asset is viewed only by authorized parties

These three properties, hallmarks of solid security, appear in the literature as early as James P. Anderson's essay on computer security [AND73] and reappear frequently in more recent computer security papers and discussions. Taken together (and rearranged), the properties are called the **C-I-A triad** or the **security triad**. ISO 7498-2 [ISO89] adds to them two more properties that are desirable, particularly in communication networks:

• **authentication:** the ability of a system to confirm the identity of a sender

• **nonrepudiation** or **accountability:** the ability of a system to confirm that a sender cannot convincingly deny having sent something The U.S. Department of Defense [DOD85] adds auditability: the ability of a system to trace all actions related to a given asset. The C-I-A triad forms a foundation for thinking about security. Authenticity and nonrepudiation extend security notions to network communications, and auditability is important in establishing individual accountability for computer activity. In this book we generally use the C-I-A triad as our security taxonomy so that we can frame threats, vulnerabilities, and controls in terms of the C-I-A properties affected. We highlight one of these other properties when it is relevant to a particular threat we are describing. For now, we focus on just the three elements of the triad.

C-I-A triad: confidentiality, integrity, availability

What can happen to harm the confidentiality, integrity, or availability of computer assets? If a thief steals your computer, you no longer have access, so you have lost availability; furthermore, if the thief looks at the pictures or documents you have stored, your confidentiality is compromised. And if the thief changes the content of your music files but then gives them back with your computer, the integrity of your data has been harmed. You can envision many scenarios based around these three properties.

The C-I-A triad can be viewed from a different perspective: the nature of the harm caused to assets. Harm can also be characterized by four acts: **interception**, **interruption**, **modification**, and **fabrication**. These four acts are depicted in Figure 1-5. From this point of view, confidentiality can suffer if someone intercepts data, availability is lost if someone or something interrupts a flow of data or access to a computer, and integrity can fail if someone or something modifies data or fabricates false data. Thinking of these four kinds of acts can help you determine what threats might exist against the computers you are trying to protect.

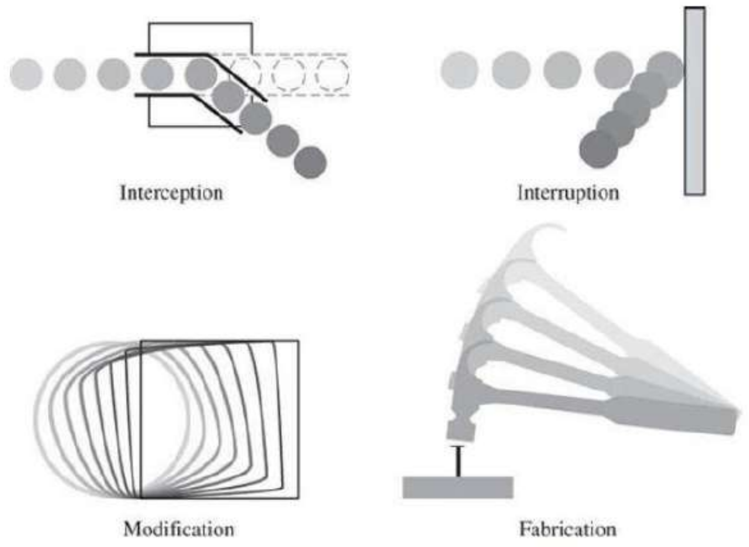


FIGURE 1-5 Four Acts to Cause Security Harm

To analyze harm, we next refine the C-I-A triad, looking more closely at each of its elements.

Confidentiality

Some things obviously need confidentiality protection. For example, students' grades, financial transactions, medical records, and tax returns are sensitive. A proud student may run out of a classroom screaming "I got an A!" but the student should be the one to choose whether to reveal that grade to others. Other things, such as diplomatic and military secrets, companies' marketing and product development plans, and educators' tests, also must be carefully controlled. Sometimes, however, it is not so obvious that something is sensitive. For example, a military food order may seem like innocuous information, but a sudden increase in the order could be a sign of incipient engagement in conflict. Purchases of food, hourly changes in location, and access to books are not things you would ordinarily consider confidential, but they can reveal something that someone wants to be kept confidential.

The definition of confidentiality is straightforward: Only authorized people or systems can access protected data. However, as we see in later chapters, ensuring confidentiality can be difficult. For example, who determines which people or systems are authorized to access the current system? By "accessing" data, do we mean that an authorized party can access a single bit? the whole collection? pieces of data out of context? Can someone who is authorized disclose data to other parties? Sometimes there is even a question of who owns the data: If you visit a web page, do you own the fact that you clicked on a link, or does the web page owner, the Internet provider, someone else, or all of you? In spite of these complicating examples, confidentiality is the security property we understand best because its meaning is narrower than that of the other two. We also understand confidentiality well because we can relate computing examples to those of preserving confidentiality in the real world.

Confidentiality relates most obviously to data, although we can think of the confidentiality of a piece of hardware (a novel invention) or a person (the whereabouts of a wanted criminal). Here are some properties that could mean a failure of data confidentiality:

• An unauthorized person accesses a data item.
• An unauthorized process or program accesses a data item.
• A person authorized to access certain data accesses other data not authorized (which is a specialized version of "an unauthorized person accesses a data item").
• An unauthorized person accesses an approximate data value (for example, not knowing someone's exact salary but knowing that the salary falls in a particular range or exceeds a particular amount).
• An unauthorized person learns the existence of a piece of data (for example, knowing that a company is developing a certain new product or that talks are underway about the merger of two companies).

Notice the general pattern of these statements: A person, process, or program is (or is not) authorized to access a data item in a particular way. We call the person, process, or program a **subject**, the data item an **object**, the kind of access (such as read, write, or execute) an **access mode**, and the authorization a **policy**, as shown in Figure 1-6. These four terms reappear throughout this book because they are fundamental aspects of computer security.

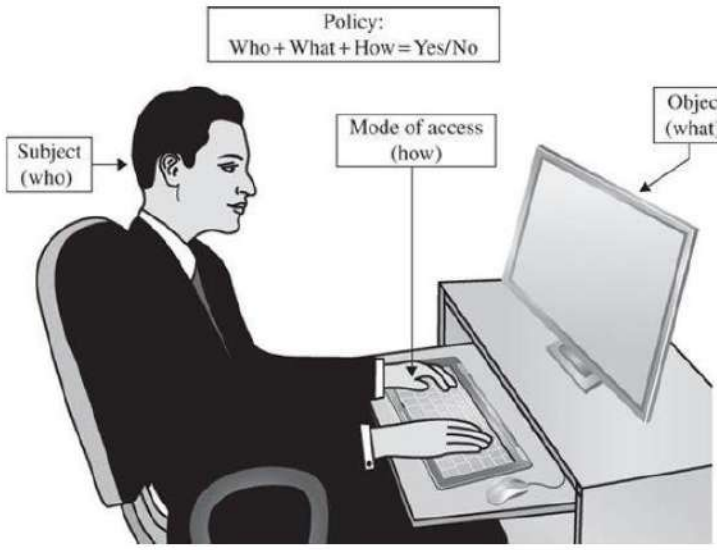


FIGURE 1-6 Access Control

One word that captures most aspects of confidentiality is *view*, although you should not take that term literally. A failure of confidentiality does not necessarily mean that someone sees an object and, in fact, it is virtually impossible to look at bits in any meaningful way (although you may look at their representation as characters or pictures). The word view does connote another aspect of confidentiality in computer security, through the association with viewing a movie or a painting in a museum: look but do not touch. In computer security, confidentiality usually means obtaining but not modifying. Modification is the subject of integrity, which we consider in the next section.

Integrity

Examples of integrity failures are easy to find. A number of years ago a malicious macro in a Word document inserted the word "not" after some random instances of the word "is;" you can imagine the havoc that ensued. Because the document was generally syntactically correct, people did not immediately detect the change. In another case, a model of the Pentium computer chip produced an incorrect result in certain circumstances of floating-point arithmetic. Although the circumstances of failure were rare, Intel decided to manufacture and replace the chips. Many of us receive mail that is misaddressed because someone typed something wrong when transcribing from a written list. A worse situation occurs when that inaccuracy is propagated to other mailing lists such that we can never seem to correct the root of the problem. Other times we find that a spreadsheet seems to be wrong, only to find that someone typed "space 123" in a cell, changing it from a numeric value to text, so the spreadsheet program misused that cell in computation. Suppose someone converted numeric data to roman numerals: One could argue that IV is the same as 4, but IV would not be useful in most applications, nor would it be obviously meaningful to someone expecting 4 as an answer. These cases show some of the breadth of examples of integrity failures. Integrity is harder to pin down than confidentiality. As Stephen Welke and Terry Mayfield [WEL90, MAY91, NCS91a] point out, integrity means different things in different contexts. When we survey the way some people use the term, we find several different meanings. For example, if we say that we have preserved the integrity of an item, we may mean that the item is

• precise
• accurate
• unmodified
• modified only in acceptable ways
• modified only by authorized people
• modified only by authorized processes
• consistent
• internally consistent
• meaningful and usable

Integrity can also mean two or more of these properties. Welke and Mayfield recognize three particular aspects of integrity—authorized actions, separation and protection of resources, and error detection and correction. Integrity can be enforced in much the same way as can confidentiality: by rigorous control of who or what can access which resources in what ways.

Availability

A computer user's worst nightmare: You turn on the switch and the computer does nothing. Your data and programs are presumably still there, but you cannot get at them. Fortunately, few of us experience that failure. Many of us do experience overload, however: access gets slower and slower; the computer responds but not in a way we consider normal or acceptable.

Availability applies both to data and to services (that is, to information and to information processing), and it is similarly complex. As with the notion of confidentiality, different people expect availability to mean different things. For example, an object or service is thought to be available if the following are true:

• It is present in a usable form.
• It has enough capacity to meet the service's needs.
• It is making clear progress, and, if in wait mode, it has a bounded waiting time.
• The service is completed in an acceptable period of time.

We can construct an overall description of availability by combining these goals.

Following are some criteria to define availability.

• There is a timely response to our request.
• Resources are allocated fairly so that some requesters are not favored over others.
• Concurrency is controlled; that is, simultaneous access, deadlock management, and exclusive access are supported as required.
• The service or system involved follows a philosophy of fault tolerance, whereby hardware or software faults lead to graceful cessation of service or to work-arounds rather than to crashes and abrupt loss of information. (Cessation does mean end; whether it is graceful or not, ultimately the system is unavailable. However, with fair warning of the system's stopping, the user may be able to move to another system and continue work.)
• The service or system can be used easily and in the way it was intended to be used. (This is a characteristic of usability, but an unusable system may also cause an availability failure.)

As you can see, expectations of availability are far-reaching. In Figure 1-7 we depict some of the properties with which availability overlaps. Indeed, the security community is just beginning to understand what availability implies and how to ensure it.

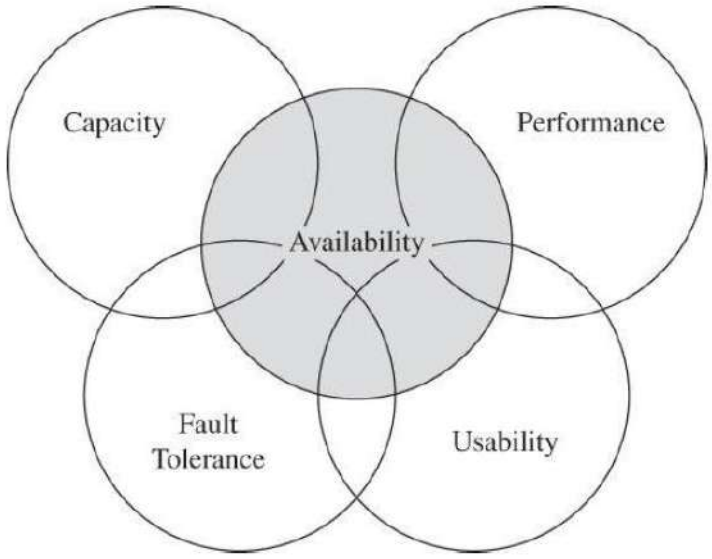


FIGURE 1-7 Availability and Related Aspects

A person or system can do three basic things with a data item: view it, modify it, or use it. Thus, viewing (confidentiality), modifying (integrity), and using (availability) are the basic modes of access that computer security seeks to preserve.

Computer security seeks to prevent unauthorized viewing (confidentiality) or modification (integrity) of data while preserving access (availability).

A paradigm of computer security is **access control**: To implement a policy, computer security controls all accesses by all subjects to all protected objects in all modes of access. A small, centralized control of access is fundamental to preserving confidentiality and integrity, but it is not clear that a single access control point can enforce availability. Indeed, experts on dependability will note that single points of control can become single points of failure, making it easy for an attacker to destroy availability by disabling the single control point. Much of computer security's past success has focused on confidentiality and integrity; there are models of confidentiality and integrity, for example, see David Bell and Leonard La Padula [BEL73, BEL76] and Kenneth Biba [BIB77]. Availability is security's next great challenge.

We have just described the C-I-A triad and the three fundamental security properties it represents. Our description of these properties was in the context of things that need protection. To motivate your understanding we gave some examples of harm and threats to cause harm. Our next step is to think about the nature of threats themselves.

Types of Threats

For some ideas of harm, look at Figure 1-8, taken from Willis Ware's report [WAR70]. Although it was written when computers were so big, so expensive, and so difficult to operate that only large organizations like universities, major corporations, or government departments would have one, Ware's discussion is still instructive today. Ware was concerned primarily with the protection of classified data, that is, preserving confidentiality. In the figure, he depicts humans such as programmers and maintenance staff gaining access to data, as well as radiation by which data can escape as signals. From the figure you can see some of the many kinds of threats to a computer system.

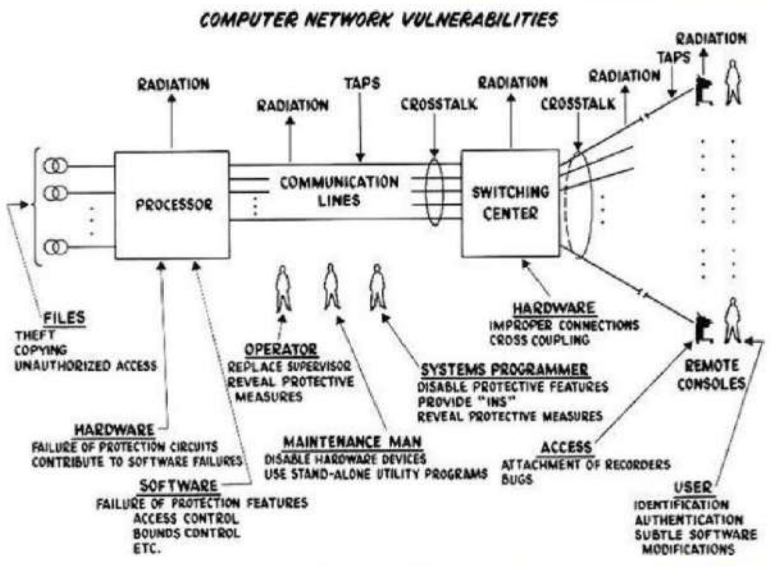


FIGURE 1-8 Computer [Network] Vulnerabilities (from [WAR70])

FIGURE 1-8 Computer [Network] Vulnerabilities (from [WAR70]) One way to analyze harm is to consider the cause or source. We call a potential cause of harm a **threat**. Harm can be caused by either nonhuman events or humans. Examples of **nonhuman threats** include natural disasters like fires or floods; loss of electrical power; failure of a component such as a communications cable, processor chip, or disk drive; or attack by a wild boar.

Threats are caused both by human and other sources.

Human threats can be either benign (nonmalicious) or malicious. **Nonmalicious** kinds of harm include someone's accidentally spilling a soft drink on a laptop, unintentionally deleting text, inadvertently sending an email message to the wrong person, and carelessly typing "12" instead of "21" when entering a phone number or clicking "yes" instead of "no" to overwrite a file. These inadvertent, human errors happen to most people; we just hope that the seriousness of harm is not too great, or if it is, that we will not repeat the mistake.

Threats can be malicious or not.

Most computer security activity relates to **malicious**, **human-caused harm**: A malicious person actually wants to cause harm, and so we often use the term *attack* for a malicious computer security event. Malicious attacks can be random or directed. In a **random attack** the attacker wants to harm any computer or user; such an attack is analogous to accosting the next pedestrian who walks down the street. An example of a random attack is malicious code posted on a website that could be visited by anybody. In a **directed attack**, the attacker intends harm to specific computers, perhaps at one organization (think of attacks against a political organization) or belonging to a specific individual (think of trying to drain a specific person's bank account, for example, by impersonation). Another class of directed attack is against a particular product, such as any computer running a particular browser. (We do not want to split hairs about whether such an attack is directed—at that one software product—or random, against any user of that product; the point is not semantic perfection but protecting against the attacks.) The range of possible directed attacks is practically unlimited. Different kinds of threats are shown in Figure 1-9.

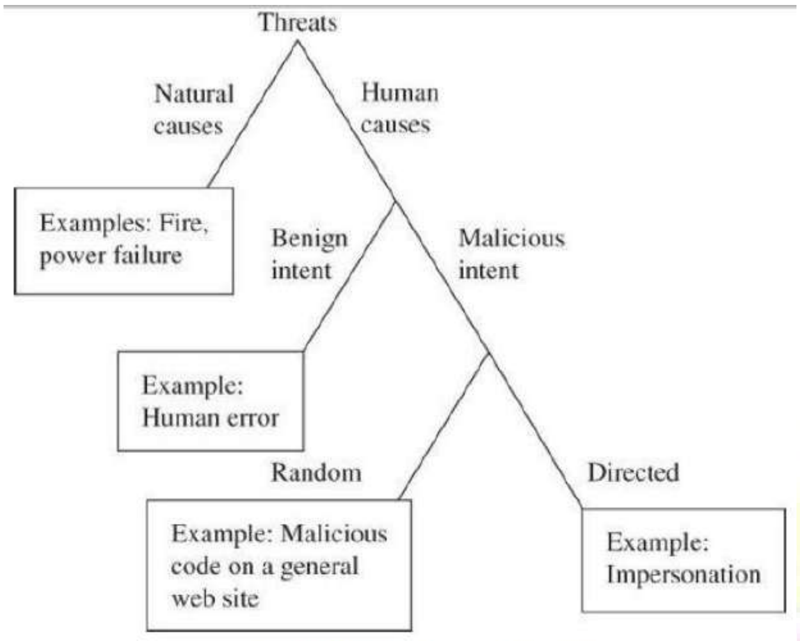


FIGURE 1-9 Kinds of Threats

Threats can be targeted or random.
Although the distinctions shown in Figure 1-9 seem clear-cut, sometimes the nature of an attack is not obvious until the attack is well underway, or perhaps even ended. A normal hardware failure can seem like a directed, malicious attack to deny access, and hackers often try to conceal their activity to look like ordinary, authorized users. As computer security experts we need to anticipate what bad things might happen, instead of waiting for the attack to happen or debating whether the attack is intentional or accidental.
Neither this book nor any checklist or method can show you *all* the kinds of harm that can happen to computer assets. There are too many ways to interfere with your use of these assets. Two retrospective lists of *known* vulnerabilities are of interest, however. The Common Vulnerabilities and Exposures (CVE) list (see http://cve.mitre.org/) is a dictionary of publicly known security vulnerabilities and exposures. CVE's common identifiers enable data exchange between security products and provide a baseline index point for evaluating coverage of security tools and services. To measure the extent of harm, the Common Vulnerability Scoring System (CVSS) (see http://nvd.nist.gov/cvss.cfm) provides a standard measurement system that allows accurate and consistent scoring of vulnerability impact.

Advanced Persistent Threat

Security experts are becoming increasingly concerned about a type of threat called **advanced persistent threat**. A lone attacker might create a random attack that snares a few, or a few million, individuals, but the resulting impact is limited to what that single attacker can organize and manage. A collection of attackers—think, for example, of the cyber equivalent of a street gang or an organized crime squad—might work together to purloin credit card numbers or similar financial assets to fund other illegal activity. Such attackers tend to be opportunistic, picking unlucky victims' pockets and moving on to other activities. Advanced persistent threat attacks come from organized, well financed, patient assailants. Often affiliated with governments or quasi-governmental groups, these attackers engage in long term campaigns. They carefully select their targets, crafting attacks that appeal to specifically those targets; email messages called spear phishing (described in Chapter 4) are intended to seduce their recipients. Typically the attacks are silent, avoiding any obvious impact that would alert a victim, thereby allowing the attacker to exploit the victim's access rights over a long time.
The motive of such attacks is sometimes unclear. One popular objective is economic espionage. A series of attacks, apparently organized and supported by the Chinese government, was used in 2012 and 2013 to obtain product designs from aerospace companies in the United States. There is evidence the stub of the attack code was loaded into victim machines long in advance of the attack; then, the attackers installed the more complex code and extracted the desired data. In May 2014 the Justice Department indicted five Chinese hackers in absentia for these attacks. In the summer of 2014 a series of attacks against J.P. Morgan Chase bank and up to a dozen similar financial institutions allowed the assailants access to 76 million names, phone numbers, and email addresses. The attackers—and even their country of origin— remain unknown, as does the motive. Perhaps the attackers wanted more sensitive financial data, such as account numbers or passwords, but were only able to get the less valuable contact information. It is also not

known if this attack was related to an attack a year earlier that disrupted service to that bank and several others.
To imagine the full landscape of possible attacks, you may find it useful to consider the kinds of people who attack computer systems. Although potentially anyone is an attacker, certain classes of people stand out because of their backgrounds or objectives. Thus, in the following sections we look at profiles of some classes of attackers.

Computer criminals

Who are attackers? As we have seen, their motivations range from chance to a specific target. Putting aside attacks from natural and benign causes, we can explore who the attackers are and what motivates them.
Most studies of attackers actually analyze computer criminals, that is, people who have actually been convicted of a crime, primarily because that group is easy to identify and study. The ones who got away or who carried off an attack without being detected may have characteristics different from those of the criminals who have been caught. Worse, by studying only the criminals we have caught, we may not learn how to catch attackers who know how to abuse the system without being apprehended.
What does a cyber criminal look like? In television and films the villains wore shabby clothes, looked mean and sinister, and lived in gangs somewhere out of town. By contrast, the sheriff dressed well, stood proud and tall, was known and respected by everyone in town, and struck fear in the hearts of most criminals.
To be sure, some computer criminals are mean and sinister types. But many more wear business suits, have university degrees, and appear to be pillars of their communities. Some are high school or university students. Others are middle-aged business executives. Some are mentally deranged, overtly hostile, or extremely committed to a cause, and they attack computers as a symbol. Others are ordinary people tempted by personal profit, revenge, challenge, advancement, or job security—like perpetrators of any crime, using a computer or not. Researchers have tried to find the psychological traits that distinguish attackers, as described in Sidebar 1-1. These studies are far from conclusive, however, and the traits they identify may show correlation but not necessarily causality. To appreciate this point, suppose a study found that a disproportionate number of people convicted of computer crime were left-handed. Does that result imply that all left-handed people are computer criminals or that only left-handed people are? Certainly not. No single profile captures the characteristics of a "typical" computer attacker, and the characteristics of some notorious attackers also match many people who are not attackers. As shown in Figure 1-10, attackers look just like anybody in a crowd.

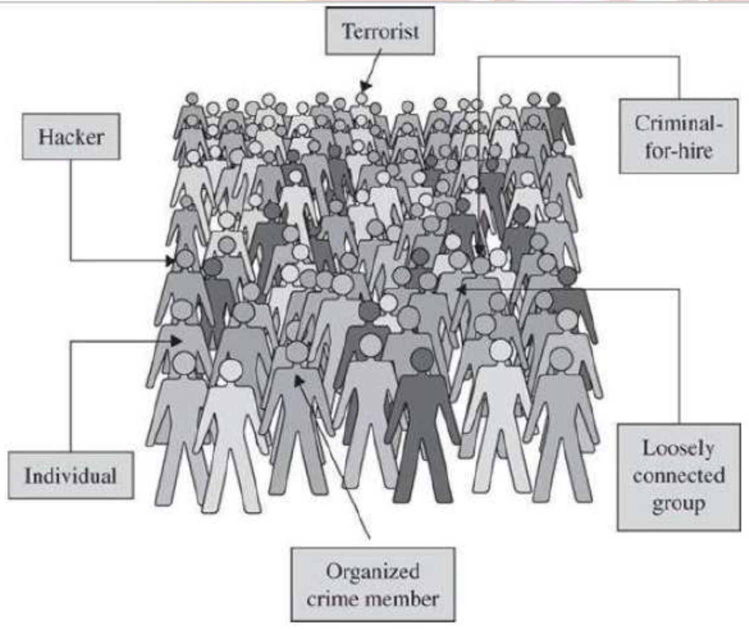


FIGURE 1-10 Attackers

No one pattern matches all attackers.
Sidebar 1-1 An Attacker's Psychological Profile?
Temple Grandin, a professor of animal science at Colorado State University and a sufferer from a mental disorder called Asperger syndrome (AS), thinks that Kevin Mitnick and several other widely described hackers show classic symptoms of Asperger syndrome. Although quick to point out that no research has established a link between AS and hacking, Grandin notes similar behavior traits among Mitnick, herself, and other AS sufferers. An article in *USA Today* (29 March 2001) lists the following AS traits:

• poor social skills, often associated with being loners during childhood; the classic "computer nerd"
• fidgeting, restlessness, inability to make eye contact, lack of response to cues in social interaction, such as facial expressions or body language
• exceptional ability to remember long strings of numbers
• ability to focus on a technical problem intensely and for a long time, although easily distracted on other problems and unable to manage several tasks at once
• deep honesty and respect for laws

Donn Parker [PAR98] has studied hacking and computer crime for many years. He states "hackers are characterized by an immature, excessively idealistic attitude … They delight in presenting themselves to the media as idealistic do-gooders, champions of the underdog."

Consider the following excerpt from an interview [SHA00] with "Mixter," the German programmer who admitted he was the author of a widespread piece of attack software called Tribal Flood Network (TFN) and its sequel TFN2K:

Q: Why did you write the software?

A: I first heard about Trin00 [another piece of attack software] in July '99 and I considered it as interesting from a technical perspective, but also potentially powerful in a negative way. I knew some facts of how Trin00 worked, and since I didn't manage to get Trin00 sources or binaries at that time, I wrote my own server-client network that was capable of performing denial of service.

Q: Were you involved … in any of the recent high-profile attacks?

A: No. The fact that I authored these tools does in no way mean that I condone their active use. I must admit I was quite shocked to hear about the latest attacks. It seems that the attackers are pretty clueless people who misuse powerful resources and tools for generally harmful and senseless activities just "because they can."

Notice that from some information about denial-of-service attacks, he wrote his own server-client network and then a sophisticated attack. But he was "quite shocked" to hear they were used for harm.

More research is needed before we can define the profile of a hacker. And even more work will be needed to extend that profile to the profile of a (malicious) attacker. Not all hackers become attackers; some hackers become extremely dedicated and conscientious system administrators, developers, or security experts. But some psychologists see in AS the rudiments of a hacker's profile.

Individuals

Originally, computer attackers were individuals, acting with motives of fun, challenge, or revenge. Early attackers acted alone. Two of the most well known among them are Robert Morris Jr., the Cornell University graduate student who brought down the Internet in 1988 [SPA89], and Kevin Mitnick, the man who broke into and stole data from dozens of computers, including the San Diego Supercomputer Center [MAR95].

Organized, Worldwide Groups

More recent attacks have involved groups of people. An attack against the government of the country of Estonia (described in more detail in Chapter 13) is believed to have been an uncoordinated outburst from a loose federation of attackers from around the world.

Kevin Poulsen [POU05] quotes Tim Rosenberg, a research professor at George Washington University, warning of "multinational groups of hackers backed by organized crime" and showing the sophistication of prohibition-era mobsters. He also reports that Christopher Painter, deputy director of the U.S. Department of Justice's computer crime section, argues that cyber criminals and serious fraud artists are increasingly working in concert or are one and the same. According to Painter, loosely connected groups of criminals all over the world work together to break into systems and steal and sell information, such as credit card numbers. For instance, in October 2004, U.S. and Canadian authorities arrested 28 people from 6 countries involved in an international, organized cybercrime ring to buy and sell credit card information and identities. Whereas early motives for computer attackers such as Morris and Mitnick were personal, such as prestige or accomplishment, recent attacks have been heavily influenced by financial gain. Security firm McAfee reports "Criminals have realized the huge financial gains to be made from the Internet with little risk. They bring the skills, knowledge, and connections needed for large scale, high-value criminal enterprise that, when combined with computer skills, expand the scope and risk of cybercrime." [MCA05]

Organized Crime
Attackers' goals include fraud, extortion, money laundering, and drug trafficking, areas in which organized crime has a well-established presence. Evidence is growing that organized crime groups are engaging in computer crime. In fact, traditional criminals are recruiting hackers to join the lucrative world of cybercrime. For example, Albert Gonzales was sentenced in March 2010 to 20 years in prison for working with a crime ring to steal 40 million credit card numbers from retailer TJMaxx and others, costing over $200 million (*Reuters*, 26 March 2010).
Organized crime may use computer crime (such as stealing credit card numbers or bank account details) to finance other aspects of crime. Recent attacks suggest that professional criminals have discovered just how lucrative computer crime can be. Mike Danseglio, a security project manager with Microsoft, said, "In 2006, the attackers want to pay the rent. They don't want to write a worm that destroys your hardware. They want to assimilate your computers and use them to make money." [NAR06a] Mikko Hyppönen, Chief Research Officer with Finnish security company f-Secure, agrees that today's attacks often come from Russia, Asia, and Brazil; the motive is now profit, not fame [BRA06]. Ken Dunham, Director of the Rapid Response Team for VeriSign says he is "convinced that groups of well-organized mobsters have taken control of a global billion-dollar crime network powered by skillful hackers." [NAR06b]
Organized crime groups are discovering that computer crime can be lucrative.
McAfee also describes the case of a hacker-for-hire: a businessman who hired a 16-year-old New Jersey hacker to attack the websites of his competitors. The hacker barraged the site for a five-month period and damaged not only the target companies but also their Internet service providers (ISPs) and other unrelated companies that used the same ISPs.
By FBI estimates, the attacks cost all the companies over $2 million; the FBI arrested both hacker and businessman in March 2005 [MCA05].
Brian Snow [SNO05] observes that hackers want a score or some kind of evidence to give them bragging rights. Organized crime wants a resource; such criminals want to stay under the radar to be able to extract profit from the system over time. These different objectives lead to different approaches to computer crime: The novice hacker can use a crude attack, whereas the professional attacker wants a neat, robust, and undetectable method that can deliver rewards for a long time.
Terrorists
The link between computer security and terrorism is quite evident. We see terrorists using computers in four ways:
• *Computer as target of attack*: Denial-of-service attacks and website defacements are popular activities for any political organization because they attract attention to the cause and bring undesired negative attention to the object of the attack. An example is the massive denial-of-service attack launched against the country of Estonia, detailed in Chapter 13.
• *Computer as method of attack*: Launching offensive attacks requires the use of computers. Stuxnet, an example of malicious computer code called a worm, is known to attack automated control systems, specifically a model of control system manufactured by Siemens. Experts say the code is designed to disable machinery used in the control of nuclear reactors in Iran [MAR10]. The persons behind the attack are unknown, but the infection is believed to have spread through USB flash drives brought in by engineers maintaining the computer controllers. (We examine the Stuxnet worm in more detail in Chapters 6 and 13.)
• *Computer as enabler of attack*: Websites, web logs, and email lists are effective, fast, and inexpensive ways to allow many people to coordinate.
According to the Council on Foreign Relations, the terrorists responsible for the November 2008 attack that killed over 200 people in Mumbai used GPS systems to guide their boats, Blackberries for their communication, and Google Earth to plot their routes.
• *Computer as enhancer of attack*: The Internet has proved to be an invaluable means for terrorists to spread propaganda and recruit agents. In October 2009 the FBI arrested Colleen LaRose, also known as JihadJane, after she had spent months using email, YouTube, MySpace, and electronic message boards to recruit radicals in Europe and South Asia to "wage violent jihad," according to a federal indictment.

We cannot accurately measure the degree to which terrorists use computers, because terrorists keep secret the nature of their activities and because our definitions and measurement tools are rather weak. Still, incidents like the one described in Sidebar 1-2 provide evidence that all four of these activities are increasing.

Sidebar 1-2 The Terrorists, Inc., IT Department

In 2001, a reporter for the *Wall Street Journal* bought a used computer in Afghanistan. Much to his surprise, he found that the hard drive contained what appeared to be files from a senior al Qaeda operative. The reporter, Alan Cullison [CUL04], reports that he turned the computer over to the FBI. In his story published in 2004 in *The Atlantic,* he carefully avoids revealing anything he thinks might be sensitive.

The disk contained over 1,000 documents, many of them encrypted with relatively weak encryption. Cullison found draft mission plans and white papers setting forth ideological and philosophical arguments for the attacks of 11 September 2001. Also found were copies of news stories on terrorist activities.

Some of the found documents indicated that al Qaeda was not originally interested in chemical, biological, or nuclear weapons, but became interested after reading public news articles accusing al Qaeda of having those capabilities.

Perhaps most unexpected were email messages of the kind one would find in a typical office: recommendations for promotions, justifications for petty cash expenditures, and arguments concerning budgets.

The computer appears to have been used by al Qaeda from 1999 to 2001. Cullison notes that Afghanistan in late 2001 was a scene of chaos, and it is likely the laptop's owner fled quickly, leaving the computer behind, where it fell into the hands of a secondhand goods merchant who did not know its contents.

But this computer's contents illustrate an important aspect of computer security and confidentiality: We can never predict the time at which a security disaster will strike, and thus we must always be prepared to act immediately if it suddenly happens. If someone on television sneezes, you do not worry about the possibility of catching a cold. But if someone standing next to you sneezes, you may become concerned. In the next section we examine the harm that can come from the presence of a computer security threat on your own computer systems.

1.3 Harm

The negative consequence of an actualized threat is **harm**; we protect ourselves against threats in order to reduce or eliminate harm. We have already described many examples of computer harm: a stolen computer, modified or lost file, revealed private letter, or denied access to data. These events cause harm that we want to avoid.

In our earlier discussion of assets, we noted that value depends on owner or outsider perception and need. Some aspects of value are immeasurable, such as the value of the paper you need to submit to your professor tomorrow; if you lose the paper (that is, if its availability is lost), no amount of money will compensate you for it. Items on which you place little or no value might be more valuable to someone else; for example, the group photograph taken at last night's party can reveal that your friend was not where he told his wife he would be. Even though it may be difficult to assign a specific number as the value of an asset, you can usually assign a value on a generic scale, such as moderate or minuscule or incredibly high, depending on the degree of harm that loss or damage to the object would cause. Or you can assign a value relative to other assets, based on comparable loss: This version of the file is more valuable to you than that version.

In their 2010 global Internet threat report, security firm Symantec surveyed the kinds of goods and services offered for sale on underground web pages. The item most frequently offered in both 2009 and 2008 was credit card numbers, at prices ranging from $0.85 to $30.00 each. (Compare those prices to an individual's effort to deal with the effect of a stolen credit card or the potential amount lost by the issuing bank.) Second most frequent was bank account credentials, at $15 to $850; these were offered for sale at 19% of websites in both years. Email accounts were next at $1 to $20, and lists of email addresses went for $1.70 to $15.00 per thousand. At position 10 in 2009 were website administration credentials, costing only $2 to $30. These black market websites demonstrate that the market price of computer assets can be dramatically different from their value to rightful owners.

The value of many assets can change over time, so the degree of harm (and therefore the severity of a threat) can change, too. With unlimited time, money, and capability, we might try to protect against all kinds of harm. But because our resources are limited, we must prioritize our protection, safeguarding only against serious threats and the ones we can control. Choosing the threats we try to mitigate involves a process called **risk management**, and it includes weighing the seriousness of a threat against our ability to protect.

Risk management involves choosing which threats to control and what resources to devote to protection.

Risk and Common Sense

The number and kinds of threats are practically unlimited because devising an attack requires an active imagination, determination, persistence, and time (as well as access and resources). The nature and number of threats in the computer world reflect life in general:

The causes of harm are limitless and largely unpredictable. Natural disasters like volcanoes and earthquakes happen with little or no warning, as do auto accidents, heart attacks, influenza, and random acts of violence. To protect against accidents or the flu, you might decide to stay indoors, never venturing outside. But by doing so, you trade one set of risks for another; while you are inside, you are vulnerable to building collapse. There are too many possible causes of harm for us to protect ourselves—or our computers— completely against all of them.

In real life we make decisions every day about the best way to provide our security. For example, although we may choose to live in an area that is not prone to earthquakes, we cannot entirely eliminate earthquake risk. Some choices are conscious, such as deciding not to walk down a dark alley in an unsafe neighborhood; other times our subconscious guides us, from experience or expertise, to take some precaution. We evaluate the likelihood and severity of harm, and then consider ways (called countermeasures or controls) to address threats and determine the controls' effectiveness. Computer security is similar. Because we cannot protect against everything, we prioritize: Only so much time, energy, or money is available for protection, so we address some risks and let others slide. Or we consider alternative courses of action, such as transferring risk by purchasing insurance or even doing nothing if the side effects of the countermeasure could be worse than the possible harm. The risk that remains uncovered by controls is called **residual risk**.

A basic model of risk management involves a user's calculating the value of all assets, determining the amount of harm from all possible threats, computing the costs of protection, selecting safeguards (that is, controls or countermeasures) based on the degree of risk and on limited resources, and applying the safeguards to optimize harm averted. This approach to risk management is a logical and sensible approach to protection, but it has significant drawbacks. In reality, it is difficult to assess the value of each asset; as we have seen, value can change depending on context, timing, and a host of other characteristics. Even harder is determining the impact of all possible threats. The range of possible threats is effectively limitless, and it is difficult (if not impossible in some situations) to know the short- and long-term impacts of an action. For instance, Sidebar 1-3 describes a study of the impact of security breaches over time on corporate finances, showing that a threat must be evaluated over time, not just at a single instance.

Sidebar 1-3 Short- and Long-term Risks of Security Breaches

It was long assumed that security breaches would be bad for business: that customers, fearful of losing their data, would veer away from insecure businesses and toward more secure ones. But empirical studies suggest that the picture is more complicated. Early studies of the effects of security breaches, such as that of Campbell [CAM03], examined the effects of breaches on stock price. They found that a breach's impact could depend on the nature of the breach itself; the effects were higher when the breach involved unauthorized access to confidential data. Cavusoglu et al. [CAV04] discovered that a breach affects the value not only of the company experiencing the breach but also of security enterprises: On average, the breached firms lost 2.1 percent of market value within two days of the breach's disclosure, but security developers' *market* value actually *increased* 1.36 percent.

Myung Ko and Carlos Dorantes [KO06] looked at the longer-term financial effects of publicly announced breaches. Based on the Campbell et al. study, they examined data for four quarters following the announcement of unauthorized access to confidential data. Ko and Dorantes note many types of possible breach-related costs:

“Examples of short-term costs include cost of repairs, cost of replacement of the system, lost business due to the disruption of business operations, and lost productivity of employees. These are also considered tangible costs. On the other hand, long-term costs include the loss of existing customers due to loss of trust, failing to attract potential future customers due to negative reputation from the breach, loss of business partners due to loss of trust, and potential legal liabilities from the breach. Most of these costs are intangible costs that are difficult to calculate but extremely important in assessing the overall security breach costs to the organization.”

Ko and Dorantes compared two groups of companies: one set (the treatment group) with data breaches, and the other (the control group) without a breach but matched for size and industry. Their findings were striking. Contrary to what you might suppose, the breached firms had no decrease in performance for the quarters following the breach, but their return on assets decreased in the third quarter. The comparison of treatment with control companies revealed that the control firms generally outperformed the breached firms. However, the breached firms outperformed the control firms in the fourth quarter.

These results are consonant with the results of other researchers who conclude that there is minimal long-term economic impact from a security breach. There are many reasons why this is so. For example, customers may think that all competing firms have the same vulnerabilities and threats, so changing to another vendor does not reduce the risk. Another possible explanation may be a perception that a breached company has better security since the breach forces the company to strengthen controls and thus reduce the likelihood of similar breaches. Yet another explanation may simply be the customers’ short attention span; as time passes, customers forget about the breach and return to business as usual.

All these studies have limitations, including small sample sizes and lack of sufficient data. But they clearly demonstrate the difficulties of quantifying and verifying the impacts of security risks, and point out a difference between shortand long-term effects.

Although we should not apply protection haphazardly, we will necessarily protect against threats we consider most likely or most damaging. For this reason, it is essential to understand how we perceive threats and evaluate their likely occurrence and impact.

Sidebar 1-4 summarizes some of the relevant research in risk perception and decisionmaking. Such research suggests that, for relatively rare instances such as high-impact security problems, we must take into account the ways in which people focus more on the impact than on the actual likelihood of occurrence.

Sidebar 1-4 Perception of the Risk of Extreme Events

When a type of adverse event happens frequently, we can calculate its likelihood and impact by examining both frequency and nature of the collective set of events. For instance, we can calculate the likelihood that it will rain this week and take an educated guess at the number of inches of precipitation we will receive; rain is a fairly frequent occurrence. But security problems are often extreme events: They happen infrequently and under a wide variety of circumstances, so it is difficult to look at them as a group and draw general conclusions.

Paul Slovic’s work on risk addresses the particular difficulties with extreme events. He points out that evaluating risk in such cases can be a political endeavor as much as a scientific one. He notes that we tend to let values, process, power, and trust influence our risk analysis [SLO99].

Beginning with Fischoff et al. [FIS78], researchers characterized extreme risk along two perception-based axes: the dread of the risk and the degree to which the risk is unknown. These feelings about risk, called *affects* by psychologists, enable researchers to discuss relative risks by placing them on a plane defined by the two perceptions as axes. A study by Loewenstein et al. [LOE01] describes how risk perceptions are influenced by association (with events already experienced) and by affect at least as much if not more than by reason. In fact, if the two influences compete, feelings usually trump reason. This characteristic of risk analysis is reinforced by prospect theory: studies of how people make decisions by using reason and feeling. Kahneman and Tversky [KAH79] showed that people tend to overestimate the likelihood of rare, unexperienced events because their feelings of dread and the unknown usually dominate analytical reasoning about the low likelihood of occurrence. By contrast, if people experience similar outcomes and their likelihood, their feeling of dread diminishes and they can actually underestimate rare events. In other words, if the impact of a rare event is high (high dread), then people focus on the impact, regardless of the likelihood. But if the impact of a rare event issmall, then they pay attention to the likelihood.

Let us look more carefully at the nature of a security threat. We have seen that one aspect—its potential harm—is the amount of damage it can cause; this aspect is the **impact** component of the risk. We also consider the magnitude of the threat's **likelihood**.

A likely threat is not just one that someone might want to pull off but rather one that could actually occur. Some people might daydream about getting rich by robbing a bank; most, however, would reject that idea because of its difficulty (if not its immorality or risk). One aspect of likelihood is feasibility: Is it even possible to accomplish the attack? If the answer is no, then the likelihood is zero, and therefore so is the risk. So a good place to start in assessing risk is to look at whether the proposed action is feasible. Three factors determine feasibility, as we describe next.

Spending for security is based on the impact and likelihood of potential harm—both of which are nearly impossible to measure precisely.

Method–Opportunity–Motive

A malicious attacker must have three things to ensure success: method, opportunity, and motive, depicted in Figure 1-11. Roughly speaking, method is the how; opportunity, the when; and motive, the why of an attack. Deny the attacker any of those three and the attack will not succeed. Let us examine these properties individually.

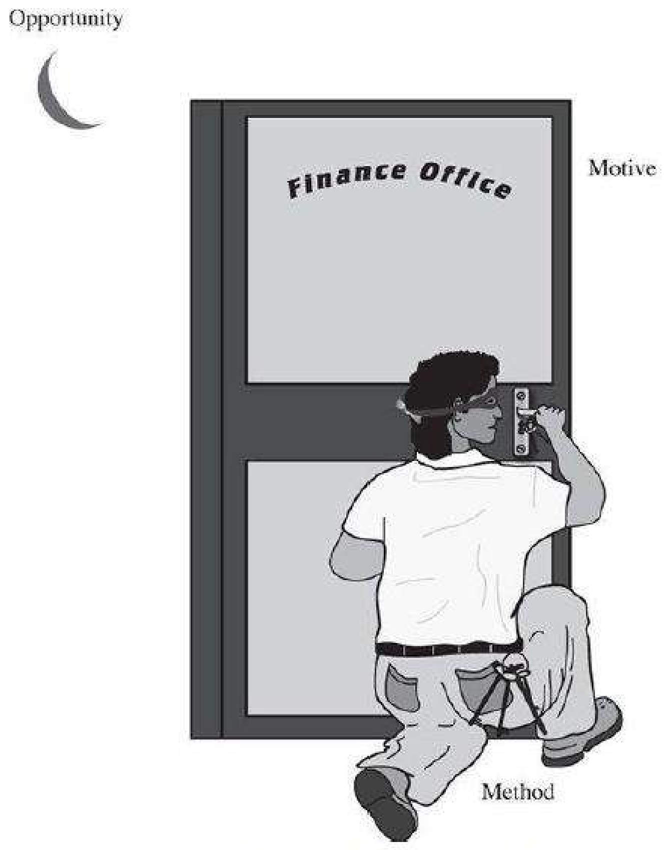


FIGURE 1-11 Method–Opportunity–Motive

Method

By **method** we mean the skills, knowledge, tools, and other things with which to perpetrate the attack. Think of comic figures that want to do something, for example, to steal valuable jewelry, but the characters are so inept that their every move is doomed to fail. These people lack the capability or method to succeed, in part because there are no classes in jewel theft or books on burglary for dummies. Anyone can find plenty of courses and books about computing, however. Knowledge of specific models of computer systems is widely available in bookstores and on the Internet. Mass-market systems (such as the Microsoft or Apple or Unix operating systems) are readily available for purchase, as are common software products, such as word processors or database management systems, so potential attackers can even get hardware and software on which to experiment and perfect an attack. Some manufacturers release detailed specifications on how the system was designed or how it operates, as guides for users and integrators who want to implement other complementary products. Various attack tools—scripts, model programs, and tools to test for weaknesses—are available from hackers' sites on the Internet, to the degree that many attacks require only the attacker's ability to download and run a program. The term **script kiddie** describes someone who downloads a complete attack code package and needs only to enter a few details to identify the target and let the script perform the attack. Often, only time and inclination limit an attacker.

Opportunity

Opportunity is the time and access to execute an attack. You hear that a fabulous apartment has just become available, so you rush to the rental agent, only to find someone else rented it five minutes earlier. You missed your opportunity.

Many computer systems present ample opportunity for attack. Systems available to the public are, by definition, accessible; often their owners take special care to make them fully available so that if one hardware component fails, the owner has spares instantly ready to be pressed into service. Other people are oblivious to the need to protect their computers, so unattended laptops and unsecured network connections give ample opportunity for attack. Some systems have private or undocumented entry points for administration or maintenance, but attackers can also find and use those entry points to attack the systems.

Motive

Finally, an attacker must have a **motive** or reason to want to attack. You probably have ample opportunity and ability to throw a rock through your neighbor's window, but you do not. Why not? Because you have no reason to want to harm your neighbor: You lack motive.

We have already described some of the motives for computer crime: money, fame, selfesteem, politics, terror. It is often difficult to determine motive for an attack. Some places are "attractive targets," meaning they are very appealing to attackers. Popular targets include law enforcement and defense department computers, perhaps because they are presumed to be well protected against attack (so they present a challenge and a successful attack shows the attacker's prowess). Other systems are attacked because they are easy to attack. And some systems are attacked at random simply because they are there.

Method, opportunity, and motive are all necessary for an attack to succeed; deny any of these and the attack will fail.

By demonstrating feasibility, the factors of method, opportunity, and motive determine whether an attack can succeed. These factors give the advantage to the attacker because they are qualities or strengths the attacker must possess. Another factor, this time giving an advantage to the defender, determines whether an attack will succeed: The attacker needs a vulnerability, an undefended place to attack. If the defender removes vulnerabilities, the attacker cannot attack.

1.4 Vulnerabilities

As we noted earlier in this chapter, a **vulnerability** is a weakness in the security of the computer system, for example, in procedures, design, or implementation, that might be exploited to cause loss or harm. Think of a bank, with an armed guard at the front door, bulletproof glass protecting the tellers, and a heavy metal vault requiring multiple keys for entry. To rob a bank, you would have to think of how to exploit a weakness not covered by these defenses. For example, you might bribe a teller or pose as a maintenance worker.

Computer systems have vulnerabilities, too. In this book we consider many, such as weak authentication, lack of access control, errors in programs, finite or insufficient resources, and inadequate physical protection. Paired with a credible attack, each of these vulnerabilities can allow harm to confidentiality, integrity, or availability. Each attack vector seeks to exploit a particular vulnerability.

Vulnerabilities are weaknesses that can allow harm to occur.

Security analysts speak of a system's **attack surface**, which is the system's full set of vulnerabilities—actual and potential. Thus, the attack surface includes physical hazards, malicious attacks by outsiders, stealth data theft by insiders, mistakes, and impersonations.

Although such attacks range from easy to highly improbable, analysts must consider all possibilities.

Our next step is to find ways to block threats by neutralizing vulnerabilities.

In television and film westerns, the bad guys always wore shabby clothes, looked mean and sinister, and lived in gangs somewhere out of town. By contrast, the sheriff dressed well, stood proud and tall, was known and respected by everyone in town, and struck fear in the hearts of most criminals.

To be sure, some computer criminals are mean and sinister types. But many more wear business suits, have university degrees, and appear to be pillars of their communities. Some are high school or university students. Others are middle-aged business executives. Some are mentally deranged, overtly hostile, or extremely committed to a cause, and they attack computers as a symbol. Others are ordinary people tempted by personal profit, revenge, challenge, advancement, or job security. No single profile captures the characteristics of a "typical" computer criminal, and many who fit the profile are not criminals at all.

Whatever their characteristics and motivations, computer criminals have access to enormous amounts of hardware, software, and data; they have the potential to cripple much of effective business and government throughout the world. In a sense, then, the purpose of computer security is to prevent these criminals from doing damage.
For the purposes of studying computer security, we say computer crime is any crime involving a computer or aided by the use of one. Although this definition is admittedly broad, it allows us to consider ways to protect ourselves, our businesses, and our communities against those who use computers maliciously.
The U.S. Federal Bureau of Investigation regularly reports uniform crime statistics. The data do not separate computer crime from crime of other sorts. Moreover, many companies do not report computer crime at all, perhaps because they fear damage to their reputation, they are ashamed to have allowed their systems to be compromised, or they have agreed not to prosecute if the criminal will "go away." These conditions make it difficult for us to estimate the economic losses we suffer as a result of computer crime; our dollar estimates are really only vague suspicions. Still, the estimates, ranging from $300 million to $500 billion per year, tell us that it is important for us to pay attention to computer crime and to try to prevent it or at least to moderate its effects.
One approach to prevention or moderation is to understand who commits these crimes and why. Many studies have attempted to determine the characteristics of computer criminals. By studying those who have already used computers to commit crimes, we may be able in the future to spot likely criminals and prevent the crimes from occurring. In this section, we examine some of these characteristics.
Amateurs
Amateurs have committed most of the computer crimes reported to date. Most embezzlers are not career criminals but rather are normal people who observe a weakness in a security system that allows them to access cash or other valuables. In the same sense, mostcomputer criminals are ordinary computer professionals or users who, while doing their jobs, discover they have access to something valuable.
When no one objects, the amateur may start using the computer at work to write letters, maintain soccer league team standings, or do accounting. This apparently innocent time-stealing may expand until the employee is pursuing a business in accounting, stock portfolio management, or desktop publishing on the side, using the employer's computing facilities. Alternatively, amateurs may become disgruntled over some negative work situation (such as a reprimand or denial of promotion) and vow to "get even" with management by wreaking havoc on a computing installation.
Crackers or Malicious Hackers
System crackersa[2] often high school or university students, attempt to access computing facilities for which they have not been authorized. Cracking a computer's defenses is seen as the ultimate victimless crime. The perception is that nobody is hurt or even endangered by a little stolen machine time. Crackers enjoy the simple challenge of trying to log in, just to see whether it can be done. Most crackers can do their harm without confronting anybody, not even making a sound. In the absence of explicit warnings not to trespass in a system, crackers infer that access is permitted. An underground network of hackers helps pass along secrets of success; as with a jigsaw puzzle, a few isolated pieces joined together may produce a large effect. Others attack for curiosity, personal gain, or self-satisfaction. And still others enjoy causing chaos, loss, or harm. There is no common profile or motivation for these attackers.
[2] The security community distinguishes between a ahacker,a someone w ho (nonmaliciously) programs, manages, or uses computing systems, and a acracker,a someone w ho attempts to access computing systems for malicious purposes. Crackers are the aevildoers.a Now, *hacker* has come to be used outside security to mean both benign and malicious users.
Career Criminals
By contrast, the career computer criminal understands the targets of computer crime. Criminals seldom change fields from arson, murder, or auto theft to computing; more often, criminals begin as computer professionals who engage in computer crime, finding the prospects and payoff good. There is some evidence that organized crime and international groups are engaging in computer crime. Recently, electronic spies and information brokers have begun to recognize that trading in companies' or individuals' secrets can be lucrative.

Recent attacks have shown that organized crime and professional criminals have discovered just how lucrative computer crime can be. Mike Danseglio, a security project manager with Microsoft, said, "In 2006, the attackers want to pay the rent. They don't want to write a worm that destroys your hardware. They want to assimilate your computers and use them to make money" [NAR06a]. Mikko Hyppönen, Chief Research Officer with the Finnish security company f-Secure, agrees that today's attacks often come from Russia, Asia, and Brazil and the motive is now profit, not fame [BRA06]. Ken Dunham, Director of the Rapid Response Team for Verisign says he is "convinced that groups of well-organized mobsters have taken control of a global billion-dollar crime network powered by skillful hackers" [NAR06b].
Snow [SNO05] observes that a hacker wants a score, bragging rights. Organized crime wants a resource; they want to stay and extract profit from the system over time. These different objectives lead to different approaches: The hacker can use a quick-and-dirty attack, whereas the professional attacker wants a neat, robust, and undetected method.
As mentioned earlier, some companies are reticent to prosecute computer criminals. In fact, after having discovered a computer crime, the companies are often thankful if the criminal quietly resigns. In other cases, the company is (understandably) more concerned about protecting its assets and so it closes down an attacked system rather than gathering evidence that could lead to identification and conviction of the criminal. The criminal is then free to continue the same illegal pattern with another company.

Terrorists

The link between computers and terrorism is quite evident. We see terrorists using computers in three ways:

- *targets of attack:* denial-of-service attacks and web site defacements are popular for any political organization because they attract attention to the cause and bring undesired negative attention to the target of the attack.
- *propaganda vehicles:* web sites, web logs, and e-mail lists are effective, fast, and inexpensive ways to get a message to many people.
- *methods of attack:* to launch offensive attacks requires use of computers.

We cannot accurately measure the amount of computer-based terrorism because our definitions and measurement tools are rather weak. Still, there is evidence that all three of these activities are increasing. (For another look at terrorists' use of computers, see Sidebar 1-6.)
Pag

Method of defense Program Security:

we investigate the legal and ethical restrictions on computer-based crime. But unfortunately, computer crime is certain to continue for the foreseeable future. For this reason, we must look carefully at controls for preserving confidentiality, integrity, and availability. Sometimes these controls can prevent or mitigate attacks; other, less powerful methods can only inform us that security has been compromised, by detecting a breach as it happens or after it occurs. Harm occurs when a threat is realized against a vulnerability. To protect against harm, then, we can neutralize the threat, close the vulnerability, or both. The possibility for harm to occur is called risk. We can deal with harm in several ways. We can seek to x prevent it, by blocking the attack or closing the vulnerability x deter it, by making the attack harder but not impossible x deflect it, by making another target more attractive (or this one less so) x detect it, either as it happens or some time after the fact x recover from its effects.
Of course, more than one of these can be done at once. So, for example, we might try to prevent intrusions. But in case we do not prevent them all, we might install a detection device to warn of an imminent attack. And we should have in place incident response procedures to help in the recovery in case an intrusion does succeed.

Controls To consider the controls or countermeasures that attempt to prevent exploiting a computing system's vulnerabilities, we begin by thinking about traditional ways to enhance physical security. In the Middle Ages, castles and fortresses were built to protect the people and valuable property inside. The fortress might have had one or more security characteristics, including x a strong gate or door, to repel invaders x heavy walls to withstand objects thrown or projected against them x a surrounding moat, to control access x arrow slits, to let archers shoot at approaching enemies x crenellations to allow inhabitants to lean out from the roof and pour hot or vile liquids on attackers x a drawbridge to limit access to authorized people x gatekeepers to verify that only authorized people and goods could enter Similarly, today we use a multipronged approach to protect our homes and offices. We may combine strong locks on the doors with a burglar alarm, reinforced windows, and even a nosy neighbor to keep an eye on our valuables. In each case, we select one or more ways to deter an intruder or attacker, and we base our selection not only on the value of what we protect but also on the effort we think an attacker or intruder will expend to get inside. Computer security has the same characteristics. We have many controls at our disposal. Some are easier than others to use or implement. Some are cheaper than others to use or implement. And some are more difficult than others for intruders to override. Figure 1-6 illustrates how we use a combination of controls to secure our valuable resources. We use one or more controls, according to what we are protecting, how the cost of protection compares with the risk of loss, and how hard we think intruders will work to get what they want.

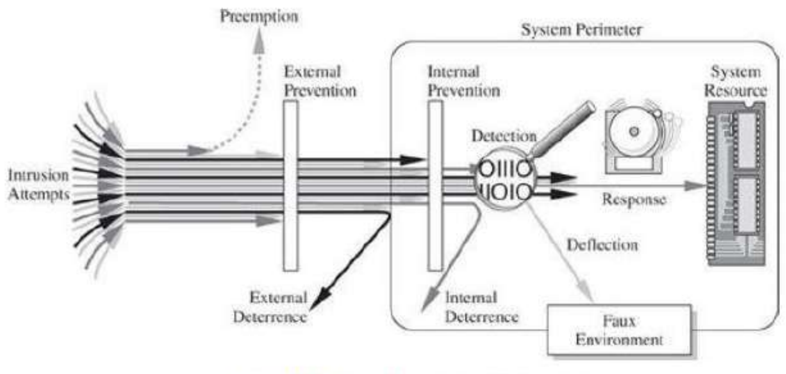


FIGURE 1-12 Effects of Controls

In this section, we present an overview of the controls available to us. In later chapters, we examine each control in much more detail. Encryption We noted earlier that we seek to protect hardware, software, and data. We can make it particularly hard for an intruder to find data useful if we somehow scramble the data so that interpretation is meaningless without the intruder's knowing how the scrambling was done. Indeed, the most powerful tool in providing computer security is this scrambling or encoding. Encryption is the formal name for the scrambling process. We take data in their normal, unscrambled state, called cleartext, and transform them so that they are unintelligible to the outside observer; the transformed data are called enciphered text or ciphertext. Using encryption, security professionals can virtually nullify the value of an interception and the possibility of effective modification or fabrication. In Chapters 2 and 12 we study many ways of devising and applying these transformations. Encryption clearly addresses the need for confidentiality of data. Additionally, it can be used to

ensure integrity; data that cannot be read generally cannot easily be changed in a meaningful manner. Furthermore, as we see throughout this book, encryption is the basis of protocols that enable us to provide security while accomplishing an important system or network task. A protocol is an agreed-on sequence of actions that leads to a desired result. For example, some operating system protocols ensure availability of resources as different tasks and users request them. Thus, encryption can also be thought of as supporting availability. That is, encryption is at the heart of methods for ensuring all aspects of computer security. Although encryption is an important tool in any computer security tool kit, we should not overrate its importance. Encryption does not solve all computer security problems, and other tools must complement its use. Furthermore, if encryption is not used properly, it may have no effect on security or could even degrade the performance of the entire system. Weak encryption can actually be worse than no encryption at all, because it gives users an unwarranted sense of protection. Therefore, we must understand those situations in which encryption is most useful as well as ways to use it effectively.
Software Controls If encryption is the primary way of protecting valuables, programs themselves are the second facet of computer security. Programs must be secure enough to prevent outside attack. They must also be developed and maintained so that we can be confident of the programs' dependability. Program controls include the following: x internal program controls: parts of the program that enforce security restrictions, such as access limitations in a database management program x operating system and network system controls: limitations enforced by the operating system or network to protect each user from all other users x independent control programs: application programs, such as password checkers, intrusion detection utilities, or virus scanners, that protect against certain types of vulnerabilities x development controls: quality standards under which a program is designed, coded, tested, and maintained to prevent software faults from becoming exploitable vulnerabilities We can implement software controls by using tools and techniques such as hardware components, encryption, or information gathering. Software controls frequently affect users directly, such as when the user is interrupted and asked for a password before being given access to a program or data. For this reason, we often think of software controls when we think of how systems have been made secure in the past. Because they influence the way users interact with a computing system, software controls must be carefully designed. Ease of use and potency are often competing goals in the design of a collection of software controls.

Hardware Controls

Numerous hardware devices have been created to assist in providing computer security. These devices include a variety of means, such as x hardware or smart card implementations of encryption x locks or cables limiting access or deterring theft x devices to verify users' identities x firewalls x intrusion detection systems x circuit boards that control access to storage media Policies and Procedures Sometimes, we can rely on agreed-on procedures or policies among users rather than enforcing security through hardware or software means. In fact, some of the simplest controls, such as frequent changes of passwords, can be achieved at essentially no cost but with tremendous effect. Training and administration follow immediately after establishment of policies, to reinforce the importance of security policy and to ensure their proper use. We must not forget the value of community standards and expectations when we consider how to enforce security. There are many acts that most thoughtful people would consider harmful, and we can leverage this commonality of belief in our policies. For this reason, legal and ethical controls are an important part of computer security. However, the law is slow to evolve, and the technology involving computers has emerged relatively suddenly. Although legal protection is necessary and desirable, it may not be as dependable in this area as it would be when applied to more well-understood and long-standing crimes. Society in general and the computing community in particular have not adopted formal standards of ethical behavior. As we see in Chapter 11, some organizations have devised codes of ethics for computer professionals. However, before codes of ethics can become widely accepted and effective, the computing community and the general public must discuss and make clear what kinds of behavior are inappropriate and why. Physical Controls Some of the easiest, most effective, and least expensive controls are physical controls. Physical controls include locks on doors, guards at entry points, backup copies of important software and data, and physical site planning that reduces the risk of natural disasters. Often the simple physical controls are overlooked while we seek more sophisticated approaches. Effectiveness of Controls Merely having controls does no good unless they are used properly. Let us consider several aspects that can enhance the effectiveness of controls. Awareness of Problem People using controls must be convinced of the need for security. That is, people will willingly cooperate with security requirements only if they understand why security is appropriate in a given situation. However, many users are unaware of the need for security, especially in situations in which a group has recently undertaken a computing task that was previously performed with lax or no apparent security. Likelihood of Use Of course, no control is effective unless it is used. The lock on a computer room door does no good if people block the door open. As Sidebar 1-7 tells, some computer systems are seriously uncontrolled.

Principle of Effectiveness:
Controls must be usedand used properlyto be effective. They must be efficient, easy to use, and appropriate. This principle implies that computer security controls must be efficient enough, in terms of time, memory space, human activity, or other resources used, that using the control does not seriously affect the task being protected. Controls should be selective so that they do not exclude legitimate accesses.

Overlapping Controls As we have seen with fortress or home security, several different controls may apply to address a single vulnerability. For example, we may choose to implement security for a microcomputer application by using a combination of controls on program access to the data, on physical access to the microcomputer and storage media, and even by file locking to control access to the processing programs. Periodic Review

Few controls are permanently effective. Just when the security specialist finds a way to secure assets against certain kinds of attacks, the opposition doubles its efforts in an attempt to defeat the security mechanisms. Thus, judging the effectiveness of a control is an ongoing task. (Sidebar 1-8 reports on periodic reviews of computer security.) Seldom, if ever, are controls perfectly effective. Controls fail, controls are incomplete, or people circumvent or misuse controls, for example. For that reason, we use overlapping controls, sometimes called a layered defense, in the expectation that one control will compensate for a failure of another. In some cases, controls do nicely complement each other. But two controls are not always better than one and, in some cases, two can even be worse than one. This brings us to another security principle. Principle of Weakest Link: Security can be no stronger than its weakest link. Whether it is the power supply that powers the firewall or the operating system under the security application or the human who plans, implements, and administers controls, a failure of any control can lead to a security failure.

Program Security

In the first two chapters, we learned about the need for computer security and we studied encryption, a fundamental tool in implementing many kinds of security controls. In this chapter, we begin to study how to apply security in computing. We start with *why* we need security at the program level and *how* we can achieve it.
In one form or another, protecting programs is at the heart of computer security because programs constitute so much of a computing system (the operating system, device drivers, the network infrastructure, database management systems and other applications, even executable commands on web pages). For now, we call all these pieces of code "programs." So we need to ask two important questions:
• How do we keep programs free from flaws?
• How do we protect computing resources against programs that contain flaws?
In later chapters, we examine particular types of programs including operating systems, database management systems, and network implementations and the specific kinds of security issues that are raised by the nature of their design and functionality.
In this chapter, we address more general themes, most of which carry forward to these special-purpose systems. Thus, this chapter not only lays the groundwork for future chapters but also is significant on its own.
This chapter deals with the writing of programs. It defers to a later chapter what may be a much larger issue in program security: trust. The trust problem can be framed as follows: Presented with a finished program, for example, a commercial software package, how can you tell how secure it is or how to use it in its most secure way? In part the answer to these questions is independent, third-party evaluations, presented for operating systems (but applicable to other programs, as well) in Chapter 5. The reporting and fixing of discovered flaws is discussed in Chapter 11, as are liability and software warranties. For now, however, the unfortunate state of commercial software development is largely a case of trust your source, and buyer beware.

Secure programs

Consider what we mean when we say that a program is "secure." We saw in Chapter 1 that security implies some degree of trust that the program enforces expected confidentiality, integrity, and availability. From the point of view of a program or a programmer, how can we look at a software component or code fragment and assess its security? This question is, of course, similar to the problem of assessing software quality in general. One way to assess security or quality is to ask people to name the characteristics of software that contribute to its overall security. However, we are likely to get different answers from different people. This difference occurs because the importance of the characteristics depends on who is analyzing the software. For example, one person may decide that code is secure because it takes too long to break through its security controls. And someone else may decide code is secure if it has run for a period of time with no apparent failures. But a third person may decide that *any* potential fault in meeting security requirements makes code insecure.

An assessment of security can also be influenced by someone's general perspective on software quality. For example, if your manager's idea of quality is conformance to specifications, then she might consider the code secure if it meets security requirements, whether or not the requirements are complete or correct. This security view played a role when a major computer manufacturer delivered all its machines with keyed locks, since a keyed lock was written in the requirements. But the machines were not secure, because all locks were configured to use the same keya Thus, another view of security is fitness for purpose; in this view, the manufacturer clearly had room for improvement.

In general, practitioners often look at quantity and types of faults for evidence of a product's quality (or lack of it). For example, developers track the number of faults found in requirements, design, and code inspections and use them as indicators of the likely quality of the final product. Sidebar 3-1 explains the importance of separating the faultsthe causes of problemsfrom the failures, the effects of the faults.

Fixing Faults

One approach to judging quality in security has been fixing faults. You might argue that a module in which 100 faults were discovered and fixed is better than another in which only 20 faults were discovered and fixed, suggesting that more rigorous analysis and testing had led to the finding of the larger number of faults. *Au contraire,* challenges your friend: a piece of software with 100 discovered faults is inherently full of problems and could clearly have hundreds more waiting to appear. Your friend's opinion is confirmed by the software testing literature; software that has many faults early on is likely to have many others still waiting to be found.

Early work in computer security was based on the paradigm of "penetrate and patch," in which analysts searched for and repaired faults. Often, a top-quality "tiger team" would be convened to test a system's security by attempting to cause it to fail. The test was considered to be a "proof" of security; if the system withstood the attacks, it was considered secure. Unfortunately, far too often the proof became a counterexample, in which not just one but several serious security problems were uncovered. The problem discovery in turn led to a rapid effort to "patch" the system to repair or restore the security. (See Schell's analysis in [SCH79].) However, the patch efforts were largely useless, making the system less secure rather than more secure because they frequently introduced *new* faults. There are at least four reasons why.

- The pressure to repair a specific problem encouraged a narrow focus on the fault itself and not on its context. In particular, the analysts paid attention to the immediate cause of the failure and not to the underlying design or requirements faults.
- The fault often had nonobvious side effects in places other than the immediate area of the fault.
- Fixing one problem often caused a failure somewhere else, or the patch addressed the problem in only one place, not in other related places.
- The fault could not be fixed properly because system functionality or performance would suffer as a consequence.

Unexpected Behavior

The inadequacies of penetrate-and-patch led researchers to seek a better way to be confident that code meets its security requirements. One way to do that is to compare the requirements with the behavior. That is, to understand program security, we can examine programs to see whether they behave as their designers intended or users expected. We call such unexpected behavior a program security flaw; it is inappropriate program behavior caused by a program vulnerability. Unfortunately, the terminology in the computer security field is not consistent with the IEEE standard described in Sidebar 3-1; the terms "vulnerability" and "flaw" do not map directly to the characterization of faults and failures. A flaw can be either a fault or failure, and a vulnerability usually describes a class of flaws, such as a buffer overflow. In spite of the inconsistency, it is important for us to remember that we must view vulnerabilities and flaws from two perspectives, cause and effect, so that we see what fault caused the problem and what failure (if any) is visible to the user. For example, a Trojan horse may have been injected in a piece of codea flaw exploiting a vulnerability but the user may not yet have seen the Trojan horse's malicious behavior.

Thus, we must address program security flaws from inside and outside, to find causes not only of existing failures but also of incipient ones. Moreover, it is not enough just to identify these problems. We must also determine how to prevent harm caused by possible flaws.

Program security flaws can derive from any kind of software fault. That is, they cover everything from a misunderstanding of program requirements to a one-character error in coding or even typing. The flaws can result from problems in a single code component or from the failure of several programs or program pieces to interact compatibly through a shared interface. The security flaws can reflect code that was intentionally designed or coded to be malicious or code that was simply developed in a sloppy or misguided way. Thus, it makes sense to divide program flaws into two separate logical categories: inadvertent human errors versus malicious, intentionally induced flaws.

These categories help us understand some ways to prevent the inadvertent and intentional insertion of flaws into future code, but we still have to address their effects, regardless of intention. That is, in the words of Sancho Panza in *Man of La Mancha*, "it doesn't matter whether the stone hits the pitcher or the pitcher hits the stone, it's going to be bad for the pitcher." An inadvertent error can cause just as much harm to users and their organizations as can an intentionally induced flaw. Furthermore, a system attack often exploits an unintentional security flaw to perform intentional damage. From reading the popular press (see Sidebar 3-2), you might conclude that intentional security incidents (called cyber attacks) are the biggest security threat today. In fact, plain, unintentional human errors are more numerous and cause much more damage.

Regrettably, we do not have techniques to eliminate or address all program security flaws. As Gasser [GAS88] notes, security *is* fundamentally hard, security often conflicts with usefulness and performance, there is no ""silver bullet" to achieve security effortlessly, and false security solutions impede real progress toward more secure programming. There are two reasons for this distressing situation.

1. Program controls apply at the level of the individual program and programmer. When we test a system, we try to make sure that the functionality prescribed in the requirements is implemented in the code. That is, we take a "should do" checklist and verify that the code does what it is supposed to do. However, security is also about *preventing* certain actions: a "shouldn't do" list. A system shouldn't do anything not on its "should do" list. It is almost impossible to ensure that a program does precisely what its designer or user intended, and nothing more. Regardless of designer or programmer intent, in a large and complex system, the pieces that have to fit together properly interact in an unmanageably large number of ways. We are forced to examine and test the code for typical or likely cases; we cannot exhaustively test every state and data combination to verify a system's behavior. So sheer size and complexity preclude total flaw prevention or mediation. Programmers intending to implant malicious code can take advantage of this incompleteness and hide some flaws successfully, despite our best efforts.

2. Programming and software engineering techniques change and evolve far more rapidly than do computer security techniques. So we often find ourselves trying to secure last year's technology while software developers are rapidly adopting today'sand next year'stechnology.

Still, the situation is far from bleak. Computer security has much to offer to program security.

By understanding what can go wrong and how to protect against it, we can devise techniques and tools to secure most computer applications.

Types of Flaws

To aid our understanding of the problems and their prevention or correction, we can define categories that distinguish one kind of problem from another. For example, Landwehr et al. [LAN94] present a taxonomy of program flaws, dividing them first into intentional and inadvertent flaws. They further divide intentional flaws into malicious and nonmalicious ones.

In the taxonomy, the inadvertent flaws fall into six categories:

- *validation* error (incomplete or inconsistent): permission checks
- *domain* error: controlled access to data
- *serialization* and *aliasing*: program flow order
- inadequate *identification and authentication*: basis for authorization

 boundary condition violation: failure on first or last case
 other exploitable *logic errors*
Other authors, such as Tsipenyuk et al. [TSI05], the OWASP project [OWA05], and Landwehr [LAN93], have produced similar lists. This list gives us a useful overview of the ways in which programs can fail to meet their security requirements. We leave our discussion of the pitfalls of identification and authentication for Chapter 4, in which we also investigate separation into execution domains. In this chapter, we address the other categories, each of which has interesting examples.

Non-malicious program errors-

Being human, programmers and other developers make many mistakes, most of which are unintentional and nonmalicious. Many such errors cause program malfunctions but do not lead to more serious security vulnerabilities. However, a few classes of errors have plagued programmers and security professionals for decades, and there is no reason to believe they will disappear. In this section we consider three classic error types that have enabled many recent security breaches. We explain each type, why it is relevant to security, and how it can be prevented or mitigated.

Buffer Overflows

A buffer overflow is the computing equivalent of trying to pour two liters of water into a one-liter pitcher: Some water is going to spill out and make a mess. And in computing, what a mess these errors have made. a

Definition

A buffer (or array or string) is a space in which data can be held. A buffer resides in memory.
Because memory is finite, a buffer's capacity is finite. For this reason, in many programming languages the programmer must declare the buffer's maximum size so that the compiler can set aside that amount of space.
Let us look at an example to see how buffer overflows can happen. Suppose a C language program contains the declaration:
char sample[10];
The compiler sets aside 10 bytes to store this buffer, one byte for each of the 10 elements of the array, sample[0] tHRough sample[9]. Now we execute the statement:
sample[10] = 'B';
The subscript is out of bounds (that is, it does not fall between 0 and 9), so we have a problem. The nicest outcome (from a security perspective) is for the compiler to detect the problem and mark the error during compilation. However, if the statement were
sample[i] = 'B';
we could not identify the problem until i was set during execution to a too-big subscript. It would be useful if, during execution, the system produced an error message warning of a subscript out of bounds. Unfortunately, in some languages, buffer sizes do not have to be predefined, so there is no way to detect an out-of-bounds error. More importantly, the code needed to check each subscript against its potential maximum value takes time and space during execution, and the resources are applied to catch a problem that occurs relatively infrequently. Even if the compiler were careful in analyzing the buffer declaration and use, this same problem can be caused with pointers, for which there is no reasonable way to define a proper limit. Thus, some compilers do not generate the code to check for exceeding bounds.
Let us examine this problem more closely. It is important to recognize that the potential overflow causes a serious problem only in some instances. The problem's occurrence depends on what is adjacent to the array sample. For example, suppose each of the ten elements of the array sample is filled with the letter A and the erroneous reference uses the letter B, as follows:
```
for (i=0; i<=9; i++)
sample[i] = 'A';
sample[10] = 'B'
```
All program and data elements are in memory during execution, sharing space with the operating system, other code, and resident routines. So there are four cases to consider in deciding where the 'B' goes, as shown in Figure 3-1. If the extra character overflows into

the user's data space, it simply overwrites an existing variable value (or it may be written into an as-yet unused location), perhaps affecting the program's result, but affecting no other program or data.

In the second case, the 'B' goes into the user's program area. If it overlays an already executed instruction (which will not be executed again), the user should perceive no effect. If it overlays an instruction that is not yet executed, the machine will try to execute an instruction with operation code 0x42, the internal code for the character 'B'. If there is no instruction with operation code 0x42, the system will halt on an illegal instruction exception.
Otherwise, the machine will use subsequent bytes as if they were the rest of the instruction, with success or failure depending on the meaning of the contents. Again, only the user is likely to experience an effect.
The most interesting cases occur when the system owns the space immediately after the array that overflows. Spilling over into system data or code areas produces similar results to those for the user's space: computing with a faulty value or trying to execute an improper operation.
Security Implication
In this section we consider program flaws from unintentional or nonmalicious causes.
Remember, however, that even if a flaw came from an honest mistake, the flaw can still cause serious harm. A malicious attacker can exploit these flaws.
Let us suppose that a malicious person understands the damage that can be done by a buffer overflow; that is, we are dealing with more than simply a normal, errant programmer. The malicious programmer looks at the four cases illustrated in Figure 3-1 and thinks deviously about the last two: What data values could the attacker insert just after the buffer to cause mischief or damage, and what planned instruction codes could the system be forced to execute? There are many possible answers, some of which are more malevolent than others.
Here, we present two buffer overflow attacks that are used frequently. (See [ALE96] for more details.)
First, the attacker may replace code in the system space. Remember that every program is invoked by the operating system and that the operating system may run with higher privileges than those of a regular program. Thus, if the attacker can gain control by masquerading as the operating system, the attacker can execute many commands in a powerful role.
Therefore, by replacing a few instructions right after returning from his or her own procedure, the attacker regains control from the operating system, possibly with raised privileges. If the buffer overflows into system code space, the attacker merely inserts overflow data that correspond to the machine code for instructions.
On the other hand, the attacker may make use of the stack pointer or the return register. Subprocedure calls are handled with a stack, a data structure in which the most recent item inserted is the next one removed (last arrived, first served). This structure works well because procedure calls can be nested, with each return causing control to transfer back to the immediately preceding routine at its point of execution. Each time a procedure is called, its parameters, the return address (the address immediately after its call), and other local values are pushed onto a stack. An old stack pointer is also pushed onto the stack, and a stack pointer register is reloaded with the address of these new values. Control is then transferred to the subprocedure.
As the subprocedure executes, it fetches parameters that it finds by using the address pointed to by the stack pointer. Typically, the stack pointer is a register in the processor. Therefore, by causing an overflow into the stack, the attacker can change either the old stack pointer (changing the context for the calling procedure) or the return address (causing control to transfer where the attacker wants when the subprocedure returns). Changing the context or return address allows the attacker to redirect execution to a block of code the attacker wants.
In both these cases, a little experimentation is needed to determine where the overflow is and how to control it. But the work to be done is relatively smallprobably a day or two for a competent analyst. These buffer overflows are carefully explained in a paper by Mudge [MUD95] of the famed l0pht computer security group. Pincus and Baker [PIN04] reviewed

buffer overflows ten years after Mudge and found that, far from being a minor aspect of attack, buffer overflows have been a very significant attack vector and have spawned several other new attack types.

An alternative style of buffer overflow occurs when parameter values are passed into a routine, especially when the parameters are passed to a web server on the Internet.

Parameters are passed in the URL line, with a syntax similar to

http://www.somesite.com/subpage/userinput.asp?parm1=(808)555-1212
&parm2=2009Jan17

In this example, the page userinput receives two parameters, parm1 with value (808)555-1212 (perhaps a U.S. telephone number) and parm2 with value 2009Jan17 (perhaps a date). The web browser on the caller's machine will accept values from a user who probably completes fields on a form. The browser encodes those values and transmits them back to the server's web site.

The attacker might question what the server would do with a really long telephone number, say, one with 500 or 1000 digits. But, you say, no telephone in the world has such a number;

that is probably exactly what the developer thought, so the developer may have allocated 15 or 20 bytes for an expected maximum length telephone number. Will the program crash with 500 digits? And if it crashes, can it be made to crash in a predictable and usable way? (For the answer to this question, see Litchfield's investigation of the Microsoft *dialer* program [LIT99].) Passing a very long string to a web server is a slight variation on the classic buffer overflow, but no less effective.

As noted earlier, buffer overflows have existed almost as long as higher-level programming languages with arrays. For a long time they were simply a minor annoyance to programmers and users, a cause of errors and sometimes even system crashes. Rather recently, attackers have used them as vehicles to cause first a system crash and then a controlled failure with a serious security implication. The large number of security vulnerabilities based on buffer overflows shows that developers must pay more attention now to what had previously been thought to be just a minor annoyance.

Incomplete Mediation

Incomplete mediation is another security problem that has been with us for decades. Attackers are exploiting it to cause security problems.

Definition

Consider the example of the previous section:

http://www.somesite.com/subpage/userinput.asp?parm1=(808)555-1212
&parm2=2009Jan17

The two parameters look like a telephone number and a date. Probably the client's (user's) web browser enters those two values in their specified format for easy processing on the server's side. What would happen if parm2 were submitted as 1800Jan01? Or 1800Feb30? Or 2048Min32? Or 1Aardvark2Many?

Something would likely fail. As with buffer overflows, one possibility is that the system would fail catastrophically, with a routine's failing on a data type error as it tried to handle a month named "Min" or even a year (like 1800) that was out of range. Another possibility is that the receiving program would continue to execute but would generate a very wrong result. (For example, imagine the amount of interest due today on a billing error with a start date of 1 Jan 1800.) Then again, the processing server might have a default condition, deciding to treat 1Aardvark2Many as 3 July 1947. The possibilities are endless.

One way to address the potential problems is to try to anticipate them. For instance, the programmer in the examples above may have written code to check for correctness on the *client*'s side (that is, the user's browser). The client program can search for and screen out errors. Or, to prevent the use of nonsense data, the program can restrict choices only to valid ones. For example, the program supplying the parameters might have solicited them by using a drop-down box or choice list from which only the twelve conventional months would have been possible choices. Similarly, the year could have been tested to ensure that the value was between 1995 and 2015, and date numbers would have to have been appropriate for the months in which they occur (no 30th of February, for example). Using these verification techniques, the programmer may have felt well insulated from the possible problems a careless or malicious user could cause.

However, the program is still vulnerable. By packing the result into the return URL, the programmer left these data fields in a place the user can access (and modify). In particular, the user could edit the URL line, change any parameter values, and resend the line. On the server side, there is no way for the server to tell if the response line came from the client's browser or as a result of the user's editing the URL directly. We say in this case that the data values are not completely mediated: The sensitive data (namely, the parameter values) are in an exposed, uncontrolled condition.

Security Implication

Incomplete mediation is easy to exploit, but it has been exercised less often than buffer overflows. Nevertheless, unchecked data values represent a serious potential vulnerability. To demonstrate this flaw's security implications, we use a real example; only the name of the vendor has been changed to protect the guilty. Things, Inc., was a very large, international vendor of consumer products, called Objects. The company was ready to sell its Objects through a web site, using what appeared to be a standard e-commerce application. The management at Things decided to let some of its in-house developers produce the web site so that its customers could order Objects directly from the web.

To accompany the web site, Things developed a complete price list of its Objects, including pictures, descriptions, and drop-down menus for size, shape, color, scent, and any other properties. For example, a customer on the web could choose to buy 20 of part number 555A Objects. If the price of one such part were $10, the web server would correctly compute the price of the 20 parts to be $200. Then the customer could decide whether to have the Objects shipped by boat, by ground transportation, or sent electronically. If the customer were to choose boat delivery, the customer's web browser would complete a form with parameters like these:

http://www.things.com/order.asp?custID=101&part=555A&qy=20&price
=10&ship=boat&shipcost=5&total=205

So far, so good; everything in the parameter passage looks correct. But this procedure leaves the parameter statement open for malicious tampering. Things should not need to pass the price of the items back to itself as an input parameter; presumably Things knows how much its Objects cost, and they are unlikely to change dramatically since the time the price was quoted a few screens earlier.

A malicious attacker may decide to exploit this peculiarity by supplying instead the following URL, where the price has been reduced from $205 to $25:

http://www.things.com/order.asp?custID=101&part=555A&qy=20&price
=1&ship=boat&shipcost=5&total=25

Surprisea It worked. The attacker could have ordered Objects from Things in any quantity at any price. And yes, this code was running on the web site for a while before the problem was detected. From a security perspective, the most serious concern about this flaw was the length of time that it could have run undetected. Had the whole world suddenly made a rush to Things's web site and bought Objects at a fraction of their price, Things probably would have noticed. But Things is large enough that it would never have detected a few customers a day choosing prices that were similar to (but smaller than) the real price, say 30 percent off. The e-commerce division would have shown a slightly smaller profit than other divisions, but the difference probably would not have been enough to raise anyone's eyebrows; the vulnerability could have gone unnoticed for years. Fortunately, Things hired a consultant to do a routine review of its code, and the consultant found the error quickly. This web program design flaw is easy to imagine in other web settings. Those of us interested in security must ask ourselves how many similar problems are there in running code today?

And how will those vulnerabilities ever be found?

Time-of-Check to Time-of-Use Errors

The third programming flaw we investigate involves synchronization. To improve efficiency, modern processors and operating systems usually change the order in which instructions and procedures are executed. In particular, instructions that appear to be adjacent may not actually be executed immediately after each other, either because of intentionally changed order or because of the effects of other processes in concurrent execution.

Definition

Access control is a fundamental part of computer security; we want to make sure that only those who should access an object are allowed that access. (We explore the access control mechanisms in operating systems in greater detail in Chapter 4.) Every requested access must be governed by an access policy stating who is allowed access to what; then the request must be mediated by an access-policy-enforcement agent. But an incomplete mediation problem occurs when access is not checked universally. The time-of-check to time-of-use (TOCTTOU) flaw concerns mediation that is performed with a "bait and switch" in the middle.

It is also known as a serialization or synchronization flaw.

To understand the nature of this flaw, consider a person's buying a sculpture that costs $100.

The buyer removes five $20 bills from a wallet, carefully counts them in front of the seller, and lays them on the table. Then the seller turns around to write a receipt. While the seller's back is turned, the buyer takes back one $20 bill. When the seller turns around, the buyer hands over the stack of bills, takes the receipt, and leaves with the sculpture. Between the time the security was checked (counting the bills) and the access (exchanging the sculpture for the bills), a condition changed: What was checked is no longer valid when the object (that is, the sculpture) is accessed.

A similar situation can occur with computing systems. Suppose a request to access a file were presented as a data structure, with the name of the file and the mode of access presented in the structure. An example of such a structure is shown in Figure 3-2.

Figure 3-2. Data Structure for File Access.

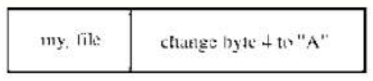


Figure 3-2. Data Structure for File Access.

The data structure is essentially a "work ticket," requiring a stamp of authorization; once authorized, it is put on a queue of things to be done. Normally the access control mediator receives the data structure, determines whether the access should be allowed, and either rejects the access and stops or allows the access and forwards the data structure to the file handler for processing.

To carry out this authorization sequence, the access control mediator would have to look up the file name (and the user identity and any other relevant parameters) in tables. The mediator could compare the names in the table to the file name in the data structure to determine whether access is appropriate. More likely, the mediator would copy the file name into its own local storage area and compare from there. Comparing from the copy leaves the data structure in the user's area, under the user's control.

It is at this point that the incomplete mediation flaw can be exploited. While the mediator is checking access rights for the file my_file, the user could change the file name descriptor to your_file, the value shown in Figure 3-3. Having read the work ticket once, the mediator would not be expected to reread the ticket before approving it; the mediator would approve the access and send the now-modified descriptor to the file handler.

Figure 3-3. Modified Data.

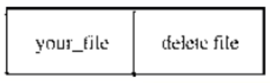


Figure 3-3. Modified Data.

The problem is called a time-of-check to time-of-use flaw because it exploits the delay between the two times. That is, between the time the access was checked and the time the result of the check was used, a change occurred, invalidating the result of the check.

Security Implication

The security implication here is pretty clear: Checking one action and performing another is an example of ineffective access control. We must be wary whenever a time lag or loss of control occurs, making sure that there is no way to corrupt the check's results during that interval.

Fortunately, there are ways to prevent exploitation of the time lag. One way is to ensure that critical parameters are not exposed during any loss of control. The access checking software must own the request data until the requested action is complete. Another way is to ensure serial integrity; that is, to allow no interruption (loss of control) during the validation. Or the validation routine can initially copy data from the user's space to the routine's areaout of the user's reachand perform validation checks on the copy. Finally, the validation routine can seal the request data with a checksum to detect modification.

Combinations of Nonmalicious Program Flaws

These three vulnerabilities are bad enough when each is considered on its own. But perhaps the worst aspect of all three flaws is that they can be used together as one step in a multistep attack. An attacker may not be content with causing a buffer overflow. Instead the attacker may begin a three-pronged attack by using a buffer overflow to disrupt all execution of arbitrary code on a machine. At the same time, the attacker may exploit a time-of-check to time-of-use flaw to add a new user ID to the system. The attacker then logs in as the new user and exploits an incomplete mediation flaw to obtain privileged status, and so forth. The clever attacker uses flaws as common building blocks to build a complex attack. For this reason, we must know about and protect against even simple flaws. (See Sidebar 3-3 for other examples of the effects of unintentional errors.) Unfortunately, these kinds of flaws are widespread and dangerous. As we see in the next section, innocuous-seeming program flaws can be exploited by malicious attackers to plant intentionally harmful code.

Viruses and other malicious code-

By themselves, programs are seldom security threats. The programs operate on data, taking action only when data and state changes trigger it. Much of the work done by a program is invisible to users who are not likely to be aware of any malicious activity. For instance, when was the last time you saw a bit? Do you know in what form a document file is stored? If you know a document resides somewhere on a disk, can you find it? Can you tell if a game program does anything in addition to its expected interaction with you? Which files are modified by a word processor when you create a document? Which programs execute when you start your computer or open a web page? Most users cannot answer these questions.

However, since users usually do not see computer data directly, malicious people can make programs serve as vehicles to access and change data and other programs. Let us look at the possible effects of malicious code and then examine in detail several kinds of programs that can be used for interception or modification of data.

Why Worry About Malicious Code?

None of us like the unexpected, especially in our programs. Malicious code behaves in unexpected ways, thanks to a malicious programmer's intention. We think of the malicious code as lurking inside our system: all or some of a program that we are running or even a nasty part of a separate program that somehow attaches itself to another (good) program. How can such a situation arise? When you last installed a major software package, such as a word processor, a statistical package, or a plug-in from the Internet, you ran one command, typically called INSTALL or SETUP. From there, the installation program took control, creating some files, writing in other files, deleting data and files, and perhaps renaming a few that it would change. A few minutes and a quite a few disk accesses later, you had plenty of new code and data, all set up for you with a minimum of human intervention. Other than the general descriptions on the box, in documentation files, or on web pages, you had absolutely no idea exactly what "gifts" you had received. You hoped all you received was good, and it probably was. The same uncertainty exists when you unknowingly download an application, such as a Java applet or an ActiveX control, while viewing a web site. Thousands or even millions of bytes of programs and data are transferred, and hundreds of modifications may be made to your existing files, all occurring without your explicit consent or knowledge.

Malicious Code Can Do Much (Harm)

Malicious code can do anything any other program can, such as writing a message on a computer screen, stopping a running program, generating a sound, or erasing a stored file. Or malicious code can do nothing at all right now; it can be planted to lie dormant, undetected, until some event triggers the code to act. The trigger can be a time or date, an interval (for example, after 30 minutes), an event (for example, when a particular program is executed), a condition (for example, when communication occurs on a network interface), a count (for example, the fifth time something happens), some combination of these, or a random situation. In fact, malicious code can do different things each time, or nothing most of the time with something dramatic on occasion. In general, malicious code can act with all the predictability of a two-year-old child: We know in general what two-year-olds do, we may even know what a specific two-year-old often does in certain situations, but two-year-olds have an amazing capacity to do the unexpected.

Malicious code runs under the user's authority. Thus, malicious code can touch everything the user can touch, and in the same ways. Users typically have complete control over their own program code and data files; they can read, write, modify, append, and even delete them.

And well they should. But malicious code can do the same, without the user's permission or even knowledge.

Malicious Code Has Been Around a Long Time

The popular literature and press continue to highlight the effects of malicious code as if it were a relatively recent phenomenon. It is not. Cohen [COH84] is sometimes credited with the discovery of viruses, but in fact Cohen gave a name to a phenomenon known long before. For example, Thompson, in his 1984 Turing Award lecture, "Reflections on Trusting Trust" [THO84], described code that can be passed by a compiler. In that lecture, he refers to an earlier Air Force document, the Multics security evaluation by Karger and Schell [KAR74, KAR02]. In fact, references to virus behavior go back at least to 1970. Ware's 1970 study (publicly released in 1979 [WAR79]) and Anderson's planning study for the U.S. Air Force [AND72] *still* accurately describe threats, vulnerabilities, and program security flaws, especially intentional ones. What *is* new about malicious code is the number of distinct instances and copies that have appeared and the speed with which exploit code appears. (See Sidebar 3-4 on attack timing.)

So malicious code is still around, and its effects are more pervasive. It is important for us to learn what it looks like and how it works so that we can take steps to prevent it from doing damage or at least mediate its effects. How can malicious code take control of a system? How can it lodge in a system? How does malicious code spread? How can it be recognized? How can it be detected? How can it be stopped? How can it be prevented? We address these questions in the following sections.

Kinds of Malicious Code

Malicious code or rogue program is the general name for unanticipated or undesired effects in programs or program parts, caused by an agent intent on damage. This definition excludes unintentional errors, although they can also have a serious negative effect. This definition also excludes coincidence, in which two benign programs combine for a negative effect. The agent is the writer of the program or the person who causes its distribution. By this definition, most faults found in software inspections, reviews, and testing do not qualify as malicious code, because we think of them as unintentional. However, keep in mind as you read this chapter that unintentional faults can in fact invoke the same responses as intentional malevolence; a benign cause can still lead to a disastrous effect.

You are likely to have been affected by a virus at one time or another, either because your computer was infected by one or because you could not access an infected system while its administrators were cleaning up the mess one made. In fact, your virus might actually have been a worm: The terminology of malicious code is sometimes used imprecisely. A virus is a program that can replicate itself and pass on malicious code to other nonmalicious programs by modifying them. The term "virus" was coined because the affected program acts like a biological virus: It infects other healthy subjects by attaching itself to the program and either destroying it or coexisting with it. Because viruses are insidious, we cannot assume that a clean program yesterday is still clean today. Moreover, a good program can be modified to include a copy of the virus program, so the infected good program itself begins to act as a virus, infecting other programs. The infection usually spreads at a geometric rate, eventually overtaking an entire computing system and spreading to all other connected systems.

A virus can be either transient or resident. A transient virus has a life that depends on the life of its host; the virus runs when its attached program executes and terminates when its attached program ends. (During its execution, the transient virus may spread its infection to other programs.) A resident virus locates itself in memory; then it can remain active or be activated as a stand-alone program, even after its attached program ends.

A Trojan horse is malicious code that, in addition to its primary effect, has a second, nonobvious malicious effect.[1] As an example of a computer Trojan horse, consider a login script that solicits a user's identification and password, passes the identification information on to the rest of the system for login processing, but also retains a copy of the information for later, malicious use. In this example, the user sees only the login occurring as expected, so there is no evident reason to suspect that any other action took place.

[1] The name is a reference to the Greek legends of the Trojan w ar. Legend tells how the Greeks tricked the Trojans into breaking their defense w all to take a w ooden horse, filled w ith the bravest of Greek soldiers, into their citadel. In the night, the soldiers descended and signaled their troops that the w ay in w as now clear, and Troy w as captured.

A logic bomb is a class of malicious code that "detonates" or goes off when a specified condition occurs. A time bomb is a logic bomb whose trigger is a time or date.
other than by the obvious, direct call, perhaps with special privileges. For instance, an automated bank teller program might allow anyone entering the number 990099 on the keypad to process the log of everyone's transactions at that machine. In this example, the trapdoor could be intentional, for maintenance purposes, or it could be an illicit way for the implementer to wipe out any record of a crime.
A worm is a program that spreads copies of itself through a network. Shock and Hupp [SHO82] are apparently the first to describe a worm, which, interestingly, was for nonmalicious purposes. The primary difference between a worm and a virus is that a worm operates through networks, and a virus can spread through any medium (but usually uses copied program or data files). Additionally, the worm spreads copies of itself as a stand-alone program, whereas the virus spreads copies of itself as a program that attaches to or embeds in other programs.
White et al. [WHI89] also define a rabbit as a virus or worm that self-replicates without bound, with the intention of exhausting some computing resource. A rabbit might create copies of itself and store them on disk in an effort to completely fill the disk, for example.
These definitions match current careful usage. The distinctions among these terms are small, and often the terms are confused, especially in the popular press. The term "virus" is often used to refer to any piece of malicious code. Furthermore, two or more forms of malicious code can be combined to produce a third kind of problem. For instance, a virus can be a time bomb if the viral code that is spreading will trigger an event after a period of time has passed.
The kinds of malicious code are summarized in Table 3-1.

Table 3-1. Types of Malicious Code.

Code Type	Characteristics
Virus	Attaches itself to program and propagates copies of itself to other programs
Trojan horse	Contains unexpected, additional functionality
Logic bomb	Triggers action when condition occurs
Time bomb	Triggers action when specified time occurs
Trapdoor	Allows unauthorized access to functionality
Worm	Propagates copies of itself through a network
Rabbit	Replicates itself without limit to exhaust resources

Because "virus" is the popular name given to all forms of malicious code and because fuzzy lines exist between different kinds of malicious code, we are not too restrictive in the following discussion. We want to look at how malicious code spreads, how it is activated, and what effect it can have. A virus is a convenient term for mobile malicious code, so in the following sections we use the term "virus" almost exclusively. The points made apply also to other forms of malicious code.

How Viruses Attach

A printed copy of a virus does nothing and threatens no one. Even executable virus code sitting on a disk does nothing. What triggers a virus to start replicating? For a virus to do its malicious work and spread itself, it must be activated by being executed. Fortunately for virus writers but unfortunately for the rest of us, there are many ways to ensure that programs will be executed on a running computer.
For example, recall the SETUP program that you initiate on your computer. It may call dozens or hundreds of other programs, some on the distribution medium, some already residing on the computer, some in memory. If any one of these programs contains a virus, the virus code could be activated. Let us see how. Suppose the virus code were in a program on the distribution medium, such as a CD; when executed, the virus could install itself on a permanent storage medium (typically, a hard disk) and also in any and all executing programs in memory. Human intervention is necessary to start the process; a human being puts the virus on the distribution medium, and perhaps another initiates the execution of the program to which the virus is attached. (It is possible for execution to occur without human intervention, though, such as when execution is triggered by a date or the passage

of a certain amount of time.) After that, no human intervention is needed; the virus can spread by itself.
A more common means of virus activation is as an attachment to an e-mail message. In this attack, the virus writer tries to convince the victim (the recipient of the e-mail message) to open the attachment. Once the viral attachment is opened, the activated virus can do its work. Some modern e-mail handlers, in a drive to "help" the receiver (victim), automatically open attachments as soon as the receiver opens the body of the e-mail message. The virus can be executable code embedded in an executable attachment, but other types of files are equally dangerous. For example, objects such as graphics or photo images can contain code to be executed by an editor, so they can be transmission agents for viruses. In general, it is safer to force users to open files on their own rather than automatically; it is a bad idea for programs to perform potentially security-relevant actions without a user's consent. However, ease-of-use often trumps security, so programs such as browsers, e-mail handlers, and viewers often "helpfully" open files without asking the user first.

Appended Viruses

A program virus attaches itself to a program; then, whenever the program is run, the virus is activated. This kind of attachment is usually easy to program. In the simplest case, a virus inserts a copy of itself into the executable program file before the first executable instruction. Then, all the virus instructions execute first; after the last virus instruction, control flows naturally to what used to be the first program instruction. Such a situation is shown in Figure 3-4.
This kind of attachment is simple and usually effective. The virus writer does not need to know anything about the program to which the virus will attach, and often the attached program simply serves as a carrier for the virus. The virus performs its task and then transfers to the original program. Typically, the user is unaware of the effect of the virus if the original program still does all that it used to. Most viruses attach in this manner.

Viruses That Surround a Program

An alternative to the attachment is a virus that runs the original program but has control before and after its execution. For example, a virus writer might want to prevent the virus from being detected. If the virus is stored on disk, its presence will be given away by its file name, or its size will affect the amount of space used on the disk. The virus writer might arrange for the virus to attach itself to the program that constructs the listing of files on the disk. If the virus regains control after the listing program has generated the listing but before the listing is displayed or printed, the virus could eliminate its entry from the listing and falsify space counts so that it appears not to exist. A surrounding virus is shown in Figure 3-5.

Integrated Viruses and Replacements

A third situation occurs when the virus replaces some of its target, integrating itself into the original code of the target. Such a situation is shown in Figure 3-6. Clearly, the virus writer has to know the exact structure of the original program to know where to insert which pieces of the virus.
Finally, the virus can replace the entire target, either mimicking the effect of the target or ignoring the expected effect of the target and performing only the virus effect. In this case, the user is most likely to perceive the loss of the original program.

Document Viruses

Currently, the most popular virus type is what we call the document virus, which is implemented within a formatted document, such as a written document, a database, a slide presentation, a picture, or a spreadsheet. These documents are highly structured files that contain both data (words or numbers) and commands (such as formulas, formatting controls, links). The commands are part of a rich programming language, including macros, variables and procedures, file accesses, and even system calls. The writer of a document virus uses any of the features of the programming language to perform malicious actions.
The ordinary user usually sees only the content of the document (its text or data), so the virus writer simply includes the virus in the commands part of the document, as in the integrated program virus.

How Viruses Gain Control

The virus (V) has to be invoked instead of the target (T). Essentially, the virus either has to seem to be T, saying effectively "I am T" or the virus has to push T out of the way and become a substitute for T, saying effectively "Call me instead of T." A more blatant virus can simply say "invoke me [you fool]."
The virus can assume T's name by replacing (or joining to) T's code in a file structure; this invocation technique is most appropriate for ordinary programs. The virus can overwrite T in storage (simply replacing the copy of T in storage, for example). Alternatively, the virus can change the pointers in the file table so that the virus is located instead of T whenever T is accessed through the file system. These two cases are shown in Figure 3-7.
Figure 3-7. Virus Completely Replacing a Program.
The virus can supplant T by altering the sequence that would have invoked T to now invoke the virus V; this invocation can be used to replace parts of the resident operating system by modifying pointers to those resident parts, such as the table of handlers for different kinds of interrupts.

Homes for Viruses

The virus writer may find these qualities appealing in a virus:
 It is hard to detect.
 It is not easily destroyed or deactivated.
 It spreads infection widely.
 It can reinfect its home program or other programs.
 It is easy to create.
 It is machine independent and operating system independent.
Few viruses meet all these criteria. The virus writer chooses from these objectives when deciding what the virus will do and where it will reside. Just a few years ago, the challenge for the virus writer was to write code that would be executed repeatedly so that the virus could multiply. Now, however, one execution is enough to ensure widespread distribution. Many viruses are transmitted by e-mail, using either of two routes. In the first case, some virus writers generate a new e-mail message to all addresses in the victim's address book. These new messages contain a copy of the virus so that it propagates widely. Often the message is a brief, chatty, nonspecific message that would encourage the new recipient to open the attachment from a friend (the first recipient). For example, the subject line or message body may read "I thought you might enjoy this picture from our vacation." In the second case, the virus writer can leave the infected file for the victim to forward unknowingly. If the virus's effect is not immediately obvious, the victim may pass the infected file unwittingly to other victims.
Let us look more closely at the issue of viral residence.

One-Time Execution

The majority of viruses today execute only once, spreading their infection and causing their effect in that one execution. A virus often arrives as an e-mail attachment of a document virus. It is executed just by being opened.

Boot Sector Viruses

A special case of virus attachment, but formerly a fairly popular one, is the so-called boot sector virus. When a computer is started, control begins with firmware that determines which hardware components are present, tests them, and transfers control to an operating system.
A given hardware platform can run many different operating systems, so the operating system is not coded in firmware but is instead invoked dynamically, perhaps even by a user's choice, after the hardware test.
The operating system is software stored on disk. Code copies the operating system from disk to memory and transfers control to it; this copying is called the bootstrap (often boot) load because the operating system figuratively pulls itself into memory by its bootstraps. The firmware does its control transfer by reading a fixed number of bytes from a fixed location on the disk (called the boot sector) to a fixed address in memory and then jumping to that address (which will turn out to contain the first instruction of the bootstrap loader). The bootstrap loader then reads into memory the rest of the operating system from disk. To run a different operating system, the user just inserts a disk with the new operating system and a bootstrap loader. When the user reboots from this new disk, the loader there brings

in and runs another operating system. This same scheme is used for personal computers, workstations, and large mainframes.
To allow for change, expansion, and uncertainty, hardware designers reserve a large amount of space for the bootstrap load. The boot sector on a PC is slightly less than 512 bytes, but since the loader will be larger than that, the hardware designers support "chaining," in which each block of the bootstrap is chained to (contains the disk location of) the next block. This chaining allows big bootstraps but also simplifies the installation of a virus. The virus writer simply breaks the chain at any point, inserts a pointer to the virus code to be executed, and reconnects the chain after the virus has been installed. This situation is shown in Figure 3-8.
The boot sector is an especially appealing place to house a virus. The virus gains control very early in the boot process, before most detection tools are active, so that it can avoid, or at least complicate, detection. The files in the boot area are crucial parts of the operating system. Consequently, to keep users from accidentally modifying or deleting them with disastrous results, the operating system makes them "invisible" by not showing them as part of a normal listing of stored files, preventing their deletion. Thus, the virus code is not readily noticed by users.
Memory-Resident Viruses
Some parts of the operating system and most user programs execute, terminate, and disappear, with their space in memory being available for anything executed later. For very frequently used parts of the operating system and for a few specialized user programs, it would take too long to reload the program each time it was needed. Such code remains in memory and is called "resident" code. Examples of resident code are the routine that interprets keys pressed on the keyboard, the code that handles error conditions that arise during a program's execution, or a program that acts like an alarm clock, sounding a signal at a time the user determines. Resident routines are sometimes called TSRs or "terminate and stay resident" routines.
Virus writers also like to attach viruses to resident code because the resident code is activated many times while the machine is running. Each time the resident code runs, the virus does too. Once activated, the virus can look for and infect uninfected carriers. For example, after activation, a boot sector virus might attach itself to a piece of resident code. Then, each time the virus was activated it might check whether any removable disk in a disk drive was infected and, if not, infect it. In this way the virus could spread its infection to all removable disks used during the computing session.
A virus can also modify the operating system's table of programs to run. On a Windows machine the registry is the table of all critical system information, including programs to run at startup. If the virus gains control once, it can insert a registry entry so that it will be reinvoked each time the system restarts. In this way, even if the user notices and deletes the executing copy of the virus from memory, the virus will return on the next system restart.
Other Homes for Viruses
A virus that does not take up residence in one of these cozy establishments has to fend more for itself. But that is not to say that the virus will go homeless. One popular home for a virus is an application program. Many applications, such as word processors and spreadsheets, have a "macro" feature, by which a user can record a series of commands and repeat them with one invocation. Such programs also provide a "startup macro" that is executed every time the application is executed. A virus writer can create a virus macro that adds itself to the startup directives for the application. It also then embeds a copy of itself in data files so that the infection spreads to anyone receiving one or more of those files.
Libraries are also excellent places for malicious code to reside. Because libraries are used by many programs, the code in them will have a broad effect. Additionally, libraries are often shared among users and transmitted from one user to another, a practice that spreads the infection. Finally, executing code in a library can pass on the viral infection to other transmission media. Compilers, loaders, linkers, runtime monitors, runtime debuggers, and even virus control programs are good candidates for hosting viruses because they are widely shared.

Virus Signatures

A virus cannot be completely invisible. Code must be stored somewhere, and the code must be in memory to execute. Moreover, the virus executes in a particular way, using certain methods to spread. Each of these characteristics yields a telltale pattern, called a signature, creating a program, called a virus scanner, that can detect and, in some cases, remove viruses. The scanner searches memory and long-term storage, monitoring execution and watching for the telltale signatures of viruses. For example, a scanner looking for signs of the Code Red worm can look for a pattern containing the following characters:

```
/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
%u9090%u6858%ucbd3
%u7801%u9090%u6858%ucdb3%u7801%u9090%u6858
%ucbd3%u7801%u9090
%u9090%u8190%u00c3%u0003%ub00%u531b%u53ff
%u0078%u0000%u00=a
HTTP/1.0
```

When the scanner recognizes a known virus's pattern, it can then block the virus, inform the user, and deactivate or remove the virus. However, a virus scanner is effective only if it has been kept up to date with the latest information on current viruses. Sidebar 3-5 describes how viruses were the primary security breach among companies surveyed in 2001.

Storage Patterns

Most viruses attach to programs that are stored on media such as disks. The attached virus piece is invariant, so the start of the virus code becomes a detectable signature. The attached piece is always located at the same position relative to its attached file. For example, the virus might always be at the beginning, 400 bytes from the top, or at the bottom of the infected file. Most likely, the virus will be at the beginning of the file because the virus writer wants to obtain control of execution before the bona fide code of the infected program is in charge. In the simplest case, the virus code sits at the top of the program, and the entire virus does its malicious duty before the normal code is invoked. In other cases, the virus infection consists of only a handful of instructions that point or jump to other, more detailed instructions elsewhere. For example, the infected code may consist of condition testing and a jump or call to a separate virus module. In either case, the code to which control is transferred will also have a recognizable pattern. Both of these situations are shown in Figure 3-9.

Figure 3-9. Recognizable Patterns in Viruses.

A virus may attach itself to a file, in which case the file's size grows. Or the virus may obliterate all or part of the underlying program, in which case the program's size does not change but the program's functioning will be impaired. The virus writer has to choose one of these detectable effects.

The virus scanner can use a code or checksum to detect changes to a file. It can also look for suspicious patterns, such as a JUMP instruction as the first instruction of a system program (in case the virus has positioned itself at the bottom of the file but is to be executed first, as in Figure 3-9).

Execution Patterns

A virus writer may want a virus to do several things at the same time, namely, spread infection, avoid detection, and cause harm. These goals are shown in Table 3-2, along with ways each goal can be addressed. Unfortunately, many of these behaviors are perfectly normal and might otherwise go undetected. For instance, one goal is modifying the file directory; many normal programs create files, delete files, and write to storage media. Thus, no key signals point to the presence of a virus.

Table 3-2. Virus Effects and Causes.

Virus Effect	How It Is Caused
Attach to executable program	• Modify file directory • Write to executable program file
Attach to data or control file	

 Modify directory
 Rewrite data
 Append to data
 Append data to self

Table 3-2. Virus Effects and Causes.

Virus Effect How It Is Caused
Remain in memory
 Intercept interrupt by modifying interrupt handler address table
 Load self in nontransient memory area
Infect disks
 Intercept interrupt
 Intercept operating system call (to format disk, for example)
 Modify system file
 Modify ordinary executable program
Conceal self Intercept system calls that would reveal self and falsify result
 Classify self as "hidden" file
Spread infection Infect boot sector
 Infect systems program
 Infect ordinary program
 Infect data ordinary program reads to control its
execution
Prevent deactivation Activate before deactivating program and block
deactivation
 Store copy to reinfect after deactivation

Most virus writers seek to avoid detection for themselves and their creations. Because a disk's boot sector is not visible to normal operations (for example, the contents of the boot sector do not show on a directory listing), many virus writers hide their code there. A resident virus can monitor disk accesses and fake the result of a disk operation that would show the virus hidden in a boot sector by showing the data that *should* have been in the boot sector (which the virus has moved elsewhere).

There are no limits to the harm a virus can cause. On the modest end, the virus might do nothing; some writers create viruses just to show they can do it. Or the virus can be relatively benign, displaying a message on the screen, sounding the buzzer, or playing music.

From there, the problems can escalate. One virus can erase files, another an entire disk; one virus can prevent a computer from booting, and another can prevent writing to disk. The damage is bounded only by the creativity of the virus's author.

Transmission Patterns

A virus is effective only if it has some means of transmission from one location to another. As we have already seen, viruses can travel during the boot process by attaching to an executable file or traveling within data files. The travel itself occurs during execution of an already infected program. Since a virus can execute any instructions a program can, virus travel is not confined to any single medium or execution pattern. For example, a virus can arrive on a disk or from a network connection, travel during its host's execution to a hard disk boot sector, reemerge next time the host computer is booted, and remain in memory to infect other disks as they are accessed.

Polymorphic Viruses

The virus signature may be the most reliable way for a virus scanner to identify a virus. If a particular virus always begins with the string 47F0F00E08 (in hexadecimal) and has string 00113FFF located at word 12, it is unlikely that other programs or data files will have these exact characteristics. For longer signatures, the probability of a correct match increases. If the virus scanner will always look for those strings, then the clever virus writer can cause something other than those strings to be in those positions. Many instructions cause no effect, such as adding 0 to a number, comparing a number to itself, or jumping to the next instruction. These instructions, sometimes called *no-ops*, can be sprinkled into a piece of code to distort any pattern. For example, the virus could have two alternative but equivalent beginning words; after being installed, the virus will choose one of the two words for its initial word. Then, a virus scanner would have to look for both patterns. A virus that can

change its appearance is called a polymorphic virus. (*Poly* means "many" and *morph* means "form.")
A two-form polymorphic virus can be handled easily as two independent viruses. Therefore, the virus writer intent on preventing detection of the virus will want either a large or an unlimited number of forms so that the number of possible forms is too large for a virus scanner to search for. Simply embedding a random number or string at a fixed place in the executable version of a virus is not sufficient, because the signature of the virus is just the constant code excluding the random part. A polymorphic virus has to randomly reposition all parts of itself and randomly change all fixed data. Thus, instead of containing the fixed (and therefore searchable) string "HAa INFECTED BY A VIRUS," a polymorphic virus has to change even that pattern sometimes.
Trivially, assume a virus writer has 100 bytes of code and 50 bytes of data. To make two virus instances different, the writer might distribute the first version as 100 bytes of code followed by all 50 bytes of data. A second version could be 99 bytes of code, a jump instruction, 50 bytes of data, and the last byte of code. Other versions are 98 code bytes jumping to the last two, 97 and three, and so forth. Just by moving pieces around, the virus writer can create enough different appearances to fool simple virus scanners. Once the scanner writers became aware of these kinds of tricks, however, they refined their signature definitions.
A simple variety of polymorphic virus uses encryption under various keys to make the stored form of the virus different. These are sometimes called encrypting viruses. This type of virus must contain three distinct parts: a decryption key, the (encrypted) object code of the virus, and the (unencrypted) object code of the decryption routine. For these viruses, the decryption routine itself, or a call to a decryption library routine, must be in the clear so that becomes the signature.
To avoid detection, not every copy of a polymorphic virus has to differ from every other copy.
If the virus changes occasionally, not every copy will match a signature of every other copy.

The Source of Viruses

Since a virus can be rather small, its code can be "hidden" inside other larger and more complicated programs. Two hundred lines of a virus could be separated into one hundred packets of two lines of code and a jump each; these one hundred packets could be easily hidden inside a compiler, a database manager, a file manager, or some other large utility.
Virus discovery could be aided by a procedure to determine if two programs are equivalent. However, theoretical results in computing are very discouraging when it comes to the complexity of the equivalence problem. The general question "Are these two programs equivalent?" is undecidable (although that question *can* be answered for many specific pairs of programs). Even ignoring the general undecidability problem, two modules may produce subtly different results that mayor may notbe security relevant. One may run faster, or the first may use a temporary file for workspace whereas the second performs all its computations in memory. These differences could be benign, or they could be a marker of an infection.
Therefore, we are unlikely to develop a screening program that can separate infected modules from uninfected ones.
Although the general is dismaying, the particular is not. If we know that a particular virus may infect a computing system, we can check for it and detect it if it is there. Having found the virus, however, we are left with the task of cleansing the system of it. Removing the virus in a running system requires being able to detect and eliminate its instances faster than it can spread.

Prevention of Virus Infection

The only way to prevent the infection of a virus is not to receive executable code from an infected source. This philosophy used to be easy to follow because it was easy to tell if a file was executable or not. For example, on PCs, a *.exe* extension was a clear sign that the file was executable. However, as we have noted, today's files are more complex, and a seemingly nonexecutable file may have some executable code buried deep within it. For example, a word processor may have commands within the document file; as we noted earlier, these commands, called macros, make it easy for the user to do complex or repetitive things. But they are really executable code embedded in the context of the document. Similarly,

spreadsheets, presentation slides, other office- or business-related files, and even media files can contain code or scripts that can be executed in various waysand thereby harbor viruses.
And, as we have seen, the applications that run or use these files may try to be helpful by automatically invoking the executable code, whether you want it run or nota Against the principles of good security, e-mail handlers can be set to automatically open (without performing access control) attachments or embedded code for the recipient, so your e-mail message can have animated bears dancing across the top.
Another approach virus writers have used is a little-known feature in the Microsoft file design.
Although a file with a *.doc* extension is expected to be a Word document, in fact, the true document type is hidden in a field at the start of the file. This convenience ostensibly helps a user who inadvertently names a Word document with a *.ppt* (Power-Point) or any other extension. In some cases, the operating system will try to open the associated application but, if that fails, the system will switch to the application of the hidden file type. So, the virus writer creates an executable file, names it with an inappropriate extension, and sends it to the victim, describing it is as a picture or a necessary code add-in or something else desirable.
The unwitting recipient opens the file and, without intending to, executes the malicious code.
More recently, executable code has been hidden in files containing large data sets, such as pictures or read-only documents. These bits of viral code are not easily detected by virus scanners and certainly not by the human eye. For example, a file containing a photograph may be highly granular; if every sixteenth bit is part of a command string that can be executed, then the virus is very difficult to detect.
Because you cannot always know which sources are infected, you should assume that any outside source is infected. Fortunately, you know when you are receiving code from an outside source; unfortunately, it is not feasible to cut off all contact with the outside world. In their interesting paper comparing computer virus transmission with human disease transmission, Kephart et al. [KEP93] observe that individuals' efforts to keep their computers free from viruses lead to communities that are generally free from viruses because members of the community have little (electronic) contact with the outside world. In this case, transmission is contained not because of limited contact but because of limited contact outside the community. Governments, for military or diplomatic secrets, often run disconnected network communities. The trick seems to be in choosing one's community prudently. However, as use of the Internet and the World Wide Web increases, such separation is almost impossible to maintain.
Nevertheless, there are several techniques for building a reasonably safe community for electronic contact, including the following:
 Use only commercial software acquired from reliable, well-established vendors. There is always a chance that you might receive a virus from a large manufacturer with a name everyone would recognize. However, such enterprises have significant reputations that could be seriously damaged by even one bad incident, so they go to some degree of trouble to keep their products virus-free and to patch any problem-causing code right away. Similarly, software distribution companies will be careful about products they handle.
 Test all new software on an isolated computer. If you must use software from a questionable source, test the software first on a computer that is not connected to a network and contains no sensitive or important data. Run the software and look for unexpected behavior, even simple behavior such as unexplained figures on the screen. Test the computer with a copy of an up-to-date virus scanner created before the suspect program is run. Only if the program passes these tests should you install it on a less isolated machine.
 Open attachments only when you know them to be safe. What constitutes "safe" is up to you, as you have probably already learned in this chapter. Certainly, an attachment from an unknown source is of questionable safety. You might also distrust an attachment from a known source but with a peculiar message.
 Make a recoverable system image and store it safely. If your system does become infected, this clean version will let you reboot securely because it overwrites the corrupted system files with clean copies. For this reason, you must keep the image write-protected during

reboot. Prepare this image now, before infection; after infection it is too late. For safety, prepare an extra copy of the safe boot image.

 Make and retain backup copies of executable system files. This way, in the event of a virus infection, you can remove infected files and reinstall from the clean backup copies (stored in a secure, offline location, of course). Also make and retain backups of important data files that might contain infectable code; such files include word-processor documents, spreadsheets, slide presentations, pictures, sound files, and databases. Keep these backups on inexpensive media, such as CDs or DVDs so that you can keep old backups for a long time. In case you find an infection, you want to be able to start from a clean backupthat is, one taken before the infection.

 Use virus detectors (often called virus scanners) regularly and update them daily. Many of the available virus detectors can both detect and eliminate infection from viruses. Several scanners are better than one because one may detect the viruses that others miss. Because scanners search for virus signatures, they are constantly being revised as new viruses are discovered. New virus signature files or new versions of scanners are distributed frequently; often, you can request automatic downloads from the vendor's web site. Keep your detector's signature file up to date.

Truths and Misconceptions About Viruses

Because viruses often have a dramatic impact on the computer-using community, they are often highlighted in the press, particularly in the business section. However, there is much misinformation in circulation about viruses. Let us examine some of the popular claims about them.

 Viruses can infect only Microsoft Windows systems. False. Among students and office workers, PCs running Windows are popular computers, and there may be more people writing software (and viruses) for them than for any other kind of processor. Thus, the PC is most frequently the target when someone decides to write a virus. However, the principles of virus attachment and infection apply equally to other processors, including Macintosh computers, Unix and Linux workstations, and mainframe computers. Cell phones and PDAs are now also virus targets. In fact, no writeable stored-program computer is immune to possible virus attack. As we noted in Chapter 1, this situation means that *all* devices containing computer code, including automobiles, airplanes, microwave ovens, radios, televisions, voting machines, and radiation therapy machines have the potential for being infected by a virus.

 Viruses can modify ahiddena or aread-onlya files. True. We may try to protect files by using two operating system mechanisms. First, we can make a file a hidden file so that a user or program listing all files on a storage device will not see the file's name. Second, we can apply a read-only protection to the file so that the user cannot change the file's contents. However, each of these protections is applied by software, and virus software can override the native software's protection. Moreover, software protection is layered, with the operating system providing the most elementary protection. If a secure operating system obtains control *before* a virus contaminator has executed, the operating system can prevent contamination as long as it blocks the attacks the virus will make.

 Viruses can appear only in data files, or only in Word documents, or only in programs. False. What are data? What is an executable file? The distinction between these two concepts is not always clear, because a data file can control how a program executes and even cause a program to execute. Sometimes a data file lists steps to be taken by the program that reads the data, and these steps can include executing a program.

For example, some applications contain a configuration file whose data are exactly such steps. Similarly, word-processing document files may contain startup commands to execute when the document is opened; these startup commands can contain malicious code. Although, strictly speaking, a virus can activate and spread only when a program executes, in fact, data files are acted on by programs. Clever virus writers have been able to make data control files that cause programs to do many things, including pass along copies of the virus to other data files.

 Viruses spread only on disks or only through e-mail. False. File-sharing is often done as one user provides a copy of a file to another user by writing the file on a transportable disk. However, any means of electronic file transfer will work. A file can be placed in a network's library or posted on a bulletin board. It can be attached to an e-mail message or made available for download from a web site. Any mechanism for sharing filesof programs, data, documents, and so forthcan be used to transfer a virus.

 Viruses cannot remain in memory after a complete power off/power on reboot. Truea but . . . If a virus is resident in memory, the virus is lost when the memory loses power. That is, computer memory (RAM) is volatile, so all contents are deleted when power is lost.[2] However, viruses written to disk certainly can remain through a reboot cycle.

Thus, you can receive a virus infection, the virus can be written to disk (or to network storage), you can turn the machine off and back on, and the virus can be reactivated during the reboot. Boot sector viruses gain control when a machine reboots (whether it is a hardware or software reboot), so a boot sector virus may remain through a reboot cycle because it activates immediately when a reboot has completed.

[2] Some very low-evel hardw are settings (for example, the size of disk installed) are retained in memory called anonvolatile RAM,a but these locations are not directly accessible by programs and are w ritten only by programs run from read-only memory (ROM) during hardw are initialization. Thus, they are highly immune to virus attack.

 Viruses cannot infect hardware. True. Viruses can infect only things they can modify; memory, executable files, and data are the primary targets. If hardware contains writeable storage (so-called firmware) that can be accessed under program control, that storage *is* subject to virus attack. There have been a few instances of firmware viruses. Because a virus can control hardware that is subject to program control, it may seem as if a hardware device has been infected by a virus, but it is really the software driving the hardware that has been infected. Viruses can also exercise hardware in any way a program can. Thus, for example, a virus could cause a disk to loop incessantly, moving to the innermost track then the outermost and back again to the innermost.

 Viruses can be malevolent, benign, or benevolent. True. Not all viruses are bad. For example, a virus might locate uninfected programs, compress them so that they occupy less memory, and insert a copy of a routine that decompresses the program when its execution begins. At the same time, the virus is spreading the compression function to other programs. This virus could substantially reduce the amount of storage required for stored programs, possibly by up to 50 percent. However, the compression would be done at the request of the virus, not at the request, or even knowledge, of the program owner.

To see how viruses and other types of malicious code operate, we examine four types of malicious code that affected many users worldwide: the Brain, the Internet worm, the Code Red worm, and web bugs.

First Example of Malicious Code: The Brain Virus

One of the earliest viruses is also one of the most intensively studied. The so-called Brain virus was given its name because it changes the label of any disk it attacks to the word "BRAIN." This particular virus, believed to have originated in Pakistan, attacks PCs running an old Microsoft operating system. Numerous variants have been produced; because of the number of variants, people believe that the source code of the virus was released to the underground virus community.

What It Does

The Brain, like all viruses, seeks to pass on its infection. This virus first locates itself in upper memory and then executes a system call to reset the upper memory bound below itself so that it is not disturbed as it works. It traps interrupt number 19 (disk read) by resetting the interrupt address table to point to it and then sets the address for interrupt number 6 (unused) to the former address of the interrupt 19. In this way, the virus screens disk read calls, handling any that would read the boot sector (passing back the original boot contents that were moved to one of the bad sectors); other disk calls go to the normal disk read handler, through interrupt 6.

The Brain virus appears to have no effect other than passing its infection, as if it were an experiment or a proof of concept. However, variants of the virus erase disks or destroy the file allocation table (the table that shows which files are where on a storage medium).

How It Spreads

The Brain virus positions itself in the boot sector and in six other sectors of the disk. One of the six sectors will contain the original boot code, moved there from the original boot sector, while two others contain the remaining code of the virus. The remaining three sectors contain a duplicate of the others. The virus marks these six sectors "faulty" so that the operating system will not try to use them. (With low-level calls, you can force the disk drive

to read from what the operating system has marked as bad sectors.) The virus allows the boot process to continue.
Once established in memory, the virus intercepts disk read requests for the disk drive under attack. With each read, the virus reads the disk boot sector and inspects the fifth and sixth bytes for the hexadecimal value 1234 (its signature). If it finds that value, it concludes that the disk is infected; if not, it infects the disk as described in the previous paragraph.

What Was Learned

This virus uses some of the standard tricks of viruses, such as hiding in the boot sector, and intercepting and screening interrupts. The virus is almost a prototype for later efforts. In fact, many other virus writers seem to have patterned their work on this basic virus. Thus, one could say it was a useful learning tool for the virus writer community. Sadly, its infection did not raise public consciousness of viruses, other than a certain amount of fear and misunderstanding. Subsequent viruses, such as the Lehigh virus that swept through the computers of Lehigh University, the nVIR viruses that sprang from prototype code posted on bulletin boards, and the Scores virus that was first found at NASA in Washington D.C. circulated more widely and with greater effect. Fortunately, most viruses seen to date have a modest effect, such as displaying a message or emitting a sound. That is, however, a matter of luck, since the writers who could put together the simpler viruses obviously had all the talent and knowledge to make much more malevolent viruses.
There is no general cure for viruses. Virus scanners are effective against today's known viruses and general patterns of infection, but they cannot counter tomorrow's variant. The only sure prevention is complete isolation from outside contamination, which is not feasible; in fact, you may even get a virus from the software applications you buy from reputable vendors.

Example: The Internet Worm

On the evening of 2 November 1988, a worm was released to the Internet,[3] causing serious damage to the network. Not only were many systems infected, but also when word of the problem spread, many more uninfected systems severed their network connections to prevent themselves from getting infected. Spafford and his team at Purdue University [SPA89] and Eichen and Rochlis at M.I.T. [EIC89] studied the worm extensively, and Orman [ORM03] did an interesting retrospective analysis 15 years after the incident.

[3] Note: This incident is normally called a aw orm,a although it shares most of the characteristics of viruses.

The perpetrator was Robert T. Morris, Jr., a graduate student at Cornell University who created and released the worm. He was convicted in 1990 of violating the 1986 Computer Fraud and Abuse Act, section 1030 of U.S. Code Title 18. He received a fine of $10,000, a three-year suspended jail sentence, and was required to perform 400 hours of community service. (See Denning [DEN90b] for a discussion of this punishment.)

What It Did

Judging from its code, Morris programmed the Internet worm to accomplish three main objectives:
1. Determine where it could spread to.
2. Spread its infection.
3. Remain undiscovered and undiscoverable.

What Effect It Had

The worm's primary effect was resource exhaustion. Its source code indicated that the worm was supposed to check whether a target host was already infected; if so, the worm would negotiate so that either the existing infection or the new infector would terminate. However, because of a supposed flaw in the code, many new copies did not terminate. As a result, an infected machine soon became burdened with many copies of the worm, all busily attempting to spread the infection. Thus, the primary observable effect was serious degradation in performance of affected machines.
A second-order effect was the disconnection of many systems from the Internet. System administrators tried to sever their connection with the Internet, either because their machines were already infected and the system administrators wanted to keep the worm's processes from looking for sites to which to spread or because their machines were not yet infected and the staff wanted to avoid having them become so.
The disconnection led to a third-order effect: isolation and inability to perform necessary work. Disconnected systems could not communicate with other systems to carry on the

normal research, collaboration, business, or information exchange users expected. System administrators on disconnected systems could not use the network to exchange information with their counterparts at other installations, so status and containment or recovery information was unavailable.
The worm caused an estimated 6,000 installations to shut down or disconnect from the Internet. In total, several thousand systems were disconnected for several days, and several hundred of these systems were closed to users for a day or more while they were disconnected. Estimates of the cost of the damage range from $100,000 to $97 million.
How It Worked
The worm exploited several known flaws and configuration failures of Berkeley version 4 of the Unix operating system. It accomplished or had code that appeared to try to accomplishits three objectives.
Determine where to spread. The worm had three techniques for locating potential machines to victimize. It first tried to find user accounts to invade on the target machine. In parallel, the worm tried to exploit a bug in the *finger* program and then to use a trapdoor in the *sendmail* mail handler. *All three of these security flaws were well known in the general Unix community.*
The first security flaw was a joint user and system error, in which the worm tried guessing passwords and succeeded when it found one. The Unix password file is stored in encrypted form, but the ciphertext in the file is readable by anyone. (This visibility is the system error.) The worm encrypted various popular passwords and compared their ciphertext to the ciphertext of the stored password file. The worm tried the account name, the owner's name, and a short list of 432 common passwords (such as "guest," "password," "help," "coffee," "coke," "aaa"). If none of these succeeded, the worm used the dictionary file stored on the system for use by application spelling checkers. (Choosing a recognizable password is the user error.) When it got a match, the worm could log in to the corresponding account by presenting the plaintext password. Then, as a user, the worm could look for other machines to which the user could obtain access. (See the article by Robert T. Morris, Sr. and Ken Thompson [MOR79] on selection of good passwords, published a decade before the worm, and the section in Chapter 4 on passwords people choose.)
The second flaw concerned *fingerd*, the program that runs continuously to respond to other computers' requests for information about system users. The security flaw involved causing the input buffer to overflow, spilling into the return address stack. Thus, when the *finger* call terminated, *fingerd* executed instructions that had been pushed there as another part of the buffer overflow, causing the worm to be connected to a remote shell.
The third flaw involved a trapdoor in the *sendmail* program. Ordinarily, this program runs in the background, awaiting signals from others wanting to send mail to the system. When it receives such a signal, *sendmail* gets a destination address, which it verifies, and then begins a dialog to receive the message. However, when running in debugging mode, the worm causes *sendmail* to receive and execute a command string instead of the destination address.
Spread infection. Having found a suitable target machine, the worm would use one of these three methods to send a bootstrap loader to the target machine. This loader consisted of 99 lines of C code to be compiled and executed on the target machine. The bootstrap loader would then fetch the rest of the worm from the sending host machine. An element of good computer securityor stealthwas built into the exchange between the host and the target.
When the target's bootstrap requested the rest of the worm, the worm supplied a one-time password back to the host. Without this password, the host would immediately break the connection to the target, presumably in an effort to ensure against "rogue" bootstraps (ones that a real administrator might develop to try to obtain a copy of the rest of the worm for subsequent analysis).
Remain undiscovered and undiscoverable. The worm went to considerable lengths to prevent its discovery once established on a host. For instance, if a transmission error occurred while the rest of the worm was being fetched, the loader zeroed and then deleted all code already transferred and then exited.
As soon as the worm received its full code, it brought the code into memory, encrypted it, and deleted the original copies from disk. Thus, no traces were left on disk, and even a memory dump would not readily expose the worm's code. The worm periodically changed its

name and process identifier so that no single name would run up a large amount of computing time.

What Was Learned

The Internet worm sent a shock wave through the Internet community, which at that timewas largely populated by academics and researchers. The affected sites closed some of the loopholes exploited by the worm and generally tightened security. Some users changed passwords. Two researchers, Farmer and Spafford [FAR90], developed a program for system administrators to check for some of the same flaws the worm exploited. However, security analysts checking for site vulnerabilities across the Internet find that many of the same security flaws still exist today. A new attack on the Internet would not succeed on the same scale as the Internet worm, but it could still cause significant inconvenience to many.

The Internet worm was benign in that it only spread to other systems but did not destroy any part of them. It collected sensitive data, such as account passwords, but it did not retain them. While acting as a user, the worm could have deleted or overwritten files, distributed them elsewhere, or encrypted them and held them for ransom. The next worm may not be so benign.

The worm's effects stirred several people to action. One positive outcome from this experience was development of an infrastructure for reporting and correcting malicious and nonmalicious code flaws. The Internet worm occurred at about the same time that Cliff Stoll [STO89] reported his problems in tracking an electronic intruder (and his subsequent difficulty in finding anyone to deal with the case). The computer community realized it needed to organize. The resulting Computer Emergency Response Team (CERT) at Carnegie Mellon University was formed; it and similar response centers around the world have done an excellent job of collecting and disseminating information on malicious code attacks and their countermeasures. System administrators now exchange information on problems and solutions.

Security comes from informed protection and action, not from ignorance and inaction.

More Malicious Code: Code Red

Code Red appeared in the middle of 2001, to devastating effect. On July 29, the U.S. Federal Bureau of Investigation proclaimed in a news release that "on July 19, the Code Red worm infected more than 250,000 systems in just nine hours. . . . This spread has the potential to disrupt business and personal use of the Internet for applications such as e-commerce, e-mail and entertainment" [BER01]. Indeed, "the Code Red worm struck faster than any other worm in Internet history," according to a research director for a security software and services vendor. The first attack occurred on July 12; overall, 750,000 servers were affected, including 400,000 just in the period from August 1 to 10 [HUL01]. Thus, of the 6 million web servers running code subject to infection by Code Red, about one in eight were infected. Michael Erbschloe, vice president of Computer Economics, Inc., estimates that Code Red's damage will exceed $2 billion [ERB01].

Code Red was more than a worm; it included several kinds of malicious code, and it mutated from one version to another. Let us take a closer look at how Code Red worked.

What It Did

There are several versions of Code Red, malicious software that propagates itself on web servers running Microsoft's Internet Information Server (IIS) software. Code Red takes two steps: infection and propagation. To infect a server, the worm takes advantage of a vulnerability in Microsoft's IIS. It overflows the buffer in the dynamic link library *idq.dll* to reside in the server's memory. Then, to propagate, Code Red checks IP addresses on port 80 of the PC to see if that web server is vulnerable.

What Effect It Had

The first version of Code Red was easy to spot because it defaced web sites with the following text:

HELLOa
Welcome to
http://www.worm.com a
Hacked by Chinesea

The rest of the original Code Red's activities were determined by the date. From day 1 to 19 of the month, the worm spawned 99 threads that scanned for other vulnerable computers, starting at the same IP address. Then, on days 20 to 27, the worm launched a distributed denial-of-service attack at the U.S. web site, *www.whitehouse.gov*. A denial-of-service

attack floods the site with large numbers of messages in an attempt to slow down or stop the site because the site is overwhelmed and cannot handle the messages. Finally, from day 28 to the end of the month, the worm did nothing.
However, there were several variants. The second variant was discovered near the end of July 2001. It did not deface the web site, but its propagation was randomized and optimized to infect servers more quickly. A third variant, discovered in early August, seemed to be a substantial rewrite of the second. This version injected a Trojan horse in the target and modified software to ensure that a remote attacker could execute any command on the server. The worm also checked the year and month so that it would automatically stop propagating in October 2002. Finally, the worm rebooted the server after 24 or 48 hours, wiping itself from memory but leaving the Trojan horse in place.

How It Worked

The Code Red worm looked for vulnerable personal computers running Microsoft IIS software.
Exploiting the unchecked buffer overflow, the worm crashed Windows NT-based servers but executed code on Windows 2000 systems. The later versions of the worm created a trapdoor on an infected server; the system was then open to attack by other programs or malicious users. To create the trapdoor, Code Red copied *%windir%\cmd.exe* to four locations:
c:\inetpub\scripts\root.ext
c:\progra~1\common~1\system\MSADC\root.exe
d:\inetpub\scripts\root.ext
d:\progra~1\common~1\system\MSADC\root.exe
Code Red also included its own copy of the file *explorer.exe*, placing it on the c: and d: drives so that Windows would run the malicious copy, not the original copy. This Trojan horse first ran the original, untainted version of *explorer.exe*, but it modified the system registry to disable certain kinds of file protection and to ensure that some directories have read, write, and execute permission. As a result, the Trojan horse had a virtual path that could be followed even when *explorer.exe* was not running. The Trojan horse continued to run in background, resetting the registry every 10 minutes; thus, even if a system administrator noticed the changes and undid them, the changes were applied again by the malicious code.
To propagate, the worm created 300 or 600 threads (depending on the variant) and tried for 24 or 48 hours to spread to other machines. After that, the system was forcibly rebooted, flushing the worm in memory but leaving the backdoor and Trojan horse in place.
To find a target to infect, the worm's threads worked in parallel. Although the early version of Code Red targeted *www.whitehouse.gov*, later versions chose a random IP address close to the host computer's own address. To speed its performance, the worm used a nonblocking socket so that a slow connection would not slow down the rest of the threads as they scanned for a connection.

What Was Learned

As of this writing, more than 6 million servers use Microsoft's IIS software. The Code Red variant that allowed unlimited root access made Code Red a virulent and dangerous piece of malicious code. Microsoft offered a patch to fix the overflow problem and prevent infection by Code Red, but many administrators neglected to apply the patch. (See Sidebar 3-6.)
Some security analysts suggested that Code Red might be "a beta test for information warfare," meaning that its powerful combination of attacks could be a prelude to a large-scale, intentional effort targeted at particular countries or groups [HUL01a]. For this reason, users and developers should pay more and careful attention to the security of their systems. Forno [FOR01] warns that security threats such as Code Red stem from our general willingness to buy and install code that does not meet minimal quality standards and from our reluctance to devote resources to the large and continuing stream of patches and corrections that flows from the vendors. As we see in Chapter 11, this problem is coupled with a lack of legal standing for users who experience seriously faulty code.

Malicious Code on the Web: Web Bugs

With the web pervading the lives of average citizens everywhere, malicious code in web pages has become a serious problem. But sometimes the malice is not always clear; code can be used to good or bad ends, depending on your perspective. In this section, we look at a generic type of code, called a web bug, to see how it can affect the code in which it is embedded.

What They Do

A web bug, sometimes called a pixel tag, clear gif, one-by-one gif, invisible gif, or beacon gif, is a hidden image on any document that can display HTML tags, such as a web page, an HTML e-mail message, or even a spreadsheet. Its creator intends the bug to be invisible, unseen by users but very useful nevertheless because it can track the activities of a web user.

For example, if you visit the Blue Nile home page, *www.bluenile.com*, the following web bug code is automatically downloaded as a one-by-one pixel image from Avenue A, a marketing agency:

<img height=1 width=1 src="http://switch.avenuea.com/action/
bluenile_homepage/v2/a/AD7029944">

What Effect They Have

Suppose you are surfing the web and load the home page for *Commercial.com*, a commercial establishment selling all kinds of houseware. If this site contains a web bug for *Market.com*, a marketing and advertising firm, then the bug places a file called a cookie on your system's hard drive. This cookie, usually containing a numeric identifier unique to you, can be used to track your surfing habits and build a demographic profile. In turn, that profile can be used to direct you to retailers in whom you may be interested. For example, *Commercial.com* may create a link to other sites, display a banner advertisement to attract you to its partner sites, or offer you content customized for your needs.

How They Work

On the surface, web bugs do not seem to be malicious. They plant numeric data but do not track personal information, such as your name and address. However, if you purchase an item at *Commercial.com*, you may be asked to supply such information. Thus, the web server can capture things such as

- your computer's IP address
- the kind of web browser you use
- your monitor's resolution
- other browser settings, such as whether you have enabled Java technology
- connection time
- previous cookie values

and more.

buying habits are, or what your personal information may be. More maliciously, the web bug can be cleverly used to review the web server's log files and to determine your IP address opening your system to hacking via the target IP address.

What Was Learned

Web bugs raise questions about privacy, and some countries are considering legislation to protect specifically from probes by web bugs. In the meantime, the Privacy Foundation has made available a tool called Bugnosis to locate web bugs and bring them to a user's attention.

We will study the privacy aspects of web bugs more in Chapter 10.

In addition, users can invoke commands from their web browsers to block cookies or at least make the users aware that a cookie is about to be placed on a system. Each option offers some inconvenience. Cookies can be useful in recording information that is used repeatedly, such as name and address. Requesting a warning message can mean almost continual interruption as web bugs attempt to place cookies on your system. Another alternative is to allow cookies but to clean them off your system periodically, either by hand or by using a commercial product.

Targeted malicious code-

So far, we have looked at anonymous code written to affect users and machines indiscriminately. Another class of malicious code is written for a particular system, for a particular application, and for a particular purpose. Many of the virus writers' techniques apply, but there are also some new ones. Bradbury [BRA06] looks at the change over time in objectives and skills of malicious code authors.

Trapdoors

A trapdoor is an undocumented entry point to a module. Developers insert trapdoors during code development, perhaps to test the module, to provide "hooks" by which to connect future modifications or enhancements, or to allow access if the module should fail in the future. In addition to these legitimate uses, trapdoors can allow a programmer access to a program once it is placed in production.

Examples of Trapdoors

Because computing systems are complex structures, programmers usually develop and test systems in a methodical, organized, modular manner, taking advantage of the way the system is composed of modules or components. Often, programmers first test each small component of the system separate from the other components, in a step called unit testing, to ensure that the component works correctly by itself. Then, developers test components together during integration testing, to see how they function as they send messages and data from one to the other. Rather than paste all the components together in a "big bang" approach, the testers group logical clusters of a few components, and each cluster is tested in a way that allows testers to control and understand what might make a component or its interface fail.

(For a more detailed look at testing, see Pfleeger and Atlee [PFL06a].)

To test a component on its own, the developer or tester cannot use the surrounding routines that prepare input or work with output. Instead, it is usually necessary to write "stubs" and "drivers," simple routines to inject data in and extract results from the component being tested. As testing continues, these stubs and drivers are discarded because they are replaced by the actual components whose functions they mimic. For example, the two modules MODA and MODB in Figure 3-10 are being tested with the driver MAIN and the stubs SORT, OUTPUT, and NEWLINE.

During both unit and integration testing, faults are usually discovered in components. Sometimes, when the source of a problem is not obvious, the developers insert debugging code in suspicious modules; the debugging code makes visible what is going on as the components execute and interact. Thus, the extra code may force components to display the intermediate results of a computation, to print the number of each step as it is executed, or to perform extra computations to check the validity of previous components.

To control stubs or invoke debugging code, the programmer embeds special control sequences in the component's design, specifically to support testing. For example, a component in a text formatting system might be designed to recognize commands such as .PAGE, .TITLE, and .SKIP. During testing, the programmer may have invoked the debugging code, using a command with a series of parameters of the form *var* = *value*. This command allows the programmer to modify the values of internal program variables during execution, either to test corrections to this component or to supply values passed to components this one calls.

Command insertion is a recognized testing practice. However, if left in place after testing, the extra commands can become a problem. They are undocumented control sequences that produce side effects and can be used as trapdoors. In fact, the Internet worm spread its infection by using just such a debugging trapdoor in an electronic mail program.

Poor error checking is another source of trapdoors. A good developer will design a system so that any data value is checked before it is used; the checking involves making sure the data type is correct as well as ensuring that the value is within acceptable bounds. But in some poorly designed systems, unacceptable input may not be caught and can be passed on for use in unanticipated ways. For example, a component's code may check for one of three expected sequences; finding none of the three, it should recognize an error. Suppose the developer uses a CASE statement to look for each of the three possibilities. A careless programmer may allow a failure simply to fall through the CASE without being flagged as an error. The *fingerd* flaw exploited by the Morris worm occurs exactly that way: A C library I/O routine fails to check whether characters are left in the input buffer before returning a pointer to a supposed next character.

Here, it often happens that not all possible binary opcode values have matching machine instructions. The undefined opcodes sometimes implement peculiar instructions, either because of an intent to test the processor design or because of an oversight by the processor designer. Undefined opcodes are the hardware counterpart of poor error checking for software.

As with viruses, trapdoors are not always bad. They can be very useful in finding security flaws. Auditors sometimes request trapdoors in production programs to insert fictitious but identifiable transactions into the system. Then, the auditors trace the flow of these transactions through the system. However, trapdoors must be documented, access to them should be strongly controlled, and they must be designed and used with full understanding of the potential consequences.

Causes of Trapdoors

Developers usually remove trapdoors during program development, once their intended usefulness is spent. However, trapdoors can persist in production programs because the developers

 forget to remove them
 intentionally leave them in the program for *testing*
 intentionally leave them in the program for *maintenance* of the finished program, or
 intentionally leave them in the program as a *covert means of access* to the component after it becomes an accepted part of a production system

The first case is an unintentional security blunder, the next two are serious exposures of the system's security, and the fourth is the first step of an outright attack. It is important to remember that the fault is not with the trapdoor itself, which can be a useful technique for program testing, correction, and maintenance. Rather, the fault is with the system development process, which does not ensure that the trapdoor is "closed" when it is no longer needed. That is, the trapdoor becomes a vulnerability if no one notices it or acts to prevent or control its use in vulnerable situations.

In general, trapdoors are a vulnerability when they expose the system to modification during execution. They can be exploited by the original developers or used by anyone who discovers the trapdoor by accident or through exhaustive trials. A system is not secure when someone believes that no one else would find the hole.

Salami Attack

We noted in Chapter 1 an attack known as a salami attack. This approach gets its name from the way odd bits of meat and fat are fused in a sausage or salami. In the same way, a salami attack merges bits of seemingly inconsequential data to yield powerful results. For example, programs often disregard small amounts of money in their computations, as when there are fractional pennies as interest or tax is calculated. Such programs may be subject to a salami attack, because the small amounts are shaved from each computation and accumulated elsewheresuch as in the programmer's bank accounta The shaved amount is so small that an individual case is unlikely to be noticed, and the accumulation can be done so that the books still balance overall. However, accumulated amounts can add up to a tidy sum, supporting a programmer's early retirement or new car. It is often the resulting expenditure, not the shaved amounts, that gets the attention of the authorities.

Examples of Salami Attacks

The classic tale of a salami attack involves interest computation. Suppose your bank pays 6.5 percent interest on your account. The interest is declared on an annual basis but is calculated monthly. If, after the first month, your bank balance is $102.87, the bank can calculate the interest in the following way. For a month with 31 days, we divide the interest rate by 365 to get the daily rate, and then multiply it by 31 to get the interest for the month. Thus, the total interest for 31 days is 31/365*0.065*102.87 = $0.5495726. Since banks deal only in full cents, a typical practice is to round down if a residue is less than half a cent, and round up if a residue is half a cent or more. However, few people check their interest computation closely, and fewer still would complain about having the amount $0.5495 rounded down to $0.54, instead of up to $0.55. Most programs that perform computations on currency recognize that because of rounding, a sum of individual computations may be a few cents different from the computation applied to the sum of the balances.

What happens to these fractional cents? The computer security folk legend is told of a programmer who collected the fractional cents and credited them to a single account: hersa The interest program merely had to balance total interest paid to interest due on the total of the balances of the individual accounts. Auditors will probably not notice the activity in one specific account. In a situation with many accounts, the roundoff error can be substantial, and the programmer's account pockets this roundoff.

But salami attacks can net more and be far more interesting. For example, instead of shaving fractional cents, the programmer may take a few cents from each account, again assuming that no individual has the desire or understanding to recompute the amount the bank reports.

Most people finding a result a few cents different from that of the bank would accept the bank's figure, attributing the difference to an error in arithmetic or a misunderstanding of the conditions under which interest is credited. Or a program might record a $20 fee for a particular service, while the company standard is $15. If unchecked, the extra $5 could be credited to an account of the programmer's choice. The amounts shaved are not necessarily small: One attacker was able to make withdrawals of $10,000 or more against accounts that had shown little recent activity; presumably the attacker hoped the owners were ignoring their accounts.

Why Salami Attacks Persist

Computer computations are notoriously subject to small errors involving rounding and truncation, especially when large numbers are to be combined with small ones. Rather than document the exact errors, it is easier for programmers and users to accept a small amount of error as natural and unavoidable. To reconcile accounts, the programmer includes an error correction in computations. Inadequate auditing of these corrections is one reason why the salami attack may be overlooked.

Usually the source code of a system is too large or complex to be audited for salami attacks, unless there is reason to suspect one. Size and time are definitely on the side of the malicious programmer.

Rootkits and the Sony XCP

A later variation on the virus theme is the rootkit. A rootkit is a piece of malicious code that goes to great lengths not to be discovered or, if discovered and removed, to reestablish itself whenever possible. The name rootkit refers to the code's attempt to operate as *root*, the superprivileged user of a Unix system.

A typical rootkit will interfere with the normal interaction between a user and the operating system as follows. Whenever the user executes a command that would show the rootkit's presence, for example, by listing files or processes in memory, the rootkit intercepts the call and filters the result returned to the user so that the rootkit does not appear. For example, if a directory contains six files, one of which is the rootkit, the rootkit will pass the directory command to the operating system, intercept the result, delete the listing for itself, and display to the user only the five other files. The rootkit will also adjust such things as file size totals to conceal itself. Notice that the rootkit needs to intercept this data between the result and the presentation interface (the program that formats results for the user to see).

Ah, two can play that game. Suppose you suspect code is interfering with your file display program. Then you write a program that displays files, then examines the disk and file system directly to enumerate files, and compares these two results. A rootkit revealer is just such a program.

A computer security expert named Mark Russinovich developed a rootkit revealer, which he ran on one of his systems. He was surprised to find a rootkit [RUS05]. On further investigation he determined the rootkit had been installed when he loaded and played a music CD on his computer. Felten and Halderman [FEL06] extensively examined this rootkit, named XCP (short for extended copy protection).

What XCP Does

The XCP rootkit prevents a user from copying a music CD, while allowing the CD to be played as music. To do this, it includes its own special music player that is allowed to play the CD. But XCP interferes with any other access to the protected music CD by garbling the result any other process would obtain in trying to read from the CD.

The rootkit has to install itself when the CD is first inserted in the PC's drive. To do this, XCP depends on a "helpful" feature of Windows: With "autorun" Windows looks for a file with a specific name, and if it finds that, it opens and executes the file without the user's involvement. (The file name can be configured in Windows, although it is *autorun.exe* by default.) You can disable the autorun feature; see [FEL06] for details.

XCP has to hide from the user so that the user cannot just remove it. So the rootkit does as we just described: It blocks display of any program whose name begins with $sys$ (which is how it is named). Unfortunately for Sony, this feature concealed not just XCP but any program beginning with $sys$ from any source, malicious or not. So any virus writer could conceal a virus just by naming it $sys$virus-1, for example.

Sony did two things wrong: First, as we just observed, it distributed code that inadvertently opens an unsuspecting user's system to possible infection by other writers of malicious code.

Second, Sony installs that code without the user's knowledge, much less consent, and it employs strategies to prevent the code's removal.

Patching the Penetration

The story of XCP became very public in November 2005 when Russinovich described what he found and several news services picked up the story. Faced with serious negative publicity, Sony decided to release an uninstaller for the XCP rootkit. Remember, however, from the start of this chapter why "penetrate and patch" was abandoned as a security strategy? The pressure for a quick repair sometimes led to shortsighted solutions that addressed the immediate situation and not the underlying cause: Fixing one problem often caused a failure somewhere else.

Sony's uninstaller itself opened serious security holes. It was presented as a web page that downloaded and executed the uninstaller. But the programmers did not check what code they were executing, so the web page would run any code from any source, not just the intended uninstaller. And worse, the downloading code remained even after uninstalling XCP, meaning that the vulnerability persisted. (In fact, Sony used two different rootkits from two different sources and, remarkably, the uninstallers for both rootkits had this same vulnerability.)

How many computers were infected by this rootkit? Nobody knows for sure. Kaminsky [KAM06] found 500,000 references in DNS tables to the site the rootkit contacts, but some of those DNS entries could support accesses by hundreds or thousands of computers. How many users of computers on which the rootkit was installed are aware of it? Again nobody knows, nor does anybody know how many of those installations might not yet have been removed.

Felten and Halderman [FEL06] present an interesting analysis of this situation, examining how digital rights management (copy protection for digital media such as music CDs) leads to requirements very similar to those for a malicious code developer. Levine et al. [LEV06] consider the full potential range of rootkit behavior as a way of determining how to defend against them.

Schneier [SCH06b] considers everyone who, maliciously or not, wants to control a PC: Automatic software updates, antivirus tools, spyware, even applications all do many things without the user's express permission or even knowledge. They also conspire against the user:

Sony worked with major antivirus vendors so its rootkit would not be detected, because keeping the user uninformed was better for all of them.

Privilege Escalation

Programs run in a context: Their access rights and privileges are controlled by that context. Most programs run in the context of the invoking user. If system access rights are set up correctly, you can create, modify, or delete items you own, but critical system objects are protected by being outside your context. Malicious code writers want to be able to access not just your objects but those outside your context as well. To do this, the malicious code has to run with privileges higher than you have. A privilege escalation attack is a means for malicious code to be launched by a user with lower privileges but run with higher privileges.

A Privilege Escalation Example

In April 2006, Symantec announced a fix to a flaw in their software (bulletin Sym06-007). Symantec produces security software, such as virus scanners and blockers, e-mail spam filters, and system integrity tools. So that a user's product will always have up-to-date code and supporting data (such as virus definition files), Symantec has a Live Update option by which the product periodically fetches and installs new versions from a Symantec location. A user can also invoke Live Update at any time to get up-to-the-minute updates. The Live Update feature has to run with elevated privileges because it will download and install programs in the system program directory. The update process actually involves executing several programs, which we will call LU1, LU2, Sys3, and Sys4; LU1 and LU2 are components of Live Update, and Sys3 and Sys4 are standard components of the operating system. These four pieces complete the downloading and installation.

Operating systems use what is called a search path to find programs to execute. The search path is a list of directories or folders in which to look for a program that is called. When a program A calls a program B, the operating system looks for B in the first directory specified in the search path. If the operating system finds such a program, it executes it; otherwise, it

continues looking in the subsequent directories in the search path until it finds B or it fails to find B by the end of the list. The operating system uses the first B it finds. The user can change the search path so a user's program B would be run instead of another program of the same name in another directory. You can always specify a program's location explicitly for example, c:\program files\ symantec\LU1to control precisely which version runs.
In some releases for the Macintosh, Symantec allowed Live Update to find programs from the search path instead of by explicit location. Remember that Live Update runs with elevated privileges; it passes those elevated privileges along to Sys3 and Sys4. But if the user sets a search path starting in the user's space and the user happens to have a program named Sys3, *the user's* version of Sys3 runs with elevated privileges.

Impact of Privilege Escalation

A malicious code writer likes a privilege escalation. Creating, installing, or modifying a system file is difficult, but it is easier to load a file into the user's space. In this example, the malicious code writer only has to create a small shell program, name it Sys3, store it anywhere (even in a temporary directory), reset the search path, and invoke a program (Live Update). Each of these actions is common for nonmalicious downloaded code.
The result of running this attack is that the malicious version of Sys3 receives control in privileged mode, and from that point it can replace operating system files, download and install new code, modify system tables, and inflict practically any other harm. Having run once with higher privilege, the malicious code can set a flag to receive elevated privileges in the future.

Interface Illusions

The name for this attack is borrowed from Elias Levy [LEV04]. An interface illusion is a spoofing attack in which all or part of a web page is false. The object of the attacker is to convince the user to do something inappropriate, for example, to enter personal banking information on a site that is not the bank's, to click *yes* on a button that actually means *no*,or simply to scroll the screen to activate an event that causes malicious software to be installed on the victim's machine. Levy's excellent article gives other excellent examples.
The problem is that every dot of the screen is addressable. So if a genuine interface can paint dot 17 red, so can a malicious interface. Given that, a malicious interface can display fake address bars, scroll bars that are not scroll bars, and even a display that looks identical to the real thing, because it is identical in all ways the attacker wants it to be.
Nothing here is new, of course. People diligently save copies of e-mail messages as proof that they received such a message when, in fact, a simple text editor will produce any authentic-looking message you want. System pranksters like to send facetious messages to unsuspecting users, warning that the computer is annoyed. These all derive from the same point: There is nothing unique, no trusted path assured to be a private and authentic communication channel directly to the user.

Keystroke Logging

Remember the movies in which a detective would spy a note pad on a desk, hold it up to the light, and read the faint impression of a message that had been written and then torn off that pad? There is a computer counterpart of that tactic, too.
First, recognize that there is not a direct path between a key you press on your keyboard and the program (let's say a word processor) that handles that keystroke. When you press A, it activates a switch that generates a signal that is received by a device driver, converted and analyzed and passed along, until finally your word processor receives the A; there is still more conversion, analysis, and transmission until the A appears on your screen. Many programs cooperate in this chain. At several points along the way you could change a program so that A would appear on the screen when you pressed W if you wanted.
If all programs work as intended, they receive and send characters efficiently and discard each character as soon as it is sent and another arrives. A malicious program called a keystroke logger retains a surreptitious copy of all keys pressed. Most keystrokes are uninteresting, but we may want to protect the privacy of identification numbers, authentication strings, and love notes.
A keystroke logger can be independent (retaining a log of every key pressed) or it can be tied to a certain program, retaining data only when a particular program (such as a banking application) runs.

Man-in-the-Middle Attacks

A keystroke logger is a special form of the more general man-in-the-middle attack. There are two versions of this attack: we cover the application type here and then expand on the concept in Chapter 7 on networks.
A man-in-the-middle attack is one in which a malicious program interjects itself between two other programs, typically between a user's input and an application's result. One example of a man-in-the-middle attack could be a program that operated between your word processor and the file system, so that each time you thought you were saving your file, the middle program prevented that, or scrambled your text or encrypted your file. What ransom would you be willing to pay to get back the paper on which you had been working for the last week?

Timing Attacks

Computers are fast, and they work far faster than humans can follow. But, as we all know,the time it takes a computer to perform a task depends on the size of the task: Creating 20 database records takes approximately twice as long as creating 10. So, in theory at least, if we could measure computer time precisely, and we could control other things being done in the computer, we could infer the size of the computer's input. In most situations size is relatively uninteresting to the attacker. But in cryptography, even the smallest bit of information can be significant.
Brumley and Boneh [BRU05] investigated a program that does RSA encryption for web sites.
The authors try to derive the key by successive guesses of increasing value as possibilities for the key. Although the details of the attack are beyond the scope of this book, the idea is to use a trick in the optimization of RSA encryption. Grossly oversimplified, encryption with numbers less than the key take successively longer amounts of time as the numbers get closer to the key, but then the time to encrypt drops sharply once the key value is passed.
Brute force guessing is prohibitive in time. But the authors show that you don't have to try all values. You infer the key a few bits at a time from the left (most significant bit). So you might try 00xxx, 01xxx, 10xxx, and 11xxx, noticing that the time to compute rises from 00xxx to 01xxx, rises from 01xxx to 10xxx, and falls between 10xxx and 11xxx. This tells you the key value is between 10xxx and 11xxx. The attack works with much longer keys (on the order of 1000 bits) and the authors use about a million possibilities for the xxx portion. Still, this technique allows the authors to infer the key a bit at a time, all based on the amount of time the encryption takes. The authors performed their experiments on a network, not with precise local timing instruments, and still they were able to deduce keys.
Cryptography is the primary area in which speed and size are information that should not be revealed. But you should be aware that malicious code can perform similar attacks undetected.

Covert Channels: Programs That Leak Information

So far, we have looked at malicious code that performs unwelcome actions. Next, we turn to programs that communicate information to people who should not receive it. The communication travels unnoticed, accompanying other, perfectly proper, communications. The general name for these extraordinary paths of communication is covert channels. The concept of a covert channel comes from a paper by Lampson [LAM73]; Millen [MIL88] presents a good taxonomy of covert channels.
Suppose a group of students is preparing for an exam for which each question has four choices (a, b, c, d); one student in the group, Sophie, understands the material perfectly and she agrees to help the others. She says she will reveal the answers to the questions, in order, by coughing once for answer "a," sighing for answer "b," and so forth. Sophie uses a communications channel that outsiders may not notice; her communications are hidden in an open channel. This communication is a human example of a covert channel.
We begin by describing how a programmer can create covert channels. The attack is more complex than one by a lone programmer accessing a data source. A programmer who has direct access to data can usually just read the data and write it to another file or print it out. If, however, the programmer is one step removed from the datafor example, outside the organization owning the datathe programmer must figure how to get at the data. One way is to supply a bona fide program with a built-in Trojan horse; once the horse is enabled, it finds and transmits the data. However, it would be too bold to generate a report labeled "Send this report to Jane Smith in Camden, Maine"; the programmer has to arrange to

extract the data more surreptitiously. Covert channels are a means of extracting data clandestinely.
Figure 3-11 shows a "service program" containing a Trojan horse that tries to copy information from a legitimate user (who is allowed access to the information) to a "spy" (who ought not be allowed to access the information). The user may not know that a Trojan horse is running and may not be in collusion to leak information to the spy.

Covert Channel Overview

A programmer should not have access to sensitive data that a program processes after the program has been put into operation. For example, a programmer for a bank has no need to access the names or balances in depositors' accounts. Programmers for a securities firm have no need to know what buy and sell orders exist for the clients. During program testing, access to the real data may be justifiable, but not after the program has been accepted for regular use.
Still, a programmer might be able to profit from knowledge that a customer is about to sell a large amount of a particular stock or that a large new account has just been opened. Sometimes a programmer may want to develop a program that secretly communicates some of the data on which it operates. In this case, the programmer is the "spy," and the "user" is whoever ultimately runs the program written by the programmer.

How to Create Covert Channels

A programmer can always find ways to communicate data values covertly. Running a program that produces a specific output report or displays a value may be too obvious. For example, in some installations, a printed report might occasionally be scanned by security staff before it is delivered to its intended recipient.
If printing the data values themselves is too obvious, the programmer can encode the data values in another innocuous report by varying the format of the output, changing the lengths of lines, or printing or not printing certain values. For example, changing the word "TOTAL" to "TOTALS" in a heading would not be noticed, but this creates a 1-bit covert channel. The absence or presence of the S conveys one bit of information. Numeric values can be inserted in insignificant positions of output fields, and the number of lines per page can be changed.
Examples of these subtle channels are shown in Figure 3-12.

Storage Channels

Some covert channels are called storage channels because they pass information by using the presence or absence of objects in storage.
A simple example of a covert channel is the file lock channel. In multiuser systems, files can be "locked" to prevent two people from writing to the same file at the same time (which could corrupt the file, if one person writes over some of what the other wrote). The operating system or database management system allows only one program to write to a file at a time by blocking, delaying, or rejecting write requests from other programs. A covert channel can signal one bit of information by whether or not a file is locked.
Remember that the service program contains a Trojan horse written by the spy but run by the unsuspecting user. As shown in Figure 3-13, the service program reads confidential data (to which the spy should not have access) and signals the data one bit at a time by locking or not locking some file (any file, the contents of which are arbitrary and not even modified). The service program and the spy need a common timing source, broken into intervals. To signal a 1, the service program locks the file for the interval; for a 0, it does not lock. Later in the interval the spy tries to lock the file itself. If the spy program cannot lock the file, it knows the service program must have locked the file, and thus the spy program concludes the service program is signaling a 1; if the spy program can lock the file, it knows the service program is signaling a 0.
This same approach can be used with disk storage quotas or other resources. With disk storage, the service program signals a 1 by creating an enormous file, so large that it consumes most of the available disk space. The spy program later tries to create a large file. If it succeeds, the spy program infers that the service program did not create a large file, and so the service program is signaling a 0; otherwise, the spy program infers a 1. Similarly the existence of a file or other resource of a particular name can be used to signal. Notice that the spy does not need access to a file itself; the mere existence of the file is adequate to signal. The spy can determine the existence of a file it cannot read by trying to create a file

of the same name; if the request to create is rejected, the spy determines that the service program has such a file.

To signal more than one bit, the service program and the spy program signal one bit in each time interval. Figure 3-14 shows a service program signaling the string 100 by toggling the existence of a file.

bakeries, banks, and other commercial establishments have a machine to distribute numbered tickets so that customers can be served in the order in which they arrived. Some computing systems provide a similar server of unique identifiers, usually numbers, used to name temporary files, to tag and track messages, or to record auditable events. Different processes can request the next unique identifier from the server. But two cooperating processes can use the server to send a signal: The spy process observes whether the numbers it receives are sequential or whether a number is missing. A missing number implies that the service program also requested a number, thereby signaling 1.

In all of these examples, the service program and the spy need access to a shared resource (such as a file, or even knowledge of the existence of a file) and a shared sense of time. As shown, shared resources are common in multiuser environments, where the resource may be as seemingly innocuous as whether a file exists, a device is free, or space remains on disk. A source of shared time is also typically available, since many programs need access to the current system time to set timers, to record the time at which events occur, or to synchronize activities. Karger and Wray [KAR91b] give a real-life example of a covert channel in the movement of a disk's arm and then describe ways to limit the potential information leakage from this channel.

Transferring data one bit at a time must seem awfully slow. But computers operate at such speeds that even the minuscule rate of 1 bit per millisecond (1/1000 second) would never be noticed but could easily be handled by two processes. At that rate of 1000 bits per second (which is unrealistically conservative), this entire book could be leaked in about two days.

Increasing the rate by an order of magnitude or two, which is still quite conservative, reduces the transfer time to minutes.

Timing Channels

Other covert channels, called timing channels, pass information by using the speed at which things happen. Actually, timing channels are shared resource channels in which the shared resource is time.

A service program uses a timing channel to communicate by using or not using an assigned amount of computing time. In the simple case, a multiprogrammed system with two user processes divides time into blocks and allocates blocks of processing alternately to one process and the other. A process is offered processing time, but if the process is waiting for another event to occur and has no processing to do, it rejects the offer. The service process either uses its block (to signal a 1) or rejects its block (to signal a 0). Such a situation is shown in Figure 3-15, first with the service process and the spy's process alternating, and then with the service process communicating the string 101 to the spy's process. In the second part of the example, the service program wants to signal 0 in the third time block. It will do this by using just enough time to determine that it wants to send a 0 and then pause.

The spy process then receives control for the remainder of the time block. multiuser computing systems typically have more than just two active processes. The only complications added by more processes are that the two cooperating processes must adjust their timings and deal with the possible interference from others. For example, with the unique identifier channel, other processes will also request identifiers. If on average *n* other processes will request *m* identifiers each, then the service program will request more than *n*m* identifiers for a 1 and no identifiers for a 0. The gap dominates the effect of all other processes. Also, the service process and the spy's process can use sophisticated coding techniques to compress their communication and detect and correct transmission errors caused by the effects of other unrelated processes.

Identifying Potential Covert Channels

In this description of covert channels, ordinary things, such as the existence of a file or time used for a computation, have been the medium through which a covert channel communicates. Covert channels are not easy to find because these media are so numerous and frequently used. Two relatively old techniques remain the standards for locating potential covert channels. One works by analyzing the resources of a system, and the other works at the source code level.

Shared Resource Matrix

Since the basis of a covert channel is a shared resource, the search for potential covert channels involves finding all shared resources and determining which processes can write to and read from the resources. The technique was introduced by Kemmerer [KEM83]. Although laborious, the technique can be automated.

To use this technique, you construct a matrix of resources (rows) and processes that can access them (columns). The matrix entries are R for "can read (or observe) the resource" and M for "can set (or modify, create, delete) the resource." For example, the file lock channel has the matrix shown in Table 3-3.

	Service Process	**Spy's Process**
Locked	R, M	R, M
Confidential data	R	

You then look for two columns and two rows having the following pattern:

	M		R	
	R			

This pattern identifies two resources and two processes such that the second process is not allowed to read from the second resource. However, the first process can pass the information to the second by reading from the second resource and signaling the data through the first resource. Thus, this pattern implies the potential information flow as shown here.

	M		R	
	R		$\boxed{R}$	

Next, you complete the shared resource matrix by adding these implied information flows, and analyzing the matrix for undesirable flows. Thus, you can tell that the spy's process can read the confidential data by using a covert channel through the file lock, as shown in Table 3-4.

Table 3-4. Complete Information Flow Matrix.

	Service Process	**Spy's Process**
Locked	R, M	R, M
Confidential data	R	R

Information Flow Method

Denning [DEN76a] derived a technique for flow analysis from a program's syntax. Conveniently, this analysis can be automated within a compiler so that information flow potentials can be detected while a program is under development.

Using this method, we can recognize nonobvious flows of information between statements in a program. For example, we know that the statement B:=A, which assigns the value of A to the variable B, obviously supports an information flow from A to B. This type of flow is called an "explicit flow." Similarly, the pair of statements B:=A; C:=B indicates an information flow from A to C (by way of B). The conditional statement IF D=1 THEN B:=A has two flows: from A to B because of the assignment, but also from D to B, because the

value of B can change if and only if the value of D is 1. This second flow is called an "implicit flow."
The statement B:=*fcn*(*args*) supports an information flow from the function *fcn* to B. At a superficial level, we can say that there is a potential flow from the arguments *args* to B. However, we could more closely analyze the function to determine whether the function's value depended on all of its arguments and whether any global values, not part of the argument list, affected the function's value. These information flows can be traced from the bottom up: At the bottom there must be functions that call no other functions, and we can analyze them and then use those results to analyze the functions that call them. By looking at the elementary functions first, we could say definitively whether there is a potential information flow from each argument to the function's result and whether there are any flows from global variables. Table 3-5 lists several examples of syntactic information flows.

Table 3-5. Syntactic Information Flows.

Statement	Flow
B:=A	from A to B
IF C=1 THEN B:=A	from A to B; from C to B
FOR K:=1 to N DO *stmts* END	from K to *stmts*
WHILE K>0 DO *stmts* END	from K to *stmts*
CASE (*exp*) *val1*: *stmts*	from *exp* to *stmts*
B:=*fcn*(*args*)	from *fcn* to B
OPEN FILE *f*	none
READ (*f*, X)	from file *f* to X
WRITE (*f*, X)	from X to file *f*

Finally, we put all the pieces together to show which outputs are affected by which inputs. Although this analysis sounds frightfully complicated, it can be automated during the syntax analysis portion of compilation. This analysis can also be performed on the higher-level design specification.

Covert Channel Conclusions

Covert channels represent a real threat to secrecy in information systems. A covert channel attack is fairly sophisticated, but the basic concept is not beyond the capabilities of even an average programmer. Since the subverted program can be practically any user service, such as a printer utility, planting the compromise can be as easy as planting a virus or any other kind of Trojan horse. And recent experience has shown how readily viruses can be planted. Capacity and speed are not problems; our estimate of 1000 bits per second is unrealistically low, but even at that rate much information leaks swiftly. With modern hardware architectures, certain covert channels inherent in the hardware design have capacities of millions of bits per second. And the attack does not require significant finance. Thus, the attack could be very effective in certain situations involving highly sensitive data.
For these reasons, security researchers have worked diligently to develop techniques for closing covert channels. The closure results have been bothersome; in ordinarily open environments, there is essentially no control over the subversion of a service program, nor is there an effective way of screening such programs for covert channels. And other than in a few very high security systems, operating systems cannot control the flow of information from a covert channel. The hardware-based channels cannot be closed, given the underlying hardware architecture.
For variety (or sobriety), Kurak and McHugh [KUR92] present a very interesting analysis of covert signaling through graphic images.[4] In their work they demonstrate that two different images can be combined by some rather simple arithmetic on the bit patterns of digitized pictures. The second image in a printed copy is undetectable to the human eye, but it can easily be separated and reconstructed by the spy receiving the digital version of the image. Byers [BYE04] explores the topic in the context of covert data passing through pictures on the Internet.

[4] This form of data communication is called *steganography,* w hich means the art of concealing data in clear sight.

Although covert channel demonstrations are highly speculativereports of actual covert channel attacks just do not existthe analysis is sound. The mere possibility of their existence calls for more rigorous attention to other aspects of security, such as program development analysis, system architecture analysis, and review of output.

Controls against program threats.

The picture we have just described is not pretty. There are many ways a program can fail and many ways to turn the underlying faults into security failures. It is of course better to focus on prevention than cure; how do we use controls during software development the specifying, designing, writing, and testing of the programto find and eliminate the sorts of exposures we have discussed? The discipline of software engineering addresses this question more globally, devising approaches to ensure the quality of software. In this book, we provide an overview of several techniques that can prove useful in finding and fixing security flaws.

For more depth, we refer you to texts such as Pfleeger et al. [PFL01] and Pfleeger and Atlee [PFL06a].

In this section we look at three types of controls: developmental, operating system, and administrative. We discuss each in turn.

Developmental Controls

Many controls can be applied during software development to ferret out and fix problems. So let us begin by looking at the nature of development itself, to see what tasks are involved in specifying, designing, building, and testing software.

The Nature of Software Development

Software development is often considered a solitary effort; a programmer sits with a specification or design and grinds out line after line of code. But in fact, software development is a collaborative effort, involving people with different skill sets who combine their expertise to produce a working product. Development requires people who can

- *specify* the system, by capturing the requirements and building a model of how the system should work from the users' point of view
- *design* the system, by proposing a solution to the problem described by the requirements and building a model of the solution
- *implement* the system, by using the design as a blueprint for building a working solution
- *test* the system, to ensure that it meets the requirements and implements the solution as called for in the design
- *review* the system at various stages, to make sure that the end products are consistent with the specification and design models
- *document* the system, so that users can be trained and supported
- *manage* the system, to estimate what resources will be needed for development and to track when the system will be done
- *maintain* the system, tracking problems found, changes needed, and changes made, and evaluating their effects on overall quality and functionality

One person could do all these things. But more often than not, a team of developers works together to perform these tasks. Sometimes a team member does more than one activity; a tester can take part in a requirements review, for example, or an implementer can write documentation. Each team is different, and team dynamics play a large role in the team's success.

Keep in mind the kinds of sophisticated attacks described in the previous section. Balfanz [BAL04] reminds us that we must design systems that are both secure and usable, recommending these points:

- You can't retrofit usable security.
- Tools aren't solutions.
- Mind the upper layers.
- Keep the customers satisfied.
- Think locally; act locally.

We can examine product and process to see how both contribute to quality and in particular to security as an aspect of quality. Let us begin with the product, to get a sense of how we recognize high-quality secure software.

Modularity, Encapsulation, and Information Hiding

Code usually has a long shelf-life and is enhanced over time as needs change and faults are found and fixed. For this reason, a key principle of software engineering is to create a design or code in small, self-contained units, called components or modules; when a system is written this way, we say that it is modular. Modularity offers advantages for program development in general and security in particular.

If a component is isolated from the effects of other components, then it is easier to trace a problem to the fault that caused it and to limit the damage the fault causes. It is also easier

to maintain the system, since changes to an isolated component do not affect other components. And it is easier to see where vulnerabilities may lie if the component is isolated.
We call this isolation encapsulation.
Information hiding is another characteristic of modular software. When information is hidden, each component hides its precise implementation or some other design decision from the others. Thus, when a change is needed, the overall design can remain intact while only the necessary changes are made to particular components.
Let us look at these characteristics in more detail.

Modularity

Modularization is the process of dividing a task into subtasks. This division is done on a logical or functional basis. Each component performs a separate, independent part of the task. Modularity is depicted in Figure 3-16. The goal is to have each component meet four conditions:
 single-purpose: performs one function
 small: consists of an amount of information for which a human can readily grasp both structure and content
 simple: is of a low degree of complexity so that a human can readily understand the purpose and structure of the module
 independent: performs a task isolated from other modules
Other component characteristics, such as having a single input and single output or using a limited set of programming constructs, indicate modularity. From a security standpoint, modularity should improve the likelihood that an implementation is correct.
In particular, smallness is an important quality that can help security analysts understand what each component does. That is, in good software, design and program units should be only as large as needed to perform their required functions. There are several advantages to having small, independent components.
 Maintenance. If a component implements a single function, it can be replaced easily with a revised one if necessary. The new component may be needed because of a change in requirements, hardware, or environment. Sometimes the replacement is an enhancement, using a smaller, faster, more correct, or otherwise better module. The interfaces between this component and the remainder of the design or code are few and well described, so the effects of the replacement are evident.
 Understandability. A system composed of many small components is usually easier to comprehend than one large, unstructured block of code.
 Reuse. Components developed for one purpose can often be reused in other systems.
Reuse of correct, existing design or code components can significantly reduce the difficulty of implementation and testing.
 Correctness. A failure can be quickly traced to its cause if the components perform only one task each.
 Testing. A single component with well-defined inputs, outputs, and function can be tested exhaustively by itself, without concern for its effects on other modules (other than the expected function and output, of course).
Security analysts must be able to understand each component as an independent unit and be assured of its limited effect on other components.
A modular component usually has high cohesion and low coupling. By cohesion, we mean that all the elements of a component have a logical and functional reason for being there; every aspect of the component is tied to the component's single purpose. A highly cohesive component has a high degree of focus on the purpose; a low degree of cohesion means that the component's contents are an unrelated jumble of actions, often put together because of time-dependencies or convenience.
Coupling refers to the degree with which a component depends on other components in the system. Thus, low or loose coupling is better than high or tight coupling because the loosely coupled components are free from unwitting interference from other components. This difference in coupling is shown in Figure 3-17.

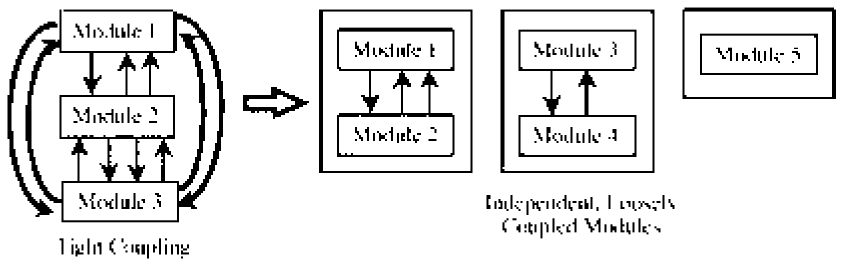


Figure 3-17. Coupling.

Encapsulation

Encapsulation hides a component's implementation details, but it does not necessarily mean complete isolation. Many components must share information with other components, usually with good reason. However, this sharing is carefully documented so that a component is affected only in known ways by others in the system. Sharing is minimized so that the fewest interfaces possible are used. Limited interfaces reduce the number of covert channels that can be constructed.

An encapsulated component's protective boundary can be translucent or transparent, as needed. Berard [BER00] notes that encapsulation is the "technique for packaging the information [inside a component] in such a way as to hide what should be hidden and make visible what is intended to be visible."

Information Hiding

Developers who work where modularization is stressed can be sure that other components will have limited effect on the ones they write. Thus, we can think of a component as a kind of black box, with certain well-defined inputs and outputs and a well-defined function. Other components' designers do not need to know *how* the module completes its function; it is enough to be assured that the component performs its task in some correct manner.

This concealment is the information hiding, depicted in Figure 3-18. Information hiding is desirable because developers cannot easily and maliciously alter the components of others if they do not know how the components work.

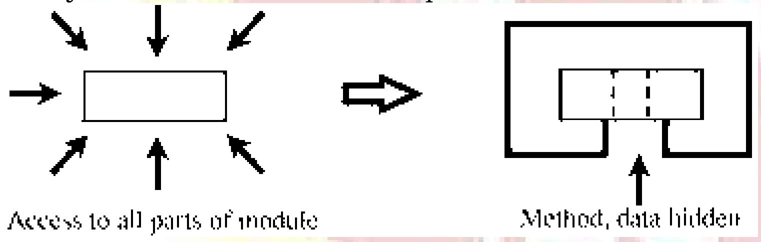


Figure 3-18. Information Hiding.

These three characteristics modularity, encapsulation, and information hidingare fundamental principles of software engineering. They are also good security practices because they lead to modules that can be understood, analyzed, and trusted.

Mutual Suspicion

Programs are not always trustworthy. Even with an operating system to enforce access limitations, it may be impossible or infeasible to bound the access privileges of an untested program effectively. In this case, the user U is legitimately suspicious of a new program P. However, program P may be invoked by another program, Q. There is no way for Q to know that P is correct or proper, any more than a user knows that of P.

Therefore, we use the concept of mutual suspicion to describe the relationship between two programs. Mutually suspicious programs operate as if other routines in the system were malicious or incorrect. A calling program cannot trust its called subprocedures to be correct, and a called subprocedure cannot trust its calling program to be correct. Each protects its interface data so that the other has only limited access. For example, a procedure to sort the entries in a list cannot be trusted not to modify those elements, while that procedure cannot trust its caller to provide any list at all or to supply the number of elements predicted.

Confinement

Confinement is a technique used by an operating system on a suspected program. A confined program is strictly limited in what system resources it can access. If a program is not trustworthy, the data it can access are strictly limited. Strong confinement would be helpful in limiting the spread of viruses. Since a virus spreads by means of transitivity and shared data, all the data and programs within a single compartment of a confined program can affect only the data and programs in the same compartment. Therefore, the virus can spread only to things in that compartment; it cannot get outside the compartment.

Genetic Diversity

At your local electronics shop you can buy a combination printerscannercopierfax machine. It comes at a good price (compared to costs of the four separate components) because there is considerable overlap in functionality among those four. It is compact, and you need only install one thing on your system, not four. But if any part of it fails, you lose a lot of capabilities all at once.

Related to the argument for modularity and information hiding and reuse or interchangeability of software components, some people recommend genetic diversity: it is risky having many components of a system come from one source, they say.

Geer at al. [GEE03a] wrote a report examining the monoculture of computing dominated by one manufacturer: Microsoft today, IBM yesterday, unknown tomorrow. They look at the parallel in agriculture where an entire crop is vulnerable to a single pathogen. Malicious code from the Morris worm to the Code Red virus was especially harmful because a significant proportion of the world's computers ran versions of the same operating systems (Unix for Morris, Windows for Code Red). Geer refined the argument in [GEE03b], which was debated by Whitaker [WHI03b] and Aucsmith [AUC03].

Tight integration of products is a similar concern. The Windows operating system is tightly linked to Internet Explorer, the Office Suite, and the Outlook e-mail handler. A vulnerability in one of these can also affect the others. Because of the tight integration, fixing a vulnerability in one can have an impact on the others, whereas a vulnerability in another vendor's browser, for example, can affect Word only to the extent they communicate through a well-defined interface.

Peer Reviews

We turn next to the process of developing software. Certain practices and techniques can assist us in finding real and potential security flaws (as well as other faults) and fixing them before we turn the system over to the users. Pfleeger et al. [PFL01] recommend several key techniques for building what they call "solid software":

- peer reviews
- hazard analysis
- testing
- good design
- prediction
- static analysis
- configuration management
- analysis of mistakes

Here, we look at each practice briefly, and we describe its relevance to security controls. We begin with peer reviews.

You have probably been doing some form of review for as many years as you have been writing code: desk-checking your work or asking a colleague to look over a routine to ferret out any problems. Today, a software review is associated with several formal process steps to make it more effective, and we review any artifact of the development process, not just code. But the essence of a review remains the same: sharing a product with colleagues able to comment about its correctness. There are careful distinctions among three types of peer reviews:

- *Review:* The artifact is presented informally to a team of reviewers; the goal is consensus and buy-in before development proceeds further.
- *Walk-through:* The artifact is presented to the team by its creator, who leads and controls the discussion. Here, education is the goal, and the focus is on learning about a single document.
- *Inspection*: This more formal process is a detailed analysis in which the artifact is checked against a prepared list of concerns. The creator does not lead the discussion, and the fault identification and correction are often controlled by statistical measurements.

A wise engineer who finds a fault can deal with it in at least three ways:

1. by learning how, when, and why errors occur
2. by taking action to prevent mistakes

3. by scrutinizing products to find the instances and effects of errors that were missed

Peer reviews address this problem directly. Unfortunately, many organizations give only lip service to peer review, and reviews are still not part of mainstream software engineering activities.

But there are compelling reasons to do reviews. An overwhelming amount of evidence suggests that various types of peer review in software engineering can be extraordinarily effective. For example, early studies at Hewlett-Packard in the 1980s revealed that those developers performing peer review on their projects enjoyed a significant advantage over those relying only on traditional dynamic testing techniques, whether black box or white box. Figure 3-19 compares the fault discovery rate (that is, faults discovered per hour) among white-box testing, black-box testing, inspections, and software execution. It is clear that inspections discovered far more faults in the same period of time than other alternatives [GRA87]. This result is particularly compelling for large, secure systems, where live running for fault discovery may not be an option.

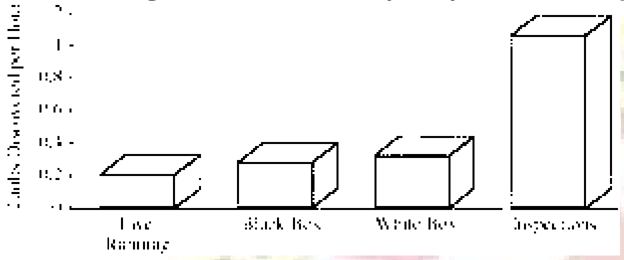


Figure 3-19. Fault Discovery Rate Reported at Hewlett-Packard.

Researchers and practitioners have repeatedly shown the effectiveness of reviews. For instance, Jones [JON91] summarized the data in his large repository of project information to paint a picture of how reviews and inspections find faults relative to other discovery activities. Because products vary so wildly by size, Table 3-6 presents the fault discovery rates relative to the number of thousands of lines of code in the delivered product.

Table 3-6. Faults Found During Discovery Activities.

Discovery Activity	**Faults Found (Per Thousand Lines of Code)**
Requirements review	2.5
Design review	5.0
Code inspection	10.0
Integration test	3.0
Acceptance test	2.0

The inspection process involves several important steps: planning, individual preparation, a logging meeting, rework, and reinspection. Details about how to perform reviews and inspections can be found in software engineering books such as [PFL01] and [PFL06a]. During the review process, someone should keep careful track of what each reviewer discovers and how quickly he or she discovers it. This log suggests not only whether particular reviewers need training but also whether certain kinds of faults are harder to find than others.

Additionally, a root cause analysis for each fault found may reveal that the fault could have been discovered earlier in the process. For example, a requirements fault that surfaces during a code review should probably have been found during a requirements review. If there are no requirements reviews, you can start performing them. If there are requirements reviews, you can examine why this fault was missed and then improve the requirements review process.

The fault log can also be used to build a checklist of items to be sought in future reviews. The review team can use the checklist as a basis for questioning what can go wrong and where.

In particular, the checklist can remind the team of security breaches, such as unchecked buffer overflows, that should be caught and fixed before the system is placed in the field. A rigorous design or code review can locate trapdoors, Trojan horses, salami attacks, worms, viruses, and other program flaws. A crafty programmer can conceal some of these flaws, but the chance of discovery rises when competent programmers review the design and code, especially when the components are small and encapsulated. Management should use demanding reviews throughout development to ensure the ultimate security of the programs.

Hazard Analysis

Hazard analysis is a set of systematic techniques intended to expose potentially hazardous system states. In particular, it can help us expose security concerns and then identify prevention or mitigation strategies to address them. That is, hazard analysis ferrets out likely causes of problems so that we can then apply an appropriate technique for preventing the problem or softening its likely consequences. Thus, it usually involves developing hazard lists, as well as procedures for exploring "what if" scenarios to trigger consideration of nonobvious hazards. The sources of problems can be lurking in any artifacts of the development or maintenance process, not just in the code, so a hazard analysis must be broad in its domain of investigation; in other words, hazard analysis is a system issue, not just a code issue.

Similarly, there are many kinds of problems, ranging from incorrect code to unclear consequences of a particular action. A good hazard analysis takes all of them into account.

Although hazard analysis is generally good practice on any project, it is required in some regulated and critical application domains, and it can be invaluable for finding security flaws. It is never too early to be thinking about the sources of hazards; the analysis should begin when you first start thinking about building a new system or when someone proposes a significant upgrade to an existing system. Hazard analysis should continue throughout the system life cycle; you must identify potential hazards that can be introduced during system design, installation, operation, and maintenance.

A variety of techniques support the identification and management of potential hazards. Among the most effective are hazard and operability studies (HAZOP), failure modes and effects analysis (FMEA), and fault tree analysis (FTA). HAZOP is a structured analysis technique originally developed for the process control and chemical plant industries. Over the last few years it has been adapted to discover potential hazards in safety-critical software systems. FMEA is a bottom-up technique applied at the system component level. A team identifies each component's possible faults or fault modes; the team then determines what could trigger the fault and what systemwide effects each fault might have. By keeping system consequences in mind, the team often finds possible system failures that are not made visible by other analytical means. FTA complements FMEA. It is a top-down technique that begins with a postulated hazardous system malfunction. Then, the FTA team works backward to identify the possible precursors to the mishap. By tracing back from a specific hazardous malfunction, the team can locate unexpected contributors to mishaps, and can then look for opportunities to mitigate the risks.

Each of these techniques is clearly useful for finding and preventing security breaches. We decide which technique is most appropriate by understanding how much we know about causes and effects. For example, Table 3-7 suggests that when we know the cause and effect of a given problem, we can strengthen the description of how the system should behave. This clearer picture will help requirements analysts understand how a potential problem is linked to other requirements. It also helps designers understand exactly what the system should do and helps testers know how to test to verify that the system is behaving properly. If we can describe a known effect with unknown cause, we use deductive techniques such as fault tree analysis to help us understand the likely causes of the unwelcome behavior. Conversely, we may know the cause of a problem but not understand all the effects; here, we use inductive techniques such as failure modes and effects analysis to help us trace from cause to all possible effects. For example, suppose we know that a subsystem is unprotected and might lead to a security failure, but we do not know how that failure will affect the rest of the system. We can use FMEA to generate a list of possible effects and then evaluate the tradeoffs between extra protection and possible problems. Finally, to find problems about which we may not yet be aware, we can perform an exploratory analysis such as a hazard and operability study.

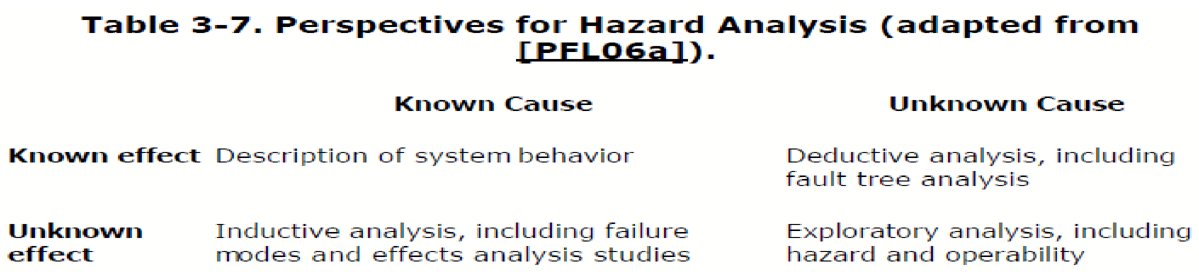


Table 3-7. Perspectives for Hazard Analysis (adapted from [PFL06a]).

	Known Cause	Unknown Cause
Known effect	Description of system behavior	Deductive analysis, including fault tree analysis
Unknown effect	Inductive analysis, including failure modes and effects analysis studies	Exploratory analysis, including hazard and operability

Table 3-7. Perspectives for Hazard Analysis (adapted from
We see in Chapter 8 that hazard analysis is also useful for determining vulnerabilities and mapping them to suitable controls.

Testing

Testing is a process activity that homes in on product quality: making the product failure free or failure tolerant. Each software problem (especially when it relates to security) has the potential not only for making software fail but also for adversely affecting a business or a life.

Thomas Young, head of NASA's investigation of the Mars lander failure, noted that "One of the things we kept in mind during the course of our review is that in the conduct of space missions, you get only one strike, not three. Even if thousands of functions are carried out flawlessly, just one mistake can be catastrophic to a mission" [NAS00]. This same sentiment is true for security: The failure of one control exposes a vulnerability that is not ameliorated by any number of functioning controls. Testers improve software quality by finding as many faults as possible and by writing up their findings carefully so that developers can locate the causes and repair the problems if possible.

Do not ignore a point from Thompson [THO03]: Security testing is hard. Side effects, dependencies, unpredictable users, and flawed implementation bases (languages, compilers, infrastructure) all contribute to this difficulty. But the essential complication with security testing is that we cannot look at just the one behavior the program gets right; we also have to look for the hundreds of ways the program might go wrong.

Testing usually involves several stages. First, each program component is tested on its own, isolated from the other components in the system. Such testing, known as *module testing, component testing,* or *unit testing,* verifies that the component functions properly with the types of input expected from a study of the component's design. Unit testing is done in a controlled environment whenever possible so that the test team can feed a predetermined set of data to the component being tested and observe what output actions and data are produced. In addition, the test team checks the internal data structures, logic, and boundary conditions for the input and output data.

When collections of components have been subjected to unit testing, the next step is ensuring that the interfaces among the components are defined and handled properly. Indeed, process of verifying that the system components work together as described in the system and program design specifications.

Once we are sure that information is passed among components in accordance with the design, we test the system to ensure that it has the desired functionality. A function test evaluates the system to determine whether the functions described by the requirements specification are actually performed by the integrated system. The result is a functioning system. The function test compares the system being built with the functions described in the developers' requirements specification. Then, a performance test compares the system with the remainder of these software and hardware requirements. It is during the function and performance tests that security requirements are examined, and the testers confirm that the system is as secure as it is required to be.

When the performance test is complete, developers are certain that the system functions according to their understanding of the system description. The next step is conferring with the customer to make certain that the system works according to customer expectations. Developers join the customer to perform an acceptance test, in which the system is checked against the customer's requirements description. Upon completion of acceptance testing, the accepted system is installed in the environment in which it will be used. A final installation test is run to make sure that the system still functions as it should. However,

security requirements often state that a system should not do something. As Sidebar 3-7 demonstrates, it is difficult to demonstrate absence rather than presence.
The objective of unit and integration testing is to ensure that the code implemented the design properly; that is, that the programmers have written code to do what the designers intended. System testing has a very different objective: to ensure that the system does what the customer wants it to do. Regression testing, an aspect of system testing, is particularly important for security purposes. After a change is made to enhance the system or fix a problem, regression testing ensures that all remaining functions are still working and that performance has not been degraded by the change.
Each of the types of tests listed here can be performed from two perspectives: black box and clear box (sometimes called white box). Black-box testing treats a system or its components as black boxes; testers cannot "see inside" the system, so they apply particular inputs and verify that they get the expected output. Clear-box testing allows visibility. Here, testers can examine the design and code directly, generating test cases based on the code's actual construction. Thus, clear-box testing knows that component X uses CASE statements and can look for instances in which the input causes control to drop through to an unexpected line. Black-box testing must rely more on the required inputs and outputs because the actual code is not available for scrutiny.
The mix of techniques appropriate for testing a given system depends on the system's size, application domain, amount of risk, and many other factors. But understanding the effectiveness of each technique helps us know what is right for each particular system. For example, Olsen [OLS93] describes the development at Contel IPC of a system containing 184,000 lines of code. He tracked faults discovered during various activities, and found differences:
 17.3 percent of the faults were found during inspections of the system design
 19.1 percent during component design inspection
 15.1 percent during code inspection
 29.4 percent during integration testing
 16.6 percent during system and regression testing
Only 0.1 percent of the faults were revealed after the system was placed in the field. Thus, Olsen's work shows the importance of using different techniques to uncover different kinds of faults during development; it is not enough to rely on a single method for catching all problems.
Who does the testing? From a security standpoint, independent testing is highly desirable; it may prevent a developer from attempting to hide something in a routine or keep a subsystem from controlling the tests that will be applied to it. Thus, independent testing increases the likelihood that a test will expose the effect of a hidden feature.
One type of testing is unique to computer security: penetration testing. In this form of testing, testers specifically try to make software fail. That is, instead of testing to see that software does do what it *is* expected to (as is the goal in the other types of testing we just listed), the testers try to see if the software does what it *is not* supposed to do, which is to fail or, more specifically, fail to enforce security. Because penetration testing usually applies to full systems, not individual applications, we study penetration testing in Chapter 5.
Good Design
We saw earlier in this chapter that modularity, information hiding, and encapsulation are characteristics of good design. Several design-related process activities are particularly helpful in building secure software:
 using a philosophy of *fault tolerance*
 having a consistent *policy* for handling failures
 capturing the *design rationale* and history
 using *design patterns*
We describe each of these activities in turn.
Designers should try to anticipate faults and handle them in ways that minimize disruption and maximize safety and security. Ideally, we want our system to be fault free. But in reality, we must assume that the system will fail, and we make sure that unexpected failure does not bring the system down, destroy data, or destroy life. For example, rather than waiting for the system to fail (called passive fault detection), we might construct the system so that it reacts in an acceptable way to a failure's occurrence. Active fault detection could be practiced by, for instance, adopting a philosophy of mutual suspicion. Instead of

assuming that data passed from other systems or components are correct, we can always check that the data are within bounds and of the right type or format. We can also use redundancy, comparing the results of two or more processes to see that they agree, before we use their result in a task.
If correcting a fault is too risky, inconvenient, or expensive, we can choose instead to practice fault tolerance: isolating the damage caused by the fault and minimizing disruption to users. Although fault tolerance is not always thought of as a security technique, it supports the idea, discussed in Chapter 8, that our security policy allows us to choose to mitigate the effects of a security problem instead of preventing it. For example, rather than install expensive security controls, we may choose to accept the risk that important data may be corrupted. If in fact a security fault destroys important data, we may decide to isolate the damaged data set and automatically revert to a backup data set so that users can continue to perform system functions.
More generally, we can design or code defensively, just as we drive defensively, by constructing a consistent policy for handling failures. Typically, failures include failing to provide a service
 providing the wrong service or data
 corrupting data
We can build into the design a particular way of handling each problem, selecting from one of three ways:
1. *Retrying*: restoring the system to its previous state and performing the service again, using a different strategy
2. *Correcting*: restoring the system to its previous state, correcting some system characteristic, and performing the service again, using the same strategy
3. *Reporting*: restoring the system to its previous state, reporting the problem to an error-handling component, and not providing the service again
This consistency of design helps us check for security vulnerabilities; we look for instances that are different from the standard approach.
Design rationales and history tell us the reasons the system is built one way instead of another. Such information helps us as the system evolves, so we can integrate the design of our security functions without compromising the integrity of the system's overall design. Moreover, the design history enables us to look for patterns, noting what designs work best in which situations. For example, we can reuse patterns that have been successful in preventing buffer overflows, in ensuring data integrity, or in implementing user password checks.
Prediction
Among the many kinds of prediction we do during software development, we try to predict the risks involved in building and using the system. As we see in depth in Chapter 8, we must postulate which unwelcome events might occur and then make plans to avoid them or at least mitigate their effects. Risk prediction and management are especially important for security, where we are always dealing with unwanted events that have negative consequences. Our predictions help us decide which controls to use and how many. For example, if we think the risk of a particular security breach is small, we may not want to invest a large amount of money, time, or effort in installing sophisticated controls. Or we may use the likely risk impact to justify using several controls at once, a technique called "defense in depth."
Static Analysis
Before a system is up and running, we can examine its design and code to locate and repair security flaws. We noted earlier that the peer review process involves this kind of scrutiny. But static analysis is more than peer review, and it is usually performed before peer review. We can use tools and techniques to examine the characteristics of design and code to see if the characteristics warn us of possible faults lurking within. For example, a large number of levels of nesting may indicate that the design or code is hard to read and understand, making it easy for a malicious developer to bury dangerous code deep within the system. To this end, we can examine several aspects of the design and code:
 control flow structure
 data flow structure
 data structure

The control flow is the sequence in which instructions are executed, including iterations and loops. This aspect of design or code can also tell us how often a particular instruction or routine is executed.
Data flow follows the trail of a data item as it is accessed and modified by the system. Many times, transactions applied to data are complex, and we use data flow measures to show us how and when each data item is written, read, and changed.
The data structure is the way in which the data are organized, independent of the system itself. For instance, if the data are arranged as lists, stacks, or queues, the algorithms for manipulating them are likely to be well understood and well defined.
There are many approaches to static analysis, especially because there are so many ways to create and document a design or program. Automated tools are available to generate not only numbers (such as depth of nesting or cyclomatic number) but also graphical depictions of control flow, data relationships, and the number of paths from one line of code to another.
These aids can help us see how a flaw in one part of a system can affect other parts.
Configuration Management
When we develop software, it is important to know who is making which changes to what and when:
 corrective changes: maintaining control of the system's day-to-day functions
 adaptive changes: maintaining control over system modifications
 perfective changes: perfecting existing acceptable functions
 preventive changes: preventing system performance from degrading to unacceptable levels
We want some degree of control over the software changes so that one change does not inadvertently undo the effect of a previous change. And we want to control what is often a proliferation of different versions and releases. For instance, a product might run on several different platforms or in several different environments, necessitating different code to support the same functionality. Configuration management is the process by which we control changes during development and maintenance, and it offers several advantages in security. In particular, configuration management scrutinizes new and changed code to ensure, among other things, that security flaws have not been inserted, intentionally or accidentally.
Four activities are involved in configuration management:
1. configuration identification
2. configuration control and change management
3. configuration auditing
4. status accounting
Configuration identification sets up baselines to which all other code will be compared after changes are made. That is, we build and document an inventory of all components that comprise the system. The inventory includes not only the code you and your colleagues may have created, but also database management systems, third-party software, libraries, test cases, documents, and more. Then, we "freeze" the baseline and carefully control what happens to it. When a change is proposed and made, it is described in terms of how the baseline changes.
Configuration control and configuration management ensure we can coordinate separate, related versions. For example, there may be closely related versions of a system to execute on 16-bit and 32-bit processors. Three ways to control the changes are separate files, deltas, and conditional compilation. If we use separate files, we have different files for each release or version. For example, we might build an encryption system in two configurations: one that uses a short key length, to comply with the law in certain countries, and another that uses a long key. Then, version 1 may be composed of components A_1 through A_k and B_1, while version 2 is A_1 through A_k and B_2, where B_1 and B_2 do key length. That is, the versions are the same except for the separate key processing files.
Alternatively, we can designate a particular version as the main version of a system and then define other versions in terms of what is different. The difference file, called a delta, contains editing commands to describe the ways to transform the main version into the variation.
Finally, we can do conditional compilation, whereby a single code component addresses all versions, relying on the compiler to determine which statements to apply to which versions.

This approach seems appealing for security applications because all the code appears in one place. However, if the variations are very complex, the code may be very difficult to read and understand.
Once a configuration management technique is chosen and applied, the system should be audited regularly. A configuration audit confirms that the baseline is complete and accurate, that changes are recorded, that recorded changes are made, and that the actual software (that is, the software as used in the field) is reflected accurately in the documents. Audits are usually done by independent parties taking one of two approaches: reviewing every entry in the baseline and comparing it with the software in use or sampling from a larger set just to confirm compliance. For systems with strict security constraints, the first approach is preferable, but the second approach may be more practical.
Finally, status accounting records information about the components: where they came from (for instance, purchased, reused, or written from scratch), the current version, the change history, and pending change requests.
All four sets of activities are performed by a configuration and change control board, or CCB. The CCB contains representatives from all organizations with a vested interest in the system, perhaps including customers, users, and developers. The board reviews all proposed changes and approves changes based on need, design integrity, future plans for the software, cost, and more. The developers implementing and testing the change work with a program librarian to control and update relevant documents and components; they also write detailed documentation about the changes and test results.
Configuration management offers two advantages to those of us with security concerns: protecting against unintentional threats and guarding against malicious ones. Both goals are addressed when the configuration management processes protect the integrity of programs and documentation. Because changes occur only after explicit approval from a configuration management authority, all changes are also carefully evaluated for side effects. With configuration management, previous versions of programs are archived, so a developer can retract a faulty change when necessary.
Malicious modification is made quite difficult with a strong review and configuration management process in place. In fact, as presented in Sidebar 3-8, poor configuration control has resulted in at least one system failure; that sidebar also confirms the principle of easiest penetration from Chapter 1. Once a reviewed program is accepted for inclusion in a system, the developer cannot sneak in to make small, subtle changes, such as inserting trapdoors. The developer has access to the running production program only through the CCB, whose members are alert to such security breaches.

Lessons from Mistakes

One of the easiest things we can do to enhance security is learn from our mistakes. As we design and build systems, we can document our decisionsnot only what we decided to do and why, but also what we decided *not* to do and why. Then, after the system is up and running, we can use information about the failures (and how we found and fixed the underlying faults) to give us a better understanding of what leads to vulnerabilities and their exploitation.
From this information, we can build checklists and codify guidelines to help ourselves and others. That is, we do not have to make the same mistake twice, and we can assist other developers in staying away from the mistakes we made. The checklists and guidelines can be invaluable, especially during reviews and inspections, in helping reviewers look for typical or common mistakes that can lead to security flaws. For instance, a checklist can remind a designer or programmer to make sure that the system checks for buffer overflows. Similarly, the guidelines can tell a developer when data require password protection or some other type of restricted access.

Proofs of Program Correctness

A security specialist wants to be certain that a given program computes a particular result, computes it correctly, and does nothing beyond what it is supposed to do. Unfortunately, results in computer science theory (see [PFL85] for a description) indicate that we cannot know with certainty that two programs do exactly the same thing. That is, there can be no general decision procedure which, given any two programs, determines if the two are equivalent. This difficulty results from the "halting problem," which states that there is no general technique to determine whether an arbitrary program will halt when processing an arbitrary input.

In spite of this disappointing general result, a technique called program verification can demonstrate formally the "correctness" of certain specific programs. Program verification involves making initial assertions about the inputs and then checking to see if the desired output is generated. Each program statement is translated into a logical description about its contribution to the logical flow of the program. Finally, the terminal statement of the program is associated with the desired output. By applying a logic analyzer, we can prove that the initial assumptions, through the implications of the program statements, produce the terminal condition. In this way, we can show that a particular program achieves its goal.
presents the case for appropriate use of formal proof techniques. We study an example of program verification in Chapter 5.
Proving program correctness, although desirable and useful, is hindered by several factors. (For more details see [PFL94].)
 Correctness proofs depend on a programmer or logician to translate a program's statements into logical implications. Just as programming is prone to errors, so also is this translation.
 Deriving the correctness proof from the initial assertions and the implications of statements is difficult, and the logical engine to generate proofs runs slowly. The speed of the engine degrades as the size of the program increases, so proofs of correctness are even less appropriate for large programs.
 The current state of program verification is less well developed than code production.
large production systems.
Program verification systems are being improved constantly. Larger programs are being verified in less time than before. As program verification continues to mature, it may become a more important control to ensure the security of programs.

Programming Practice Conclusions

None of the development controls described here can guarantee the security or quality of a system. As Brooks often points out [BRO87], the software development community seeks, but is not likely to find, a "silver bullet": a tool, technique, or method that will dramatically improve the quality of software developed. "There is no single development in either technology or management technique that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity." He bases this conjecture on the fact that software is complex, it must conform to the infinite variety of human requirements, and it is abstract or invisible, leading to its being hard to draw or envision. While software development technologies design tools, process improvement models, development methodologies help the process, software development is inherently complicated and, therefore, prone to errors. This uncertainty does not mean that we should not seek ways to improve; we should. However, we should be realistic and accept that no technique is sure to prevent erroneous software.
We should incorporate in our development practices those techniques that reduce uncertainty and reduce risk. At the same time, we should be skeptical of new technology, making sure each one can be shown to be reliable and effective.
In the early 1970s, Paul Karger and Roger Schell led a team to evaluate the security of the Multics system for the U.S. Air Force. They republished their original report [KAR74] thirty years later with a thoughtful analysis of how the security of Multics compares to the security of current systems [KAR02]. Among their observations were that buffer overflows were almost impossible in Multics because of support from the programming language, and security was easier to ensure because of the simplicity and structure of the Multics design. Karger and Schell argue that we can and have designed and implemented systems with both functionality and security.

Standards of Program Development

No software development organization worth its salt allows its developers to produce code at any time in any manner. The good software development practices described earlier in this chapter have all been validated by many years of practice. Although none is Brooks's mythical "silver bullet" that guarantees program correctness, quality, or security, they all add demonstrably to the strength of programs. Thus, organizations prudently establish standards for how programs are developed. Even advocates of agile methods, which give developers an unusual degree of flexibility and autonomy, encourage goal-directed behavior based on past experience and past success. Standards and guidelines can capture wisdom from previous projects and increase the likelihood that the resulting system will be correct. In addition, we want to ensure that the systems we build are reasonably easy to maintain and are compatible with the systems with which they interact.

We can exercise some degree of administrative control over software development by considering several kind of standards or guidelines:

 standards of *design,* including using specified design tools, languages, or methodologies, using design diversity, and devising strategies for error handling and fault tolerance

 standards of *documentation, language,* and *coding style*, including layout of code on the page, choices of names of variables, and use of recognized program structures

 standards of *programming,* including mandatory peer reviews, periodic code audits for correctness, and compliance with standards

 standards of *testing,* such as using program verification techniques, archiving test results for future reference, using independent testers, evaluating test thoroughness, and encouraging test diversity

 standards of *configuration management*, to control access to and changes of stable or completed program units

Standardization improves the conditions under which all developers work by establishing a common framework so that no one developer is indispensable. It also allows carryover from one project to another; lessons learned on previous projects become available for use by all on the next project. Standards also assist in maintenance, since the maintenance team can find required information in a well-organized program. However, we must take care that the standards do not unnecessarily constrain the developers.

Firms concerned about security and committed to following software development standards often perform security audits. In a security audit, an independent security evaluation team arrives unannounced to check each project's compliance with standards and guidelines. The team reviews requirements, designs, documentation, test data and plans, and code. Knowing that documents are routinely scrutinized, a developer is unlikely to put suspicious code in a component in the first place.

Process Standards

You have two friends. Sonya is extremely well organized, she keeps lists of things to do, she always knows where to find a tool or who has a particular book, and everything is done before it is needed. Dorrie, on the other hand, is a mess. She can never find her algebra book, her desk has so many piles of papers you cannot see the top, and she seems to deal with everything as a crisis because she tends to ignore things until the last minute. Who would you choose to organize and run a major social function, a new product launch, or a multiple-author paper? Most people would pick Sonya, concluding that her organization skills are crucial. There is no guarantee that Sonya would do a better job than Dorrie, but you might assume the chances are better with Sonya.

We know that software development is difficult in part because it has inherently human aspects that are very difficult to judge in advance. Still, we may conclude that software built in an orderly manner has a better chance of being good or secure.

The Software Engineering Institute developed the Capability Maturity Model (CMM) to assess organizations, not products (see [HUM88] and [PAU93]). The International Standards Organization (ISO) developed process standard ISO 9001 [ISO94], which is somewhat similar to the CMM (see [PAU95]). Finally the U.S. National Security Agency (NSA) developed the System Security Engineering CMM (SSE CMM, see [NSA95a]). All of these are process models, in that they examine *how* an organization does something, not *what* it does. Thus, they judge consistency, and many people extend consistency to quality. For views on that subject, see Bollinger and McGowan [BOL91] and Curtis [CUR87]. El Emam [ELE95] has also looked at the reliability of measuring a process.

Now go back to the original descriptions of Sonya and Dorrie. Who would make the better developer? That question is tricky because many of us have friends like Dorrie who are fabulous programmers, but we may also know great programmers who resemble Sonya. And some successful teams have both. Order, structure, and consistency *may* lead to good software projects, but it is not sure to be the only way to go.

Program Controls in General

This section has explored how to control for faults during the program development process.

Some controls apply to how a program is developed, and others establish restrictions on the

program's use. The best is a combination, the classic layered defense.
Is one control essential? Can one control be skipped if another is used? Although these are valid questions, the security community does not have answers. Software development is both an art and a science. As a creative activity, it is subject not only to the variety of human minds, but also to the fallibility of humans. We cannot rigidly control the process and get the same results time after time, as we can with a machine.
But creative humans can learn from their mistakes and shape their creations to account for fundamental principles. Just as a great painter will achieve harmony and balance in a painting, a good software developer who truly understands security will incorporate security into all phases of development. Thus, even if you never become a security professional, this exposure to the needs and shortcomings of security will influence many of your future actions.
Unfortunately, many developers do not have the opportunity to become sensitive to security issues, which probably accounts for many of the unintentional security faults in today's programs.

Unit 2 : Operating System Security:

Protected objects and methods of protection

4.1. Protected Objects and Methods of Protection

We begin by reviewing the history of protection in operating systems. This background helps us understand what kinds of things operating systems can protect and what methods are available for protecting them. (Readers who already have a good understanding of operating system capabilities may want to jump to Section 4.3.)

A Bit of History

Once upon a time, there were no operating systems: Users entered their programs directly into the machine in binary by means of switches. In many cases, program entry was done by physical manipulation of a toggle switch; in other cases, the entry was performed with a more complex electronic method, by means of an input device such as a keyboard. Because each user had exclusive use of the computing system, users were required to schedule blocks of time for running the machine. These users were responsible for loading their own libraries of support routinesassemblers, compilers, shared subprogramsand "cleaning up" after use by removing any sensitive code or data.
The first operating systems were simple utilities, called executives, designed to assist individual programmers and to smooth the transition from one user to another. The early executives provided linkers and loaders for relocation, easy access to compilers and assemblers, and automatic loading of subprograms from libraries. The executives handled the tedious aspects of programmer support, focusing on a single programmer during execution.
Operating systems took on a much broader role (and a different name) as the notion of multiprogramming was implemented. Realizing that two users could interleave access to the resources of a single computing system, researchers developed concepts such as scheduling, sharing, and parallel use. Multiprogrammed operating systems, also known as monitors, oversaw each program's execution. Monitors took an active role, whereas executives were passive. That is, an executive stayed in the background, waiting to be called into service by a requesting user. But a monitor actively asserted control of the computing system and gave resources to the user only when the request was consistent with general good use of the system. Similarly, the executive waited for a request and provided service on demand; the monitor maintained control over all resources, permitting or denying all computing and loaning resources to users as they needed them.
Multiprogramming brought another important change to computing. When a single person was using a system, the only force to be protected against was the user himself or herself. A user making an error may have felt foolish, but one user could not adversely affect the computation of any other user. However, multiple users introduced more complexity and risk.

User A might rightly be angry if User B's programs or data had a negative effect on A's program's execution. Thus, protecting one user's programs and data from other users' programs became an important issue in multiprogrammed operating systems.

Protected Objects

In fact, the rise of multiprogramming meant that several aspects of a computing system required protection:

- memory
- sharable I/O devices, such as disks
- serially reusable I/O devices, such as printers and tape drives
- sharable programs and subprocedures
- networks
- sharable data

As it assumed responsibility for controlled sharing, the operating system had to protect these objects. In the following sections, we look at some of the mechanisms with which operating systems have enforced these objects' protection. Many operating system protection mechanisms have been supported by hardware. But, as noted in Sidebar 4-1, that approach is not always possible.

Security Methods of Operating Systems

The basis of protection is separation: keeping one user's objects separate from other users. Rushby and Randell [RUS83] note that separation in an operating system can occur in several ways:

- *physical separation*, in which different processes use different physical objects, such as separate printers for output requiring different levels of security
- *temporal separation*, in which processes having different security requirements are executed at different times
- *logical separation*, in which users operate under the illusion that no other processes exist, as when an operating system constrains a program's accesses so that the program cannot access objects outside its permitted domain
- *cryptographic separation*, in which processes conceal their data and computations in such a way that they are unintelligible to outside processes

Of course, combinations of two or more of these forms of separation are also possible. The categories of separation are listed roughly in increasing order of complexity to implement, and, for the first three, in decreasing order of the security provided. However, the first two approaches are very stringent and can lead to poor resource utilization. Therefore, we would like to shift the burden of protection to the operating system to allow concurrent execution of processes having different security needs.

But separation is only half the answer. We want to separate users and their objects, but we also want to be able to provide sharing for some of those objects. For example, two users with different security levels may want to invoke the same search algorithm or function call. We would like the users to be able to share the algorithms and functions without compromising their individual security needs. An operating system can support separation and sharing in several ways, offering protection at any of several levels.

- *Do not protect.* Operating systems with no protection are appropriate when sensitive procedures are being run at separate times.
- *Isolate.* When an operating system provides isolation, different processes running concurrently are unaware of the presence of each other. Each process has its own address space, files, and other objects. The operating system must confine each process somehow so that the objects of the other processes are completely concealed.
- *Share all or share nothing.* With this form of protection, the owner of an object declares it to be public or private. A public object is available to all users, whereas a private object is available only to its owner.
- *Share via access limitation.* With protection by access limitation, the operating system checks the allowability of each user's potential access to an object. That is, access control is implemented for a specific user and a specific object. Lists of acceptable actions guide the operating system in determining whether a particular user should have access to a particular object. In some sense, the operating system acts as a guard between users and objects, ensuring that only authorized accesses occur.

 Share by capabilities. An extension of limited access sharing, this form of protection allows dynamic creation of sharing rights for objects. The degree of sharing can depend on the owner or the subject, on the context of the computation, or on the object itself.
 Limit use of an object. This form of protection limits not just the access to an object but the use made of that object after it has been accessed. For example, a user may be allowed to view a sensitive document, but not to print a copy of it. More powerfully, a user may be allowed access to data in a database to derive statistical summaries (such as average salary at a particular grade level), but not to determine specific data values (salaries of individuals).
Again, these modes of sharing are arranged in increasing order of difficulty to implement, but also in increasing order of fineness of protection they provide. A given operating system may provide different levels of protection for different objects, users, or situations.
When we think about data, we realize that access can be controlled at various levels: the bit, the byte, the element or word, the field, the record, the file, or the volume. Thus, the granularity of control concerns us. The larger the level of object controlled, the easier it is to implement access control. However, sometimes the operating system must allow access to more than the user needs. For example, with large objects, a user needing access only to part of an object (such as a single record in a file) must be given access to the entire object (the whole file).
Let us examine in more detail several different kinds of objects and their specific kinds of protection.

Memory address protection

4.2. Memory and Address Protection

The most obvious problem of multiprogramming is preventing one program from affecting the data and programs in the memory space of other users. Fortunately, protection can be built into the hardware mechanisms that control efficient use of memory, so solid protection can be provided at essentially no additional cost.

Fence

The simplest form of memory protection was introduced in single-user operating systems to prevent a faulty user program from destroying part of the resident portion of the operating system. As its name implies, a fence is a method to confine users to one side of a boundary. In one implementation, the fence was a predefined memory address, enabling the operating system to reside on one side and the user to stay on the other. An example of this situation is shown in Figure 4-1. Unfortunately, this kind of implementation was very restrictive because a predefined amount of space was always reserved for the operating system, whether it was needed or not. If less than the predefined space was required, the excess space was wasted.
Conversely, if the operating system needed more space, it could not grow beyond the fence boundary.

Figure 4-1. Fixed Fence.

Another implementation used a hardware register, often called a fence register, containing the address of the end of the operating system. In contrast to a fixed fence, in this scheme the location of the fence could be changed. Each time a user program generated an address for data modification, the address was automatically compared with the fence address. If the address was greater than the fence address (that is, in the user area), the instruction was executed; if it was less than the fence address (that is, in the operating system area), an error condition was raised. The use of fence registers is shown in Figure 4-2.

Figure 4-2. Variable Fence Register.

A fence register protects only in one direction. In other words, an operating system can be protected from a single user, but the fence cannot protect one user from another user. Similarly, a user cannot identify certain areas of the program as inviolable (such as the code of the program itself or a read-only data area).

Relocation

If the operating system can be assumed to be of a fixed size, programmers can write their code assuming that the program begins at a constant address. This feature of the operating system makes it easy to determine the address of any object in the program. However, it also makes it essentially impossible to change the starting address if, for example, a new version of the operating system is larger or smaller than the old. If the size of the operating system is allowed to change, then programs must be written in a way that does not depend on placement at a specific location in memory.

Relocation is the process of taking a program written as if it began at address 0 and changing all addresses to reflect the actual address at which the program is located in memory. In many instances, this effort merely entails adding a constant relocation factor to each address of the program. That is, the relocation factor is the starting address of the memory assigned for the program.

Conveniently, the fence register can be used in this situation to provide an important extra benefit: The fence register can be a hardware relocation device. The contents of the fence register are added to each program address. This action both relocates the address and guarantees that no one can access a location lower than the fence address. (Addresses are treated as unsigned integers, so adding the value in the fence register to any number is guaranteed to produce a result at or above the fence address.) Special instructions can be added for the few times when a program legitimately intends to access a location of the operating system.

Base/Bounds Registers

A major advantage of an operating system with fence registers is the ability to relocate; this characteristic is especially important in a multiuser environment. With two or more users, none can know in advance where a program will be loaded for execution. The relocation register solves the problem by providing a base or starting address. All addresses inside a program are offsets from that base address. A variable fence register is generally known as a base register.

Fence registers provide a lower bound (a starting address) but not an upper one. An upper bound can be useful in knowing how much space is allotted and in checking for overflows into "forbidden" areas. To overcome this difficulty, a second register is often added, as shown in Figure 4-3. The second register, called a bounds register, is an upper address limit, in the same way that a base or fence register is a lower address limit. Each program address is forced to be above the base address because the contents of the base register are added to the address; each address is also checked to ensure that it is below the bounds address. In this way, a program's addresses are neatly confined to the space between the base and the bounds registers.

Figure 4-3. Pair of Base/Bounds Registers.

This technique protects a program's addresses from modification by another user. When execution changes from one user's program to another's, the operating system must change the contents of the base and bounds registers to reflect the true address space for that user.

This change is part of the general preparation, called a context switch, that the operating system must perform when transferring control from one user to another. With a pair of base/bounds registers, a user is perfectly protected from outside users, or, more correctly, outside users are protected from errors in any other user's program. Erroneous addresses *inside* a user's address space can still affect that program because the base/bounds checking guarantees only that each address is inside the user's address space. For example, a user error might occur when a subscript is out of range or an undefined variable generates an address reference within the user's space but, unfortunately, inside the executable instructions of the user's program. In this manner, a user can accidentally store data on top of instructions. Such an error can let a user inadvertently destroy a program, but (fortunately) only the user's own program.

We can solve this overwriting problem by using another pair of base/bounds registers, one for the instructions (code) of the program and a second for the data space. Then, only instruction fetches (instructions to be executed) are relocated and checked with the first register pair, and only data accesses (operands of instructions) are relocated and checked with the second register pair. The use of two pairs of base/bounds registers is shown in Figure 4-4.

Although two pairs of registers do not prevent all program errors, they limit the effect of data-manipulating instructions to the data space. The pairs of registers offer another more important advantage: the ability to split a program into two pieces that can be relocated separately.

Figure 4-4. Two Pairs of Base/Bounds Registers.

These two features seem to call for the use of three or more pairs of registers: one for code, one for read-only data, and one for modifiable data values. Although in theory this concept can be extended, two pairs of registers are the limit for practical computer design. For each additional pair of registers (beyond two), something in the machine code of each instruction must indicate which relocation pair is to be used to address the instruction's operands. That is, with more than two pairs, each instruction specifies one of two or more data spaces. But with only two pairs, the decision can be automatic: instructions with one pair, data with the other.

Tagged Architecture

Another problem with using base/bounds registers for protection or relocation is their contiguous nature. Each pair of registers confines accesses to a consecutive range of addresses. A compiler or loader can easily rearrange a program so that all code sections are adjacent and all data sections are adjacent.

However, in some cases you may want to protect *some* data values but not *all*. For example, a personnel record may require protecting the field for salary but not office location and phone number. Moreover, a programmer 1may want to ensure the integrity of certain data values by allowing them to be written when the program is initialized but prohibiting the program from modifying them later. This scheme protects against errors in the programmer's own code. A programmer may also want to invoke a shared subprogram from a common library. We can address some of these issues by using good design, both in the operating system and in the other programs being run. Recall that in Chapter 3 we studied good design characteristics such as information hiding and modularity in program design. These characteristics dictate that one program module must share with another module only the *minimum* amount of data necessary for both of them to do their work.

Additional, operating-system-specific design features can help, too. Base/bounds registers create an all-or-nothing situation for sharing: Either a program makes all its data available to be accessed and modified or it prohibits access to all. Even if there were a third set of registers for shared data, all data would need to be located together. A procedure could not effectively share data items *A, B,* and *C* with one module, *A, C,* and *D* with a second, and *A, B,* and *D* with a third. The only way to accomplish the kind of sharing we want would be to move each appropriate set of data values to some contiguous space. However, this solution would not be acceptable if the data items were large records, arrays, or structures.

An alternative is tagged architecture, in which every word of machine memory has one or more extra bits to identify the access rights to that word. These access bits can be set only by privileged (operating system) instructions. The bits are tested every time an instruction accesses that location.

For example, as shown in Figure 4-5, one memory location may be protected as execute-only (for example, the object code of instructions), whereas another is protected for fetch-only (for example, read) data access, and another accessible for modification (for example, write).

In this way, two adjacent locations can have different access rights. Furthermore, with a few extra tag bits, different classes of data (numeric, character, address or pointer, and undefined) can be separated, and data fields can be protected for privileged (operating system) access only.

Figure 4-5. Example of Tagged Architecture

This protection technique has been used on a few systems, although the number of tag bits has been rather small. The Burroughs B6500-7500 system used three tag bits to separate data words (three types), descriptors (pointers), and control words (stack pointers and addressing control words). The IBM System/38 used a tag to control both integrity and access.

A variation used one tag that applied to a group of consecutive locations, such as 128 or 256 bytes. With one tag for a block of addresses, the added cost for implementing tags was not as high as with one tag per location. The Intel I960 extended architecture processor used a tagged architecture with a bit on each memory word that marked the word as a "capability," not as an ordinary location for data or instructions. A capability controlled access to a variable-sized memory block or segment. This large number of possible tag values supported memory segments that ranged in size from 64 to 4 billion bytes, with a potential 2^{256} different protection domains.

Compatibility of code presented a problem with the acceptance of a tagged architecture. A tagged architecture may not be as useful as more modern approaches, as we see shortly. Some of the major computer vendors are still working with operating systems that were designed and implemented many years ago for architectures of that era. Indeed, most manufacturers are locked into a more conventional memory architecture because of the wide availability of components and a desire to maintain compatibility among operating systems and machine families. A tagged architecture would require fundamental changes to substantially all the operating system code, a requirement that can be prohibitively expensive. But as the price of memory continues to fall, the implementation of a tagged architecture becomes more feasible.

Segmentation

We present two more approaches to protection, each of which can be implemented on top of a conventional machine structure, suggesting a better chance of acceptance. Although these approaches are ancient by computing's standardsthey were designed between 1965 and 1975they have been implemented on many machines since then. Furthermore, they offer important advantages in addressing, with memory protection being a delightful bonus.

The first of these two approaches, segmentation, involves the simple notion of dividing a program into separate pieces. Each piece has a logical unity, exhibiting a relationship among all of its code or data values. For example, a segment may be the code of a single procedure, the data of an array, or the collection of all local data values used by a particular module. Segmentation was developed as a feasible means to produce the effect of the equivalent of an unbounded number of base/bounds registers. In other words, segmentation allows a program to be divided into many pieces having different access rights. Each segment has a unique name. A code or data item within a segment is addressed as the pair <*name*, *offset*>, where *name* is the name of the segment containing the data item and *offset* is its location within the segment (that is, its distance from the start of the segment).

Logically, the programmer pictures a program as a long collection of segments. Segments can be separately relocated, allowing any segment to be placed in any available memory locations.

The relationship between a logical segment and its true memory position is shown in Figure 4-6.

Figure 4-6. Logical and Physical Representation of Segments.

The operating system must maintain a table of segment names and their true addresses in memory. When a program generates an address of the form <*name*, *offset*>, the operating system looks up *name* in the segment directory and determines its real beginning memory address. To that address the operating system adds *offset*, giving the true memory address of the code or data item. This translation is shown in Figure 4-7. For efficiency there is usually one operating system segment address table for each process in execution. Two processes that need to share access to a single segment would have the same segment name and address in their segment tables.

Figure 4-7. Translation of Segment Address.

Thus, a user's program does not know what true memory addresses it uses. It has no way and no needto determine the actual address associated with a particular <*name*, *offset*>. The <*name*, *offset*> pair is adequate to access any data or instruction to which a program should have access.

This hiding of addresses has three advantages for the operating system.

 The operating system can place any segment at any location or move any segment to any location, even after the program begins to execute. Because it translates all address references by a segment address table, the operating system needs only update the address in that one table when a segment is moved.

 A segment can be removed from main memory (and stored on an auxiliary device) if it is not being used currently.

 Every address reference passes through the operating system, so there is an opportunity to check each one for protection.

Because of this last characteristic, a process can access a segment only if that segment appears in that process's segment translation table. The operating system controls which programs have entries for a particular segment in their segment address tables. This control provides strong protection of segments from access by unpermitted processes. For example, program *A* might have access to segments *BLUE* and *GREEN* of user *X* but not to other segments of that user or of any other user. In a straightforward way we can allow a user to have different protection classes for different segments of a program. For example, one segment might be read-only data, a second might be execute-only code, and a third might be writeable data. In a situation like this one, segmentation can approximate the goal of separate protection of different pieces of a program, as outlined in the previous section on tagged architecture.

Segmentation offers these security benefits:

- Each address reference is checked for protection.
- Many different classes of data items can be assigned different levels of protection.
- Two or more users can share access to a segment, with potentially different access rights.
- A user cannot generate an address or access to an unpermitted segment.

One protection difficulty inherent in segmentation concerns segment size. Each segment has a particular size. However, a program can generate a reference to a valid segment *name*, but with an *offset* beyond the end of the segment. For example, reference <*A*,9999> looks perfectly valid, but in reality segment *A* may be only 200 bytes long. If left unplugged, this security hole could allow a program to access any memory address beyond the end of a segment just by using large values of *offset* in an address.

This problem cannot be stopped during compilation or even when a program is loaded, because effective use of segments requires that they be allowed to grow in size during execution. For example, a segment might contain a dynamic data structure such as a stack. Therefore, secure implementation of segmentation requires checking a generated address to verify that it is not beyond the current end of the segment referenced. Although this checking results in extra expense (in terms of time and resources), segmentation systems must perform this check; the segmentation process must maintain the current segment length in the translation table and compare every address generated.

Thus, we need to balance protection with efficiency, finding ways to keep segmentation as efficient as possible. However, efficient implementation of segmentation presents two problems: Segment names are inconvenient to encode in instructions, and the operating system's lookup of the name in a table can be slow. To overcome these difficulties, segment names are often converted to numbers by the compiler when a program is translated; the compiler also appends a linkage table matching numbers to true segment names.

Unfortunately, this scheme presents an implementation difficulty when two procedures need to share the same segment because the assigned segment numbers of data accessed by that segment must be the same.

Paging

One alternative to segmentation is paging. The program is divided into equal-sized pieces called pages, and memory is divided into equal-sized units called page frames. (For implementation reasons, the page size is usually chosen to be a power of two between 512 and 4096 bytes.) As with segmentation, each address in a paging scheme is a two-part object, this time consisting of <*page*, *offset*>.

Each address is again translated by a process similar to that of segmentation: The operating system maintains a table of user page numbers and their true addresses in memory. The *page* portion of every <*page*, *offset*> reference is converted to a page frame address by a table lookup; the *offset* portion is added to the page frame address to produce the real memory address of the object referred to as <*page*, *offset*>. This process is illustrated in Figure 4-8.

Figure 4-8. Page Address Translation.

Unlike segmentation, all pages in the paging approach are of the same fixed size, so fragmentation is not a problem. Each page can fit in any available page in memory, and thus there is no problem of addressing beyond the end of a page. The binary form of a <*page*,
offset> address is designed so that the *offset* values fill a range of bits in the address.

Therefore, an *offset* beyond the end of a particular page results in a carry into the *page* portion of the address, which changes the address.
To see how this idea works, consider a page size of 1024 bytes ($1024 = 2_{10}$), where 10 bits are allocated for the *offset* portion of each address. A program cannot generate an *offset* value larger than 1023 in 10 bits. Moving to the next location after <*x*,1023> causes a carry into the *page* portion, thereby moving translation to the next page. During the translation, the paging process checks to verify that a <*page*, *offset*>reference does not exceed the maximum number of pages the process has defined.
With a segmentation approach, a programmer must be conscious of segments. However, a programmer is oblivious to page boundaries when using a paging-based operating system. Moreover, with paging there is no logical unity to a page; a page is simply the next 2_n bytes of the program. Thus, a change to a program, such as the addition of one instruction, pushes all subsequent instructions to lower addresses and moves a few bytes from the end of each page to the start of the next. This shift is not something about which the programmer need be concerned because the entire mechanism of paging and address translation is hidden from the programmer.
However, when we consider protection, this shift is a serious problem. Because segments are logical units, we can associate different segments with individual protection rights, such as read-only or execute-only. The shifting can be handled efficiently during address translation.
But with paging there is no necessary unity to the items on a page, so there is no way to establish that all values on a page should be protected at the same level, such as read-only or execute-only.

Combined Paging with Segmentation

We have seen how paging offers implementation efficiency, while segmentation offers logical protection characteristics. Since each approach has drawbacks as well as desirable features, the two approaches have been combined.
The IBM 390 family of mainframe systems used a form of paged segmentation. Similarly, the Multics operating system (implemented on a GE-645 machine) applied paging on top of segmentation. In both cases, the programmer could divide a program into logical segments. Each segment was then broken into fixed-size pages. In Multics, the segment *name* portion of an address was an 18-bit number with a 16-bit *offset*. The addresses were then broken into 1024-byte pages. The translation process is shown in Figure 4-9. This approach retained the logical unity of a segment and permitted differentiated protection for the segments, but it added an additional layer of translation for each address. Additional hardware improved the efficiency of the implementation.

Figure 4-9. Paged Segmentation.

Control of access to general objects

4.3. Control of Access to General Objects

Protecting memory is a specific case of the more general problem of protecting *objects*. As multiprogramming has developed, the numbers and kinds of objects shared have also increased. Here are some examples of the kinds of objects for which protection is desirable:
 memory
 a file or data set on an auxiliary storage device
 an executing program in memory
 a directory of files
 a hardware device
 a data structure, such as a stack
 a table of the operating system
 instructions, especially privileged instructions
 passwords and the user authentication mechanism
 the protection mechanism itself
The memory protection mechanism can be fairly simple because every memory access is guaranteed to go through certain points in the hardware. With more general objects, the number of points of access may be larger, a central authority through which all accesses pass may be lacking, and the kind of access may not simply be limited to read, write, or execute.

Furthermore, all accesses to memory occur through a program, so we can refer to the program or the programmer as the accessing agent. In this book, we use terms like *the user* or *the subject* in describing an access to a general object. This user or subject could be a person who uses a computing system, a programmer, a program, another object, or something else that seeks to use an object.
There are several complementary goals in protecting objects.
 Check every access. We may want to revoke a user's privilege to access an object. If we have previously authorized the user to access the object, we do not necessarily intend that the user should retain indefinite access to the object. In fact, in some situations, we may want to prevent further access immediately after we revoke authorization. For this reason, every access by a user to an object should be checked.
 Enforce least privilege. The principle of least privilege states that a subject should have access to the smallest number of objects necessary to perform some task. Even if extra information would be useless or harmless if the subject were to have access, the subject should not have that additional access. For example, a program should not have access to the absolute memory address to which a page number reference translates, even though the program could not use that address in any effective way.
Not allowing access to unnecessary objects guards against security weaknesses if a part of the protection mechanism should fail.
 Verify acceptable usage. Ability to access is a yes-or-no decision. But it is equally important to check that the activity to be performed on an object is appropriate. For example, a data structure such as a stack has certain acceptable operations, including *push*, *pop*, *clear*, and so on. We may want not only to control who or what has access to a stack but also to be assured that the accesses performed are legitimate stack accesses.
In the next section we consider protection mechanisms appropriate for general objects of unspecified types, such as the kinds of objects listed above. To make the explanations easier to understand, we sometimes use an example of a specific object, such as a file. Note, however, that a general mechanism can be used to protect any of the types of objects listed.

Directory

One simple way to protect an object is to use a mechanism that works like a file directory. Imagine we are trying to protect files (the set of objects) from users of a computing system (the set of subjects). Every file has a unique owner who possesses "control" access rights (including the rights to declare who has what access) and to revoke access to any person at any time. Each user has a file directory, which lists all the files to which that user has access.
Clearly, no user can be allowed to write in the file directory because that would be a way to forge access to a file. Therefore, the operating system must maintain all file directories, under commands from the owners of files. The obvious rights to files are the common *read*, *write*, and *execute* familiar on many shared systems. Furthermore, another right, *owner*, is possessed by the owner, permitting that user to grant and revoke access rights. Figure 4-10 shows an example of a file directory.

Figure 4-10. Directory Access.

This approach is easy to implement because it uses one list per user, naming all the objects that user is allowed to access. However, several difficulties can arise. First, the list becomes too large if many shared objects, such as libraries of subprograms or a common table of users, are accessible to all users. The directory of each user must have one entry for each such shared object, even if the user has no intention of accessing the object. Deletion must be reflected in all directories. (See Sidebar 4-2 for a different issue concerning deletion of objects.)
A second difficulty is revocation of access. If owner *A* has passed to user *B* the right to read file *F*, an entry for *F* is made in the directory for *B*. This granting of access implies a level of *trust* between *A* and *B*. If *A* later questions that trust, *A* may want to revoke the access right of *B*. The operating system can respond easily to the single request to delete the right of *B* to access *F* because that action involves deleting one entry from a specific directory. But if *A* wants to remove the rights of *everyone* to access *F*, the operating system must search each individual directory for the entry *F*, an activity that can be time consuming on a large system.

For example, large timesharing systems or networks of smaller systems can easily have 5,000 to 10,000 active accounts. Moreover, *B* may have passed the access right for *F* to another user, so *A* may not know that *F*s access exists and should be revoked. This problem is particularly serious in a network.

A third difficulty involves pseudonyms. Owners *A* and *B* may have two different files named *F*, and they may both want to allow access by *S*. Clearly, the directory for *S* cannot contain two entries under the same name for different files. Therefore, *S* has to be able to uniquely identify the *F* for *A* (or *B*). One approach is to include the original owner's designation as if it were part of the file name, with a notation such as *A*:*F* (or *B*:*F*).

Suppose, however, that *S* has trouble remembering file contents from the name *F*. Another approach is to allow *S* to name *F* with any name unique to the directory of *S*. Then, *F* from *A* could be called *Q* to *S*. As shown in Figure 4-11, *S* may have forgotten that *Q* is *F* from *A*, and so *S* requests access again from *A* for *F*. But by now *A* may have more trust in *S*, so *A* transfers *F* with greater rights than before. This action opens up the possibility that one subject, *S*, may have two distinct sets of access rights to *F*, one under the name *Q* and one under the name *F*. In this way, allowing pseudonyms leads to multiple permissions that are not necessarily consistent. Thus, the directory approach is probably too simple for most object protection situations.

Figure 4-11. Alternative Access Paths.

Access Control List

An alternative representation is the access control list. There is one such list for each object, and the list shows all subjects who should have access to the object and what their access is. This approach differs from the directory list because there is one access control list per *object*; a directory is created for each *subject*. Although this difference seems small, there are some significant advantages.

To see how, consider subjects *A* and *S*, both of whom have access to object *F*. The operating system will maintain just one access list for *F*, showing the access rights for *A* and *S*, as shown in Figure 4-12. The access control list can include general default entries for any users.

In this way, specific users can have explicit rights, and all other users can have a default set of rights. With this organization, a public file or program can be shared by all possible users of the system without the need for an entry for the object in the individual directory of each user.

Figure 4-12. Access Control List.

The Multics operating system used a form of access control list in which each user belonged to three protection classes: a *user*, a *group*, and a *compartment*. The user designation identified a specific subject, and the group designation brought together subjects who had a common interest, such as coworkers on a project. The compartment confined an untrusted object; a program executing in one compartment could not access objects in another compartment without specific permission. The compartment was also a way to collect objects that were related, such as all files for a single project.

To see how this type of protection might work, suppose every user who initiates access to the system identifies a group and a compartment with which to work. If Adams logs in as user *Adams* in group *Decl* and compartment *Art2*, only objects having *Adams-Decl-Art2* in the access control list are accessible in the session.

By itself, this kind of mechanism would be too restrictive to be usable. Adams cannot create general files to be used in any session. Worse yet, shared objects would have not only to list Adams as a legitimate subject but also to list Adams under all acceptable groups and all acceptable compartments for each group.

The solution is the use of wild cards, meaning placeholders that designate "any user" (or "any group" or "any compartment"). An access control list might specify access by *Adams-Decl-Art1*, giving specific rights to Adams if working in group *Decl* on compartment *Art1* . The list might also specify *Adams-*-Art1*, meaning that Adams can access the object from any group in compartment *Art1*. Likewise, a notation of **-Decl-** would mean "any user in group *Decl* in any compartment." Different placements of the wildcard notation * have the obvious interpretations.

The access control list can be maintained in sorted order, with * sorted as coming after all specific names. For example, *Adams-Decl-** would come after all specific compartment

designations for Adams. The search for access permission continues just until the first match.
In the protocol, all explicit designations are checked before wild cards in any position, so a specific access right would take precedence over a wildcard right. The last entry on an access list could be **-*-**, specifying rights allowable to any user not explicitly on the access list. By using this wildcard device, a shared public object can have a very short access list, explicitly naming the few subjects that should have access rights different from the default.

Access Control Matrix

We can think of the directory as a listing of objects accessible by a single subject, and the access list as a table identifying subjects that can access a single object. The data in these two representations are equivalent, the distinction being the ease of use in given situations.
As an alternative, we can use an access control matrix, a table in which each row represents a subject, each column represents an object, and each entry is the set of access rights for that subject to that object. An example representation of an access control matrix is shown in Table 4-1. In general, the access control matrix is sparse (meaning that most cells are empty): Most subjects do not have access rights to most objects. The access matrix can be represented as a list of triples, having the form <*subject*, *object*, *rights*>. Searching a large number of these triples is inefficient enough that this implementation is seldom used.

Table 4-1. Access Control Matrix.

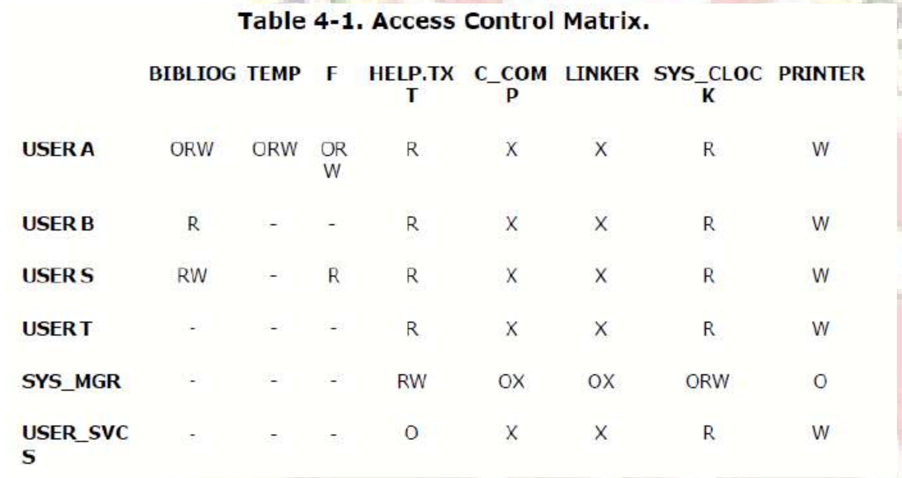


Table 4-1. Access Control Matrix.

	BIBLIOG	TEMP	F	HELP.TXT	C_COMP	LINKER	SYS_CLOCK	PRINTER
USER A	ORW	ORW	ORW	R	X	X	R	W
USER B	R	-	-	R	X	X	R	W
USER S	RW	-	R	R	X	X	R	W
USER T	-	-	-	R	X	X	R	W
SYS_MGR	-	-	-	RW	OX	OX	ORW	O
USER_SVCS	-	-	-	O	X	X	R	W

Capability

So far, we have examined protection schemes in which the operating system must keep track of all the protection objects and rights. But other approaches put some of the burden on the user. For example, a user may be required to have a ticket or pass that enables access, much like a ticket or identification card that cannot be duplicated. More formally, we say that a capability is an unforgeable token that gives the possessor certain rights to an object. The Multics [SAL74], CAL [LAM76], and Hydra [WUL74] systems used capabilities for access control. In theory, a subject can create new objects and can specify the operations allowed on those objects. For example, users can create objects, such as files, data segments, or subprocesses, and can also specify the acceptable kinds of operations, such as *read*, *write*, and *execute*. But a user can also create completely new objects, such as new data structures, and can define types of accesses previously unknown to the system.
A capability is a ticket giving permission to a subject to have a certain type of access to an object. For the capability to offer solid protection, the ticket must be unforgeable. One way to make it unforgeable is to not give the ticket directly to the user. Instead, the operating system holds all tickets on behalf of the users. The operating system returns to the user a pointer to an operating system data structure, which also links to the user. A capability can be created only by a specific request from a user to the operating system. Each capability also identifies the allowable accesses.
Alternatively, capabilities can be encrypted under a key available only to the access control mechanism. If the encrypted capability contains the identity of its rightful owner, user *A* cannot copy the capability and give it to user *B*.
One possible access right to an object is *transfer* or *propagate*. A subject having this right can pass copies of capabilities to other subjects. In turn, each of these capabilities also has

a list of permitted types of accesses, one of which might also be *transfer*. In this instance, process *A* can pass a copy of a capability to *B,* who can then pass a copy to *C*. *B* can prevent further distribution of the capability (and therefore prevent further dissemination of the access right) by omitting the *transfer* right from the rights passed in the capability to *C*. *B* might still pass certain access rights to *C*, but not the right to propagate access rights to other subjects.

As a process executes, it operates in a domain or local name space. The domain is the collection of objects to which the process has access. A domain for a user at a given time might include some programs, files, data segments, and I/O devices such as a printer and a terminal. An example of a domain is shown in Figure 4-13.

Figure 4-13. Process Execution Domain.

As execution continues, the process may call a subprocedure, passing some of the objects to which it has access as arguments to the subprocedure. The domain of the subprocedure is not necessarily the same as that of its calling procedure; in fact, a calling procedure may pass only some of its objects to the subprocedure, and the subprocedure may have access rights to other objects not accessible to the calling procedure. The caller may also pass only some of its access rights for the objects it passes to the subprocedure. For example, a procedure might pass to a subprocedure the right to read but not modify a particular data value.

Because each capability identifies a single object in a domain, the collection of capabilities defines the domain. When a process calls a subprocedure and passes certain objects to the subprocedure, the operating system forms a stack of all the capabilities of the current procedure. The operating system then creates new capabilities for the subprocedure, as shown in Figure 4-14.

Figure 4-14. Passing Objects to a Subject.

Operationally, capabilities are a straightforward way to keep track of the access rights of subjects to objects during execution. The capabilities are backed up by a more comprehensive table, such as an access control matrix or an access control list. Each time a process seeks to use a new object, the operating system examines the master list of objects and subjects to determine whether the object is accessible. If so, the operating system creates a capability for that object.

Capabilities must be stored in memory inaccessible to normal users. One way of accomplishing this is to store capabilities in segments not pointed at by the user's segment table or to enclose them in protected memory as from a pair of base/bounds registers. Another approach is to use a tagged architecture machine to identify capabilities as structures requiring protection.

During execution, only the capabilities of objects that have been accessed by the current process are kept readily available. This restriction improves the speed with which access to an object can be checked. This approach is essentially the one used in Multics, as described in [FAB74].

Capabilities can be revoked. When an issuing subject revokes a capability, no further access under the revoked capability should be permitted. A capability table can contain pointers to the active capabilities spawned under it so that the operating system can trace what access rights should be deleted if a capability is revoked. A similar problem is deleting capabilities for users who are no longer active.

Kerberos

Fundamental research on capabilities laid the groundwork for subsequent production use in systems such as Kerberos [STE88] (studied in greater detail in Chapter 7). Kerberos implements both authentication and access authorization by means of capabilities, called tickets, secured with symmetric cryptography. Microsoft has based much of its access control in NT+ on Kerberos.

Kerberos requires two systems, called the authentication server (AS) and the ticket-granting server (TGS), which are both part of the key distribution center (KDC). A user presents an authenticating credential (such as a password) to the authentication server and receives a ticket showing that the user has passed authentication. Obviously, the ticket must be encrypted to prevent the user from modifying or forging one claiming to be a different user, and the ticket must contain some provision to prevent one user from acquiring another user's ticket to impersonate that user.

Now let us suppose that a user, Joe, wants to access a resource R (for example, a file, printer, or network port). Joe sends the TGS his authenticated ticket and a request to use R. Assuming Joe is allowed access, the TGS returns to Joe two tickets: One shows Joe that his access to R has been authorized, and the second is for Joe to present to R in order to access R.

Kerberos implements single sign-on; that is, a user signs on once and from that point on all the user's (allowable) actions are authorized without the user needing to sign on again. So if a user wants access to a resource in a different domain, say on a different system or in a different environment or even a different company or institution, as long as authorization rights have been established between the two domains, the user's access takes place without the user's signing on to a different system.

Kerberos accomplishes its local and remote authentication and authorization with a system of shared secret encryption keys. In fact, each user's password is used as an encryption key.

(That trick also means that passwords are never exposed, reducing the risk from interception.) We study the exact mechanism of Kerberos in Chapter 7.

Procedure-Oriented Access Control

One goal of access control is restricting not just which subjects have access to an object, but also what they can *do* to that object. Read versus write access can be controlled rather readily by most operating systems, but more complex control is not so easy to achieve.

By procedure-oriented protection, we imply the existence of a procedure that controls access to objects (for example, by performing its own user authentication to strengthen the basic protection provided by the basic operating system). In essence, the procedure forms a capsule around the object, permitting only certain specified accesses.

Procedures can ensure that accesses to an object be made through a trusted interface. For example, neither users nor general operating system routines might be allowed direct access to the table of valid users. Instead, the only accesses allowed might be through three procedures: one to add a user, one to delete a user, and one to check whether a particular name corresponds to a valid user. These procedures, especially add and delete, could use their own checks to make sure that calls to them are legitimate.

Procedure-oriented protection implements the principle of information hiding because the means of implementing an object are known only to the object's control procedure. Of course, this degree of protection carries a penalty of inefficiency. With procedure-oriented protection, there can be no simple, fast access, even if the object is frequently used.

Our survey of access control mechanisms has intentionally progressed from simple to complex. Historically, as the mechanisms have provided greater flexibility, they have done so with a price of increased overhead. For example, implementing capabilities that must be checked on each access is far more difficult than implementing a simple directory structure that is checked only on a subject's first access to an object. This complexity is apparent both to the user and to the implementer. The user is aware of additional protection features, but the naïve user may be frustrated or intimidated at having to select protection options with little understanding of their usefulness. The implementation complexity becomes apparent in slow response to users. The balance between simplicity and functionality is a continuing battle in security.

Role-Based Access Control

We have not yet distinguished among kinds of users, but we want some users (such as administrators) to have significant privileges, and we want others (such as regular users or guests) to have lower privileges. In companies and educational institutions, this can get complicated when an ordinary user becomes an administrator or a baker moves to the candlestick makers' group. Role-based access control lets us associate privileges with groups, such as all administrators can do this or candlestick makers are forbidden to do this.

Administering security is easier if we can control access by job demands, not by person. Access control keeps up with a person who changes responsibilities, and the system administrator does not have to choose the appropriate access control settings for someone. For more details on the nuances of role-based access control, see [FER03].

File protection mechanism

4.4. File Protection Mechanisms

Until now, we have examined approaches to protecting a general object, no matter the object's nature or type. But some protection schemes are particular to the type. To see how they work, we focus in this section on file protection. The examples we present are only representative; they do not cover all possible means of file protection on the market.

Basic Forms of Protection

We noted earlier that all multiuser operating systems must provide some minimal protection to keep one user from maliciously or inadvertently accessing or modifying the files of another. As the number of users has grown, so also has the complexity of these protection schemes.

AllNone Protection

In the original IBM OS operating systems, files were by default public. Any user could read, modify, or delete a file belonging to any other user. Instead of software- or hardware-based protection, the principal protection involved trust combined with ignorance. System designers supposed that users could be trusted not to read or modify others' files because the users would expect the same respect from others. Ignorance helped this situation, because a user could access a file only by name; presumably users knew the names only of those files to which they had legitimate access.

However, it was acknowledged that certain system files were sensitive and that the system administrator could protect them with a password. A normal user could exercise this feature, but passwords were viewed as most valuable for protecting operating system files. Two philosophies guided password use. Sometimes, passwords controlled all accesses (read, write, or delete), giving the system administrator complete control over all files. But at other times passwords controled only write and delete accesses because only these two actions affected other users. In either case, the password mechanism required a system operator's intervention each time access to the file began.

However, this all-or-none protection is unacceptable for several reasons.

 Lack of trust. The assumption of trustworthy users is not necessarily justified. For systems with few users who all know each other, mutual respect might suffice; but in large systems where not every user knows every other user, there is no basis for trust.

 Too coarse. Even if a user identifies a set of trustworthy users, there is no convenient way to allow access only to them.

 Rise of sharing. This protection scheme is more appropriate for a batch environment, in which users have little chance to interact with other users and in which users do their thinking and exploring when not interacting with the system. However, on shared-use systems, users interact with other users and programs representing other classes of users.

 Complexity. Because (human) operator intervention is required for this file protection, operating system performance is degraded. For this reason, this type of file protection is discouraged by computing centers for all but the most sensitive data sets.

 File listings. For accounting purposes and to help users remember for what files they are responsible, various system utilities can produce a list of all files. Thus, users are not necessarily ignorant of what files reside on the system. Interactive users may try to browse through any unprotected files.

Group Protection

Because the all-or-nothing approach has so many drawbacks, researchers sought an improved way to protect files. They focused on identifying *groups* of users who had some common relationship. In a typical Unix+ implementation, the world is divided into three classes: the user, a trusted working group associated with the user, and the rest of the users. For simplicity we can call these classes *user*, *group,* and *world*. Windows NT+ uses groups such as Administrators, Power Users, Users, and Guests. (NT+ administrators can also create other groups.)

All authorized users are separated into groups. A group may consist of several members working on a common project, a department, a class, or a single user. The basis for group membership is *need to share*. The group members have some common interest and therefore are assumed to have files to share with the other group members. In this approach, no user belongs to more than one group. (Otherwise, a member belonging to groups *A* and *B* could pass along an *A* file to another *B* group member.)

When creating a file, a user defines access rights to the file for the user, for other members of the same group, and for all other users in general. Typically, the choices for access rights are a limited set, such as {update, readexecute, read, writecreatedelete}. For a particular file, a user might declare read-only access to the general world, read and update access to the group, and all rights to the user. This approach would be suitable for a paper being developed by a group, whereby the different members of the group might modify sections being written within the group. The paper itself should be available for people outside the group to review but not change.
A key advantage of the group protection approach is its ease of implementation. A user is recognized by two identifiers (usually numbers): a user ID and a group ID. These identifiers are stored in the file directory entry for each file and are obtained by the operating system when a user logs in. Therefore, the operating system can easily check whether a proposed access to a file is requested from someone whose group ID matches the group ID for the file to be accessed.
Although this protection scheme overcomes some of the shortcomings of the all-or-nothing scheme, it introduces some new difficulties of its own.
 Group affiliation. A single user cannot belong to two groups. Suppose Tom belongs to one group with Ann and to a second group with Bill. If Tom indicates that a file is to be readable by the group, to which group(s) does this permission refer? Suppose a file of Ann's is readable by the group; does Bill have access to it? These ambiguities are most simply resolved by declaring that every user belongs to exactly one group. (This restriction does not mean that all users belong to the *same* group.)
 Multiple personalities. To overcome the one-person one-group restriction, certain people might obtain multiple accounts, permitting them, in effect, to be multiple users.
This hole in the protection approach leads to new problems because a single person can be only one user at a time. To see how problems arise, suppose Tom obtains two accounts, thereby becoming Tom1 in a group with Ann and Tom2 in a group with Bill. Tom1 is not in the same group as Tom2, so any files, programs, or aids developed under the Tom1 account can be available to Tom2 only if they are available to the entire world. Multiple personalities lead to a proliferation of accounts, redundant files, limited protection for files of general interest, and inconvenience to users.
 All groups. To avoid multiple personalities, the system administrator may decide that Tom should have access to all his files any time he is active. This solution puts the responsibility on Tom to control with whom he shares what things. For example, he may be in Group1 with Ann and Group2 with Bill. He creates a Group1 file to share with Ann. But if he is active in Group2 the next time he is logged in, he still sees the Group1 file and may not realize that it is not accessible to Bill, too.
 Limited sharing. Files can be shared only within groups or with the world. Users want to be able to identify sharing partners for a file on a per-file basis; for example, sharing one file with ten people and another file with twenty others.

Individual Permissions

In spite of their drawbacks, the file protection schemes we have described are relatively simple and straightforward. The simplicity of implementing them suggests other easy-to-manage methods that provide finer degrees of security while associating permission with a single file.

Persistent Permission

From other contexts you are familiar with persistent permissions. The usual implementation of such a scheme uses a name (you claim a dinner reservation under the name of Sanders), a token (you show your driver's license or library card), or a secret (you say a secret word or give the club handshake). Similarly, in computing you are allowed access by being on the access list, presenting a token or ticket, or giving a password. User access permissions can be required for any access or only for modifications (write access).
All these approaches present obvious difficulties in revocation: Taking someone off one list is easy, but it is more complicated to find all lists authorizing someone and remove him or her.
Reclaiming a token or password is even more challenging.

Temporary Acquired Permission

Unix+ operating systems provide an interesting permission scheme based on a three-level usergroupworld hierarchy. The Unix designers added a permission called set userid (suid).

If this protection is set for a file to be executed, the protection level is that of the file's *owner*, not the *executor*. To see how it works, suppose Tom owns a file and allows Ann to execute it with *suid*. When Ann executes the file, she has the protection rights of Tom, not of herself.
This peculiar-sounding permission has a useful application. It permits a user to establish data files to which access is allowed only through specified procedures.
For example, suppose you want to establish a computerized dating service that manipulates a database of people available on particular nights. Sue might be interested in a date for Saturday, but she might have already refused a request from Jeff, saying she had other plans.
Sue instructs the service not to reveal to Jeff that she is available. To use the service, Sue, Jeff, and others must be able to read the file and write to it (at least indirectly) to determine who is available or to post their availability. But if Jeff can read the file directly, he would find that Sue has lied. Therefore, your dating service must force Sue and Jeff (and all others) to access this file only through an access program that would screen the data Jeff obtains. But if the file access is limited to read and write by you as its owner, Sue and Jeff will never be able to enter data into it.
The solution is the Unix SUID protection. You create the database file, giving only you access permission. You also write the program that is to access the database, and save it with the SUID protection. Then, when Jeff executes your program, he temporarily acquires your access permission, but only during execution of the program. Jeff never has direct access to the file because your program will do the actual file access. When Jeff exits from your program, he regains his own access rights and loses yours. Thus, your program can access the file, but the program must display to Jeff only the data Jeff is allowed to see.
This mechanism is convenient for system functions that general users should be able to perform only in a prescribed way. For example, only the system should be able to modify the file of users' passwords, but individual users should be able to change their own passwords any time they wish. With the SUID feature, a password change program can be owned by the system, which will therefore have full access to the system password table. The program to change passwords also has SUID protection so that when a normal user executes it, the program can modify the password file in a carefully constrained way on behalf of the user.

Per-Object and Per-User Protection

The primary limitation of these protection schemes is the ability to create meaningful groups of related users who should have similar access to related objects. The access control lists or access control matrices described earlier provide very flexible protection. Their disadvantage is for the user who wants to allow access to many users and to many different data sets;
such a user must still specify each data set to be accessed by each user. As a new user is added, that user's special access rights must be specified by all appropriate users.

Authentication:

Authentication basics-

An operating system bases much of its protection on knowing who a user of the system is. In real-life situations, people commonly ask for identification from people they do not know: A bank employee may ask for a driver's license before cashing a check, library employees may require some identification before charging out books, and immigration officials ask for passports as proof of identity. In-person identification is usually easier than remote identification. For instance, some universities do not report grades over the telephone because the office workers do not necessarily know the students calling. However, a professor who recognizes the voice of a certain student can release that student's grades. Over time, organizations and systems have developed means of authentication, using documents, voice recognition, fingerprint and retina matching, and other trusted means of identification.
In computing, the choices are more limited and the possibilities less secure. Anyone can attempt to log in to a computing system. Unlike the professor who recognizes a student's voice, the computer cannot recognize electrical signals from one person as being any different from those of anyone else. Thus, most computing authentication systems must be based on some knowledge shared only by the computing system and the user.

Authentication mechanisms use any of three qualities to confirm a user's identity.
1. Something the user *knows*a Passwords, PIN numbers, passphrases, a secret handshake, and mother's maiden name are examples of what a user may know.
2. Something the user *has*a Identity badges, physical keys, a driver's license, or a uniform are common examples of things people have that make them recognizable.
3. Something the user *is*a These authenticators, called biometrics, are based on a physical characteristic of the user, such as a fingerprint, the pattern of a person's voice, or a face (picture). These authentication methods are old (we recognize friends in person by their faces or on a telephone by their voices) but are just starting to be used in computer authentications. See Sidebar 4-3 for a glimpse at some of the promising approaches.
Two or more forms can be combined for more solid authentication; for example, a bank card and a PIN combine something the user has with something the user knows.

Password

Passwords as Authenticators

The most common authentication mechanism for user to operating system is a password, a "word" known to computer and user. Although password protection seems to offer a relatively secure system, human practice sometimes degrades its quality. In this section we consider passwords, criteria for selecting them, and ways of using them for authentication. We conclude by noting other authentication techniques and by studying problems in the authentication process, notably Trojan horses masquerading as the computer authentication process.

Use of Passwords

Passwords are mutually agreed-upon code words, assumed to be known only to the user and the system. In some cases a user chooses passwords; in other cases the system assigns them. The length and format of the password also vary from one system to another.
Even though they are widely used, passwords suffer from some difficulties of use:
 Loss. Depending on how the passwords are implemented, it is possible that no one will be able to replace a lost or forgotten password. The operators or system administrators can certainly intervene and unprotect or assign a particular password, but often they cannot determine what password a user has chosen; if the user loses the password, a new one must be assigned.
 Use. Supplying a password for each access to a file can be inconvenient and time consuming.
 Disclosure. If a password is disclosed to an unauthorized individual, the file becomes immediately accessible. If the user then changes the password to reprotect the file, all the other legitimate users must be informed of the new password because their old password will fail.
 Revocation. To revoke one user's access right to a file, someone must change the password, thereby causing the same problems as disclosure.
The use of passwords is fairly straightforward. A user enters some piece of identification, such as a name or an assigned user ID; this identification can be available to the public or easy to guess because it does not provide the real security of the system. The system then requests a password from the user. If the password matches that on file for the user, the user is authenticated and allowed access to the system. If the password match fails, the system requests the password again, in case the user mistyped.

Additional Authentication Information

In addition to the name and password, we can use other information available to authenticate users. Suppose Adams works in the accounting department during the shift between 8:00 a.m.
and 5:00 p.m., Monday through Friday. Any legitimate access attempt by Adams should be made during those times, through a workstation in the accounting department offices. By limiting Adams to logging in under those conditions, the system protects against two problems:
 Someone from outside might try to impersonate Adams. This attempt would be thwarted by either the time of access or the port through which the access was attempted.
 Adams might attempt to access the system from home or on a weekend, planning to use resources not allowed or to do something that would be too risky with other people around.

Limiting users to certain workstations or certain times of access can cause complications (as when a user legitimately needs to work overtime, a person has to access the system while out of town on a business trip, or a particular workstation fails). However, some companies use these authentication techniques because the added security they provide outweighs inconveniences.
Using additional authentication information is called multifactor authentication. Two forms of authentication (which is, not surprisingly, known as two-factor authentication) are better than one, assuming of course that the two forms are strong. But as the number of forms increases, so also does the inconvenience. (For example, think about passing through a security checkpoint at an airport.) Each authentication factor requires the system and its administrators to manage more security information.

Attacks on Passwords

How secure are passwords themselves? Passwords are somewhat limited as protection devices because of the relatively small number of bits of information they contain. Here are some ways you might be able to determine a user's password, in decreasing order of difficulty.

- Try all possible passwords.
- Try frequently used passwords.
- Try passwords likely for the user.
- Search for the system list of passwords.
- Ask the user.

Loose-Lipped Systems

So far the process seems secure, but in fact it has some vulnerabilities. To see why, consider the actions of a would-be intruder. Authentication is based on knowing the <*name*, *password* > pair A complete outsider is presumed to know nothing of the system. Suppose the intruder attempts to access a system in the following manner. (In the following examples, the system messages are in uppercase, and the user's responses are in lowercase.)

```
WELCOME TO THE XYZ COMPUTING SYSTEMS
ENTER USER NAME: adams
INVALID USER NAMEUNKNOWN USER
ENTER USER NAME:
```

We assumed that the intruder knew nothing of the system, but without having to do much, the intruder found out that *adams* is not the name of an authorized user. The intruder could try other common names, first names, and likely generic names such as *system* or *operator* to build a list of authorized users.
An alternative arrangement of the login sequence is shown below.

```
WELCOME TO THE XYZ COMPUTING SYSTEMS
ENTER USER NAME: adams
ENTER PASSWORD: john
INVALID ACCESS
ENTER USER NAME:
```

This system notifies a user of a failure only after accepting both the user name and the password. The failure message should not indicate whether it is the user name or password that is unacceptable. In this way, the intruder does not know which failed.
These examples also gave a clue as to which computing system is being accessed. The true outsider has no right to know that, and legitimate insiders already know what system they have accessed. In the example below, the user is given no information until the system is assured of the identity of the user.

```
ENTER USER NAME: adams
ENTER PASSWORD: john
INVALID ACCESS
ENTER USER NAME: adams
ENTER PASSWORD: johnq
WELCOME TO THE XYZ COMPUTING SYSTEMS
```

Exhaustive Attack

In an exhaustive or brute force attack, the attacker tries all possible passwords, usually in some automated fashion. Of course, the number of possible passwords depends on the implementation of the particular computing system. For example, if passwords are words

consisting of the 26 characters AZ and can be of any length from 1 to 8 characters, there are 26_1 passwords of 1 character, 26_2 passwords of 2 characters, and 26_8 passwords of 8 characters. Therefore, the system as a whole has $26_1 + 26_2 + ... + 26_8 = 26_9 - 1\ 5 * 10_{12}$
or five million million possible passwords. That number seems intractable enough. If we were to use a computer to create and try each password at a rate of checking one password per millisecond, it would take on the order of 150 years to test all passwords. But if we can speed up the search to one password per microsecond, the work factor drops to about two months.
This amount of time is reasonable if the reward is large. For instance, an intruder may try to break the password on a file of credit card numbers or bank account information.
But the break-in time can be made more tractable in a number of ways. Searching for a single particular password does not necessarily require all passwords to be tried; an intruder needs to try only until the correct password is identified. If the set of all possible passwords were evenly distributed, an intruder would likely need to try only half of the password space: the expected number of searches to find any particular password. However, an intruder can also use to advantage the fact that passwords are not evenly distributed. Because a password has to be remembered, people tend to pick simple passwords. This feature reduces the size of the password space.

Probable Passwords

Think of a word.
Is the word you thought of long? Is it uncommon? Is it hard to spell or to pronounce? The answer to all three of these questions is probably no. Penetrators searching for passwords realize these very human characteristics and use them to their advantage. Therefore, penetrators try techniques that are likely to lead to rapid success. If people prefer short passwords to long ones, the penetrator will plan to try all passwords but to try them in order by length. There are only $26_1 + 26_2 + 26_3$=18,278
passwords of length 3 or less. At the assumed rate of one password per millisecond, all of these passwords can be checked in 18.278 seconds, hardly a challenge with a computer. Even expanding the tries to 4 or 5 characters raises the count only to 475 seconds (about 8 minutes) or 12,356 seconds (about 3.5 hours), respectively.
This analysis assumes that people choose passwords such as *vxlag* and *msms* as often as they pick *enter* and *beer*. However, people tend to choose names or words they can remember. Many computing systems have spelling checkers that can be used to check for spelling errors and typographic mistakes in documents. These spelling checkers sometimes carry online dictionaries of the most common English words. One contains a dictionary of 80,000 words. Trying all of these words as passwords takes only 80 seconds.

Passwords Likely for a User

If Sandy is selecting a password, she is probably not choosing a word completely at random. Most likely Sandy's password is something meaningful to her. People typically choose personal passwords, such as the name of a spouse, a child, a brother or sister, a pet, a street name, or something memorable or familiar. If we restrict our password attempts to just names of people (first names), streets, projects, and so forth, we generate a list of only a few hundred possibilities at most. Trying this number of passwords takes under a seconda Even a person working by hand could try ten likely candidates in a minute or two.
Thus, what seemed formidable in theory is in fact quite vulnerable in practice, and the likelihood of successful penetration is frightening. Morris and Thompson [MOR79] confirmed our fears in their report on the results of having gathered passwords from many users, shown in Table 4-2. Figure 4-15 (based on data from that study) shows the characteristics of the 3,289 passwords gathered. The results from that study are distressing, and the situation today is likely to be the same. Of those passwords, 86 percent could be uncovered in about one week's worth of 24-hour-a-day testing, using the very generous estimate of 1 millisecond per password check.

Table 4-2. Distribution of Actual Passwords.

15	0.5%	were a single(a) ASCII character
72	2%	were two ASCII characters
464	14%	were three ASCII characters
477	14%	were four alphabetic letters
706	21%	were five alphabetic letters, all the same case
605	18%	were six lowercase alphabetic letters

492	15%	were words in dictionaries or lists of names
2831	86%	total of all above categories

Figure 4-15. Users' Password Choices.

Lest you dismiss these results as dated (they were reported in 1979), Klein repeated the experiment in 1990 [KLE90] and Spafford in 1992 [SPA92]. Each collected approximately 15,000 passwords. Klein reported that 2.7 percent of the passwords were guessed in only 15 minutes of machine time and 21 percent were guessed within a weeka Spafford found the average password length was 6.8 characters, and 28.9 percent consisted of only lowercase alphabetic characters. Notice that both these studies were done *after* the Internet worm (described in Chapter 3) succeeded, in part by breaking weak passwords.

Even in 2002, the British online bank Egg found users still choosing weak passwords [BUX02].

A full 50 percent of passwords for their online banking service were family members' names: 23 percent children's names, 19 percent a spouse or partner, and 9 percent their own. Alas, pets came in at only 8 percent, while celebrities and football (soccer) stars tied at 9 percent each.

And in 1998, Knight and Hartley [KNI98] reported that approximately 35 percent of passwords are deduced from syllables and initials of the account owner's name.

Two friends we know have told us their passwords as we helped them administer their systems, and their passwords would both have been among the first we would have guessed.

But, you say, these are amateurs unaware of the security risk of a weak password. At a recent meeting, a security expert related this experience: He thought he had chosen a solid password, so he invited a class of students to ask him a few questions and offer some guesses as to his password. He was amazed that they asked only a few questions before they had deduced the password. And this was a security expert.

Several news articles have claimed that the four most common passwords are "God," "sex," "love,"and "money" (the order among those is unspecified). The perhaps apocryphal list of common passwords at geodsoft.com/howto/password/common.htm appears at several other places on the Internet. Or see the default password list at www.phenoelit.de/dpl/dpl.html.

Whether these are really passwords we do not know. Still, it warrants a look because similar lists are bound to be built into some hackers' tools.

Several network sites post dictionaries of phrases, science fiction characters, places, mythological names, Chinese words, Yiddish words, and other specialized lists. All these lists are posted to help site administrators identify users who have chosen weak passwords, but the same dictionaries can also be used by attackers of sites that do not have such attentive administrators. The COPS [FAR90], Crack [MUF92], and SATAN [FAR95] utilities allow an administrator to scan a system for weak passwords. But these same utilities, or other homemade ones, allow attackers to do the same. Now Internet sites offer so-called password recovery software as freeware or shareware for under $20. (These are password-cracking programs.)

People think they can be clever by picking a simple password and replacing certain characters, such as 0 (zero) for letter O, 1 (one) for letter I or L, 3 (three) for letter E or @ (at) for letter A. But users aren't the only people who could think up these substitutions. Knight and Hartley [KNI98] list, in order, 12 steps an attacker might try in order to determine a password. These steps are in increasing degree of difficulty (number of guesses), so they indicate the amount of work to which the attacker must go to derive a password. Here are their password guessing steps:

•. no password
•. the same as the user ID
•. is, or is derived from, the user's name
•. common word list (for example, "password," "secret," "private") plus common names and patterns (for example, "asdfg," "aaaaaa")
•. short college dictionary
•. complete English word list
•. common non-English language dictionaries
•. short college dictionary with capitalizations (PaSsWorD) and substitutions (0 for O, and so forth)

•. complete English with capitalizations and substitutions
•. common non-English dictionaries with capitalization and substitutions
•. brute force, lowercase alphabetic characters
•. brute force, full character set
Although the last step will always succeed, the steps immediately preceding it are so time consuming that they will deter all but the dedicated attacker for whom time is not a limiting factor.

Plaintext System Password List

To validate passwords, the system must have a way of comparing entries with actual passwords. Rather than trying to guess a user's password, an attacker may instead target the system password file. Why guess when with one table you can determine all passwords with total accuracy?

On some systems, the password list is a file, organized essentially as a two-column table of user IDs and corresponding passwords. This information is certainly too obvious to leave out in the open. Various security approaches are used to conceal this table from those who should not see it.

You might protect the table with strong access controls, limiting access to the operating system. But even this tightening of control is looser than it should be, because not every operating system module needs or deserves access to this table. For example, the operating system scheduler, accounting routines, or storage manager have no need to know the table's contents. Unfortunately, in some systems, there are *n*+1 known users: *n* regular users and the operating system. The operating system is not partitioned, so all its modules have access to all privileged information. This monolithic view of the operating system implies that a user who exploits a flaw in one section of the operating system has access to all the system's deepest secrets. A better approach is to limit table access to the modules that need access: the user authentication module and the parts associated with installing new users, for example.

If the table is stored in plain sight, an intruder can simply dump memory at a convenient time to access it. Careful timing may enable a user to dump the contents of all of memory and, by exhaustive search, find values that look like the password table.

System backups can also be used to obtain the password table. To be able to recover from system errors, system administrators periodically back up the file space onto some auxiliary medium for safe storage. In the unlikely event of a problem, the file system can be reloaded from a backup, with a loss only of changes made since the last backup. Backups often contain only file contents, with no protection mechanism to control file access. (Physical security and access controls to the backups themselves are depended on to provide security for the contents of backup media.) If a regular user can access the backups, even ones from several weeks, months, or years ago, the password tables stored in them may contain entries that are still valid.

Finally, the password file is a copy of a file stored on disk. Anyone with access to the disk or anyone who can overcome file access restrictions can obtain the password file.

Encrypted Password File

There is an easy way to foil an intruder seeking passwords in plain sight: encrypt them. Frequently, the password list is hidden from view with conventional encryption or one-way ciphers.

With conventional encryption, either the entire password table is encrypted or just the password column. When a user's password is received, the stored password is decrypted, and the two are compared.

Even with encryption, there is still a slight exposure because for an instant the user's password is available in plaintext in main memory. That is, the password is available to anyone who could obtain access to all of memory.

A safer approach uses one-way encryption, defined in Chapter 2. The password table's entries are encrypted by a one-way encryption and then stored. When the user enters a password, it is also encrypted and then compared with the table. If the two values are equal, the authentication succeeds. Of course, the encryption has to be such that it is unlikely that two passwords would encrypt to the same ciphertext, but this characteristic is true for most secure encryption algorithms.

With one-way encryption, the password file can be stored in plain view. For example, the password table for the Unix operating system can be read by any user unless special access

controls have been installed. Because the contents are encrypted, backup copies of the password table are no longer a problem.
There is always the possibility that two people might choose the same password, thus creating two identical entries in the password file. Even though the entries are encrypted, each user will know the plaintext equivalent. For instance, if Bill and Kathy both choose their passwords on April 1, they might choose APRILFOOL as a password. Bill might read the password file and notice that the encrypted version of his password is the same as Kathy's.
Unix+ circumvents this vulnerability by using a password extension, called the salt. The salt is a 12-bit number formed from the system time and the process identifier. Thus, the salt is likely to be unique for each user, and it can be stored in plaintext in the password file. The salt is concatenated to Bill's password (*pw*) when he chooses it; *E*(*pw*+salt$_B$) is stored for Bill, and his salt value is also stored. When Kathy chooses her password, the salt is different because the time or the process number is different. Call this new one salt$_K$. For her, *E*(*pw*+salt$_K$) and salt$_K$ are stored. When either person tries to log in, the system fetches the appropriate salt from the password table and combines that with the password before performing the encryption. The encrypted versions of (*pw*+salt) are very different for these two users. When Bill looks down the password list, the encrypted version of his password will not look at all like Kathy's. Storing the password file in a disguised form relieves much of the pressure to secure it. Better still is to limit access to processes that legitimately need access. In this way, the password file is protected to a level commensurate with the protection provided by the password itself.
Someone who has broken the controls of the file system has access to data, not just passwords, and that is a serious threat. But if an attacker successfully penetrates the outer security layer, the attacker still must get past the encryption of the password file to access the useful information in it.

Indiscreet Users

Guessing passwords and breaking encryption can be tedious or daunting. But there is a simple way to obtain a password: Get it directly from the usera People often tape a password to the side of a terminal or write it on a card just inside the top desk drawer. Users are afraid they will forget their passwords, or they cannot be bothered trying to remember them. It is particularly tempting to write the passwords down when users have several accounts.
Users sharing work or data may also be tempted to share passwords. If someone needs a file, it is easier to say "my password is *x*; get the file yourself" than to arrange to share the file.
This situation is a result of user laziness, but it may be brought about or exacerbated by a system that makes sharing inconvenient.
In an admittedly unscientific poll done by Verisign [TEC05], two-thirds of people approached on the street volunteered to disclose their password for a coupon good for a cup of coffee, and 79 percent admitted they used the same password for more than one system or web site.

Password Selection Criteria

At the RSA Security Conference in 2006, Bill Gates, head of Microsoft, described his vision of a world in which passwords would be obsolete, having gone the way of the dinosaur. In their place sophisticated multifactor authentication technologies would offer far greater security than passwords ever could. But that is Bill Gates' view of the future; despite decades of articles about their weakness, passwords are with us still and will be for some time.
So what can we conclude about passwords? They should be hard to guess and difficult to determine exhaustively. But the degree of difficulty should be appropriate to the security needs of the situation. To these ends, we present several guidelines for password selection:
 Use characters other than just AZ. If passwords are chosen from the letters AZ, there are only 26 possibilities for each character. Adding digits expands the number of possibilities to 36. Using both uppercase and lowercase letters plus digits expands the number of possible characters to 62. Although this change seems small, the effect is large when someone is testing a full space of all possible combinations of characters. It takes about 100 hours to test all 6-letter words chosen from letters of one case only, but it takes about 2 years to test

all 6-symbol passwords from upper- and lowercase letters and digits. Although 100 hours is reasonable, 2 years is oppressive enough to make this attack far less attractive.
 Choose long passwords. The combinatorial explosion of passwords begins at length 4 or 5. Choosing longer passwords makes it less likely that a password will be uncovered.
Remember that a brute force penetration can stop as soon as the password is found. Some penetrators will try the easy casesknown words and short passwordsand move on to another target if those attacks fail.
 Avoid actual names or words. Theoretically, there are 26_6 or about 300 million 6-letter "words", but there are only about 150,000 words in a good collegiate dictionary, ignoring length. By picking one of the 99.95 percent nonwords, you force the attacker to use a longer brute force search instead of the abbreviated dictionary search.
 Choose an unlikely password. Password choice is a double bind. To remember thepassword easily, you want one that has special meaning to you. However, you don't want someone else to be able to guess this special meaning. One easy-to-remember password is 2Brn2B. That unlikely looking jumble is a simple transformation of "to be or not to be." The first letters of words from a song, a few letters from different words of a private phrase, or a memorable basketball score are examples of reasonable passwords. But don't be too obvious. Password-cracking tools also test replacements of 0 (zero) for o or O (letter "oh") and 1 (one) for l (letter "ell") or $ for S (letter "ess").
So I10veu is already in the search file.
 Change the password regularly. Even if there is no reason to suspect that the password has been compromised, change is advised. A penetrator may break a password system by obtaining an old list or working exhaustively on an encrypted list.
 Don't write it down. (Note: This time-honored advice is relevant only if physical security is a serious risk. People who have accounts on many different machines and servers, not to mention bank and charge card PINs, may have trouble remembering all the access codes. Setting all codes the same or using insecure but easy-to-remember passwords may be more risky than writing passwords on a reasonably well protected list.)
 Don't tell anyone else. The easiest attack is social engineering, in which the attackercontacts the system's administrator or a user to elicit the password in some way. For example, the attacker may phone a user, claim to be "system administration," and ask the user to verify the user's password. Under no circumstances should you ever give out your private password; legitimate administrators can circumvent your password if need be, and others are merely trying to deceive you.
To help users select good passwords, some systems provide meaningless but pronounceable passwords. For example, the VAX VMS system randomly generates five passwords from which the user chooses one. They are pronounceable, so that the user should be able to repeat and memorize them. However, the user may misremember a password because of having interchanged syllables or letters of a meaningless string. (The sound "bliptab" is no more easily misremembered than "blaptib" or "blabtip.")
Yan et al. [YAN04] did experiments to determine whether users could remember passwords or passphrases better. First, they found that users are poor at remembering random passwords.
And instructions to users about the importance of selecting good passwords had little effect. But when they asked users to select their own password based on some mnemonic phrase they chose themselves, the users selected passwords that were harder to guess than regular (not based on a phrase) passwords.
Other systems encourage users to change their passwords regularly. The regularity of password change is usually a system parameter, which can be changed for the characteristics of a given installation. Suppose the frequency is set at 30 days. Some systems begin to warn the user after 25 days that the password is about to expire. Others wait until 30 days and inform the user that the password has expired. Some systems nag without end, whereas other systems cut off a user's access if a password has expired. Still others force the user immediately into the password change utility on the first login after 30 days.
Grampp and Morris [GRA84a] argue that this reminder process is not necessarily good.
Choosing passwords is not difficult, but under pressure a user may adopt any password, just to satisfy the system's demand for a new one. Furthermore, if this is the only time a

password can be changed, a bad password choice cannot be changed until the next scheduled time.
Sometimes when systems force users to change passwords periodically, users with favorite passwords will alternate between two passwords each time a change is required. To prevent password reuse, Microsoft Windows 2000 systems refuse to accept any of the *k* most recently used passwords. One user of such a system went through 24 password changes each month, just to cycle back to the favorite password.

One-Time Passwords

A one-time password is one that changes every time it is used. Instead of assigning a static phrase to a user, the system assigns a static mathematical function. The system provides an argument to the function, and the user computes and returns the function value. Such systems are also called challengeresponse systems because the system presents a challenge to the user and judges the authenticity of the user by the user's response. Here are some simple examples of one-time password functions; these functions are overly simplified to make the explanation easier. Very complex functions can be used in place of these simple ones for host authentication in a network.

 $f(x) = x + 1$. With this function, the system prompts with a value for x, and the user enters the value $x + 1$. The kinds of mathematical functions used are limited only by the ability of the user to compute the response quickly and easily. Other similar possibilities are $f(x) = 3x_2 - 9x + 2$, $f(x) = p_x$, where p_x is the xth prime number, or $f(x) = d * h$, where d is the date and h is the hour of the current time. (Alas, many users cannot perform simple arithmetic in their heads.)

 $f(x) = r(x)$. For this function, the receiver uses the argument as the seed for a random number generator (available to both the receiver and host). The user replies with the value of the first random number generated. A variant of this scheme uses x as a number of random numbers to generate. The receiver generates x random numbers and sends the xth of these to the host.

 $f(a_1a_2a_3a_4a_5a_6) = a_3a_1a_1a_4$. With this function, the system provides a character string, which the user must transform in some predetermined manner. Again, many different character operations can be used.

 $f(E(x)) = E(D(E(x)) + 1)$. In this function, the computer sends an encrypted value, $E(x)$. The user must decrypt the value, perform some mathematical function, and encrypt the result to return it to the system. Clearly, for human use, the encryption function must be something that can be done easily by hand, unlike the strong encryption algorithms in Chapter 2. For machine-to-machine authentication, however, an encryption algorithm such as DES or AES is appropriate.

One-time passwords are very important for authentication because (as becomes clear in Chapter 7) an intercepted password is useless because it cannot be reused. However, theirusefulness is limited by the complexity of algorithms people can be expected to remember. A password-generating device can implement more complex functions. Several models are readily available at reasonable prices. They are very effective at countering the threat of transmitting passwords in plaintext across a network. (See Sidebar 4-4 for another dilemma in remote authentication.)

The Authentication Process

Authentication usually operates as described previously. However, users occasionally mistype their passwords. A user who receives a message of INCORRECT LOGIN will carefully retype the login and gain access to the system. Even a user who is a terrible typist should be able to log in successfully in a few tries.
Some authentication procedures are intentionally slow. A legitimate user will not complain if the login process takes 5 or 10 seconds. To a penetrator who is trying an exhaustive search or a dictionary search, however, 5 or 10 seconds per trial makes this class of attack generally infeasible.
Someone whose login attempts continually fail may not be an authorized user. Systems commonly disconnect a user after a small number of failed logins, forcing the user to reestablish a connection with the system. (This action will slow down a penetrator who is trying to penetrate the system by telephone. After a small number of failures, the penetrator must reconnect, which takes a few seconds.)
In more secure installations, stopping penetrators is more important than tolerating users' mistakes. For example, some system administrators assume that all legitimate users can

type their passwords correctly within three tries. After three successive password failures, the account for that user is disabled and only the security administrator can reenable it. This action identifies accounts that may be the target of attacks by penetrators.

Fixing Flaws in the Authentication Process

Password authentication assumes that anyone who knows a password is the user to whom the password belongs. As we have seen, passwords can be guessed, deduced, or inferred. Some people give out their passwords for the asking. Other passwords have been obtained just by someone watching a user typing in the password. The password can be considered as a preliminary or first-level piece of evidence, but skeptics will want more convincing proof.

There are several ways to provide a second level of protection, including another round of passwords or a challengeresponse interchange.

Challenge-response

ChallengeResponse Systems

As we have just seen, the login is usually time invariant. Except when passwords are changed, each login looks like every other. A more sophisticated login requires a user ID and password,

followed by a challengeresponse interchange. In such an interchange, the system prompts the user for a reply that will be different each time the user logs in. For example, the system might display a four-digit number, and the user would have to correctly enter a function such as the sum or product of the digits. Each user is assigned a different challenge function to compute. Because there are many possible challenge functions, a penetrator who captures the user ID and password cannot necessarily infer the proper function.

A physical device similar to a calculator can be used to implement a more complicated response function. The user enters the challenge number, and the device computes and displays the response for the user to type in order to log in. (For more examples, see Chapter 7's discussion of network authentication.)

Impersonation of Login

In the systems we have described, the proof is one-sided. The system demands certain identification of the user, but the user is supposed to trust the system. However, a programmer can easily write a program that displays the standard prompts for user ID and password, captures the pair entered, stores the pair in a file, displays SYSTEM ERROR; DISCONNECTED, and exits. This attack is a type of Trojan horse. The perpetrator sets it up, leaves the terminal unattended, and waits for an innocent victim to attempt a login. The naïve victim may not even suspect that a security breach has occurred.

To foil this type of attack, the user should be sure the path to the system is reinitialized each time the system is used. On some systems, turning the terminal off and on again or pressing the BREAK key generates a clear signal to the computer to halt any running process for the terminal. (Microsoft chose <CTRLALTDELETE> as the path to the secure authorization mechanism for this reason.) Not every computer recognizes power-off or BREAK as an interruption of the current process, though. And computing systems are often accessed through networks, so physical reinitialization is impossible.

Alternatively, the user can be suspicious of the computing system, just as the system is suspicious of the user. The user will not enter confidential data (such as a password) until convinced that the computing system is legitimate. Of course, the computer acknowledges the user only after passing the authentication process. A computing system can display some information known only by the user and the system. For example, the system might read the user's name and reply "YOUR LAST LOGIN WAS 10 APRIL AT 09:47." The user can verify that the date and time are correct before entering a secret password. If higher security is desired, the system can send an encrypted timestamp. The user decrypts this and discovers that the time is current. The user then replies with an encrypted timestamp and password, to convince the system that a malicious intruder has not intercepted a password from some prior login.

Biometrics.

Biometrics: Authentication Not Using Passwords

Some sophisticated authentication devices are now available. These devices include handprint detectors, voice recognizers, and identifiers of patterns in the retina. Authentication with such devices uses unforgeable physical characteristics to authenticate

users. The cost continues to fall as these devices are adopted by major markets; the devices are useful in very high security situations. In this section we consider a few of the approaches available.
Biometrics are biological authenticators, based on some physical characteristic of the human body. The list of biometric authentication technologies is still growing. Now there are devices to recognize the following biometrics: fingerprints, hand geometry (shape and size of fingers), retina and iris (parts of the eye), voice, handwriting, blood vessels in the finger, and face.
Authentication with biometrics has advantages over passwords because a biometric cannot be lost, stolen, forgotten, lent, or forged and is always available, always at hand, so to speak.

Identification versus Authentication

Two concepts are easily confused: identification and authentication. Biometrics are very reliable for authentication but much less reliable for authentication. The reason is mathematical. All biometric readers operate in two phases: First, a user registers with the reader, during which time a characteristic of the user (for example, the geometry of the hand)
is captured and reduced to a template or pattern. During registration, the user may be asked to present the hand several times so that the registration software can adjust for variations, such as how the hand is positioned. Second, the user later seeks authentication from the system, during which time the system remeasures the hand and compares the new measurements with the stored template. If the new measurement is close enough to the template, the system accepts the authentication; otherwise, the system rejects it. Every template is thus a pattern of some number of measurements.
Unless every template is unique, that is, no two people have the same measured hand geometry, the system cannot uniquely identify subjects. However, as long as it is unlikely that an imposter will have the same biometric template as the real user, the system can authenticate. The difference is between a system that looks at a hand geometry and says "this is Captain Hook" (identification) versus a man who says "I, Captain Hook, present my hand to prove who I am" and the system confirms "this hand matches Captain Hook's template" (authentication). Biometric authentication is feasible today; biometric identification is largely still a research topic.

Problems with Biometrics

There are several problems with biometrics:
 Biometrics are relatively new, and some people find their use intrusive. Hand geometry and face recognition (which can be done from a camera across the room) are scarcely invasive, but people have real concerns about peering into a laser beam or sticking a finger into a slot. (See [SCH06a] for some examples of people resisting biometrics.)
 Biometric recognition devices are costly, although as the devices become more popular, their costs go down. Still, outfitting every user's workstation with a reader can be expensive for a large company with many employees.
 All biometric readers use sampling and establish a threshold for when a match is close enough to accept. The device has to sample the biometric, measure often hundreds of key points, and compare that set of measurements with a template. There is normal variability if, for example, your face is tilted, you press one side of a finger more than another, or your voice is affected by an infection. Variation reduces accuracy.
 Biometrics can become a single point of failure. Consider a retail application in which a biometric recognition is linked to a payment scheme: As one user puts it, "If my credit card fails to register, I can always pull out a second card, but if my fingerprint is not recognized, I have only that one finger." Forgetting a password is a user's fault; failing biometric authentication is not.
 Although equipment is improving, there are still false readings. We label a "false positive" or "false accept" a reading that is accepted when it should be rejected (that is, the authenticator does not match) and a "false negative" or "false reject" one that rejects when it should accept. Often, reducing a false positive rate increases false negatives, and vice versa. The consequences for a false negative are usually less than for a false positive, so an acceptable system may have a false positive rate of 0.001 percent but a false negative rate of 1 percent.

 The speed at which a recognition must be done limits accuracy. We might ideally like to take several readings and merge the results or evaluate the closest fit. But authentication is done to allow a user to do something: Authentication is not the end goal but a gate keeping the user from the goal. The user understandably wants to get past the gate and becomes frustrated and irritated if authentication takes too long.
 Although we like to think of biometrics as unique parts of an individual, forgeries are possible. The most famous example was an artificial fingerprint produced by researchers in Japan [MAT02]. Although difficult and uncommon, forgery will be an issue whenever the reward for a false positive is high enough. Sometimes overlooked in the authentication discussion is that credibility is a two-sided issue:
The system needs assurance that the user is authentic, but the user needs that same assurance about the system. This second issue has led to a new class of computer fraud called phishing, in which an unsuspecting user submits sensitive information to a malicious system impersonating a trustworthy one. Common targets of phishing attacks are banks and other financial institutions because fraudsters use the sensitive data they obtain from customers to take customers' money from the real institutions. We consider phishing in more detail in Chapter 7.
Authentication is essential for an operating system because accurate user identification is the key to individual access rights. Most operating systems and computing system administrators have applied reasonable but stringent security measures to lock out illegal users before they can access system resources. But, as reported in Sidebar 4-5, sometimes an inappropriate mechanism is forced into use as an authentication device.

Unit 3 :

Database Security:

Protecting data is at the heart of many secure systems, and many users (people, programs, or systems) rely on a database management system (DBMS) to manage the protection. For this reason, we devote this chapter to the security of database management systems, as an example of how application security can be designed and implemented for a specific task.

There is substantial current interest in DBMS security because databases are newer than programming and operating systems. Databases are essential to many business and government organizations, holding data that reflect the organization's core competencies. Often, when business processes are reengineered to make them more effective and more in tune with new or revised goals, one of the first systems to receive careful scrutiny is the set of databases supporting the business processes. Thus, databases are more than software-related repositories. Their organization and contents are considered valuable corporate assets that must be carefully protected.

However, the protection provided by database management systems has had mixed results. Over time, we have improved our understanding of database security problems, and several good controls have been developed. But, as you will see, there are still more security concerns for which there are no available controls.

We begin this chapter with a brief summary of database terminology. Then we consider the security requirements for database management systems. Two major security Problems integrity and secrecy are explained in a database context. We continue the chapter by studying two major (but related) database security problems, the inference problem and the multilevel problem. Both problems are complex, and there are no immediate solutions.

However, by understanding the problems, we become more sensitive to ways of reducing potential threats to the data. Finally, we conclude the chapter by looking at data mining, a technology for deriving patterns from one or more databases. Data mining involves many of the security issues we raise in this chapter.

Security requirements

The basic security requirements of database systems are not unlike those of other computing systems we have studied. The basic problems access control, exclusion of spurious data, authentication of users, and reliability have appeared in many contexts so far in this book. Following is a list of requirements for database security.

 Physical database integrity. The data of a database are immune to physical problems, such as power failures, and someone can reconstruct the database if it is destroyed through a catastrophe.
 Logical database integrity. The structure of the database is preserved. With logical integrity of a database, a modification to the value of one field does not affect other fields, for example.
 Element integrity. The data contained in each element are accurate.
 Auditability. It is possible to track who or what has accessed (or modified) the elements in the database.
 Access control. A user is allowed to access only authorized data, and different users can be restricted to different modes of access (such as read or write).
 User authentication. Every user is positively identified, both for the audit trail and for permission to access certain data.
 Availability. Users can access the database in general and all the data for which they are authorized.
We briefly examine each of these requirements.

Integrity of the Database

If a database is to serve as a central repository of data, users must be able to trust the accuracy of the data values. This condition implies that the database administrator must be assured that updates are performed only by authorized individuals. It also implies that the data must be protected from corruption, either by an outside illegal program action or by an outside force such as fire or a power failure. Two situations can affect the integrity of a database: when the whole database is damaged (as happens, for example, if its storage medium is damaged) or when individual data items are unreadable.

Integrity of the database as a whole is the responsibility of the DBMS, the operating system, and the (human) computing system manager. From the perspective of the operating system and the computing system manager, databases and DBMSs are files and programs, respectively. Therefore, one way of protecting the database as a whole is to regularly back up all files on the system. These periodic backups can be adequate controls against catastrophic failure.

Sometimes it is important to be able to reconstruct the database at the point of a failure. For instance, when the power fails suddenly, a bank's clients may be in the middle of making transactions or students may be in the midst of registering online for their classes. In these cases, we want to be able to restore the systems to a stable point without forcing users to redo their recently completed transactions. To handle these situations, the DBMS must maintain a log of transactions. For example, suppose the banking system is designed so that a message is generated in a log (electronic or paper or both) each time a transaction is processed. In the event of a system failure, the system can obtain accurate account balances by reverting to a backup copy of the database and reprocessing all later transactions from the log.

Element Integrity

The integrity of database elements is their correctness or accuracy. Ultimately, authorized users are responsible for entering correct data into databases. However, users and programs make mistakes collecting data, computing results, and entering values. Therefore, DBMSs sometimes take special action to help catch errors as they are made and to correct errors after they are inserted.

This corrective action can be taken in three ways. First, the DBMS can apply field checks, activities that test for appropriate values in a position. A field might be required to be numeric, an uppercase letter, or one of a set of acceptable characters. The check ensures that a value falls within specified bounds or is not greater than the sum of the values in two other fields. These checks prevent simple errors as the data are entered. (Sidebar 6-1 demonstrates the importance of element integrity.)

A second integrity action is provided by access control. To see why, consider life without databases. Data files may contain data from several sources, and redundant data may be stored in several different places. For example, a student's home address may be stored in many different campus files: at class registration, for dining hall privileges, at the bookstore, and in the financial aid office. Indeed, the student may not even be aware that each separate office has the address on file. If the student moves from one residence to another, each of the separate files requires correction. Without a database, there are several risks to the data's integrity. First, at a given time, there could be some data files with the old address (they have not yet been updated) and some simultaneously with the new address (they have already been updated). Second, there is always the possibility that the data fields were changed incorrectly, again leading to files with incorrect information. Third, there may be files of which the student is unaware, so he or she does not know to notify the file owner about updating the address information. These problems are solved by databases. They enable collection and control of this data at one central source, ensuring the student and users of having the correct address.

However, the centralization is easier said than done. Who owns this shared central file? Who has authorization to update which elements? What if two people apply conflicting modifications? What if modifications are applied out of sequence? How are duplicate records detected? What action is taken when duplicates are found? These are policy questions that must be resolved by the database administrator. Sidebar 6-2 describes how these issues are addressed for managing the configuration of programs; similar formal processes are needed for managing changes in databases.

The third means of providing database integrity is maintaining a change log for the database. A change log lists every change made to the database; it contains both original and modified values. Using this log, a database administrator can undo any changes that were made in error. For example, a library fine might erroneously be posted against Charles W. Robertson, instead of Charles M. Robertson, flagging Charles W. Robertson as ineligible to participate in varsity athletics. Upon discovering this error, the database administrator obtains Charles W.'s original eligibility value from the log and corrects the database.

Auditability

For some applications it may be desirable to generate an audit record of all access (read or write) to a database. Such a record can help to maintain the database's integrity, or at least to discover after the fact who had affected which values and when. A second advantage, as we see later, is that users can access protected data incrementally; that is, no single access reveals protected data, but a set of sequential accesses viewed together reveals the data, much like discovering the clues in a detective novel. In this case, an audit trail can identify which clues a user has already been given, as a guide to whether to tell the user more.

As we noted in Chapters 4 and 5, granularity becomes an impediment in auditing. Audited events in operating systems are actions like *open file* or *call procedure*; they are seldom as specific as *write record* 3 or *execute instruction* I. To be useful for maintaining integrity, database audit trails should include accesses at the record, field, and even element levels.

This detail is prohibitive for most database applications. Furthermore, it is possible for a record to be accessed but not reported to a user, as when the user performs a select operation. (Accessing a record or an element without transferring to the user the data received is called the pass-through problem.) Also, you can determine the values of some elements without accessing them directly. (For example, you can ask for the average salary in a group of employees when you know the number of employees in the group is only one.) Thus, a log of all records accessed directly may both overstate and understate what a user actually knows.

Access Control

Databases are often separated logically by user access privileges. For example, all users can be granted access to general data, but only the personnel department can obtain salary data and only the marketing department can obtain sales data. Databases are very useful because they centralize the storage and maintenance of data. Limited access is both a responsibility and a benefit of this centralization.

The database administrator specifies who should be allowed access to which data, at the view, relation, field, record, or even element level. The DBMS must enforce this policy, granting access to all specified data or no access where prohibited. Furthermore, the number of modes of access can be many. A user or program may have the right to read, change, delete, or append to a value, add or delete entire fields or records, or reorganize the entire database.

Superficially, access control for a database seems like access control for operating systems or any other component of a computing system. However, the database problem is more complicated, as we see throughout this chapter. Operating system objects, such as files, are unrelated items, whereas records, fields, and elements are related. Although a user cannot determine the contents of one file by reading others, a user might be able to determine one data element just by reading others. The problem of obtaining data values from others is called inference, and we consider it in depth later in this chapter.

It is important to notice that you can access data by inference without needing direct access to the secure object itself. Restricting inference may mean prohibiting certain paths to prevent possible inferences. However, restricting access to control inference also limits queries from users who do not intend unauthorized access to values. Moreover, attempts to check requested accesses for possible unacceptable inferences may actually degrade the DBMS's performance.

Finally, size or granularity is different between operating system objects and database objects. An access control list of several hundred files is much easier to implement than an access control list for a database with several hundred files of perhaps a hundred fields each.

Size affects the efficiency of processing.

User Authentication

The DBMS can require rigorous user authentication. For example, a DBMS might insist that a user pass both specific password and time-of-day checks. This authentication supplements the authentication performed by the operating system. Typically, the DBMS runs as an application program on top of the operating system. This system design means that there is no trusted path from the DBMS to the operating system, so the DBMS must be suspicious of any data it receives, including user authentication. Thus, the DBMS is forced to do its own authentication.

Availability

A DBMS has aspects of both a program and a system. It is a program that uses other hardware and software resources, yet to many users it is the only application run. Users often take the DBMS for granted, employing it as an essential tool with which to perform particular tasks. But when the system is not available busy serving other users or down to be repaired or upgraded the users are very aware of a DBMS's unavailability. For example, two users may request the same record, and the DBMS must arbitrate; one user is bound to be denied access for a while. Or the DBMS may withhold unprotected data to avoid revealing protected data, leaving the requesting user unhappy. We examine these problems in more detail later in this chapter. Problems like these result in high availability requirements for a DBMS.

Integrity/Confidentiality/Availability

The three aspects of computer security integrity, confidentiality, and availability clearly relate to database management systems. As we have described, integrity applies to the individual elements of a database as well as to the database as a whole. Thus, integrity is a major concern in the design of database management systems. We look more closely at integrity issues in the next section.

Confidentiality is a key issue with databases because of the inference problem, whereby a user can access sensitive data indirectly. Inference and access control are covered later in this chapter.

Finally, availability is important because of the shared access motivation underlying database development. However, availability conflicts with confidentiality. The last sections of the chapter address availability in an environment in which confidentiality is also important.

Reliability and integrity

Databases amalgamate data from many sources, and users expect a DBMS to provide access to the data in a reliable way. When software engineers say that software has reliability, they mean that the software runs for very long periods of time without failing. Users certainly expect a DBMS to be reliable, since the data usually are key to business or organizational needs. Moreover, users entrust their data to a DBMS and rightly expect it to protect the data from loss or damage. Concerns for reliability and integrity are general security issues, but they are more apparent with databases.

A DBMS guards against loss or damage in several ways that we study them in this section. However, the controls we consider are not absolute: No control can prevent an authorized user from inadvertently entering an acceptable but incorrect value.

Database concerns about reliability and integrity can be viewed from three dimensions:

 Database integrity: concern that the database as a whole is protected against damage, as from the failure of a disk drive or the corruption of the master database index. These concerns are addressed by operating system integrity controls and recovery procedures.

 Element integrity: concern that the value of a specific data element is written or changed only by authorized users. Proper access controls protect a database from corruption by unauthorized users.

 Element accuracy: concern that only correct values are written into the elements of a database. Checks on the values of elements can help prevent insertion of improper values. Also, constraint conditions can detect incorrect values.

Protection Features from the Operating System

In Chapter 4 we discussed the protection an operating system provides for its users. A responsible system administrator backs up the files of a database periodically along with other user files. The files are protected during normal execution against outside access by the operating system's standard access control facilities. Finally, the operating system performs certain integrity checks for all data as a part of normal read and write operations for I/O devices. These controls provide basic security for databases, but the database manager must enhance them.

Two-Phase Update

A serious problem for a database manager is the failure of the computing system in the middle of modifying data. If the data item to be modified was a long field, half of the field might show the new value, while the other half would contain the old. Even if errors of this type were spotted easily (which they are not), a more subtle problem occurs when several fields are updated and no single field appears to be in obvious error. The solution to this problem, proposed first by Lampson and Sturgis [LAM76] and adopted by most DBMSs, uses a two-phase update.

Update Technique

During the first phase, called the intent phase, the DBMS gathers the resources it needs to perform the update. It may gather data, create dummy records, open files, lock out other users, and calculate final answers; in short, it does everything to prepare for the update, but it makes no changes to the database. The first phase is repeatable an unlimited number of times because it takes no permanent action. If the system fails during execution of the first phase, no harm is done because all these steps can be restarted and repeated after the system resumes processing.

The last event of the first phase, called committing, involves the writing of a commit flag to the database. The commit flag means that the DBMS has passed the point of no return:

After committing, the DBMS begins making permanent changes. The second phase makes the permanent changes. During the second phase, no actions from before the commit can be repeated, but the update activities of phase two can also be repeated as often as needed. If the system fails during the second phase, the database may contain incomplete data, but the system can repair these data by performing all activities of the second phase. After the second phase has been completed, the database is again complete.

Two-Phase Update Example

Suppose a database contains an inventory of a company's office supplies. The company's central stockroom stores paper, pens, paper clips, and the like, and the different departments requisition items as they need them. The company buys in bulk to obtain the best prices. Each department has a budget for office supplies, so there is a charging mechanism by which the cost of supplies is recovered from the department. Also, the central stockroom monitors quantities of supplies on hand so as to order new supplies when the stock becomes low.

Suppose the process begins with a requisition from the accounting department for 50 boxes of paper clips. Assume that there are 107 boxes in stock and a new order is placed if the quantity in stock ever falls below 100. Here are the steps followed after the stockroom receives the requisition.

1. The stockroom checks the database to determine that 50 boxes of paper clips are on hand. If not, the requisition is rejected and the transaction is finished.
2. If enough paper clips are in stock, the stockroom deducts 50 from the inventory figure in the database (107 - 50 = 57).
3. The stockroom charges accounting's supplies budget (also in the database) for 50 boxes of paper clips.
4. The stockroom checks its remaining quantity on hand (57) to determine whether the remaining quantity is below the reorder point. Because it is, a notice to order more paper clips is generated, and the item is flagged as "on order" in the database.
5. A delivery order is prepared, enabling 50 boxes of paper clips to be sent to accounting.

All five of these steps must be completed in the order listed for the database to be accurate and for the transaction to be processed correctly. Suppose a failure occurs while these steps are being processed. If the failure occurs before step 1 is complete, there is no harm because the entire transaction can be restarted.

However, during steps 2, 3, and 4, changes are made to elements in the database. If a failure occurs then, the values in the database are inconsistent. Worse, the transaction cannot be reprocessed because a requisition would be deducted twice, or a department would be charged twice, or two delivery orders would be prepared.

When a two-phase commit is used, shadow values are maintained for key data points. A shadow data value is computed and stored locally during the intent phase, and it is copied to the actual database during the commit phase. The operations on the database would be performed as follows for a two-phase commit.

Intent:

1. Check the value of COMMIT-FLAG in the database. If it is set, this phase cannot be performed. Halt or loop, checking COMMIT-FLAG until it is not set.
2. Compare number of boxes of paper clips on hand to number requisitioned; if more are requisitioned than are on hand, halt.
3. Compute TCLIPS = ONHAND - REQUISITION.
4. Obtain BUDGET, the current supplies budget remaining for accounting department. Compute TBUDGET = BUDGET - COST, where COST is the cost of 50 boxes of clips.
5. Check whether TCLIPS is below reorder point; if so, set TREORDER = TRUE; else set TREORDER = FALSE.

Commit:

1. Set COMMIT-FLAG in database.
2. Copy TCLIPS to CLIPS in database.
3. Copy TBUDGET to BUDGET in database.
4. Copy TREORDER to REORDER in database.
5. Prepare notice to deliver paper clips to accounting department. Indicate transaction completed in log.
6. Unset COMMIT-FLAG.

With this example, each step of the intent phase depends only on unmodified values from the database and the previous results of the intent phase. Each variable beginning with T is a shadow variable used only in this transaction. The steps of the intent phase can be repeated an unlimited number of times without affecting the integrity of the database.

Once the DBMS begins the commit phase, it writes a commit flag. When this flag is set, the DBMS will not perform any steps of the intent phase. Intent steps cannot be performed after committing because database values are modified in the commit phase. Notice, however, that the steps of the commit phase can be repeated an unlimited number of times, again with no negative effect on the correctness of the values in the database.

The one remaining flaw in this logic occurs if the system fails after writing the "transaction complete" message in the log but before clearing the commit flag in the database. It is a simple matter to work backward through the transaction log to find completed transactions for which the commit flag is still set and to clear those flags.

Redundancy/Internal Consistency

Many DBMSs maintain additional information to detect internal inconsistencies in data. The additional information ranges from a few check bits to duplicate or shadow fields, depending on the importance of the data.

Error Detection and Correction Codes

One form of redundancy is error detection and correction codes, such as parity bits, Hamming codes, and cyclic redundancy checks. These codes can be applied to single fields, records, or the entire database. Each time a data item is placed in the database, the appropriate check codes are computed and stored; each time a data item is retrieved, a similar check code is computed and compared to the stored value. If the values are unequal, they signify to the DBMS that an error has occurred in the database. Some of these codes point out the place of the error; others show precisely what the correct value should be. The more information provided, the more space required to store the codes.

Shadow Fields

Entire attributes or entire records can be duplicated in a database. If the data are irreproducible, this second copy can provide an immediate replacement if an error is detected.

Obviously, redundant fields require substantial storage space.

Recovery

In addition to these error correction processes, a DBMS can maintain a log of user accesses, particularly changes. In the event of a failure, the database is reloaded from a backup copy and all later changes are then applied from the audit log.

Concurrency/Consistency

Database systems are often multiuser systems. Accesses by two users sharing the same database must be constrained so that neither interferes with the other. Simple locking is done by the DBMS. If two users attempt to read the same data item, there is no conflict because both obtain the same value.

If both users try to modify the same data items, we often assume that there is no conflict because each knows what to write; the value to be written does not depend on the previous value of the data item. However, this supposition is not quite accurate.

To see how concurrent modification can get us into trouble, suppose that the database consists of seat reservations for a particular airline flight. Agent A, booking a seat for passenger Mock, submits a query to find which seats are still available. The agent knows that Mock prefers a right aisle seat, and the agent finds that seats 5D, 11D, and 14D are open. At the same time, Agent B is trying to book seats for a family of three traveling together. In response to a query, the database indicates that 8ABC and 11DEF are the two remaining groups of three adjacent unassigned seats. Agent A submits the update command

```
SELECT (SEAT-NO = '11D')
ASSIGN 'MOCK,E' TO PASSENGER-NAME
```

while Agent B submits the update sequence

```
SELECT (SEAT-NO = '11D')
ASSIGN 'EHLERS,P' TO PASSENGER-NAME
```

as well as commands for seats 11E and 11F. Then two passengers have been booked into the same seat (which would be uncomfortable, to say the least).

Both agents have acted properly: Each sought a list of empty seats, chose one seat from the list, and updated the database to show to whom the seat was assigned. The difficulty in this situation is the time delay between reading a value from the database and writing a modification of that value. During the delay time, another user has accessed the same data.

To resolve this problem, a DBMS treats the entire query update cycle as a single atomic operation. The command from the agent must now resemble "read the current value of seat PASSENGER-NAME for seat 11D; if it is 'UNASSIGNED', modify it to 'MOCK,E' (or 'EHLERS,P')."

The read modify cycle must be completed as an uninterrupted item without allowing any other users access to the PASSENGER-NAME field for seat 11D. The second agent's request to book would not be considered until after the first agent's had been completed; at that time, the value of PASSENGERNAME would no longer be 'UNASSIGNED'.

A final problem in concurrent access is readwrite. Suppose one user is updating a value when a second user wishes to read it. If the read is done while the write is in progress, the reader may receive data that are only partially updated. Consequently, the DBMS locks any read requests until a write has been completed.

Monitors

The monitor is the unit of a DBMS responsible for the structural integrity of the database. A monitor can check values being entered to ensure their consistency with the rest of the database or with characteristics of the particular field. For example, a monitor might reject alphabetic characters for a numeric field. We discuss several forms of monitors.

Range Comparisons

A range comparison monitor tests each new value to ensure that the value is within an acceptable range. If the data value is outside the range, it is rejected and not entered into the database. For example, the range of dates might be 131, "/," 112, "/," 19002099. An even more sophisticated range check might limit the day portion to 130 for months with 30 days, or it might take into account leap year for February.

Range comparisons are also convenient for numeric quantities. For example, a salary field might be limited to $200,000, or the size of a house might be constrained to be between 500 and 5,000 square feet. Range constraints can also apply to other data having a predictable form.

Range comparisons can be used to ensure the internal consistency of a database. When used in this manner, comparisons are made between two database elements. For example, a grade level from K8 would be acceptable if the record described a student at an elementary school, whereas only 912 would be acceptable for a record of a student in high school. Similarly, a person could be assigned a job qualification score of 75100 only if the person had completed college or had had at least ten years of work experience. Filters or patterns are more general types of data form checks. These can be used to verify that an automobile plate is two letters followed by four digits, or the sum of all digits of a credit card number is a multiple of 9.

Checks of these types can control the data allowed in the database. They can also be used to test existing values for reasonableness. If you suspect that the data in a database have been corrupted, a range check of all records could identify those having suspicious values.

State Constraints

State constraints describe the condition of the entire database. At no time should the database values violate these constraints. Phrased differently, if these constraints are not met, some value of the database is in error.

In the section on two-phase updates, we saw how to use a commit flag, which is set at the start of the commit phase and cleared at the completion of the commit phase. The commit flag can be considered a state constraint because it is used at the end of every transaction for which the commit flag is not set. Earlier in this chapter, we described a process to reset the commit flags in the event of a failure after a commit phase. In this way, the status of the commit flag is an integrity constraint on the database.

For another example of a state constraint, consider a database of employees' classifications. At any time, at most one employee is classified as "president." Furthermore, each employee has an employee number different from that of every other employee. If a mechanical or software failure causes portions of the database file to be duplicated, one of these uniqueness constraints might be violated. By testing the state of the database, the DBMS could identify records with duplicate employee numbers or two records classified as "president."

Transition Constraints

State constraints describe the state of a correct database. Transition constraints describe conditions necessary before changes can be applied to a database. For example, before a new employee can be added to the database, there must be a position number in the database with status "vacant." (That is, an empty slot must exist.) Furthermore, after the employee is added, exactly one slot must be changed from "vacant" to the number of the new employee.

Simple range checks and filters can be implemented within most database management systems. However, the more sophisticated state and transition constraints can require special procedures for testing. Such user-written procedures are invoked by the DBMS each time an action must be checked.

Sensitive data

Table 6-6. Sample Database.

Name	Sex	Race	Aid	Fines	Drugs	Dorm
Adams	M	C	5000	45.	1	Holmes
Bailey	M	B	0	0.	0	Grey
Chin	F	A	3000	20.	0	West
Dewitt	M	B	1000	35.	3	Grey
Earhart	F	C	2000	95.	1	Holmes
Fein	F	C	1000	15.	0	West
Groff	M	C	4000	0.	3	West
Hill	F	B	5000	10.	2	Holmes
Koch	F	C	0	0.	1	West
Liu	F	A	0	10.	2	Grey
Majors	M	C	2000	0.	2	Grey

Some databases contain what is called sensitive data. As a working definition, let us say that sensitive data are data that should not be made public. Determining which data items and fields are sensitive depends both on the individual database and the underlying meaning of the data. Obviously, some databases, such as a public library catalog, contain no sensitive data; other databases, such as defense-related ones, are totally sensitive. These two cases nothing sensitive and everything sensitive are the easiest to handle because they can be covered by access controls to the database as a whole. Someone either is or is not an authorized user.

These controls are provided by the operating system. The more difficult problem, which is also the more interesting one, is the case in which *some but not all* of the elements in the database are sensitive. There may be varying degrees of sensitivity. For example, a university database might contain student data consisting of name, financial aid, dorm, drug use, sex, parking fines, and race. An example of this database is shown in Table 6-6. Name and dorm are probably the least sensitive; financial aid, parking fines, and drug use the most; sex and race somewhere in between. That is, many people may have legitimate access to name, some to sex and race, and relatively few to financial aid, parking fines, or drug use. Indeed, knowledge of the existence of some fields, such as drug use, may itself be sensitive. Thus, security concerns not only the data elements but also their context and meaning.

Furthermore, we must take into account different degrees of sensitivity. For instance, although they are all highly sensitive, the financial aid, parking fines, and drug-use fields may not have the same kinds of access restrictions. Our security requirements may demand that a few people be authorized to see each field, but no one be authorized to see all three. The challenge of the access control problem is to limit users' access so that they can obtain only the data to which they have legitimate access. Alternatively, the access control problem forces us to ensure that sensitive data are not to be released to unauthorized people.

Several factors can make data sensitive.

 Inherently sensitive. The value itself may be so revealing that it is sensitive. Examples are the locations of defensive missiles or the median income of barbers in a town with only one barber.

 From a sensitive source. The source of the data may indicate a need for confidentiality. An example is information from an informer whose identity would be compromised if the information were disclosed.

 Declared sensitive. The database administrator or the owner of the data may have declared the data to be sensitive. Examples are classified military data or the name of the anonymous donor of a piece of art.

 Part of a sensitive *attribute* or a sensitive *record.* In a database, an entire attribute or record may be classified as sensitive. Examples are the salary attribute of a personnel database or a record describing a secret space mission.

 Sensitive *in relation to previously disclosed information.* Some data become sensitive in the presence of other data. For example, the longitude coordinate of a secret gold mine reveals little, but the longitude coordinate in conjunction with the latitude coordinate pinpoints the mine.

All of these factors must be considered to determine the sensitivity of the data.

Access Decisions

Remember that a database administrator is a *person* who decides *what* data should be in the database and *who* should have access to it. The database administrator considers the need for different users to know certain information and decides who should have what access.

Decisions of the database administrator are based on an access *policy*. The database manager or DBMS is a *program* that operates on the database and auxiliary control information to implement the decisions of the access policy. We say that the database manager decides to permit user x to access data y. Clearly, a program or machine cannot decide anything; it is more precise to say that the program performs the instructions by which x accesses y as a way of implementing the policy established by the database administrator. (Now you see why we use the simpler wording.) To keep explanations concise, we occasionally describe programs as if they can carry out human thought processes.

The DBMS may consider several factors when deciding whether to permit an access. These factors include availability of the data, acceptability of the access, and authenticity of the user. We expand on these three factors below.

Availability of Data

One or more required elements may be inaccessible. For example, if a user is updating several fields, other users' accesses to those fields must be blocked temporarily. This blocking ensures that users do not receive inaccurate information, such as a new street address with an old city and state, or a new code component with old documentation. Blocking is usually temporary. When performing an update, a user may have to block access to several fields or several records to ensure the consistency of data for others.

Notice, however, that if the updating user aborts the transaction while the update is in progress, the other users may be permanently blocked from accessing the record. This indefinite postponement is also a security problem, resulting in denial of service.

Acceptability of Access

One or more values of the record may be sensitive and not accessible by the general user. A DBMS should not release sensitive data to unauthorized individuals. Deciding what is sensitive, however, is not as simple as it sounds, because the fields may not be directly requested. A user may have asked for certain records that contain sensitive data, but the user's purpose may have been only to project the values from particular fields that are not sensitive. For example, a user of the database shown in Table 6-6 may request the NAME and DORM of any student for whom FINES is not 0. The exact value of the sensitive field FINES is not disclosed, although "not 0" is a partial disclosure. Even when a sensitive value is not explicitly given, the database manager may deny access on the grounds that it reveals information the user is not authorized to have.

Alternatively, the user may want to derive a nonsensitive statistic from the sensitive data; for example, if the average financial aid value does not reveal any individual's financial aid value, the database management system can safely return the average. However, the average of one data value discloses that value.

Assurance of Authenticity

Certain characteristics of the user external to the database may also be considered when permitting access. For example, to enhance security, the database administrator may permit someone to access the database only at certain times, such as during working hours. Previous user requests may also be taken into account; repeated requests for the same data or requests that exhaust a certain category of information may be used to find out all elements in a set when a direct query is not allowed. As we shall see, sensitive data can sometimes be revealed by combined results from several less sensitive queries.

Types of Disclosures

Data can be sensitive, but so can their characteristics. In this section, we see that even descriptive information about data (such as their existence or whether they have an element that is zero) is a form of disclosure.

Exact Data

The most serious disclosure is the *exact value of a sensitive data item* itself. The user may know that sensitive data are being requested, or the user may request general data without knowing that some of it is sensitive. A faulty database manager may even deliver sensitive data by accident, without the user's having requested it. In all of these cases the result is the same: The security of the sensitive data has been breached.

Bounds

Another exposure is disclosing bounds on a sensitive value; that is, indicating that a sensitive value, y, is between two values, L and H. Sometimes, by using a narrowing technique not unlike the binary search, the user may first determine that L y H and then see whether L y $H/2$, and so forth, thereby permitting the user to determine y to any desired precision.

In another case, merely revealing that a value such as the athletic scholarship budget or the number of CIA agents exceeds a certain amount may be a serious breach of security.

Sometimes, however, bounds are a useful way to present sensitive data. It is common to release upper and lower bounds for data without identifying the specific records. For example, a company may announce that its salaries for programmers range from $50,000 to $82,000. If you are a programmer earning $79,700, you can presume that you are fairly well off, so you have the information you want; however, the announcement does not disclose who are the highest- and lowest-paid programmers.

Negative Result

Sometimes we can word a query to determine a negative result. That is, we can learn that z is *not* the value of y. For example, knowing that 0 is not the total number of felony convictions for a person reveals that the person was convicted of a felony. The distinction between 1 and 2 or 46 and 47 felonies is not as sensitive as the distinction between 0 and 1.

Therefore, disclosing that a value is not 0 can be a significant disclosure. Similarly, if a student does not appear on the honors list, you can infer that the person's grade point average is below 3.50. This information is not too revealing, however, because the range of grade point averages from 0.0 to 3.49 is rather wide.

Existence

In some cases, the existence of data is itself a sensitive piece of data, regardless of the actual value. For example, an employer may not want employees to know that their use of long distance telephone lines is being monitored. In this case, discovering a LONG DISTANCE field in a personnel file would reveal sensitive data.

Probable Value

Finally, it may be possible to determine the probability that a certain element has a certain value. To see how, suppose you want to find out whether the president of the United States is registered in the Tory party. Knowing that the president is in the database, you submit two queries to the database:

How many people have 1600 Pennsylvania Avenue as their official residence? (Response: 4) How many people have 1600 Pennsylvania Avenue as their official residence and have YES as the value of TORY? (Response: 1) From these queries you conclude there is a 25 percent likelihood that the president is a registered Tory.

Summary of Partial Disclosure

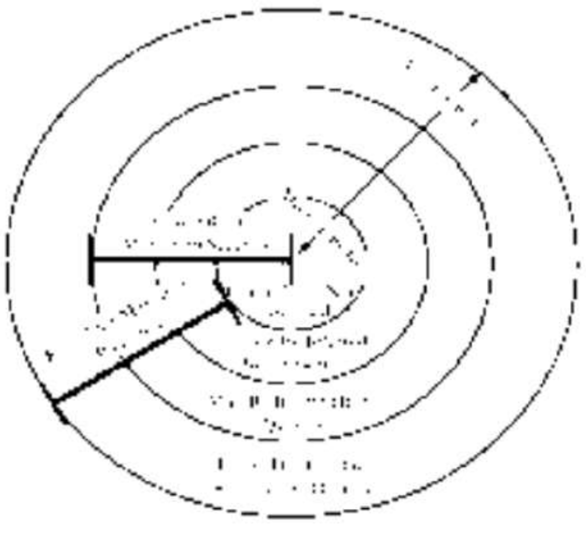
We have seen several examples of how a security problem can result if characteristics of sensitive data are revealed. Notice that some of the techniques we presented used information *about* the data, rather than direct access to the data, to infer sensitive results. A successful security strategy must protect from both direct and indirect disclosure.

Security versus Precision

Our examples have illustrated how difficult it is to determine which data are sensitive and how to protect them. The situation is complicated by a desire to share nonsensitive data. For reasons of confidentiality we want to disclose only those data that are not sensitive. Such an outlook encourages a conservative philosophy in determining what data to disclose: less is better than more.

On the other hand, consider the users of the data. The conservative philosophy suggests rejecting any query that mentions a sensitive field. We may thereby reject many reasonable and nondisclosing queries. For example, a

Figure 6-3. Security versus Precision.



researcher may want a list of grades for all students
using drugs, or a statistician may request lists of salaries for all men and for all women. These queries probably do not compromise the identity of any individual. We want to disclose as much data as possible so that users of the database have access to the data they need. This goal, called precision, aims to protect all sensitive data while revealing as much nonsensitive data as possible.

We can depict the relationship between security and precision with concentric circles. As Figure 6-3 shows, the sensitive data in the central circle should be carefully concealed. The outside band represents data we willingly disclose in response to queries. But we know that the user may put together pieces of disclosed data and infer other, more deeply hidden, data.

The figure shows us that beneath the outer layer may be yet more nonsensitive data that the user cannot infer.

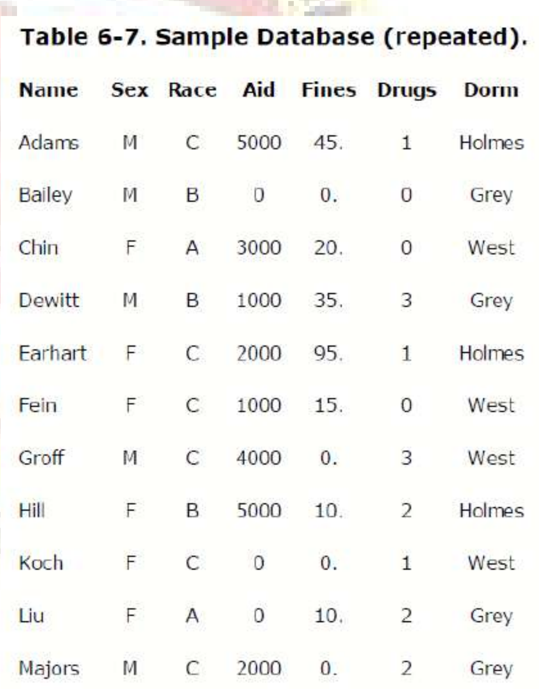
The ideal combination of security and precision allows us to maintain perfect confidentiality with maximum precision; in other words, we disclose all and only the nonsensitive data. But achieving this goal is not as easy as it might seem, as we show in the next section. Sidebar 6-3 gives an example of using imprecise techniques to improve accuracy. In the next section, we consider ways in which sensitive data can be obtained from queries that appear harmless.

Inference

Inference is a way to infer or derive sensitive data from nonsensitive data. The inference problem is a subtle vulnerability in database security. The database in Table 6-7 can help illustrate the inference problem. Recall that AID is the amount of financial aid a student is receiving. FINES is the amount of parking fines still owed.

DRUGS is the result of a drug-use survey: 0 means never used and 3 means frequent user. Obviously this information should be kept confidential. We assume that AID, FINES, and DRUGS are sensitive fields, although only when the values are related to a specific individual. In this section, we look at ways to determine sensitive data values from the database.

Table 6-7. Sample Database (repeated).



Name	Sex	Race	Aid	Fines	Drugs	Dorm
Adams	M	C	5000	45.	1	Holmes
Bailey	M	B	0	0.	0	Grey
Chin	F	A	3000	20.	0	West
Dewitt	M	B	1000	35.	3	Grey
Earhart	F	C	2000	95.	1	Holmes
Fein	F	C	1000	15.	0	West
Groff	M	C	4000	0.	3	West
Hill	F	B	5000	10.	2	Holmes
Koch	F	C	0	0.	1	West
Liu	F	A	0	10.	2	Grey
Majors	M	C	2000	0.	2	Grey

Direct Attack

In a direct attack, a user tries to determine values of sensitive fields by seeking them directly with queries that yield few records. The most successful technique is to form a query so specific that it matches exactly one data item. In Table 6-7, a sensitive query might be List NAME where SEX=M DRUGS=1

This query discloses that for record ADAMS, DRUGS=1. However, it is an obvious attack because it selects people for whom DRUGS=1, and the DBMS might reject the query because it selects records for a specific value of the sensitive attribute DRUGS.

A less obvious query is
List NAME where (SEX=M && DRUGS=1) | |
(SEX!= M && SEX!= F) | | (DORM=AYRES)

On the surface, this query looks as if it should conceal drug usage by selecting other non-drug-related records as well. However, this query still retrieves only one record, revealing a name that corresponds to the sensitive DRUG value. The DBMS needs to know that SEX has only two possible values so that the second clause will select no records. Even if that were possible, the DBMS would also need to know that no records exist with DORM=AYRES, even though AYRES might in fact be an acceptable value for DORM.

Organizations that publish personal statistical data, such as the U.S. Census Bureau, do not reveal results when a small number of people make up a large proportion of a category. The rule of "n items over k percent" means that data should be withheld if n items represent over k percent of the result reported. In the previous case, the one person selected represents 100 percent of the data reported, so there would be no ambiguity about which person matches the query.

Indirect Attack

Another procedure, used by the U.S. Census Bureau and other organizations that gather sensitive data, is to release only statistics. The organizations suppress individual names, addresses, or other characteristics by which a single individual can be recognized. Only neutral statistics, such as sum, count, and mean, are released.

The indirect attack seeks to infer a final result based on one or more intermediate statistical results. But this approach requires work outside the database itself. In particular, a statistical attack seeks to use some apparently anonymous statistical measure to infer individual data. In the following sections, we present several examples of indirect attacks on databases that report statistics.

Sum

An attack by sum tries to infer a value from a reported sum. For example, with the sample database in Table 6-7, it might seem safe to report student aid total by sex and dorm. Such a report is shown in Table 6-8. This seemingly innocent report reveals that no female living in Grey is receiving financial aid. Thus, we can infer that any female living in Grey (such as Liu) is certainly not receiving financial aid. This approach often allows us to determine a negative result.

Table 6-8. Sums of Financial Aid by Dorm and Sex.

	Holmes	Grey	West	Total
M	5000	3000	4000	12000
F	7000	0	4000	11000
Total	12000	3000	8000	23000

Count

The count can be combined with the sum to produce some even more revealing results. Often these two statistics are released for a database to allow users to determine average values. (Conversely, if count and mean are released, sum can be deduced.)

Table 6-9 shows the count of records for students by dorm and sex. This table is innocuous by itself. Combined with the sum table, however, this table demonstrates that the two males in Holmes and West are receiving financial aid in the amount of $5000 and $4000, respectively. We can obtain the names by selecting the subschema of NAME, DORM, which is not sensitive because it delivers only low-security data on the entire database.

Table 6-9. Count of Students by Dorm and Sex.

	Holmes	Grey	West	Total
M	1	3	1	5
F	2	1	3	6
Total	3	4	4	11

Mean

The arithmetic mean (average) allows exact disclosure if the attacker can manipulate the subject population. As a trivial example, consider salary. Given the number of employees, the mean salary for a company and the mean salary of all employees except the president, it is easy to compute the president's salary.

Median

By a slightly more complicated process, we can determine an individual value from medians. The attack requires finding selections having one point of intersection that happens to be exactly in the middle, as shown in Figure 6-4.

For example, in our sample database, there are five males and three persons whose drug use value is 2. Arranged in order of aid, these lists are

Table 6-10. Inference from Median of Two Lists.

Name	Sex	Drugs	Aid
Bailey	M	0	0
Dewitt	M	3	1000
Majors	M	2	2000
Groff	M	3	4000
Adams	M	1	5000
Liu	F	2	0
Majors	M	2	2000
Hill	F	2	5000

shown in Table 6-10. Notice that Majors is the only name common to both lists, and conveniently that name is in the middle of each list.

Someone working at the Health Clinic might be able to find out that Majors is a white male whose drug-use score is 2. That information identifies Majors as the intersection of these two lists and pinpoints Majors' financial aid as $2000. In this example, the queries
q = median(AID where SEX = M)
p = median(AID where DRUGS = 2)
reveal the exact financial aid amount for Majors.

Tracker Attacks

As already explained, database management systems may conceal data when a small number of entries make up a large proportion of the data revealed. A tracker attack can fool the database manager into locating the desired data by using additional queries that produce small results. The tracker adds additional records to be retrieved for two different queries; the two sets of records cancel each other out, leaving only the statistic or data desired. The approach is to use intelligent padding of two queries. In other words, instead of trying to identify a unique value, we request n - 1 other values (where there are n values in the database). Given n and n - 1, we can easily compute the desired single element.

For instance, suppose we wish to know how many female Caucasians live in Holmes Hall. A query posed might be

```
count ((SEX=F) ∧ (RACE=C) ∧ (DORM=Holmes))
```

The database management system might consult the database, find that the answer is 1, and refuse to answer that query because one record dominates the result of the query.

However, further analysis of the query allows us to track sensitive data through nonsensitive queries.

The query

```
q=count((SEX=F) ∧ (RACE=C) ∧ (DORM=Holmes))
```

is of the form

```
q = count(a ∧ b ∧ c)
```

By using the rules of logic and algebra, we can transform this query to

```
q = count(a ∧ b ∧ c) = count(a)  count(a ∧ ¬ (b ∧ c))
```

Thus, the original query is equivalent to

```
count (SEX=F)
```

minus

```
count ((SEX=F) ∧ ((RACE≠C) ∨ (DORM≠Holmes)))
```

Because count(a) = 6 and count(a ∧ ¬ (b ∧ c)) = 5, we can determine the suppressed value easily: 6 - 5 = 1. Furthermore, neither 6 nor 5 is a sensitive count.

Linear System Vulnerability

A tracker is a specific case of a more general vulnerability. With a little logic, algebra, and luck in the distribution of the database contents, it may be possible to construct a series of queries that returns results relating to several different sets. For example, the following system of five queries does not overtly reveal any single c value from the database. However, the queries' equations can be solved for each of the unknown c values, revealing them all.

$$\begin{aligned}
q_1 &= c_1 + c_2 + c_3 + c_4 + c_5 \\
q_2 &= c_1 + c_2 + c_4 \\
q_3 &= c_3 + c_4 \\
q_4 &= c_4 + c_5 \\
q_5 &= c_2 + c_5
\end{aligned}$$

To see how, use basic algebra to note that $q_1 - q_2 = c_3 + c_5$, and $q_3 - q_4 = c_3 - c_5$. Then, subtracting these two equations, we obtain $c_5 = ((q_1 - q_2) - (q_3 - q_4))/2$. Once we know c_5, we can derive the others.

In fact, this attack can also be used to obtain results *other than* numerical ones. Recall that we can apply logical rules to *and* ($\wedge$) and *or* ($\vee$), typical operators for database queries, to derive values from a series of logical expressions. For example, each expression might represent a query asking for precise data instead of counts, such as the equation

$$q = s_1 \vee s_2 \vee s_3 \vee s_4 \vee s_5$$

The result of the query is a set of records. Using logic and set algebra in a manner similar to our numerical example, we can carefully determine the actual values for each of the s_i.

Controls for Statistical Inference Attacks

Denning and Schlörer [DEN83a] present a very good survey of techniques for maintaining security in databases. The controls for all statistical attacks are similar. Essentially, there are two ways to protect against inference attacks: Either controls are applied to the queries or controls are applied to individual items within the database. As we have seen, it is difficult to determine whether a given query discloses sensitive data. Thus, query controls are effective primarily against direct attacks.

Suppression and concealing are two controls applied to data items. With suppression, sensitive data values are not provided; the query is rejected without response. With concealing, the answer provided is *close to* but not exactly the actual value.

These two controls reflect the contrast between security and precision. With suppression, any results provided are correct, yet many responses must be withheld to maintain security.

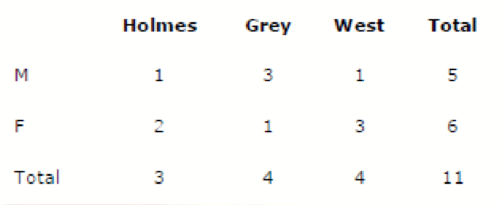
With concealing, more results can be provided, but the precision of the results is lower. The choice between suppression and concealing depends on the context of the database.

Examples of suppression and concealing follow.

Limited Response Suppression

The *n*-item *k*-percent rule eliminates certain low-frequency elements from being displayed. It is not sufficient to delete them, however, if their values can also be inferred. To see why, consider Table 6-11, which shows counts of students by dorm and sex.

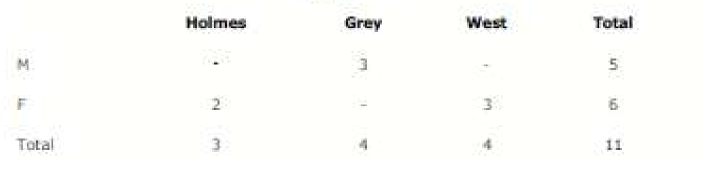
Table 6-11. Students by Dorm and Sex.



	Holmes	Grey	West	Total
M	1	3	1	5
F	2	1	3	6
Total	3	4	4	11

The data in this table suggest that the cells with counts of 1 should be suppressed; their counts are too revealing. But it does no good to suppress the MaleHoldmes cell when the value 1 can be determined by subtracting FemaleHolmes (2) from the total (3) to determine 1, as shown in Table 6-12.

Table 6-12. Students by Dorm and Sex, with Low Count Suppression.



	Holmes	Grey	West	Total
M	-	3	-	5
F	2	-	3	6
Total	3	4	4	11

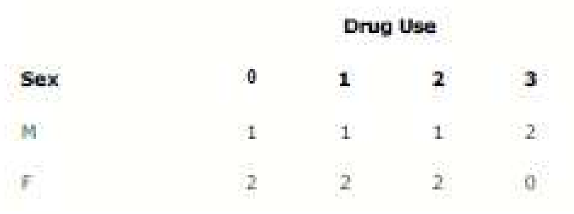
When one cell is suppressed in a table with totals for rows and columns, it is necessary to suppress at least one additional cell on the row and one on the column to provide some confusion. Using this logic, all cells (except totals) would have to be suppressed in this small sample table. When totals are not provided, single cells in a row or column can be suppressed.

Combined Results

Another control combines rows or columns to protect sensitive values. For example, Table 6-13 shows several sensitive results that identify single individuals. (Even though these counts may not seem sensitive, they can be used to infer sensitive data such as NAME; therefore, we consider them to be sensitive.)

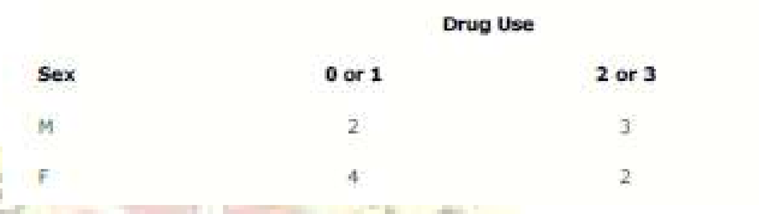
These counts, combined with other results such as sum, permit us to infer individual drug-use values for the three males, as well as to infer that no female was rated 3 for drug use. To suppress such sensitive information, it is possible to combine the attribute values for 0 and 1, and also for 2 and 3, producing the less sensitive results shown in Table 6-14. In this instance, it is impossible to identify any single value.

Table 6-13. Students by Sex and Drug Use.



	Drug Use			
Sex	0	1	2	3
M	1	1	1	2
F	2	2	2	0

Table 6-14. Suppression by Combining Revealing Values.



	Drug Use	
Sex	0 or 1	2 or 3
M	2	3
F	4	2

Another way of combining results is to present values in ranges. For example, instead of releasing exact financial aid figures, results can be released for the ranges $01999, $20003999, and $4000 and above. Even if only one record is represented by a single result, the exact value of that record is not known. Similarly, the highest and lowest financial aid values are concealed.

Yet another method of combining is by rounding. This technique is actually a fairly well-known example of combining by range. If numbers are rounded to the nearest multiple of 10, the effective ranges are 05, 615, 1625, and so on. Actual values are rounded up or down to the nearest multiple of some base.

Random Sample

With random sample control, a result is not derived from the whole database; instead the result is computed on a random sample of the database. The sample chosen is large enough to be valid. Because the sample is not the whole database, a query against this sample will not necessarily match the result for the whole database. Thus, a result of 5 percent for a particular query means that 5 percent of the records chosen for the sample for this query had the desired property. You would expect that approximately 5 percent of the entire database will have the property in question, but the actual percentage may be quite different.

So that averaging attacks from repeated, equivalent queries are prevented, the same sample set should be chosen for equivalent queries. In this way, all equivalent queries will produce the same result, although that result will be only an approximation for the entire database.

Random Data Perturbation

It is sometimes useful to perturb the values of the database by a small error. For each x_i that is the true value of data item i in the database, we can generate a small random error term ε_i and add it to x_i for statistical results. The ε values are both positive and negative, so that some reported values will be slightly higher than their true values and other reported values will be lower. Statistical measures such as sum and mean will be close but not necessarily exact. Data perturbation is easier to use than random sample selection because it is easier to store all the ε values in order to produce the same result for equivalent queries.

Query Analysis

A more complex form of security uses query analysis. Here, a query and its implications are analyzed to determine whether a result should be provided. As noted earlier, query analysis can be quite difficult. One approach involves maintaining a query history for each user and judging a query in the context of what inferences are possible given previous results.

Conclusion on the Inference Problem

There are no perfect solutions to the inference problem. The approaches to controlling it follow the three paths listed below. The first two methods can be used either to limit queries accepted or to limit data provided in response to a query. The last method applies only to data released.

 Suppress obviously sensitive information. This action can be taken fairly easily. The tendency is to err on the side of suppression, thereby restricting the usefulness of the database.
 Track what the user knows. Although possibly leading to the greatest safe disclosure, this approach is extremely costly. Information must be maintained on all users, even though most are not trying to obtain sensitive data. Moreover, this approach seldom takes into account what any two people may know together and cannot address what a single user can accomplish by using multiple IDs.
 Disguise the data. Random perturbation and rounding can inhibit statistical attacks that depend on exact values for logical and algebraic manipulation. The users of the database receive slightly incorrect or possibly inconsistent results.

It is unlikely that research will reveal a simple, easy-to-apply measure that determines exactly which data can be revealed without compromising sensitive data.

Nevertheless, an effective control for the inference problem is just knowing that it exists. As with other problems in security, recognition of the problem leads to understanding of the purposes of controlling the problem and to sensitivity to the potential difficulties caused by the problem. However, just knowing of possible database attacks does not necessarily mean people will protect against those attacks, as explained in Sidebar 6-4. It is also noteworthy that much of the research on database inference was done in the early 1980s, but this proposal appeared almost two decades later.

Aggregation

Related to the inference problem is aggregation, which means building sensitive results from less sensitive inputs. We saw earlier that knowing either the latitude or longitude of a gold mine does you no good. But if you know both latitude and longitude, you can pinpoint the mine. For a more realistic example, consider how police use aggregation frequently in solving crimes: They determine who had a motive for committing the crime, when the crime was committed, who had alibis covering that time, who had the skills, and so forth. Typically, you think of police investigation as starting with the entire population and narrowing the analysis to a single person. But if the police officers work in parallel, one may have a list of possible suspects, another may have a list with possible motive, and another may have a list of capable persons. When the intersection of these lists is a single person, the police have their prime suspect.

Addressing the aggregation problem is difficult because it requires the database management system to track which results each user had already received and conceal any result that would let the user derive a more sensitive result. Aggregation is especially difficult to counter because it can take place outside the system. For example, suppose the security policy is that anyone can have *either* the latitude or longitude of the mine, but not both. Nothing prevents you from getting one, your friend from getting the other, and the two of you talking to each other.

Recent interest in data mining has raised concern again about aggregation. Data mining is the process of sifting through multiple databases and correlating multiple data elements to find useful information. Marketing companies use data mining extensively to find consumers likely to buy a product. As Sidebar 6-5 points out, it is not only marketers who are interested in aggregation through data mining.

Aggregation was of interest to database security researchers at the same time as was inference. As we have seen, some approaches to inference have proven useful and are currently being used. But there have been few proposals for countering aggregation.

Multilevel database

So far, we have considered data in only two categories: either sensitive or nonsensitive. We have alluded to some data items being more sensitive than others, but we have allowed only yes-or-no access. Our presentation may have implied that sensitivity was a function of the *attribute,* the column in which the data appeared, although nothing we have done depended on this interpretation of sensitivity. Such a model appears in Table 6-15, where two columns are identified (by shading) as sensitive. In fact, though, sensitivity is determined not just by attribute but also in ways that we investigate in the next section.

The Case for Differentiated Security

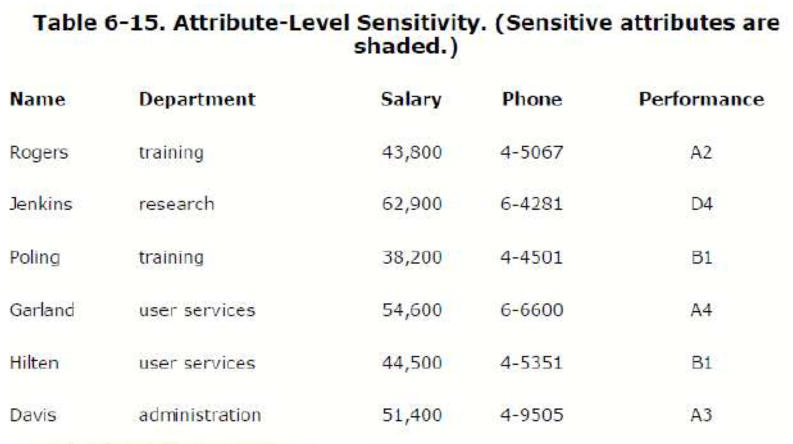


Table 6-15. Attribute-Level Sensitivity. (Sensitive attributes are shaded.)

Name	Department	Salary	Phone	Performance
Rogers	training	43,800	4-5067	A2
Jenkins	research	62,900	6-4281	D4
Poling	training	38,200	4-4501	B1
Garland	user services	54,600	6-6600	A4
Hilten	user services	44,500	4-5351	B1
Davis	administration	51,400	4-9505	A3

Consider a database containing data on U.S. government expenditures. Some of the expenditures are for paper clips, which is not sensitive information. Some salary expenditures are subject to privacy requirements. Individual salaries are sensitive, but the aggregate (for example, the total Agriculture Department payroll, which is a matter of public record) is not sensitive. Expenses of certain military operations are more sensitive; for example, the total amount the United States spends for ballistic missiles, which is not public. There are even operations known only to a few people, and so the amount spent on these operations, or even the fact that anything was spent on such an operation, is highly sensitive.

Table 6-15 lists employee information. It may in fact be the case that Davis is a temporary employee hired for a special project, and her whole record has a different sensitivity from the others. Perhaps the phone shown for Garland is her private line, not available to the public.

We can refine the sensitivity of the data by depicting it as shown in Table 6-16.

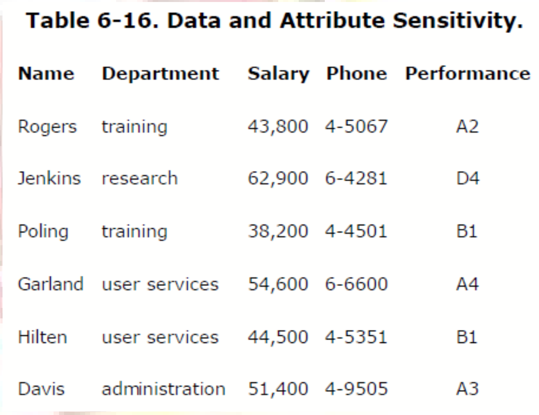


Table 6-16. Data and Attribute Sensitivity.

Name	Department	Salary	Phone	Performance
Rogers	training	43,800	4-5067	A2
Jenkins	research	62,900	6-4281	D4
Poling	training	38,200	4-4501	B1
Garland	user services	54,600	6-6600	A4
Hilten	user services	44,500	4-5351	B1
Davis	administration	51,400	4-9505	A3

From this description, three characteristics of database security emerge.

 The security of a single element may be different from the security of other elements of the same record or from other values of the same attribute. That is, the security of one element may differ from that of other elements of the same row or column. This situation implies that security should be implemented for each individual element.

 Two levelssensitive and nonsensitiveare inadequate to represent some security situations. Several grades of security may be needed. These grades may represent ranges of allowable knowledge, which may overlap. Typically, the security grades form a lattice.

 The security of an aggregate a sum, a count, or a group of values in a database may differ from the security of the individual elements. The security of the aggregate may be higher or lower than that of the individual elements.

These three principles lead to a model of security not unlike the military model of security encountered in Chapter 5, in which the sensitivity of an object is defined as one of *n* levels and is further separated into compartments by category.

Granularity

Recall that the military classification model applied originally to paper documents and was adapted to computers. It is fairly easy to classify and track a single sheet of paper or, for that matter, a paper file, a computer file, or a single program or process. It is entirely different to classify individual data items.

For obvious reasons, an entire sheet of paper is classified at one level, even though certain words, such as *and*, *the*, or *of*, would be innocuous in any context, and other words, such as code words like *Manhattan project*, might be sensitive in any context. But defining the sensitivity of each value in a database is similar to applying a sensitivity level to each individual word of a document.

And the problem is still more complicated. The word *Manhattan* by itself is not sensitive, nor is *project*. However, the combination of these words produces the sensitive codeword *Manhattan project*. A similar situation occurs in databases. Therefore, not only can every *element* of a database have a distinct sensitivity, every *combination of elements* can also have a distinct sensitivity. Furthermore, the combination can be more or less sensitive than any of its elements.

So what would we need in order to associate a sensitivity level with each value of a database? First, we need an access control policy to dictate which users may have access to what data. Typically, to implement this policy each data item is marked to show its access limitations. Second, we need a means to guarantee that the value has not been changed by an unauthorized person. These two requirements address both confidentiality and integrity.

Security Issues

In Chapter 1, we introduced three general security concerns: integrity, confidentiality, and availability. In this section, we extend the first two of these concepts to include their special roles for multilevel databases.

Integrity

Even in a single-level database in which all elements have the same degree of sensitivity, integrity is a tricky problem. In the case of multilevel databases, integrity becomes both more important and more difficult to achieve. Because of the *-property for access control, a process that reads high-level data is not allowed to write a file at a lower level. Applied to databases, however, this principle says that a high-level user should not be able to write a lower-level data element.

The problem with this interpretation arises when the DBMS must be able to read all records in the database and write new records for any of the following purposes: to do backups, to scan the database to answer queries, to reorganize the database according to a user's processing needs, or to update all records of the database.

When people encounter this problem, they handle it by using trust and common sense. People who have access to sensitive information are careful not to convey it to uncleared individuals.

In a computing system, there are two choices: Either the process cleared at a high level cannot write to a lower level or the process must be a "trusted process," the computer equivalent of a person with a security clearance.

Confidentiality

Users trust that a database will provide correct information, meaning that the data are consistent and accurate. As indicated earlier, some means of protecting confidentiality may result in small changes to the data. Although these perturbations should not affect statistical analyses, they may produce two different answers representing the same underlying data value in response to two differently formed queries. In the multilevel case, two different users operating at two different levels of security might get two different answers to the same query. To preserve confidentiality, precision is sacrificed.

Enforcing confidentiality also leads to unknowing redundancy. Suppose a personnel specialist works at one level of access permission. The specialist knows that Bob Hill works for the company. However, Bob's record does not appear on the retirement payment roster. The specialist assumes this omission is an error and creates a record for Bob.

The reason that no record for Bob appears is that Bob is a secret agent, and his employment with the company is not supposed to be public knowledge. A record on Bob actually is in the file but, because of his special position, his record is not accessible to the personnel specialist. The DBMS cannot reject the record from the personnel specialist because doing so would reveal that there already is such a record at a sensitivity too high for the specialist to see. The creation of the new record means that there are now two records for Bob Hill: one sensitive and one not, as shown in Table 6-17. This situation is called polyinstantiation, meaning that one record can appear (be instantiated) many times, with a different level of confidentiality each time.

Table 6-17. Polyinstantiated Records.

Name	Sensitivity	Assignment	Location
Hill, Bob	C	Program Mgr	London
Hill, Bob	TS	Secret Agent	South Bend

This problem is exacerbated because Bob Hill is a common enough name that there might be two different people in the database with that name. Thus, merely scanning the database (from a high-sensitivity level) for duplicate names is not a satisfactory way to find records entered unknowingly by people with only low clearances.

We might also find other reasons, unrelated to sensitivity level, that result in polyinstantiation. For example, Mark Thyme worked for Acme Corporation for 30 years and retired. He is now drawing a pension from Acme, so he appears as a retiree in one personnel record. But Mark tires of being home and is rehired as a part-time contractor; this new work generates a second personnel record for Mark. Each is a legitimate employment record. In our zeal to reduce polyinstantiation, we must be careful not to eliminate legitimate records such as these.

Proposals for multilevel security.

As you can already tell, implementing multilevel security for databases is difficult, probably more so than in operating systems, because of the small granularity of the items being controlled. In the remainder of this section, we study approaches to multilevel security for databases.

Separation

As we have already seen, separation is necessary to limit access. In this section, we study mechanisms to implement separation in databases. Then, we see how these mechanisms can help to implement multilevel security for databases.

Partitioning

The obvious control for multilevel databases is partitioning. The database is divided into separate databases, each at its own level of sensitivity. This approach is similar to maintaining separate files in separate file cabinets.

This control destroys a basic advantage of databases: elimination of redundancy and improved accuracy through having only one field to update. Furthermore, it does not address the problem of a high-level user who needs access to some low-level data combined with high-level data.

Nevertheless, because of the difficulty of establishing, maintaining, and using multilevel databases, many users with data of mixed sensitivities handle their data by using separate, isolated databases.

Encryption

If sensitive data are encrypted, a user who accidentally receives them cannot interpret the data. Thus, each level of sensitive data can be stored in a table encrypted under a key unique to the level of sensitivity. But encryption has certain disadvantages.

First, a user can mount a chosen plaintext attack. Suppose party affiliation of REP or DEM is stored in encrypted form in each record. A user who achieves access to these encrypted fields can easily decrypt them by creating a new record with party=DEM and comparing the resulting encrypted version to that element in all other records. Worse, if authentication data are encrypted, the malicious user can substitute the encrypted form of his or her own data for that of any other user. Not only does this provide access for the malicious user, but it also excludes the legitimate user whose authentication data have been changed to that of the malicious user. These possibilities are shown in Figures 6-5 and 6-6.

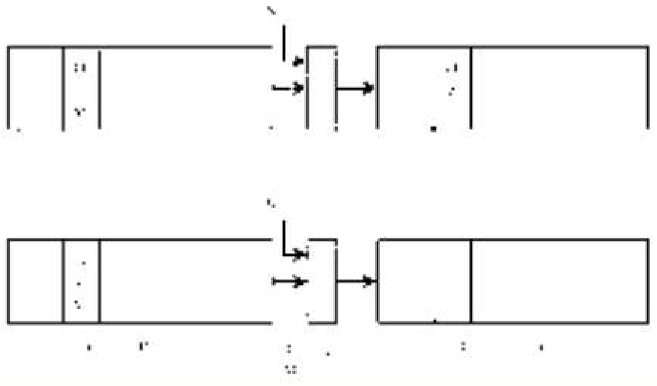


Figure 6-6. Cryptographic Separation: Block Chaining.

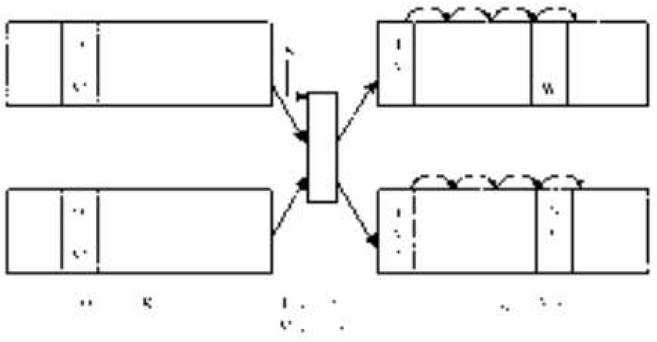


Figure 6-5. Cryptographic Separation: Different Encryption Keys.

Using a different encryption key for each record overcomes these defects. Each record's fields can be encrypted with a different key, or all fields of a record can be cryptographically linked, as with cipher block chaining.

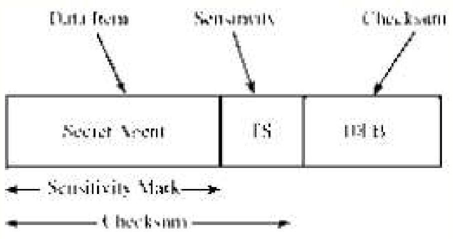
The disadvantage, then, is that each field must be decrypted when users perform standard database operations such as "select all records with SALARY > 10,000." Decrypting the SALARY field, even on rejected records, increases the time to process a query. (Consider the query that selects just one record but that must decrypt and compare one field of each record to find the one that satisfies the query.) Thus, encryption is not often used to implement separation in databases.

Integrity Lock

The integrity lock was first proposed at the U.S. Air Force Summer Study on Data Base Security [AFS83]. The lock is a way to provide both integrity and limited access for a database. The operation was nicknamed "spray paint" because each element is figuratively painted with a color that denotes its sensitivity. The coloring is maintained with the element, not in a master database table.

A model of the basic integrity lock is shown in Figure 6-7. As illustrated, each apparent data item consists of three pieces: the actual data item itself, a sensitivity label, and a checksum.

Figure 6-7. Integrity Lock.



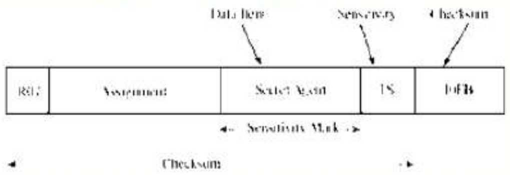
The sensitivity label defines the sensitivity of the data, and the checksum is computed across both data and sensitivity label to prevent unauthorized modification of the data item or its label. The actual data item is stored in plaintext, for efficiency because the DBMS may need to examine many fields when selecting records to match a query.

The sensitivity label should be

- *unforgeable,* so that a malicious subject cannot create a new sensitivity level for an element
- *unique,* so that a malicious subject cannot copy a sensitivity level from another element
- *concealed,* so that a malicious subject cannot even determine the sensitivity level of an arbitrary element

The third piece of the integrity lock for a field is an error-detecting code, called a cryptographic checksum. To guarantee that a data value or its sensitivity classification has not been changed, this checksum must be unique for a given element, and must contain both the element's data value and something to tie that value to a particular position in the database. As shown in Figure 6-8, an appropriate cryptographic checksum includes something unique to the record (the record number), something unique to this data field within the record (the field attribute name), the value of this element, and the sensitivity classification of the element. These four components guard against anyone's changing, copying, or moving the data. The checksum can be computed with a strong encryption algorithm or hash function.

Figure 6-8. Cryptographic Checksum.



Sensitivity Lock

The sensitivity lock shown in Figure 6-9 was designed by Graubert and Kramer [GRA84b] to meet these principles. A sensitivity lock is a combination of a unique identifier (such as the record number) and the sensitivity level. Because the identifier is unique, each lock relates to one particular record. Many different elements will have the same sensitivity level. A malicious subject should not be able to identify two elements having identical sensitivity levels or identical data values just by looking at the sensitivity level portion of the lock. Because of the encryption, the lock's contents, especially the sensitivity level, are concealed from plain view.

Thus, the lock is associated with one specific record, and it protects the secrecy of the sensitivity level of that record.

Designs of Multilevel Secure Databases

This section covers different designs for multilevel secure databases. These designs show the tradeoffs among efficiency, flexibility, simplicity, and trustworthiness.

Integrity Lock

The integrity lock DBMS was invented as a short-term solution to the security problem for multilevel databases. The intention was to be able to use any (untrusted) database manager with a trusted procedure that handles access control. The sensitive data were obliterated or concealed with encryption that protected both a data item and its sensitivity. In this way, only the access procedure would need to be trusted because only it would be able to achieve or grant access to sensitive data.

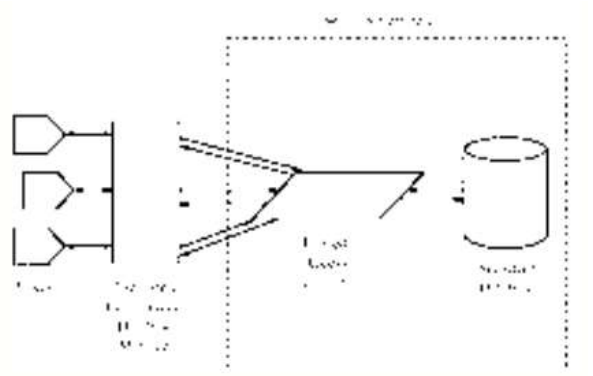


Figure 6-10. Trusted Database Manager.

The structure of such a system is shown in Figure 6-10.element must be expanded to contain the sensitivity label. Because there are several pieces in the label and one label for every element, the space required is significant.

Problematic, too, is the processing time efficiency of an integrity lock. The sensitivity label must be decoded every time a data element is passed to the user to verify that the user's access is allowable. Also, each time a value is written or modified, the label must be recomputed. Thus, substantial processing time is consumed. If the database file can be sufficiently protected, the data values of the individual elements can be left in plaintext. That approach benefits select and project queries across sensitive fields because an element need not be decrypted just to determine whether it should be selected.

A final difficulty with this approach is that the untrusted database manager sees all data, so it is subject to Trojan horse attacks by which data can be leaked through covert channels.

Trusted Front End

The model of a trusted front-end process is shown in Figure 6-11. A trusted front end is also known as a guard and operates much like the reference monitor of Chapter 5. This approach, originated by Hinke and Schaefer [HIN75], recognizes that many DBMSs have been built and put into use without consideration of multilevel security. Staff members are already trained in using these DBMSs, and they may in fact use them frequently. The front-end concept takes advantage of existing tools and expertise, enhancing the security of these existing systems with minimal change to the system. The interaction between a user, a trusted front end, and a DBMS involves the following steps.

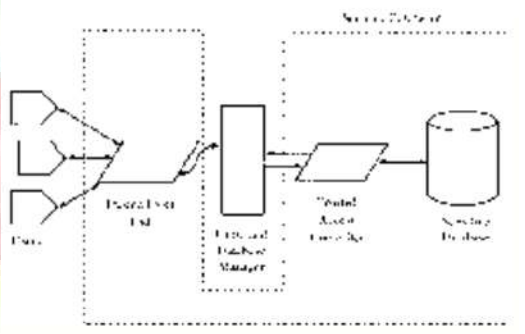


Figure 6-11. Trusted Front End.

1. A user identifies himself or herself to the front end; the front end authenticates the user's identity.
2. The user issues a query to the front end.
3. The front end verifies the user's authorization to data.
4. The front end issues a query to the database manager.
5. The database manager performs I/O access, interacting with low-level access control to achieve access to actual data.
6. The database manager returns the result of the query to the trusted front end.
7. The front end analyzes the sensitivity levels of the data items in the result and selects those items consistent with the user's security level.
8. The front end transmits selected data to the untrusted front end for formatting.

9. The untrusted front end transmits formatted data to the user.

The trusted front end serves as a one-way filter, screening out results the user should not be able to access. But the scheme is inefficient because potentially much data is retrieved and then discarded as inappropriate for the user.

Commutative Filters

The notion of a commutative filter was proposed by Denning [DEN85] as a simplification of the trusted interface to the DBMS. Essentially, the filter screens the user's request, reformatting it if necessary, so that only data of an appropriate sensitivity level are returned to the user.

A commutative filter is a process that forms an interface between the user and a DBMS.

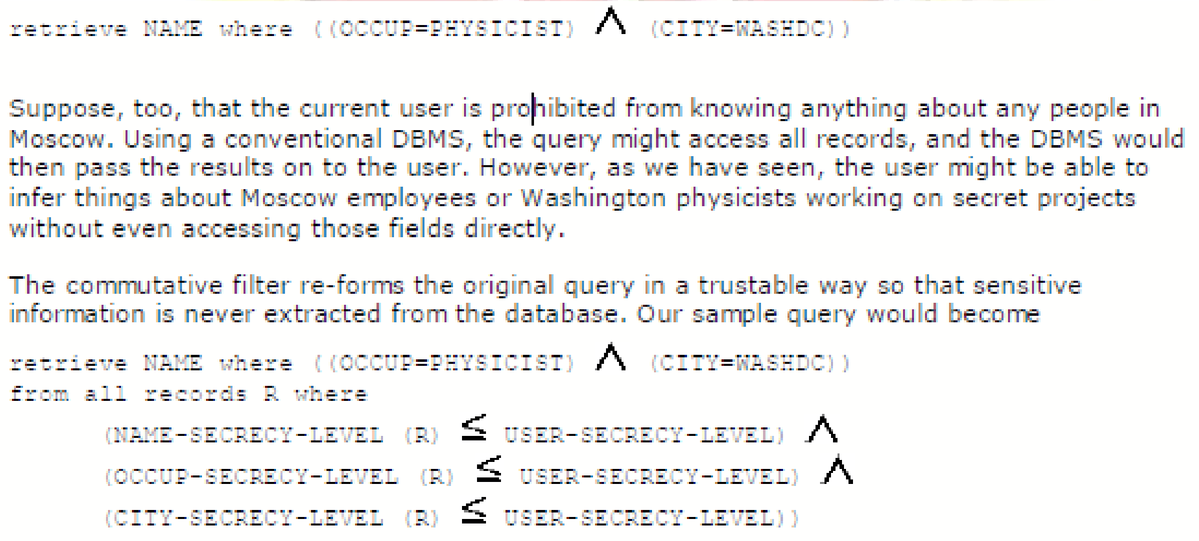
However, unlike the trusted front end, the filter tries to capitalize on the efficiency of most DBMSs. The filter reformats the query so that the database manager does as much of the work as possible, screening out many unacceptable records. The filter then provides a second screening to select only data to which the user has access.

Filters can be used for security at the record, attribute, or element level.

- When used at the record level, the filter requests desired data plus cryptographic checksum information; it then verifies the accuracy and accessibility of data to be passed to the user.
- At the attribute level, the filter checks whether all attributes in the user's query are accessible to the user and, if so, passes the query to the database manager. On return, it deletes all fields to which the user has no access rights.
- At the element level, the system requests desired data plus cryptographic checksum information. When these are returned, it checks the classification level of every element of every record retrieved against the user's level.

Suppose a group of physicists in Washington works on very sensitive projects, so the current user should not be allowed to access the physicists' names in the database. This restriction presents a problem with this query:

The filter works by restricting the query to the DBMS and then restricting the results before they are returned to the user. In this instance, the filter would request NAME, NAME-SECRECY-LEVEL, OCCUP, OCCUP-SECRECY-LEVEL, CITY, and CITY-SECRECY-LEVEL values and would then filter and return to the user only those fields and items that are of a secrecy level acceptable for the user. Although even this simple query becomes complicated because of the added terms, these terms are all added by the front-end filter,



```
retrieve NAME where ((OCCUP=PHYSICIST) ∧ (CITY=WASHDC))
```

Suppose, too, that the current user is prohibited from knowing anything about any people in Moscow. Using a conventional DBMS, the query might access all records, and the DBMS would then pass the results on to the user. However, as we have seen, the user might be able to infer things about Moscow employees or Washington physicists working on secret projects without even accessing those fields directly.

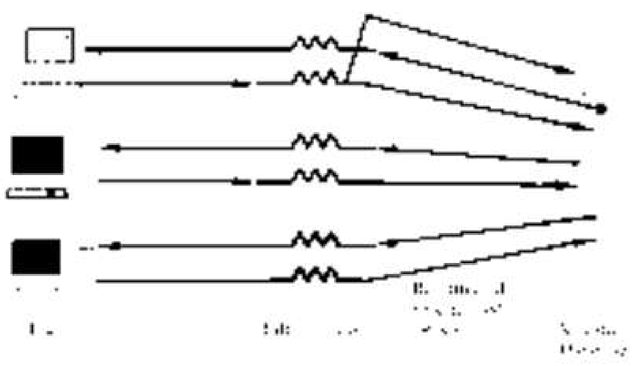
The commutative filter re-forms the original query in a trustable way so that sensitive information is never extracted from the database. Our sample query would become

```
retrieve NAME where ((OCCUP=PHYSICIST) ∧ (CITY=WASHDC))
from all records R where
      (NAME-SECRECY-LEVEL (R) ≤ USER-SECRECY-LEVEL) ∧
      (OCCUP-SECRECY-LEVEL (R) ≤ USER-SECRECY-LEVEL) ∧
      (CITY-SECRECY-LEVEL (R) ≤ USER-SECRECY-LEVEL))
```

invisible to the user.

An example of this query filtering in operation is shown in Figure 6-12. The advantage of the commutative filter is that it allows query selection, some optimization, and some subquery handling to be done by the DBMS. This delegation of duties keeps the size of the security filter small, reduces redundancy between it and the DBMS, and improves the overall efficiency of the system.

Figure 6-12. Commutative Filters.



or

Distributed Databases The distributed federated database is a fourth design for a secure multilevel database.

In this case, a trusted front end controls access to two unmodified commercial DBMSs: one for all low-sensitivity data and one for all high-sensitivity data.

The front end takes a user's query and formulates single-level queries to the databases as appropriate. For a user cleared for high-sensitivity data, the front end submits queries to both the high- and low-sensitivity databases. But if the user is not cleared for high-sensitivity data, the front end submits a query to only the low-sensitivity database. If the result is obtained from either back-end database alone, the front end passes the result back to the user. If the result comes from both databases, the front end has to combine the results appropriately. For example, if the query is a join query having some high-sensitivity terms and some low, the front end has to perform the equivalent of a database join itself.

The distributed database design is not popular because the front end, which must be trusted is complex, Potentially including most of the functionality of a full DBMS itself. In addition, the design does not scale well to many degrees of sensitivity; each sensitivity level of data must be maintained in its own separate database.

Window/View

Traditionally, one of the advantages of using a DBMS for multiple users of different interests (but not necessarily different sensitivity levels) is the ability to create a different view for each user. That is, each user is restricted to a picture of the data reflecting only what the user needs to see. For example, the registrar may see only the class assignments and grades of each student at a university, not needing to see extracurricular activities or medical records. The university health clinic, on the other hand, needs medical records and drug-use information but not scores on standardized academic tests.

The notion of a window or a view can also be an organizing principle for multilevel database access. A window is a subset of a database, containing exactly the information that a user is entitled to access. Denning [DEN87a] surveys the development of views for multilevel database security.

A view can represent a single user's subset database so that all of a user's queries access only that database. This subset guarantees that the user does not access values outside the permitted ones, because nonpermitted values are not even in the user's database. The view is specified as a set of relations in the database, so the data in the view subset change as data change in the database.

For example, a travel agent might have access to part of an airline's flight information database. Records for cargo flights

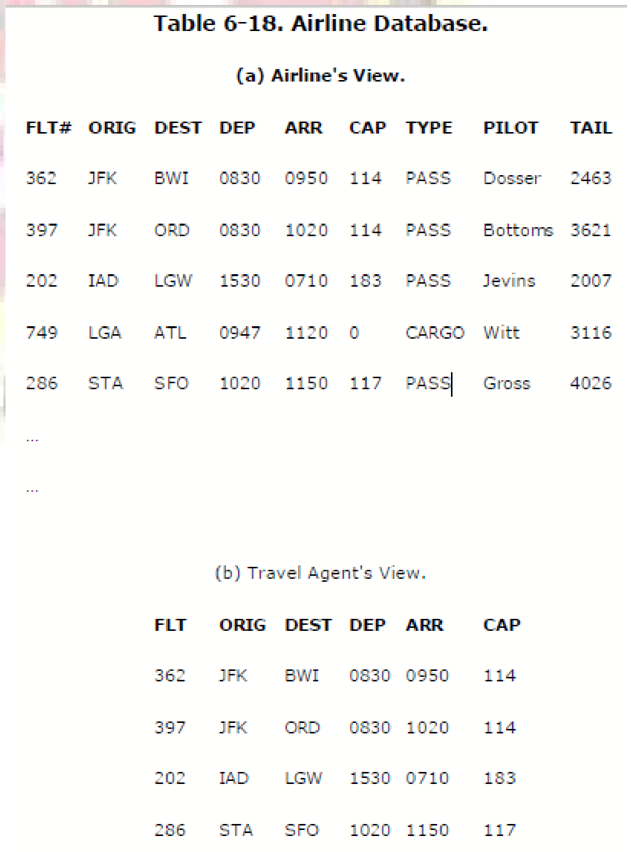


Table 6-18. Airline Database.

(a) Airline's View.

FLT#	ORIG	DEST	DEP	ARR	CAP	TYPE	PILOT	TAIL
362	JFK	BWI	0830	0950	114	PASS	Dosser	2463
397	JFK	ORD	0830	1020	114	PASS	Bottoms	3621
202	IAD	LGW	1530	0710	183	PASS	Jevins	2007
749	LGA	ATL	0947	1120	0	CARGO	Witt	3116
286	STA	SFO	1020	1150	117	PASS	Gross	4026
...								
...								

(b) Travel Agent's View.

FLT	ORIG	DEST	DEP	ARR	CAP
362	JFK	BWI	0830	0950	114
397	JFK	ORD	0830	1020	114
202	IAD	LGW	1530	0710	183
286	STA	SFO	1020	1150	117

would be excluded, as would the pilot's name and the serial number of the plane for every flight. Suppose the database contained an attribute TYPE whose value was either CARGO or PASS (for passenger). Other attributes might be flight number, origin, destination, departure time, arrival time, capacity, pilot, and tail number.

Now suppose the airline created some passenger flights with lower fares that could be booked only directly through the airline. The airline might assign their flight numbers a more sensitive rating to make these flights unavailable to travel agents. The whole database, and the agent's view, might have the logical structure shown in Table 6-18.

The travel agent's view of the database is expressed as

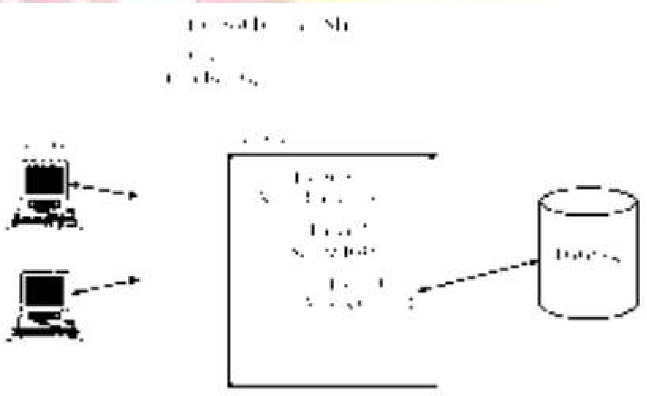
```
view AGENT-INFO
FLTNO:=MASTER.FLTNO
ORIG:=MASTER.ORIG
DEST:=MASTER.DEST
DEP:=MASTER.DEP
ARR:=MASTER.ARR
CAP:=MASTER.CAP
where MASTER.TYPE='PASS'
class AGENT
auth retrieve
```

Because the access class of this view is AGENT, more sensitive flight numbers (flights booked only through the airline) do not appear in this view. Alternatively, we could have eliminated the entire records for those flights by restricting the record selection with a *where* clause. A view may involve computation or complex selection criteria to specify subset data.

The data presented to a user is obtained by filtering of the contents of the original database. Attributes, records, and elements are stripped away so that the user sees only acceptable items. Any attribute (column) is withheld unless the user is authorized to access at least one element. Any record (row) is withheld unless the user is authorized to access at least one element. Then, for all elements that still remain, if the user is not authorized to access the element, it is replaced by UNDEFINED. This last step does not compromise any data because the user knows the existence of the attribute (there is at least one element that the user can access) and the user knows the existence of the record (again, at least one accessible element exists in the record).

In addition to elements, a view includes relations on attributes. Furthermore, a user can create new relations from new and existing attributes and elements. These new relations are accessible to other users, subject to the standard access rights. A user can operate on the subset database defined in a view only as allowed by the operations authorized in the view.

As an example, a user might be allowed to retrieve records specified in one view or to retrieve and update records as specified in another view. For instance, the airline in our example may restrict travel agents to retrieving data.



The Sea Views project described in [DEN87a, LUN90a] is the basis for a system that integrates a trusted operating system to form a trusted database manager. The layered implementation as described is shown in Figure 6-13. The lowest layer, the reference monitor, performs file interaction, enforcing the BellLa Padula access controls, and does user authentication. Part of its function is to filter data passed to higher levels. The second level performs basic indexing and computation functions of the database. The third level translates views into the base relations of the database. These three layers make up the trusted computing base (TCB) of the system. The remaining layers implement normal DBMS functions and the user interface.

Figure 6-13. Secure Database Decomposition.

This layered approach makes views both a logical division of a database and a functional one. The approach is an important step toward the design and implementation of a trustable database management system.

Practical Issues

The multilevel security problem for databases has been studied since the 1970s. Several promising research results have been identified, as we have seen in this chapter. However, as with trusted operating systems, the consumer demand has not been sufficient to support many products. Civilian users have not liked the inflexibility of the military multilevel security model, and there have been too few military users. Consequently, multilevel secure databases are primarily of research and historical interest.

The general concepts of multilevel databases are important. We do need to be able to separate data according to their degree of sensitivity. Similarly, we need ways of combining data of different sensitivities into one database (or at least into one virtual database or federation of databases). And these needs will only increase over time as larger databases contain more sensitive information, especially for privacy concerns.

In the next section we study data mining, a technique of growing significance, but one for which we need to be able to address degrees of sensitivity of the data.

Unit 4 : Security in Networks:

Networkstheir design, development, and usages are critical to our style of computing. We interact with networks daily, when we perform banking transactions, make telephone calls, or ride trains and planes. The utility companies use networks to track electricity or water usage and bill for it. When we pay for groceries or gasoline, networks enable our credit or debit card transactions and billing. Life without networks would be considerably less convenient, and many activities would be impossible. Not surprisingly, then, computing networks are attackers' targets of choice. Because of their actual and potential impact, network attacks attract the attention of journalists, managers, auditors, and the general public. For example, when you read the daily newspapers, you are likely to find a story about a network-based attack at least every month. The coverage itself evokes a sense of evil, using terms such as hijacking, distributed denial of service, and our familiar friends viruses, worms, and Trojan horses.

Because any large-scale attack is likely to put thousands of computing systems at risk, with potential losses well into the millions of dollars, network attacks make good copy. The media coverage is more than hype; network attacks are critical problems. Fortunately, your bank, your utility company, and even your Internet service provider take network security very seriously. Because they do, they are vigilant about applying the most current and most effective controls to their systems. Of equal importance, these organizations continually assess their risks and learn about the latest attack types and defense mechanisms so that they can maintain the protection of their networks.

In this chapter we describe what makes a network similar to and different from an application program or an operating system, which you have studied in earlier chapters. In investigating networks, you will learn how the concepts of confidentiality, integrity, and availability apply in networked settings. At the same time, you will see that the basic notions of identification and authentication, access control, accountability, and assurance are the basis for network security, just as they have been in other settings.

Networking is growing and changing perhaps even faster than other computing disciplines Consequently, this chapter is unlikely to present you with the most current technology, the latest attack, or the newest defense mechanism; you can read about those in daily newspapers and at web sites. But the novelty and change build on what we know today: the fundamental concepts, threats, and controls for networks. By developing an understanding of the basics, you can absorb the most current news quickly and easily. More importantly, your understanding can assist you in building, protecting, and using networks.

Threats in networks

Up to now, we have reviewed network concepts with very little discussion of their security implications. But our earlier discussion of threats and vulnerabilities, as well as outside articles and your own experiences, probably have you thinking about the many possible attacks against networks. This section describes some of the threats you have already hypothesized and perhaps presents you with some new ones. But the general thrust is the same: threats aimed to compromise confidentiality, integrity, or availability, applied against data, software, and hardware by nature, accidents, nonmalicious humans, and malicious attackers.

What Makes a Network Vulnerable?

An isolated home user or a stand-alone office with a few employees is an unlikely target for many attacks. But add a network to the mix and the risk rises sharply. Consider how a network differs from a stand-alone environment:

 Anonymity. An attacker can mount an attack from thousands of miles away and never come into direct contact with the system, its administrators, or users. The potential attacker is thus safe behind an electronic shield. The attack can be passed through many other hosts in an effort to disguise the attack's origin. And computer-to-computer authentication is not the same for computers as it is for humans; as illustrated by Sidebar 7-2, secure distributed authentication requires thought and attention to detail.

 Many points of attackboth targets and origins. A simple computing system is a self-contained unit. Access controls on one machine preserve the confidentiality of data on that processor. However, when a file is stored in a network host remote from the user, the data or the file itself may pass through many hosts to get to the user.
One host's administrator may enforce rigorous security policies, but that administrator has no control over other hosts in the network. Thus, the user must depend on the access control mechanisms in each of these systems. An attack can come from any host to any host, so that a large network offers many points of vulnerability.

 Sharing. Because networks enable resource and workload sharing, more users have the potential to access networked systems than on single computers. Perhaps worse, access is afforded to *more systems,* so that access controls for single systems may be inadequate in networks.

 Complexity of system. In Chapter 4 we saw that an operating system is a complicated piece of software. Reliable security is difficult, if not impossible, on a large operating system, especially one not designed specifically for security. A network combines two or more possibly dissimilar operating systems. Therefore, a network operating/control system is likely to be more complex than an operating system for a single computing system. Furthermore, the ordinary desktop computer today has greater computing power than did many office computers in the last two decades. The attacker can use this power to advantage by causing the victim's computer to perform part of the attack's computation. And because an average computer is so powerful, most users do not know what their computers are really doing at any moment: What processes are active in the background while you are playing *Invaders from Mars*? This complexity diminishes confidence in the network's security.

 Unknown perimeter. A network's expandability also implies uncertainty about the network boundary. One host may be a node on two different networks, so resources on one network are accessible to the users of the other network as well. Although wide accessibility is an advantage, this unknown or uncontrolled group of possibly malicious users is a security disadvantage. A similar problem occurs when new hosts can be added to the network. Every network node must be able to react to the possible presence of new, untrustable hosts. Figure 7-11 points out the problems in defining the boundaries of a network. Notice, for example, that a user on a host in network D may be unaware of the potential connections from users of networks A and B. And the host in the middle of networks A and B in fact belongs to A, B, C, and E. If there are different security rules for these networks, to what rules is that host subject?

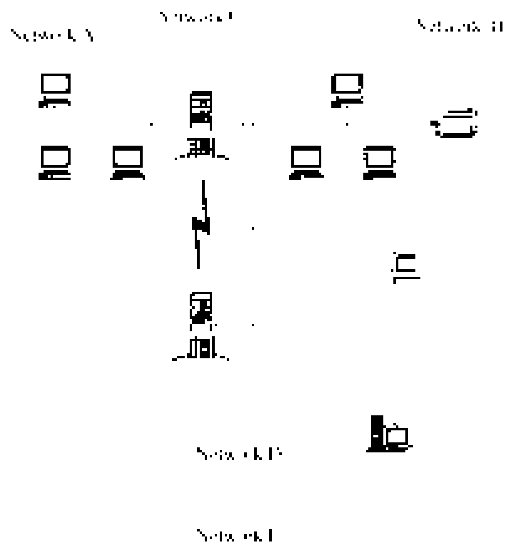
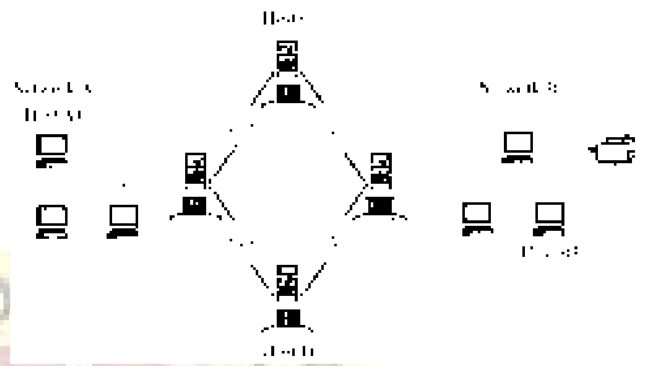


Figure 7-11. Unclear Network Boundaries.

 Unknown path. Figure 7-12 illustrates that there may be many paths from one host to another. Suppose that a user on host A1 wants to send a message to a user on host B3. That message might be routed through hosts C or D before arriving at host B3.
Host C may provide acceptable security, but not D. Network users seldom have control over the routing of their messages.



Thus, a network differs significantly from a stand-alone, local environment. Network characteristics significantly increase the security risk.

Who Attacks Networks?

Who are the attackers? We cannot list their names, just as we cannot know who are all the criminals in our city, country, or the world. Even if we knew who they were, we do not know if we could stop their behavior. (See Sidebar 7-3 for a first, tenuous link between psychological traits and hacking.) To have some idea of who the attackers might be, we return to concepts introduced in Chapter 1, where we described the three necessary components of an attack: method, opportunity, and motive.
In the next sections we explore method: tools and techniques the attackers use. Here we consider first the motives of attackers. Focusing on motive may give us some idea of who might attack a networked host or user. Four important motives are challenge or power, fame, money, and ideology.

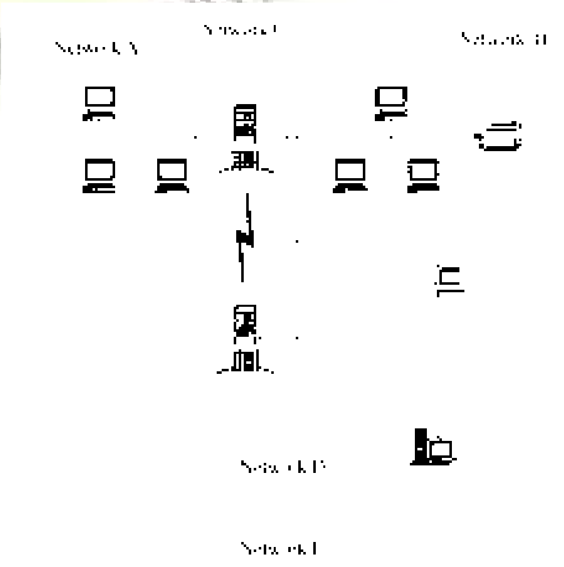
Challenge

Why do people do dangerous or daunting things, like climb mountains or swim the English Channel or engage in extreme sports? Because of the challenge. The situation is no different for someone skilled in writing or using programs. The single most significant motivation for a network attacker is the intellectual challenge. He or she is intrigued with knowing the answers to Can I defeat this network? What would happen if I tried this approach or that technique?

Some attackers enjoy the intellectual stimulation of defeating the supposedly undefeatable.

For example, Robert Morris, who perpetrated the Internet worm in 1988 (described in Chapter 3), attacked supposedly as an experiment to see if he could exploit a particular vulnerability.

Other attackers, such as the Cult of the Dead Cow, seek to demonstrate weaknesses in security defenses so that others will pay attention to strengthening security. Still other attackers are unnamed, unknown individuals working persistently just to see how far they can go in performing unwelcome activities.
However, as you will soon see, only a few attackers find previously unknown flaws. The vast majority of attackers repeat well-known and even well-documented attacks, sometimes only to see if they work against different hosts. In these cases, intellectual stimulation is certainly not the driving force, when the attacker is merely pressing [run] to activate an attack discovered, designed, and implemented by someone else.



Fame

The challenge of accomplishment is enough for some attackers. But other attackers seek recognition for their activities. That is, part of the challenge is doing the deed; another part is taking credit for it. In many cases, we do not know who the attackers really are, but they leave behind a "calling card" with a name or moniker: Mafiaboy, Kevin Mitnick, Fluffy Bunny, and members of the Chaos Computer Club, for example. The actors often retain some anonymity by using pseudonyms, but they achieve fame nevertheless. They may not be able to brag too openly, but they enjoy the personal thrill of seeing their attacks written up in the news media.

Money and Espionage

As in other settings, financial reward motivates attackers, too. Some attackers perform industrial espionage, seeking information on a company's products, clients, or long-range plans. We know industrial espionage has a role when we read about laptops and sensitive papers having been lifted from hotel rooms when other more valuable items were left behind.

Some countries are notorious for using espionage to aid their state-run industries. Sometimes industrial espionage is responsible for seemingly strange corporate behavior. For example, in July 2002, newspapers reported that a Yale University security audit had revealed that admissions officers from rival Princeton University broke into Yale's online admissions notification system. The Princeton snoops admitted looking at the confidential decisions about eleven students who had applied to both schools but who had not yet been told of their decisions by Yale. In another case, a startup company was about to activate its first application on the web. Two days before the application's unveiling, the head offices were burglarized. The only item stolen was the one computer containing the application's network design. Corporate officials had to make a difficult choice: Go online knowing that a competitor might then take advantage of knowing the internal architecture or delay the product's rollout until the network design was changed. They chose the latter. Similarly, the chief of security for a major manufacturing company has reported privately to us of evidence that one of the company's competitors had stolen information. But he could take no action because he could not determine which of three competitors was the actual culprit. Industrial espionage is illegal, but it occurs, in part because of the high potential gain. Its existence and consequences can be embarrassing for the target companies. Thus, many incidents go unreported, and there are few reliable statistics on how much industrial espionage and "dirty tricks" go on. Yearly since 1997, the Computer Security Institute and the U.S. Federal Bureau of Investigation have surveyed security professionals from companies, government agencies, universities, and organizations, asking them to report perceptions of computer incidents. About 500 responses are received for each survey. Theft of intellectual property amounted to a total loss of $31 million, with an average loss per incident of $350 thousand, making this the category of third-highest loss. That amount was more than double the amount reported in the 2004 survey. (These survey results are anecdotal, so it is hard to draw many conclusions. For full details on the survey see [CSI05].) Industrial espionage, leading to loss of intellectual property, is clearly a problem.

Organized Crime

With the growth in commercial value of the Internet, participation by organized crime has also increased. In October 2004, police arrested members of a 28-person gang of Internet criminals, called the Shadow crew, who operated out of six foreign countries and eight states in the United States. Six leaders of that group pled guilty to charges, closing an illicit business that trafficked in at least 1.5 million stolen credit and bank card numbers and resulted in losses in excess of $4 million. In July 2003, Alexey Ivanov was convicted as the supervisor of a wide-ranging, organized criminal enterprise that engaged in sophisticated manipulation of computer data, financial information, and credit card numbers. Ivanov and group were responsible for an aggregate loss of approximately $25 million. And in January 2006, Jeanson James Ancheta pled guilty to having infected 400,000 computers with malicious code and renting their use to others to use to launch attacks on others. In June 2005, the FBI and law enforcement from 10 other countries conducted over 90 searches worldwide as part of "Operation Site Down," designed to disrupt and dismantle many of the leading criminal organizations that illegally distribute and trade in copyrighted software, movies, music, and games on the Internet [DOJ06]. Brazilian police arrested 85 people in 2005 for Internet fraud.

Although money is common to these crimes, the more interesting fact is that they often involve collaborators from several countries. These more sophisticated attacks require more than one person working out of a bedroom, and so organization and individual responsibilities follow. With potential revenue in the millions of dollars and operations involving hundreds of thousands of credit card numbers and other pieces of identity, existing organized crime units are sure to take notice. As Williams [WIL01] says, "[T]here is growing evidence that organized crime groups are exploiting the new opportunities offered by the Internet."

Ideology

In the last few years, we are starting to find cases in which attacks are perpetrated to advance ideological ends. For example, many security analysts believe that the Code Red worm of 2001 was launched by a group motivated by the tension in U.S.China relations. Denning [DEN99a] has distinguished between two types of related behaviors, hactivism and cyberterrorism. Hactivism involves "operations that use hacking techniques against a target's [network] with the intent of disrupting normal operations but not causing serious damage." In some cases, the hacking is seen as giving voice to a constituency that might otherwise not be heard by the company or government organization. For example, Denning describes activities such as virtual sit-ins, in which an interest group floods an organization's web site with traffic to demonstrate support of a particular position. Cyberterrorism is more dangerous than hactivism: "politically motivated hacking operations intended to cause grave harm such as loss of life or severe economic damage."

Security and terrorism experts are seeing increasing use of the Internet as an attack vector, as a communications medium among attackers, and as a point of attack. Cullison [CUL04] presents a very interesting insight (which we overview in Sidebar 1-6, p. 24) into of the use of technology by al Qaeda.

Reconnaissance

Now that we have listed many motives for attacking, we turn to how attackers perpetrate their attacks. Attackers do not ordinarily sit down at a terminal and launch an attack. A clever attacker investigates and plans before acting. Just as you might invest time in learning about a jewelry store before entering to steal from it, a network attacker learns a lot about a potential target before beginning the attack. We study the precursors to an attack so that if we can recognize characteristic behavior, we may be able to block the attack before it is launched.

Because most vulnerable networks are connected to the Internet, the attacker begins preparation by finding out as much as possible about the target. An example of information gathering is given in [HOB97]. (Not all information gathered is accurate, however; see Sidebar 7-4 for a look at reconnaissance combined with deception.)

Port Scan

An easy way to gather network information is to use a port scan, a program that, for a particular IP address, reports which ports respond to messages and which of several known vulnerabilities seem to be present. Farmer and Venema [FAR93] are among the first to describe the technique.

A port scan is much like a routine physical examination from a doctor, particularly the initial questions used to determine a medical history. The questions and answers by themselves may not seem significant, but they point to areas that suggest further investigation.

Port scanning tells an attacker three things: which standard ports or services are running and responding on the target system, what operating system is installed on the target system, and what applications and versions of applications are present. This information is readily available for the asking from a networked system; it can be obtained quietly, anonymously, without identification or authentication, drawing little or no attention to the scan.

Port scanning tools are readily available, and not just to the underground community. The nmap scanner by Fyodor at *www.insecure.org/nmap* is a useful tool that anyone can download. Given an address, nmap will report all open ports, the service they support, and the owner (user ID) of the daemon providing the service. (The owner is significant because it implies what privileges would descend upon someone who compromised that service.) Another readily available scanner is netcat, written by Hobbit, at *www.l0pht.com/users/l0pht*. (That URL is "letter ell," "digit zero," p-h-t.) Commercial products are a little more costly, but not prohibitive. Well-known commercial scanners are Nessus (Nessus Corp. [AND03]), CyberCop Scanner (Network Associates), Secure Scanner (Cisco), and Internet Scanner (Internet Security Systems).

Social Engineering

The port scan gives an external picture of a networkwhere are the doors and windows, of what are they constructed, to what kinds of rooms do they open? The attacker also wants to know what is inside the building. What better way to find out than to ask? Suppose, while sitting at your workstation, you receive a phone call. "Hello, this is John Davis from IT support. We need to test some connections on the internal network. Could you please run the command ipconfig/all on your workstation and read to me the addresses it displays?"

The request sounds innocuous. But unless you know John Davis and his job responsibilities well, the caller could be an attacker gathering information on the inside architecture.

Social engineering involves using social skills and personal interaction to get someone to reveal security-relevant information and perhaps even to do something that permits an attack. The point of social engineering is to persuade the victim to be helpful. The attacker often impersonates someone inside the organization who is in a bind: "My laptop has just been stolen and I need to change the password I had stored on it," or "I have to get out a very important report quickly and I can't get access to the following thing." This attack works especially well if the attacker impersonates someone in a high position, such as the division vice president or the head of IT security. (Their names can sometimes be found on a public web site, in a network registration with the Internet registry, or in publicity and articles.) The attack is often directed at someone low enough to be intimidated or impressed by the high-level person. A direct phone call and expressions of great urgency can override any natural instinct to check out the story.

Because the victim has helped the attacker (and the attacker has profusely thanked the victim), the victim will think nothing is wrong and not report the incident. Thus, the damage may not be known for some time.

An attacker has little to lose in trying a social engineering attack. At worst it will raise awareness of a possible target. But if the social engineering is directed against someone who is not skeptical, especially someone not involved in security management, it may well succeed. We as humans like to help others when asked politely.

Intelligence

From a port scan the attacker knows what is open. From social engineering, the attacker knows certain internal details. But a more detailed floor plan would be nice. Intelligence is the general term for collecting information. In security it often refers to gathering discrete bits of information from various sources and then putting them together like the pieces of a puzzle.

One commonly used intelligence technique is called "dumpster diving." It involves looking through items that have been discarded in rubbish bins or recycling boxes. It is amazing what we throw away without thinking about it. Mixed with the remains from lunch might be network diagrams, printouts of security device configurations, system designs and source code, telephone and employee lists, and more. Even outdated printouts may be useful. Seldom will the configuration of a security device change completely. More often only one rule is added or deleted or modified, so an attacker has a high probability of a successful attack based on the old information.

Gathering intelligence may also involve eavesdropping. Trained spies may follow employees to lunch and listen in from nearby tables as coworkers discuss security matters. Or spies may befriend key personnel in order to co-opt, coerce, or trick them into passing on useful information.

Most intelligence techniques require little training and minimal investment of time. If an attacker has targeted a particular organization, spending a little time to collect background information yields a big payoff.

Operating System and Application Fingerprinting

The port scan supplies the attacker with very specific information. For instance, an attacker can use a port scan to find out that port 80 is open and supports HTTP, the protocol for transmitting web pages. But the attacker is likely to have many related questions, such as which commercial server application is running, what version, and what the underlying operating system and version are. Once armed with this additional information, the attacker can consult a list of specific software's known vulnerabilities to determine which particular weaknesses to try to exploit.

How can the attacker answer these questions? The network protocols are standard and vendor independent. Still, each vendor's code is implemented independently, so there may be minor variations in interpretation and behavior. The variations do not make the software noncompliant with the standard, but they are different enough to make each version distinctive. For example, each version may have different sequence numbers, TCP flags, and new options. To see why, consider that sender and receiver must coordinate with sequence numbers to implement the connection of a TCP session. Some implementations respond with a given sequence number, others respond with the number one greater, and others respond with an unrelated number. Likewise, certain flags in one version are undefined or incompatible with others. How a system responds to a prompt (for instance, by acknowledging it, requesting retransmission, or ignoring it) can also reveal the system and version. Finally, new features offer a strong clue: A new version will implement a new feature but an old version will reject the request. All these peculiarities, sometimes called the operating system or application fingerprint, can mark the manufacturer and version.
For example, in addition to performing its port scan, a scanner such as *nmap* will respond with a guess at the target operating system. For more information about how this is done, see the paper at *www.insecure.org/nmap/nmap-fingerprinting-article.html*.

Sometimes the application identifies itself. Usually a client-server interaction is handled completely within the application according to protocol rules: "Please send me this page; OK but run this support code; thanks, I just did." But the application cannot respond to a message that does not follow the expected form. For instance, the attacker might use a Telnet application to send meaningless messages to another application. Ports such as 80 (HTTP), 25 (SMTP), 110 (POP), and 21 (FTP) may respond with something like Server: Netscape-Commerce/1.12
Your browser sent a non-HTTP compliant message. Or Microsoft ESMTP MAIL Service, Version: 5.0.2195.3779
This reply tells the attacker which application and version are running.

Bulletin Boards and Chats

The Internet is probably the greatest tool for sharing knowledge since the invention of the printing press. It is probably also the most dangerous tool for sharing knowledge. Numerous underground bulletin boards and chat rooms support exchange of information. Attackers can post their latest exploits and techniques, read what others have done, and search for additional information on systems, applications, or sites. Remember that, as with everything on the Internet, anyone can post anything, so there is no guarantee that the information is reliable or accurate. And you never know who is reading from the Internet.
(See Sidebar 7-4 on law enforcement officials' "going underground" to catch malicious hackers.)

Availability of Documentation

The vendors themselves sometimes distribute information that is useful to an attacker. For example, Microsoft produces a resource kit by which application vendors can investigate a Microsoft product in order to develop compatible, complementary applications. This toolkit also gives attackers tools to use in investigating a product that can subsequently be the target of an attack.

Reconnaissance: Concluding Remarks

A good thief, that is, a successful one, spends time understanding the context of the target. To prepare for perpetrating a bank theft, the thief might monitor the bank, seeing how many guards there are, when they take breaks, when cash shipments arrive, and so forth.

Remember that time is usually on the side of the attacker. In the same way that a bank might notice someone loitering around the entrance, a computing site might notice exceptional numbers of probes in a short time. But the clever thief or attacker will collect a little information, go dormant for a while, and resurface to collect more. So many people walk past banks and peer in the windows, or scan and probe web hosts that individual peeks over time are hard to correlate.

The best defense against reconnaissance is silence. Give out as little information about your site as possible, whether by humans or machines.

Threats in Transit: Eavesdropping and Wiretapping

By now, you can see that an attacker can gather a significant amount of information about a victim before beginning the actual attack. Once the planning is done, the attacker is ready to proceed. In this section we turn to the kinds of attacks that can occur. Recall from Chapter 1 that an attacker has many ways by which to harm in a computing environment: loss of confidentiality, integrity, or availability to data, hardware or software, processes, or other assets. Because a network involves data in transit, we look first at the harm that can occur between a sender and a receiver. Sidebar 7-5 describes the ease of interception.

The easiest way to attack is simply to listen in. An attacker can pick off the content of a communication passing in the clear. The term eavesdrop implies overhearing without expending any extra effort. For example, we might say that an attacker (or a system administrator) is eavesdropping by monitoring all traffic passing through a node. The administrator might have a legitimate purpose, such as watching for inappropriate use of resources (for instance, visiting non-work-related web sites from a company network) or communication with inappropriate parties (for instance, passing files to an enemy from a military computer).

A more hostile term is wiretap, which means intercepting communications through some effort. Passive wiretapping is just "listening," much like eavesdropping. But active wiretapping means injecting something into the communication. For example, Marvin could replace Manny's communications with his own or create communications purported to be from Manny. Originally derived from listening in on telegraph and telephone communications, the term wiretapping usually conjures up a physical act by which a device extracts information as it flows over a wire. But in fact no actual contact is necessary. A wiretap can be done covertly so that neither the sender nor the receiver of a communication knows that the contents have been intercepted.

Wiretapping works differently depending on the communication medium used. Let us look more carefully at each possible choice.

Cable

At the most local level, all signals in an Ethernet or other LAN are available on the cable for anyone to intercept. Each LAN connector (such as a computer board) has a unique address; each board and its drivers are programmed to label all packets from its host with its unique address (as a sender's "return address") and to take from the net only those packets addressed to its host.

But removing only those packets addressed to a given host is mostly a matter of politeness; there is little to stop a program from examining each packet as it goes by. A device called a packet sniffer can retrieve all packets on the LAN. Alternatively, one of the interface cards can be reprogrammed to have the supposedly unique address of another existing card on the LAN so that two different cards will both fetch packets for one address. (To avoid detection, the rogue card will have to put back on the net copies of the packets it has intercepted.)

Fortunately (for now), LANs are usually used only in environments that are fairly friendly, so these kinds of attacks occur infrequently.

Clever attackers can take advantage of a wire's properties and read packets without any physical manipulation. Ordinary wire (and many other electronic components) emit radiation. By a process called inductance an intruder can tap a wire and read radiated signals without making physical contact with the cable. A cable's signals travel only short distances, and they can be blocked by other conductive materials. The equipment needed to pick up signals is inexpensive and easy to obtain, so inductance threats are a serious concern for cable-based networks. For the attack to work, the intruder must be fairly close to the cable; this form of attack is thus limited to situations with reasonable physical access.

If the attacker is not close enough to take advantage of inductance, then more hostile measures may be warranted. The easiest form of intercepting a cable is by direct cut. If a cable is severed, all service on it stops. As part of the repair, an attacker can easily splice in a secondary cable that then receives a copy of all signals along the primary cable. There are ways to be a little less obvious but accomplish the same goal. For example, the attacker might carefully expose some of the outer conductor, connect to it, then carefully expose some of the inner conductor and connect to it. Both of these operations alter the resistance, called the impedance, of the cable. In the first case, the repair itself alters the impedance, and the impedance change can be explained (or concealed) as part of the repair. In the second case, a little social engineering can explain the change. ("Hello, this is Matt, a technician with Bignetworks. We are changing some equipment on our end, and so you might notice a change in impedance.")

Signals on a network are multiplexed, meaning that more than one signal is transmitted at a given time. For example, two analog (sound) signals can be combined, like two tones in a musical chord, and two digital signals can be combined by interleaving, like playing cards being shuffled. A LAN carries distinct packets, but data on a WAN may be heavily multiplexed as it leaves its sending host. Thus, a wiretapper on a WAN needs to be able not only to intercept the desired communication but also to extract it from the others with which it is multiplexed.

While this can be done, the effort involved means it will be used sparingly. Microwave Microwave signals are not carried along a wire; they are broadcast through the air, making them more accessible to outsiders. Typically, a transmitter's signal is focused on its corresponding receiver. The signal path is fairly wide, to be sure of hitting the receiver, as shown in Figure 7-13. From a security standpoint, the wide swath is an invitation to mischief.

Not only can someone intercept a microwave transmission by interfering with the line of sight between sender and receiver, someone can also pick up an entire transmission from an antenna located close to but slightly off the direct focus point.

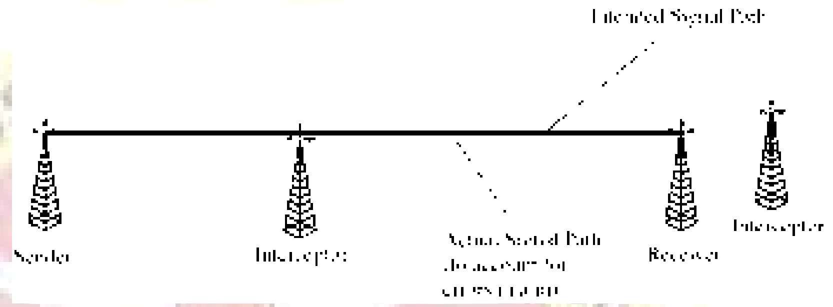


Figure 7-13. Path of Microwave Signals.

A microwave signal is usually not shielded or isolated to prevent interception. Microwave is, therefore, a very insecure medium. However, because of the large volume of traffic carried by microwave links, it is unlikelybut not impossiblethat someone will be able to separate an individual transmission from all the others interleaved with it. A privately owned microwave link, carrying only communications for one organization, is not so well protected by volume.

Satellite Communication

Satellite communication has a similar problem of being dispersed over an area greater than the intended point of reception. Different satellites have different characteristics, but some signals can be intercepted in an area several hundred miles wide and a thousand miles long.

Therefore, the potential for interception is even greater than with microwave signals. However, because satellite communications are generally heavily multiplexed, the risk is small that any one communication will be intercepted.

Optical Fiber

Optical fiber offers two significant security advantages over other transmission media. First, the entire optical network must be tuned carefully each time a new connection is made.

Therefore, no one can tap an optical system without detection. Clipping just one fiber in a bundle will destroy the balance in the network.

Second, optical fiber carries light energy, not electricity. Light does not emanate a magnetic field as electricity does. Therefore, an inductive tap is impossible on an optical fiber cable.

Just using fiber, however, does not guarantee security, any more than does using encryption. The repeaters, splices, and taps along a cable are places at which data may be available more easily than in the fiber cable itself. The connections from computing equipment to the fiber may also be points for penetration. By itself, fiber is much more secure than cable, but it has vulnerabilities too.

Wireless

Wireless networking is becoming very popular, with good reason. With wireless (also known as WiFi), people are not tied to a wired connection; they are free to roam throughout an office, house, or building while maintaining a connection. Universities, offices, and even home users like being able to connect to a network without the cost, difficulty, and inconvenience of running wires. The difficulties of wireless arise in the ability of intruders to intercept and spoof a connection.

As we noted earlier, wireless communications travel by radio. In the United States, wireless computer connections share the same frequencies as garage door openers, local radios (typically used as baby monitors), some cordless telephones, and other very short distance applications. Although the frequency band is crowded, few applications are expected to be on the band from any single user, so contention or interference is not an issue.

But the major threat is not interference; it is interception. A wireless signal is strong for approximately 100 to 200 feet. To appreciate those figures, picture an ordinary ten-story office building, ten offices "wide" by five offices "deep," similar to many buildings in office parks or on university campuses. Assume you set up a wireless base station (receiver) in the corner of the top floor. That station could receive signals transmitted from the opposite corner of the ground floor. If a similar building were adjacent, the signal could also be received throughout that building, too. (See Sidebar 7-5 on how easy it is to make a connection.) Few people would care to listen to someone else's baby monitor, but many people could and do take advantage of a passive or active wiretap of a network connection.
A strong signal can be picked up easily. And with an inexpensive, tuned antenna, a wireless signal can be picked up several miles away. In other words, someone who wanted to pick up your particular signal could do so from several streets away. Parked in a truck or van, the interceptor could monitor your communications for quite some time without arousing suspicion.

Interception

Interception of wireless traffic is always a threat, through either passive or active wiretapping. Sidebar 7-6 illustrates how software faults may make interception easier than you might think. You may react to that threat by assuming that encryption will address it.
Unfortunately, encryption is not always used for wireless communication, and the encryption built into some wireless devices is not as strong as it should be to deter a dedicated attacker.

Theft of Service

Wireless also admits a second problem: the possibility of rogue use of a network connection. Many hosts run the Dynamic Host Configuration Protocol (DHCP), by which a client negotiates a one-time IP address and connectivity with a host. This protocol is useful in office or campus settings, where not all users (clients) are active at any time. A small number of IP addresses can be shared among users. Essentially the addresses are available in a pool. A new client requests a connection and an IP address through DHCP, and the server assigns one from the pool.

This scheme admits a big problem with authentication. Unless the host authenticates users before assigning a connection, any requesting client is assigned an IP address and network access. (Typically, this assignment occurs before the user on the client workstation actually identifies and authenticates to a server, so there may not be an authenticatable identity that the DHCP server can demand.) The situation is so serious that in some metropolitan areas a map is available, showing many networks accepting wireless connections.

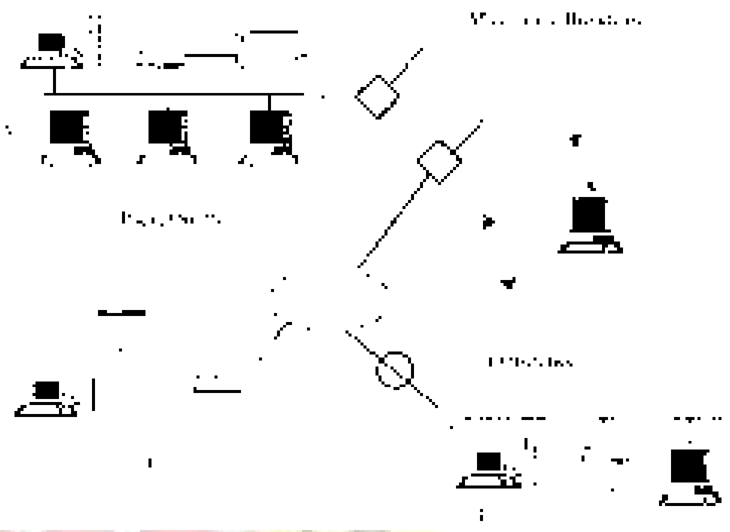
A user wanting free Internet access can often get it simply by finding a wireless LAN offering DHCP service. But is it legal? In separate cases Benjamin Smith III in Florida in July 2005 and Dennis Kauchak in Illinois in March 2006 were convicted of remotely accessing a computer wirelessly without the owner's permission. Kauchak was sentenced to a $250 fine. So, even though you are able to connect, it may not be legal to do so.

On the other hand, some cities or organizations make wireless access freely available as a community service. Free wireless cities include Albuquerque and Honolulu in the United States, Oulu in Finland, and the central districts of cities such as Hamburg, Germany, and Adelaide, Australia. The cities hope that providing free access will spur IT growth and attract tourists and business travelers.

Summary of Wiretapping

There are many points at which network traffic is available to an interceptor. Figure 7-14 illustrates how communications are exposed from their origin to their destination.

Figure 7-14. Wiretap Vulnerabilities.



From a security standpoint, you should assume that *all* communication links between network nodes can be broken. For this reason, commercial network users employ encryption to protect the confidentiality of their communications, as we demonstrate later in this chapter. Local network communications can be encrypted, although for performance reasons it may be preferable to protect local connections with strong physical and administrative security instead.

Protocol Flaws

Internet protocols are publicly posted for scrutiny by the entire Internet community. Each accepted protocol is known by its Request for Comment (RFC) number. Many problems with protocols have been identified by sharp reviewers and corrected before the protocol was established as a standard.

But protocol definitions are made and reviewed by fallible humans. Likewise, protocols are implemented by fallible humans. For example, TCP connections are established through sequence numbers. The client (initiator) sends a sequence number to open a connection, the server responds with that number and a sequence number of its own, and the client responds with the server's sequence number. Suppose (as pointed out by Morris [MOR85]) someone can guess a client's next sequence number. That person could impersonate the client in an interchange. Sequence numbers are incremented regularly, so it can be easy to predict the next number. (Similar protocol problems are summarized in [BEL89].)

Impersonation

In many instances, there is an easier way than wiretapping for obtaining information on a network: Impersonate another person or process. Why risk tapping a line, or why bother extracting one communication out of many, if you can obtain the same data directly? Impersonation is a more significant threat in a wide area network than in a local one. Local individuals often have better ways to obtain access as another user; they can, for example, simply sit at an unattended workstation. Still, impersonation attacks should not be ignored even on local area networks, because local area networks are sometimes attached to wider area networks without anyone's first thinking through the security implications.

In an impersonation, an attacker has several choices:

- Guess the identity and authentication details of the target.
- Pick up the identity and authentication details of the target from a previous communication or from wiretapping.
- Circumvent or disable the authentication mechanism at the target computer.
- Use a target that will not be authenticated.
- Use a target whose authentication data are known.

Let us look at each choice.

Authentication Foiled by Guessing

Chapter 4 reported the results of several studies showing that many users choose easy-to-guess passwords. In Chapter 3, we saw that the Internet worm of 1988 capitalized on exactly that flaw. Morris's worm tried to impersonate each user on a target machine by trying, in order, a handful of variations of the user name, a list of about 250 common passwords and, finally, the words in a dictionary. Sadly, many users' accounts are still open to these easy attacks.

A second source of password guesses is default passwords. Many systems are initially configured with default accounts having GUEST or ADMIN as login IDs;

accompanying these IDs are well-known passwords such as "guest" or "null" or "password" to enable the administrator to set up the system. Administrators often forget to delete or disable these accounts, or at least to change the passwords.

In a trustworthy environment, such as an office LAN, a password may simply be a signal that the user does not want others to use the workstation or account. Sometimes the password-protected workstation contains sensitive data, such as employee salaries or information about new products. Users may think that the password is enough to keep out a curious colleague; they see no reason to protect against concerted attacks. However, if that trustworthy environment is connected to an untrustworthy wider-area network, all users with simple passwords become easy targets. Indeed, some systems are not originally connected to a wider network, so their users begin in a less exposed situation that clearly changes when the connection occurs.

Dead accounts offer a final source of guessable passwords. To see how, suppose Professor Romine, a faculty member, takes leave for a year to teach at another university. The existing account may reasonably be kept on hold, awaiting the professor's return. But an attacker, reading a university newspaper online, finds out that the user is away. Now the attacker uses social engineering on the system administration ("Hello, this is Professor Romine calling from my temporary office at State University. I haven't used my account for quite a while, but now I need something from it urgently. I have forgotten the password. Can you please reset it to ICECREAM? No? Well, send me a new password by email to my account r1@stateuniv.edu.")

Alternatively, the attacker can try several passwords until the password guessing limit is exceeded. The system then locks the account administratively, and the attacker uses a social engineering attack. In all these ways the attacker may succeed in resetting or discovering a password.

Authentication Thwarted by Eavesdropping or Wiretapping

Because of the rise in distributed and client-server computing, some users have access privileges on several connected machines. To protect against arbitrary outsiders using these accesses, authentication is required between hosts. This access can involve the user directly, or it can be done automatically on behalf of the user through a host-to-host authentication protocol. In either case, the account and authentication details of the subject are passed to the destination host. When these details are passed on the network, they are exposed to anyone observing the communication on the network. These same authentication details can be reused by an impersonator until they are changed.

Because transmitting a password in the clear is a significant vulnerability, protocols have been developed so that the password itself never leaves a user's workstation. But, as we have seen in several other places, the details are important.

Microsoft LAN Manager was an early method for implementing networks. It had a password exchange mechanism in which the password itself was never transmitted in the clear; instead only a cryptographic hash of it was transmitted. A password could consist of up to 14 characters. It could include upper- and lowercase letters, digits, and special characters, for 67 possibilities in any one position, and 67_{14} possibilities for a whole 14-character password quite a respectable work factor. However, those 14 characters were not diffused across the entire hash; they were sent in separate substrings, representing characters 17 and 814. A 7-character or shorter password had all nulls in the second substring and was instantly recognizable. An 8-character password had 1 character and 6 nulls in the second substring, so 67 guesses would find the one character. Even in the best case, a 14-character password, the work factor fell from 67_{14} to $67_7 + 67_7 = 2 * 67_7$. These work factors differ by a factor of approximately 10 billion. (See [MUD97] for details.) LAN Manager authentication was preserved in many later systems (including Windows NT) as an option to support backward compatibility with systems such as Windows 95/98. This lesson is a good example of why security and cryptography are very precise and must be monitored by experts from concept through design and implementation.

Authentication Foiled by Avoidance

Obviously, authentication is effective only when it works. A weak or flawed authentication allows access to any system or person who can circumvent the authentication.

In a classic operating system flaw, the buffer for typed characters in a password was of fixed size, counting all characters typed, including backspaces for correction. If a user typed more characters than the buffer would hold, the overflow caused the operating system to bypass password comparison and act as if a correct authentication had been supplied. These flaws or weaknesses can be exploited by anyone seeking access.

Many network hosts, especially those that connect to wide area networks, run variants of Unix System V or BSD Unix. In a local environment, many users are not aware of which networked operating system is in use; still fewer would know of, be capable of, or be interested in exploiting flaws. However, some hackers regularly scan wide area networks for hosts running weak or flawed operating systems. Thus, connection to a wide area network, especially the Internet, exposes these flaws to a wide audience intent on exploiting them.

Nonexistent Authentication

If two computers are used by the same users to store data and run processes and if each has authenticated its users on first access, you might assume that computer-to-computer or local user-to-remote process authentication is unnecessary. These two computers and their users are a trustworthy environment in which the added complexity of repeated authentication seems excessive.

However, this assumption is not valid. To see why, consider the Unix operating system. In Unix, the file *.rhosts* lists trusted hosts and *.rlogin* lists trusted users who are allowed access without authentication. The files are intended to support computer-to-computer connection by users who have already been authenticated at their primary hosts. These "trusted hosts" can also be exploited by outsiders who obtain access to one system through an authentication weakness (such as a guessed password) and then transfer to another system that accepts the authenticity of a user who comes from a system on its trusted list.

An attacker may also realize that a system has some identities requiring no authentication. Some systems have "guest" or "anonymous" accounts to allow outsiders to access things the systems want to release to anyone. For example, a bank might post a current listing of foreign currency rates, a library with an online catalog might make that catalog available for anyone to search, or a company might allow access to some of its reports. A user can log in as "guest" and retrieve publicly available items. Typically, no password is required, or the user is shown a message requesting that the user type "GUEST" (or *your name*, which really means any string that looks like a name) when asked for a password. Each of these accounts allows access to unauthenticated users.

Well-Known Authentication

Authentication data should be unique and difficult to guess. But unfortunately, the convenience of one well-known authentication scheme sometimes usurps the protection. For example, one computer manufacturer planned to use the same password to allow its remote maintenance personnel to access any of its computers belonging to any of its customers throughout the world. Fortunately, security experts pointed out the potential danger before that idea was put in place.

The system network management protocol (SNMP) is widely used for remote management of network devices, such as routers and switches, that support no ordinary users. SNMP uses a "community string," essentially a password for the community of devices that can interact with one another. But network devices are designed especially for quick installation with minimal configuration, and many network administrators do not change the default community string installed on a router or switch. This laxity makes these devices on the network perimeter open to many SNMP attacks.

Some vendors still ship computers with one system administration account installed, having a default password. Or the systems come with a demonstration or test account, with no required password. Some administrators fail to change the passwords or delete these accounts.

Trusted Authentication

Finally, authentication can become a problem when identification is delegated to other trusted sources. For instance, a file may indicate who can be trusted on a particular host. Or the authentication mechanism for one system can "vouch for" a user. We noted earlier how the Unix *.rhosts, .rlogin,* and */etc/hosts/equiv* files indicate hosts or users that are trusted on other hosts. While these features are useful to users who have accounts on multiple machines or for network management, maintenance, and operation, they must be used very carefully.

Each of them represents a potential hole through which a remote useror a remote attacker can achieve access.

Spoofing

Guessing or otherwise obtaining the network authentication credentials of an entity (a user, an account, a process, a node, a device) permits an attacker to create a full communication under the entity's identity. Impersonation falsely represents a valid entity in a communication.

Closely related is spoofing, when an attacker falsely carries on one end of a networked interchange. Examples of spoofing are masquerading, session hijacking, and man-in-the-middle attacks.

Masquerade

In a masquerade one host pretends to be another. A common example is URL confusion. Domain names can easily be confused, or someone can easily mistype certain names. Thus xyz.com, xyz.org, and xyz.net might be three different organizations, or one bona fide organization (for example, xyz.com) and two masquerade attempts from someone who registered the similar domain names. Names with or without hyphens (coca-cola.com versus cocacola.com) and easily mistyped names (l0pht.com versus lopht.com, or citibank.com versus citybank.com) are candidates for masquerading.

From the attacker's point of view, the fun in masquerading comes before the mask is removed. For example, suppose you want to attack a real bank, First Blue Bank of Chicago. The actual bank has the domain name BlueBank.com, so you register the domain name Blue-Bank.com. Next, you put up a web page at Blue-Bank.com, perhaps using the real Blue Bank logo that you downloaded to make your site look as much as possible like that of the Chicago bank. Finally, you ask people to log in with their name, account number, and password or PIN. (This redirection can occur in many ways. For example, you can pay for a banner ad that links to your site instead of the real bank's, or you can send e-mail to Chicago residents and invite them to visit your site.) After collecting personal data from several bank users, you can drop the connection, pass the connection on to the real Blue Bank, or continue to collect more information. You may even be able to transfer this connection smoothly to an authenticated access to the real Blue Bank so that the user never realizes the deviation. (First Blue Bank would probably win a suit to take ownership of the Blue-Bank.com domain.)

A variation of this attack is called phishing. You send an e-mail message, perhaps with the real logo of Blue Bank, and an enticement to click on a link, supposedly to take the victim to the Blue Bank web site. The enticement might be that your victim's account has been suspended or that you offer your victim some money for answering a survey (and need the account number and PIN to be able to credit the money), or some other legitimate-sounding explanation. The link might be to your domain Blue-Bank.com, the link might say *click here* to access your account (where the *click here* link connects to your fraudulent site), or you might use some other trick with the URL to fool your victim, like www.redirect.com/bluebank.com.

In another version of a masquerade, the attacker exploits a flaw in the victim's web server and is able to overwrite the victim's web pages. Although there is some public humiliation at having one's site replaced, perhaps with obscenities or strong messages opposing the nature of the site (for example, a plea for vegetarianism on a slaughterhouse web site), most people would not be fooled by a site displaying a message absolutely contrary to its aims. However, a clever attacker can be more subtle. Instead of differentiating from the real site, the attacker can try to build a false site that resembles the real one, perhaps to obtain sensitive information (names, authentication numbers, credit card numbers) or to induce the user to enter into a real transaction. For example, if one bookseller's site, call it Books-R-Us, were overtaken subtly by another, called Books Depot, the orders may actually be processed, filled, and billed to the naïve users by Books Depot. Test your ability to distinguish phishing sites from real ones at http://survey.mailfrontier.com/survey/quiztest.html.

Phishing is becoming a serious problem, according to a trends report from the Anti-Phishing Working Group [APW05]. This group received over 12,000 complaints each month from March 2005 to March 2006, with the number peaking above 18,000 for March 2006.

Session Hijacking Session hijacking is intercepting and carrying on a session begun by another entity. Suppose two entities have entered into a session but then a third entity intercepts the traffic and carries on the session in the name of the other. Our example of Books-R-Us could be an instance of this technique. If Books Depot used a wiretap to intercept packets between you and Books-R-Us, Books Depot could simply monitor the information flow, letting Books-R-Us do the hard part of displaying titles for sale and convincing the user to buy. Then, when the user has completed the order, Books Depot intercepts the "I'm ready to check out" packet, and finishes the order with the user, obtaining shipping address, credit card details, and so forth.

To Books-R-Us, the transaction would look like any other incomplete transaction: The user was browsing but for some reason decided to go elsewhere before purchasing. We would say that Books Depot had hijacked the session.

A different type of example involves an interactive session, for example, using Telnet. If a system administrator logs in remotely to a privileged account, a session hijack utility could intrude in the communication and pass commands as if they came from the administrator.

Man-in-the-Middle Attack

Our hijacking example requires a third party involved in a session between two entities. A man-in-the-middle attack is a similar form of attack, in which one entity intrudes between two others. We studied one form of this attack in Chapter 3. The difference between man-in-the-middle and hijacking is that a man-in-the-middle usually participates from the start of the session, whereas a session hijacking occurs after a session has been established.

The difference is largely semantic and not too significant. Man-in-the-middle attacks are frequently described in protocols. To see how an attack works, suppose you want to exchange encrypted information with your friend. You contact the key server and ask for a secret key with which to communicate with your friend. The key server responds by sending a key to you and your friend. One man-in-the-middle attack assumes someone can see and enter into all parts of this protocol. A malicious middleman intercepts the response key and can then eavesdrop on, or even decrypt, modify, and re-encrypt any subsequent communications between you and your friend. This attack is depicted in Figure 7-15.

Figure 7-15. Key Interception by a Man-in-the-Middle Attack.

This attack would be changed with public keys, because the man-in-the-middle would not have the private key to be able to decrypt messages encrypted under your friend's public key. The man-in-the-middle attack now becomes more of the three-way interchange its name implies. The man-in-the-middle intercepts your request to the key server and instead asks for your friend's public key. The man-in-the-middle passes to you his own public key, not your friend's. You encrypt using the public key you received (from the man-in-the-middle); the man-in-the-middle intercepts and decrypts, reads, and reencrypts, using your friend's public key; and your friend receives. In this way, the man-in-the-middle reads the messages and neither you nor your friend is aware of the interception. A slight variation of this attack works for secret key distribution under a public key.

Message Confidentiality Threats

An attacker can easily violate message confidentiality (and perhaps integrity) because of the public nature of networks. Eavesdropping and impersonation attacks can lead to a confidentiality or integrity failure. Here we consider several other vulnerabilities that can affect confidentiality.

Misdelivery

Sometimes messages are misdelivered because of some flaw in the network hardware or software. Most frequently, messages are lost entirely, which is an integrity or availability issue. Occasionally, however, a destination address is modified or some handler malfunctions, causing a message to be delivered to someone other than the intended recipient. All of these "random" events are quite uncommon.

More frequent than network flaws are human errors. It is far too easy to mistype an address such as 100064,30652 as 10064,30652 or 100065,30642, or to type "idw" or "iw" instead of "diw" for David Ian Walker, who is called Ian by his friends. There is simply no justification for a computer network administrator to identify people by meaningless long numbers or cryptic initials when "iwalker" would be far less prone to human error.

Exposure

To protect the confidentiality of a message, we must track it all the way from its creation to its disposal. Along the way, the content of a message may be exposed in temporary buffers; at switches, routers, gateways, and intermediate hosts throughout the network; and in the workspaces of processes that build, format, and present the message. In earlier chapters, we considered confidentiality exposures in programs and operating systems. All of these exposures apply to networked environments as well. Furthermore, a malicious attacker can use any of these exposures as part of a general or focused attack on message confidentiality.

Passive wiretapping is one source of message exposure. So also is subversion of the structure by which a communication is routed to its destination. Finally, intercepting the message at its source, destination, or at any intermediate node can lead to its exposure.

Traffic Flow Analysis

Sometimes not only is the message itself sensitive but the fact that a message *exists* is also sensitive. For example, if the enemy during wartime sees a large amount of network traffic between headquarters and a particular unit, the enemy may be able to infer that significant action is being planned involving that unit. In a commercial setting, messages sent from the president of one company to the president of a competitor could lead to speculation about a takeover or conspiracy to fix prices. Or communications from the prime minister of one country to another with whom diplomatic relations were suspended could lead to inferences about a rapprochement between the countries. In these cases, we need to protect both the *content* of messages and the *header* information that identifies sender and receiver.

Message Integrity Threats

In many cases, the *integrity* or correctness of a communication is at least as important as its confidentiality. In fact for some situations, such as passing authentication data, the integrity of the communication is paramount. In other cases, the need for integrity is less obvious. Next we consider threats based on failures of integrity in communication.

Falsification of Messages

Increasingly, people depend on electronic messages to justify and direct actions. For example, if you receive a message from a good friend asking you to meet at the pub for a drink next Tuesday evening, you will probably be there at the appointed time. Likewise, you will comply with a message from your supervisor telling you to stop work on project A and devote your energy instead to project B. As long as it is reasonable, we tend to act on an electronic message just as we would on a signed letter, a telephone call, or a face-to-face communication.

However, an attacker can take advantage of our trust in messages to mislead us. In particular, an attacker may

- change some or all of the content of a message
- replace a message entirely, including the date, time, and sender/receiver identification
- reuse (replay) an old message
- combine pieces of different messages into one
- change the apparent source of a message
- redirect a message
- destroy or delete a message

These attacks can be perpetrated in the ways we have already examined, including

- active wiretap
- Trojan horse
- impersonation
- preempted host
- preempted workstation

Noise

Signals sent over communications media are subject to interference from other traffic on the same media, as well as from natural sources, such as lightning, electric motors, and animals.

Such unintentional interference is called noise. These forms of noise are inevitable, and they can threaten the integrity of data in a message.

Fortunately, communications protocols have been intentionally designed to overcome the negative effects of noise. For example, the TCP/IP protocol suite ensures detection of almost all transmission errors. Processes in the communications stack detect errors and arrange for retransmission, all invisible to the higher-level applications. Thus, noise is scarcely a consideration for users in security-critical applications.

Format Failures

Network communications work because of well-designed protocols that define how two computers communicate with a minimum of human intervention. The format of a message, size of a data unit, sequence of interactions, even the meaning of a single bit is precisely described in a standard. The whole network works only because everyone obeys these rules.

Almost everyone, that is. Attackers purposely break the rules to see what will happen. Or the attacker may seek to exploit an undefined condition in the standard. Software may detect the violation of structure and raise an error indicator. Sometimes, however, the malformation causes a software failure, which can lead to a security compromise, just what the attacker wants. In this section we look at several kinds of malformation.

Malformed Packets

Packets and other data items have specific formats, depending on their use. Field sizes, bits to signal continuations, and other flags have defined meanings and will be processed appropriately by network service applications called protocol handlers. These services do not necessarily check for errors, however. What happens if a packet indicates a data field is 40 characters long and the actual field length is 30 or 50? Or what if a packet reports its content is continued in the next packet and there is no next packet? Or suppose for a 2-bit flag only values 00, 01, and 10 are defined; what does the handler do if it receives the value 11?

For example, in 2003 Microsoft distributed a patch for its RPC (Remote Procedure Call) service. If a malicious user initiated an RPC session and then sent an incorrectly formatted packet, the entire RPC service failed, as well as some other Microsoft services.

Attackers try all sorts of malformations of packets. Of course, many times the protocol handler detects the malformation and raises an error condition, and other times the failure affects only the user (the attacker). But when the error causes the protocol handler to fail, the result can be denial of service, complete failure of the system, or some other serious result.

Protocol Failures and Implementation Flaws

Each protocol is a specification of a service to be provided; the service is then implemented in software, which, as discussed in Chapter 3, may be flawed. Network protocol software is basic to the operating system, so flaws in that software can cause widespread harm because of the privileges with which the software runs and the impact of the software on many users at once. Certain network protocol implementations have been the source of many security flaws; especially troublesome have been SNMP (network management), DNS (addressing service), and e-mail services such as SMTP and S/MIME. Although different vendors have implemented the code for these services themselves, they often are based on a common (flawed) prototype. For example, the CERT advisory for SNMP flaws (Vulnerability Note 107186) lists approximately 200 different implementations to which the advisory applies.

Or the protocol itself may be incomplete. If the protocol does not specify what action to take in a particular situation, vendors may produce different results. So an interaction on Windows, for example, might succeed while the same interaction on a Unix system would fail.

The protocol may have an unknown security flaw. In a classic example, Bellovin [BEL89] points out a weakness in the way packet sequence numbers are assignedan attacker could intrude into a communication in such a way that the intrusion is accepted as the real communication and the real sender is rejected.

Attackers can exploit all of these kinds of errors.

Web Site Vulnerabilities

A web site is especially vulnerable because it is almost completely exposed to the user. If you use an application program, you do not usually get to view the program's code. With a web site, the attacker can download the site's code for offline study over time. With a program, you have little ability to control in what order you access parts of the program, but a web attacker gets to control in what order pages are accessed, perhaps even accessing without first having run pages 1 through 4. The attacker can also choose what data to supply and can run experiments with different data values to see how the site will react. In short, the attacker has some advantages that can be challenging to control.

The list of web site vulnerabilities is too long to explore completely here. Hoglund and McGraw [HOG04], Andrews and Whitaker [AND06], and Howard et al. [HOW05] offer excellent analyses of how to find and fix flaws in web software. Be sure to review the code development issues in Chapter 3, because many code techniques there (such as buffer overflows and insufficient parameter checking) are applicable here.

Web Site Defacement

One of the most widely known attacks is the web site defacement attack. Because of the large number of sites that have been defaced and the visibility of the result, the attacks are often reported in the popular press.

A defacement is common not only because of its visibility but also because of the ease with which one can be done. Web sites are designed so that their code is downloaded, enabling an attacker to obtain the full hypertext document and all programs directed to the client in the loading process. An attacker can even view programmers' comments left in as they built or maintained the code. The download process essentially gives the attacker the blueprints to the web site.

The ease and appeal of a defacement are enhanced by the seeming plethora of vulnerabilities that web sites offer an attacker. For example, between December 1999 and June 2001 (the first 18 months after its release), Microsoft provided 17 *security* patches for its web server software, Internet Information Server (IIS) version 4.0. And version 4.0 was an upgrade for three previous versions, so theoretically Microsoft had a great deal of time earlier to work out its security flaws.

Buffer Overflows

Buffer overflow is alive and well on web pages, too. It works exactly the same as described in Chapter 3: The attacker simply feeds a program far more data than it expects to receive. A buffer size is exceeded, and the excess data spill over into adjoining code and data locations.

Perhaps the best-known web server buffer overflow is the file name problem known as iishack. This attack is so well known that is has been written into a procedure (see *http://www.technotronic.com*). To execute the procedure, an attacker supplies as parameters the site to be attacked and the URL of a program the attacker wants that server to execute.

Other web servers are vulnerable to extremely long parameter fields, such as passwords of length 10,000 or a long URL padded with space or null characters.

Dot-Dot-Slash Web server code should always run in a constrained environment. Ideally, the web server should never have editors, xterm and Telnet programs, or even most system utilities loaded. By constraining the environment in this way, even if an attacker escapes from the web server application, no other executable programs will help the attacker use the web server's computer and operating system to extend the attack. The code and data for web applications can be transferred manually to a web server or pushed as a raw image.

But many web applications programmers are naïve. They expect to need to edit a web application in place, so they install editors and system utilities on the server to give them a complete environment in which to program.

A second, less desirable, condition for preventing an attack is to create a fence confining the web server application. With such a fence, the server application cannot escape from its area and access other potentially dangerous system areas (such as editors and utilities). The server begins in a particular directory subtree, and everything the server needs is in that same subtree.

Enter the dot-dot. In both Unix and Windows, '..' is the directory indicator for "predecessor." And '../..' is the grandparent of the current location. So someone who can

enter file names can travel back up the directory tree one .. at a time. Cerberus Information Security analysts found just that vulnerability in the webhits.dll extension for the Microsoft Index Server. For example, passing the following URL causes the server to return the requested file, autoexec.nt, enabling an attacker to modify or delete it.
http://yoursite.com/webhits.htw?CiWebHits&File=
../../../../../winnt/system32/autoexec.nt
Application Code Errors

A user's browser carries on an intricate, undocumented protocol interchange with applications on the web server. To make its job easier, the web server passes context strings to the user, making the user's browser reply with full context. A problem arises when the user can modify that context.

To see why, consider our fictitious shopping site called CDs-R-Us, selling compact discs. At any given time, a server at that site may have a thousand or more transactions in various states of completion. The site displays a page of goods to order, the user selects one, the site displays more items, the user selects another, the site displays more items, the user selects two more, and so on until the user is finished selecting. Many people go on to complete the order by specifying payment and shipping information. But other people use web sites like this one as an online catalog or guide, with no real intention of ordering. For instance, they can use this site to find out the price of the latest CD from Cherish the Ladies; they can use an online book service to determine how many books by Iris Murdoch are in print. And even if the user is a bona fide customer, sometimes web connections fail, leaving the transaction incomplete. For these reasons, the web server often keeps track of the status of an incomplete order in parameter fields appended to the URL. These fields travel from the server to the browser and back to the server with each user selection or page request.

Assume you have selected one CD and are looking at a second web page. The web server has passed you a URL similar to http://www.CDs-r-us.com/buy.asp?i1=459012&p1=1599 This URL means you have chosen CD number 459012, and its price is $15.99. You now select a second and the URL becomes http://www.CDs-r-us.com/
buy.asp?i1=459012&p1=1599&i2=365217&p2=1499

But if you are a clever attacker, you realize that you can edit the URL in the address window of your browser. Consequently, you change each of 1599 and 1499 to 199. And when the server totals up your order, lo and behold, your two CDs cost only $1.99 each.

This failure is an example of the time-of-check to time-of-use flaw that we discussed in Chapter 3. The server sets (checks) the price of the item when you first display the price, but then it loses control of the checked data item and never checks it again. This situation arises frequently in server application code because application programmers are generally not aware of security (they haven't read Chapter 3a) and typically do not anticipate malicious behavior.
Server-Side Include

A potentially more serious problem is called a server-side include. The problem takes advantage of the fact that web pages can be organized to invoke a particular function automatically. For example, many pages use web commands to send an e-mail message in the "contact us" part of the displayed page. The commands, such as e-mail, if, goto, and include, are placed in a field that is interpreted in HTML.

One of the server-side include commands is exec, to execute an arbitrary file on the server. For instance, the server-side include command
<a#exec cmd="/usr/bin/telnet &">
opens a Telnet session from the server running in the name of (that is, with the privileges of) the server. An attacker may find it interesting to execute commands such as *chmod* (change access rights to an object), *sh* (establish a command shell), or *cat* (copy to a file).

For more web application vulnerabilities see [HOG04, AND06, and HOW05].

Denial of Service

So far, we have discussed attacks that lead to failures of confidentiality or integrity problems we have also seen in the contexts of operating systems, databases, and applications.

Availability attacks, sometimes called denial-of-service or DOS attacks, are much more significant in networks than in other contexts. There are many accidental and malicious threats to availability or continued service.

Transmission Failure

Communications fail for many reasons. For instance, a line is cut. Or network noise makes a packet unrecognizable or undeliverable. A machine along the transmission path fails for hardware or software reasons. A device is removed from service for repair or testing. A device is saturated and rejects incoming data until it can clear its overload. Many of these problems are temporary or automatically fixed (circumvented) in major networks, including the Internet.

However, some failures cannot be easily repaired. A break in the single communications line to your computer (for example, from the network to your network interface card or the telephone line to your modem) can be fixed only by establishment of an alternative link or repair of the damaged one. The network administrator will say "service to the rest of the network was unaffected," but that is of little consolation to you.

From a malicious standpoint, you can see that anyone who can sever, interrupt, or overload capacity to you can deny you service. The physical threats are pretty obvious. We consider instead several electronic attacks that can cause a denial of service.

Connection Flooding

The most primitive denial-of-service attack is flooding a connection. If an attacker sends you as much data as your communications system can handle, you are prevented from receiving any other data. Even if an occasional packet reaches you from someone else, communication to you will be seriously degraded.

More sophisticated attacks use elements of Internet protocols. In addition to TCP and UDP, there is a third class of protocols, called ICMP or Internet Control Message Protocols.

Normally used for system diagnostics, these protocols do not have associated user applications. ICMP protocols include

- *ping*, which requests a destination to return a reply, intended to show that the destination system is reachable and functioning
- *echo*, which requests a destination to return the data sent to it, intended to show that the connection link is reliable (ping is actually a version of echo)
- *destination unreachable*, which indicates that a destination address cannot be accessed
- *source quench*, which means that the destination is becoming saturated and the source should suspend sending packets for a while

These protocols have important uses for network management. But they can also be used to attack a system. The protocols are handled within the network stack, so the attacks may be difficult to detect or block on the receiving host. We examine how these protocols can be used to attack a victim.

Echo-Chargen

This attack works between two hosts. Chargen is a protocol that generates a stream of packets; it is used to test the network's capacity. The attacker sets up a chargen process on host A that generates its packets as echo packets with a destination of host B. Then, host A produces a stream of packets to which host B replies by echoing them back to host A. This series puts the network infrastructures of A and B into an endless loop. If the attacker makes B both the source and destination address of the first packet, B hangs in a loop, constantly creating and replying to its own messages.

Ping of Death

A ping of death is a simple attack. Since ping requires the recipient to respond to the ping request, all the attacker needs to do is send a flood of pings to the intended victim. The attack is limited by the smallest bandwidth on the attack route. If the attacker is on a 10-megabyte (MB) connection and the path to the victim is 100 MB or more, the attacker cannot mathematically flood the victim alone. But the attack succeeds if the numbers are reversed: The attacker on a 100-MB connection can easily flood a 10-MB victim. The ping packets will saturate the victim's bandwidth.

Smurf

The smurf attack is a variation of a ping attack. It uses the same vehicle, a ping packet, with two extra twists. First, the attacker chooses a network of unwitting victims. The attacker spoofs the source address in the ping packet so that it appears to come from the victim. Then, the attacker sends this request to the network in broadcast mode by setting the last byte of the address to all 1s; broadcast mode packets are distributed to all hosts on the network. The attack is shown in Figure 7-16.

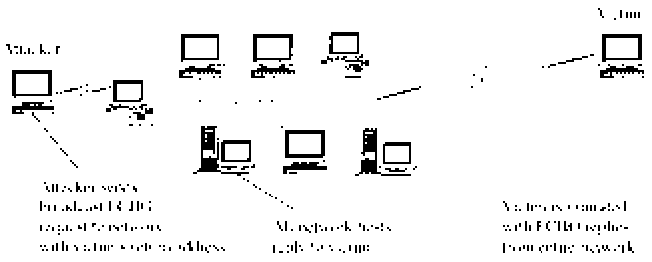
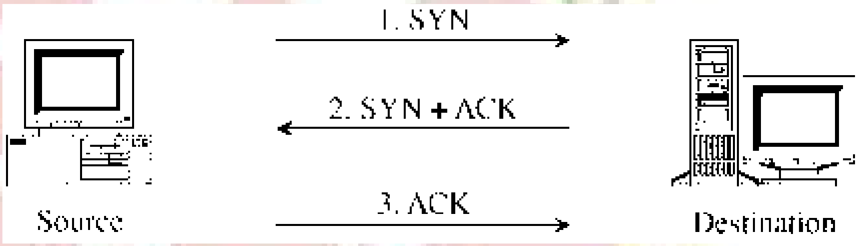


Figure 7-16. Smurf Attack.

Syn Flood

Another popular denial-of-service attack is the syn flood. This attack uses the TCP protocol suite, making the session-oriented nature of these protocols work against the victim. For a protocol such as Telnet, the protocol peers establish a virtual connection, called a session, to synchronize the back-and-forth, command-response nature of the Telnet terminal emulation. A session is established with a three-way TCP handshake. Each TCP packet has flag bits, two of which are denoted SYN and ACK. To initiate a TCP connection, the originator sends a packet with the SYN bit on. If the recipient is ready to establish a connection, it replies with a packet with both the SYN and ACK bits on. The first party then completes the exchange to demonstrate a clear and complete communication channel by sending a packet with the ACK bit on, as shown in Figure 7-17.

Figure 7-17. Three-Way Connection Handshake



Occasionally packets get lost or damaged in transmission. The destination maintains a queue called the SYN_RECV connections, tracking those items for which a SYNACK has been sent but no corresponding ACK has yet been received. Normally, these connections are completed in a short time. If the SYNACK (2) or the ACK (3) packet is lost, eventually the destination host will time out the incomplete connection and discard it from its waiting queue.

The attacker can deny service to the target by sending many SYN requests and never responding with ACKs, thereby filling the victim's SYN_RECV queue. Typically, the SYN_RECV queue is quite small, such as 10 or 20 entries. Because of potential routing delays in the Internet, typical holding times for the SYN_RECV queue can be minutes. So the attacker need only send a new SYN request every few seconds and it will fill the queue.

Attackers using this approach usually do one more thing: They spoof the nonexistent return address in the initial SYN packet. Why? For two reasons. First, the attacker does not want to disclose the real source address in case someone should inspect the packets in the SYN_RECV queue to try to identify the attacker. Second, the attacker wants to make the SYN packets indistinguishable from legitimate SYN packets to establish real connections. Choosing a different (spoofed) source address for each one makes them unique. A SYNACK packet to a nonexistent address results in an ICMP Destination Unreachable response, but this is not the ACK for which the TCP connection is waiting. (Remember that TCP and ICMP are different protocol suites, so an ICMP reply does not necessarily get back to the sender's TCP handler.)

Teardrop

The teardrop attack misuses a feature designed to improve network communication. A network IP datagram is a variable-length object. To support different applications and conditions, the datagram protocol permits a single data unit to be fragmented, that is, broken into pieces and transmitted separately. Each fragment indicates its length and relative position within the data unit. The receiving end is responsible for reassembling the fragments into a single data unit.

In the teardrop attack, the attacker sends a series of datagrams that cannot fit together properly. One datagram might say it is position 0 for length 60 bytes, another position 30 for 90 bytes, and another position 41 for 173 bytes. These three pieces overlap, so they cannot be reassembled properly. In an extreme case, the operating system locks up with these partial data units it cannot reassemble, thus leading to denial of service.

For more on these and other denial of service threats, see [CER99 and MAR05].

Traffic Redirection

As we saw earlier, at the network layer, a router is a device that forwards traffic on its way through intermediate networks between a source host's network and a destination's network. So if an attacker can corrupt the routing, traffic can disappear. Routers use complex algorithms to decide how to route traffic. No matter the algorithm, they essentially seek the best path (where "best" is measured in some combination of distance, time, cost, quality, and the like). Routers are aware only of the routers with which they share a direct network connection, and they use gateway protocols to share information about their capabilities. Each router advises its neighbors about how well it can reach other network addresses. This characteristic allows an attacker to disrupt the network.

To see how, keep in mind that, in spite of its sophistication, a router is simply a computer with two or more network interfaces. Suppose a router advertises to its neighbors that it has the best path to every other address in the whole network. Soon all routers will direct all traffic to that one router. The one router may become flooded, or it may simply drop much of its traffic.

In either case, a lot of traffic never makes it to the intended destination.

DNS Attacks

Our final denial-of-service attack is actually a class of attacks based on the concept of domain name server. A domain name server (DNS) is a table that converts domain names like ATT.COM into network addresses like 211.217.74.130; this process is called resolving the domain name. A domain name server queries other name servers to resolve domain names it does not know. For efficiency, it caches the answers it receives so it can resolve that name more rapidly in the future. A pointer to a DNS server can be retained for weeks or months.

In the most common implementations of Unix, name servers run software called Berkeley Internet Name Domain or BIND or named (a shorthand for "name daemon"). There have been numerous flaws in BIND, including the now-familiar buffer overflow.

By overtaking a name server or causing it to cache spurious entries (called DNS cache poisoning), an attacker can redirect the routing of any traffic, with an obvious implication for denial of service.

In October 2002, a massive flood of traffic inundated the top-level domain DNS servers, the servers that form the foundation of the Internet addressing structure. Roughly half the traffic came from just 200 addresses. Although some people think the problem was a set of misconfigured firewalls, nobody knows for sure what caused the attack.

An attack in March 2005 used a flaw in a Symantec firewall to allow a change in the DNS records used on Windows machines. The objective of this attack was not denial of service, however. In this attack, the poisoned DNS cache redirected users to advertising sites that received money from clients each time a user visited the site. Nevertheless, the attack also prevented users from accessing the legitimate sites.

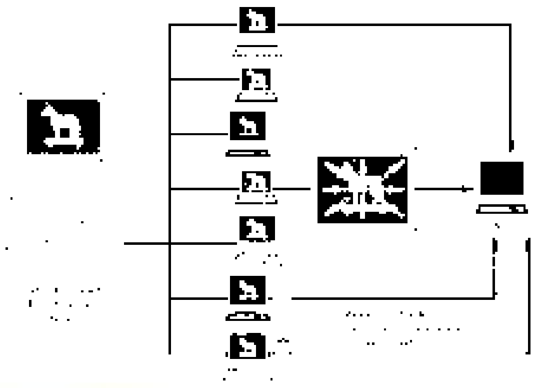
Distributed Denial of Service

The denial-of-service attacks we have listed are powerful by themselves, and Sidebar 7-7 shows us that many are launched. But an attacker can construct a two-stage attack that multiplies the effect many times. This multiplicative effect gives power to distributed denial of service.

To perpetrate a distributed denial-of-service (or DDoS) attack, an attacker does two things, as illustrated in Figure 7-18. In the first stage, the attacker uses any convenient attack (such as exploiting a buffer overflow or tricking the victim to open and install unknown code from an e-mail attachment) to plant a Trojan horse on a target machine. That Trojan horse does not necessarily cause any harm to the target machine, so it may not be noticed.

The Trojan horse file may be named for a popular editor or utility, bound to a standard operating system service, or entered into the list of processes (daemons) activated at startup. No matter how it is situated within the system, it will probably not attract any attention.

Figure 7-18. Distributed Denial-of-Service Attack.



The attacker repeats this process with many targets. Each of these target systems then becomes what is known as a zombie. The target systems carry out their normal work, unaware of the resident zombie.

At some point the attacker chooses a victim and sends a signal to all the zombies to launch the attack. Then, instead of the victim's trying to defend against one denial-of-service attack from one malicious host, the victim must try to counter *n* attacks from the *n* zombies all acting at once. Not all of the zombies need to use the same attack; for instance, some could use smurf attacks and others, could use syn floods to address different potential weaknesses.

In addition to their tremendous multiplying effect, distributed denial-of-service attacks are a serious problem because they are easily launched from scripts. Given a collection of denial-of-service attacks and a Trojan horse propagation method, one can easily write a procedure to plant a Trojan horse that can launch any or all of the denial-of-service attacks.

DDoS attack tools first appeared in mid-1999. Some of the original DDoS tools include Tribal Flood Network (TFN), Trin00, and TFN2K (Tribal Flood Network, year 2000 edition). As new vulnerabilities are discovered that allow Trojan horses to be planted and as new denial-of-service attacks are found, new combination tools appear. For more details on this topic, see [HAN00a].

According to the U.S. Computer Emergency Response Team (CERT) [HOU01a], scanning to find a vulnerable host (potential zombie) is now being included in combination tools; a single tool now identifies its zombie, installs the Trojan horse, and activates the zombie to wait for an attack signal. Recent target (zombie) selection has been largely random, meaning that attackers do not seem to care which zombies they infect. This revelation is actually bad news, because it means that no organization or accessible host is safe from attack. Perhaps because they are so numerous and because their users are assumed to be less knowledgeable about computer management and protection, Windows-based machines are becoming more popular targets for attack than other systems. Most frightening is the CERT finding that the time is shrinking between discovery of a vulnerability and its widespread exploitation.

Threats in Active or Mobile Code

Active code or mobile code is a general name for code that is pushed to the client for execution. Why should the web server waste its precious cycles and bandwidth doing simple work that the client's workstation can do? For example, suppose you want your web site to have bears dancing across the top of the page. To download the dancing bears, you could download a new image for each movement the bears take: one bit forward, two bits forward, and so forth. However, this approach uses far too much server time and bandwidth to compute the positions and download new images. A more efficient use of (server) resources is to download a program that runs on the client's machine and implements the movement of the bears.

Since you have been studying security and are aware of vulnerabilities, you probably are saying to yourself, "You mean a site I don't control, which could easily be hacked by teenagers, is going to push code to my machine that will execute without my knowledge, permission, or oversight?" Welcome to the world of (potentially malicious) mobile code. In fact, there are many different kinds of active code, and in this section we look at the related potential vulnerabilities.

Cookies

Strictly speaking, cookies are not active code. They are data files that can be stored and fetched by a remote server. However, cookies can be used to cause unexpected data transfer from a client to a server, so they have a role in a loss of confidentiality.

A cookie is a data object that can be held in memory (a per-session cookie) or stored on disk for future access (a persistent cookie). Cookies can store anything about a client that the browser can determine: keystrokes the user types, the machine name, connection details (such as IP address), date and type, and so forth. On command a browser will send to a server the cookies saved for it. Per-session cookies are deleted when the browser is closed, but persistent cookies are retained until a set expiration date, which can be years in the future.

Cookies provide context to a server. Using cookies, certain web pages can greet you with "Welcome back, James Bond" or reflect your preferences, as in "Shall I ship this order to you at 135 Elm Street?" But as these two examples demonstrate, anyone possessing someone's cookie becomes that person in some contexts. Thus, anyone intercepting or retrieving a cookie can impersonate the cookie's owner.

What information about you does a cookie contain? Even though it is your information, most of the time you cannot tell what is in a cookie, because the cookie's contents are encrypted under a key from the server. So a cookie is something that takes up space on your disk, holding information about you that you cannot see, forwarded to servers you do not know whenever the server wants it, without informing you. The philosophy behind cookies seems to be "Trust us, it's good for you."

Scripts

Clients can invoke services by executing scripts on servers. Typically, a web browser displays a page. As the user interacts with the web site via the browser, the browser organizes user inputs into parameters to a defined script; it then sends the script and parameters to a server to be executed. But all communication is done through HTML. The server cannot distinguish between commands generated from a user at a browser completing a web page and a user's handcrafting a set of orders. The malicious user can monitor the communication between a browser and a server to see how changing a web page entry affects what the browser sends and then how the server reacts. With this knowledge, the malicious user can manipulate the server's actions.

To see how easily this manipulation is done, remember that programmers do not often anticipate malicious behavior; instead, programmers assume that users will be benign and will use a program in the way it was intended to be used. For this reason, programmers neglect to filter script parameters to ensure that they are reasonable for the operation and safe to execute. Some scripts allow arbitrary files to be included or arbitrary commands to be executed. An attacker can see the files or commands in a string and experiment with changing them.

A well-known attack against web servers is the escape-character attack. A common scripting language for web servers, CGI (Common Gateway Interface), defines a machine-independent way to encode communicated data. The coding convention uses %nn to represent ASCII special characters. However, special characters may be interpreted by CGI script interpreters. So, for example, %0A (end-of-line) instructs the interpreter to accept the following characters as a new command. The following command requests a copy of the server's password file:

```
http://www.test.com/cgi-bin/query?%0a/bin/cat%20/etc/passwd
```

CGI scripts can also initiate actions directly on the server. For example, an attacker can observe a CGI script that includes a string of this form:

```
<a-#action arg1=value arg2=value ->
```

and submit a subsequent command where the string is replaced by

```
<a--#exec cmd="rm *" ->
```

to cause a command shell to execute a command to remove all files in the shell's current directory.

Microsoft uses active server pages (ASP) as its scripting capability. Such pages instruct the browser on how to display files, maintain context, and interact with the server. These pages can also be viewed at the browser end, so any programming weaknesses in the ASP code are available for inspection and attack.

The server should never trust anything received from a client, because the remote user can send the server a string crafted by hand, instead of one generated by a benign procedure the server sent the client. As with so many cases of remote access, these examples demonstrate that if you allow someone else to run a program on your machine, you can no longer be confident that your machine is secure.

Active Code

Displaying web pages started simply with a few steps: generate text, insert images, and register mouse clicks to fetch new pages. Soon, people wanted more elaborate action at their web sites: toddlers dancing atop the page, a three-dimensional rotating cube, images flashing on and off, colors changing, totals appearing. Some of these tricks, especially those involving movement, take significant computing power; they require a lot of time and communication to download from a server. But typically, the client has a capable and underutilized processor, so the timing issues are irrelevant.

To take advantage of the processor's power, the server may download code to be executed on the client. This executable code is called active code. The two main kinds of active code are Java code and ActiveX controls.

Java Code

Sun Microsystems [GOS96] designed and promoted the Java technology as a truly machine-independent programming language. A Java program consists of Java byte-code executed on a Java virtual machine (JVM) program. The bytecode programs are machine independent, and only the JVM interpreter needs to be implemented on each class of machine to achieve program portability. The JVM interpreter contains a built-in security manager that enforces a security policy. A Java program runs in a Java "sandbox," a constrained resource domain from which the program cannot escape. The Java programming language is strongly typed, meaning that the content of a data item must be of the appropriate type for which it is to be used (for example, a text string cannot be used as a numeric).

The original, Java 1.1 specification was very solid, very restrictive, and hence very unpopular. In it, a program could not write permanently to disk, nor could it invoke arbitrary procedures that had not been included in the sandbox by the security manager's policy. Thus, the sandbox was a collection of resources the user was willing to sacrifice to the uncertainties of Java code. Although very strong, the Java 1.1 definition proved unworkable. As a result, the original restrictions on the sandbox were relaxed, to the detriment of security. Koved et al. [KOV98] describe how the Java security model evolved.

The Java 1.2 specification opened the sandbox to more resources, particularly to stored disk files and executable procedures. (See, for example, [GON96, GON97].) Although it is still difficult to break its constraints, the Java sandbox contains many new toys, enabling more interesting computation but opening the door to exploitation of more serious vulnerabilities.(For more information, see [DEA96] and review the work of the Princeton University Secure Internet Programming group, *http://www.cs.princeton.edu/sip/history/index.php3*.)

Does this mean that the Java system's designers made bad decisions? No. As we have seen many times before, a product's security flaw is not necessarily a design flaw. Sometimes the designers choose to trade some security for increased functionality or ease of use. In other cases, the design is fine, but implementers fail to uphold the high security standards set out by designers. The latter is certainly true for Java technology. Problems have occurred with implementations of Java virtual machines for different platforms and in different components.

For example, a version of Netscape browser failed to implement type checking on all data types, as is required in the Java specifications. A similar vulnerability affected Microsoft Internet Explorer. Although these vulnerabilities have been patched, other problems could occur with subsequent releases.

A hostile applet is downloadable Java code that can cause harm on the client's system. Because an applet is not screened for safety when it is downloaded and because it typically runs with the privileges of its invoking user, a hostile applet can cause serious damage. Dean et al. [DEA96] list necessary conditions for secure execution of applets:

- The system must control applets' access to sensitive system resources, such as the file system, the processor, the network, the user's display, and internal state variables.
- The language must protect memory by preventing forged memory pointers and array (buffer) overflows.
- The system must prevent object reuse by clearing memory contents for new objects; the system should perform garbage collection to reclaim memory that is no longer in use.
- The system must control interapplet communication as well as applets' effects on the environment outside the Java system through system calls.

ActiveX Controls

Microsoft's answer to Java technology is the ActiveX series. Using ActiveX controls, objects of arbitrary type can be downloaded to a client. If the client has a viewer or handler for the object's type, that viewer is invoked to present the object. For example, downloading a Microsoft Word .doc file would invoke Microsoft Word on a system on which it is installed. Files for which the client has no handler cause other code to be downloaded. Thus, in theory, an attacker could invent a type, called .bomb, and cause any unsuspecting user who downloaded a web page with a .bomb file also to download code that would execute .bombs.

To prevent arbitrary downloads, Microsoft uses an authentication scheme under which downloaded code is cryptographically signed and the signature is verified before execution. But the authentication verifies only the source of the code, not its correctness or safety. Code from Microsoft (or Netscape or any other manufacturer) is not inherently safe, and code from an unknown source may be more or less safe than that from a known source. Proof of origin shows where it came from, not how good or safe it is. And some vulnerabilities allow ActiveX to bypass the authentication.

Auto Exec by Type Data files are processed by programs. For some products, the file type is implied by the file extension, such as .doc for a Word document, .pdf (Portable Document Format) for an Adobe Acrobat file, or .exe for an executable file. On many systems, when a file arrives with one of these extensions, the operating system automatically invokes the appropriate processor to handle it.

By itself, a Word document is unintelligible as an executable file. To prevent someone from running a file temp.doc by typing that name as a command, Microsoft embeds within a file what type it really is. Double-clicking the file in a Windows Explorer window brings up the appropriate program to handle that file.

But, as we noted in Chapter 3, this scheme presents an opportunity to an attacker. A malicious agent might send you a file named innocuous.doc, which you would expect to be a Word document. Because of the .doc extension, Word would try to open it. Suppose that file is renamed "innocuous" (without a .doc). If the embedded file type is .doc, then double-clicking innocuous also brings the file up in Word. The file might contain malicious macros or invoke the opening of another, more dangerous file.

Generally, we recognize that executable files can be dangerous, text files are likely to be safe, and files with some active content, such as .doc files, fall in between. If a file has no apparent file type and will be opened by its built-in file handler, we are treading on dangerous ground. An attacker can disguise a malicious active file under a nonobvious file type.

Bots

Bots, hackerese for robots, are pieces of malicious code under remote control. These code objects are Trojan horses that are distributed to large numbers of victims' machines. Because they may not interfere with or harm a user's computer (other than consuming computing and network resources), they are often undetected.

Bots coordinate with each other and with their master through ordinary network channels, such as Internet Relay Chat (IRC) channels or peer-to-peer networking (which has been used for sharing music over the Internet). Structured as a loosely coordinated web, a network of bots, called a botnet, is not subject to failure of any one bot or group of bots, and with multiple channels for communication and coordination, they are highly resilient.

Botnets are used for distributed denial-of-service attacks, launching attacks from many sites in parallel against a victim. They are also used for spam and other bulk email attacks, in which an extremely large volume of e-mail from any one point might be blocked by the sending service provider.

Complex Attacks

As if these vulnerabilities were not enough, two other phenomena multiply the risk. Scripts let people perform attacks even if the attackers do not understand what the attack is or how it is performed. Building blocks let people combine components of an attack, almost like building a house from prefabricated parts.

Script Kiddies

Attacks can be scripted. A simple smurf denial-of-service attack is not hard to implement. But an underground establishment has written scripts for many of the popular attacks. With a script, attackers need not understand the nature of the attack or even the concept of a network. The attackers merely download the attack script (no more difficult than downloading a newspaper story from a list of headlines) and execute it. The script takes care of selecting an appropriate (that is, vulnerable) victim and launching the attack. The hacker community is active in creating scripts for known vulnerabilities. For example, within three weeks of a CERT advisory for a serious SNMP vulnerability in February 2002 [CER02], scripts had appeared. These scripts probed for the vulnerability's existence in specific brands and models of network devices; then they executed attacks when a vulnerable host was found.

People who download and run attack scripts are called script kiddies. As the rather derogatory name implies, script kiddies are not well respected in the attacker community because the damage they do requires almost no creativity or innovation. Nevertheless, script kiddies can cause serious damage, sometimes without even knowing what they do.

Building Blocks

This chapter's attack types do not form an exhaustive list, but they represent the kinds of vulnerabilities being exploited, their sources, and their severity. A good attacker knows these vulnerabilities and many more.

An attacker simply out to cause minor damage to a randomly selected site could use any of the techniques we have described, perhaps under script control. A dedicated attacker who targets one location can put together several pieces of an attack to compound the damage.

Often, the attacks are done in series so that each part builds on the information gleaned from previous attacks. For example, a wiretapping attack may yield reconnaissance information with which to form an ActiveX attack that transfers a Trojan horse that monitors for sensitive data in transmission. Putting the attack pieces together like building blocks expands the number of targets and increases the degree of damage.

Summary of Network Vulnerabilities

A network has many different vulnerabilities, but all derive from an underlying model of computer, communications, and information systems security. Threats are raised against the key aspects of security: confidentiality, integrity, and availability, as shown in Table 7-4.

Table 7-4. Network Vulnerabilities.

Target Vulnerability

Precursors to attack

- Port scan
- Social engineering
- Reconnaissance
- OS and application fingerprinting

Authentication failures

- Impersonation
- Guessing
- Eavesdropping
- Spoofing
- Session hijacking
- Man-in-the-middle attack

Programming flaws

- Buffer overflow
- Addressing errors
- Parameter modification, time-of-check to time-of-use errors
- Server-side include
- Cookie
- Malicious active code: Java, ActiveX
- Malicious code: virus, worm, Trojan horse
- Malicious typed code

Confidentiality

- Protocol flaw
- Eavesdropping
- Passive wiretap

 Misdelivery

Table 7-4. Network Vulnerabilities.

Target Vulnerability

 Exposure within the network
 Traffic flow analysis
 Cookie

Integrity

 Protocol flaw
 Active wiretap
 Impersonation
 Falsification of message
 Noise
 Web site defacement
 DNS attack

Availability

 Protocol flaw
 Transmission or component failure
 Connection flooding, e.g., echo-chargen, ping of death,smurf, syn flood
 DNS attack
 Traffic redirection
 Distributed denial of service

Network security control

The list of security attacks is long, and the news media carry frequent accounts of serious security incidents. From these, you may be ready to conclude that network security is hopeless. Fortunately, that is not the case. Previous chapters have presented several strategies for addressing security concerns, such as encryption for confidentiality and integrity, reference monitors for access control, and overlapping controls for defense in depth.

These strategies are also useful in protecting networks. This section presents many excellent defenses available to the network security engineer. Subsequent sections provide detailed explanations for three particularly important controlsfirewalls, intrusion detection systems, and encrypted e-mail.

Security Threat Analysis

Recall the three steps of a security threat analysis in other situations. First, we scrutinize all the parts of a system so that we know what each part does and how it interacts with other parts. Next, we consider possible damage to confidentiality, integrity, and availability. Finally, we hypothesize the kinds of attacks that could cause this damage. We can take the same steps with a network. We begin by looking at the individual parts of a network:

 local nodes connected via
 local communications links to a
 local area network, which also has
 local data storage,
 local processes, and
 local devices.

The local network is also connected to a

 network gateway which gives access via
 network communications links to
 network control resources,
 network routers, and
 network resources, such as databases.

These functional needs are typical for network users. But now we look again at these parts, this time conjuring up the negative effects threat agents can cause. We posit a malicious agent call him Hector who wants to attack networked communications between two users, Andy and Bo. What might Hector do?

 Read communications. The messages sent and received are exposed inside Andy's machine, at all places through the network, and inside Bo's machine. Thus, a confidentiality attack can be mounted from practically any place in the network.
 Modify communications from Andy to Bo. Again, the messages are exposed at all places through the network.
 Forge communications allegedly from Andy to Bo. This action is even easier than modifying a communication because a forgery can be inserted at any place in the network. It need not originate with the ostensible sender, and it does not require that a communication be caught in transit. Since Andy does not deliver his communications personally and since Bo might even never have met Andy, Bo has little basis for judging whether a communication purportedly sent by Andy is authentic.
 Inhibit communications from Andy to Bo. Here again, Hector can achieve this result by invading Andy's machine, Bo's machine, routers between them, or communications links. He can also disrupt communications in general by flooding the network or disrupting any unique path on the network.
 Inhibit all communications passing through a point. If the point resides on a unique path to or from a node, all traffic to or from that node is blocked. If the path is not unique, blocking it shifts traffic to other nodes, perhaps overburdening them.
 Read data at some machine C between Andy and Bo. Hector can impersonate Andy (who is authorized to access data at C). Bo might question a message that seems out of character for Andy, but machine C will nevertheless apply the access controls for Andy. Alternatively, Hector can invade (run a program on) machine C to override access controls. Finally, he can search the network for machines that have weak or improperly administered access controls.
 Modify or *destroy data* at C. Here again, Hector can impersonate Andy and do anything Andy could do. Similarly, Hector can try to circumvent controls.
We summarize these threats with a list:
 intercepting data in traffic
 accessing programs or data at remote hosts
 modifying programs or data at remote hosts
 modifying data in transit
 inserting communications
 impersonating a user
 inserting a repeat of a previous communication
 blocking selected traffic
 blocking all traffic
 running a program at a remote host

Why are all these attacks possible? Size, anonymity, ignorance, misunderstanding, complexity, dedication, and programming all contribute. But we have help at hand; we look next at specific threats and their countermeasures. Later in this chapter we investigate how these countermeasures fit together into specific tools.

Design and Implementation

Throughout this book we have discussed good principles of system analysis, design, implementation, and maintenance. Chapter 3, in particular, presented techniques that have been developed by the software engineering community to improve requirements, design, and code quality. Concepts from the work of the early trusted operating systems projects (presented in Chapter 5) have natural implications for networks as well. And assurance, also discussed in Chapter 5, relates to networked systems. In general, the Open Web Applications project [OWA02, OWA05] has documented many of the techniques people can use to develop secure web applications. Thus, having addressed secure programming from several perspectives already, we do not belabor the points now.

Architecture

As with so many of the areas we have studied, planning can be the strongest control. In particular, when we build or modify computer-based systems, we can give some thought to their overall architecture and plan to "build in" security as one of the key constructs. Similarly, the architecture or design of a network can have a significant effect on its security.

Segmentation

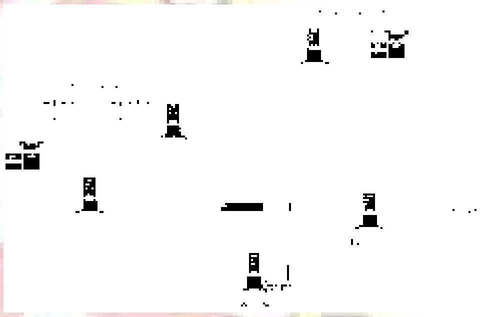
Just as segmentation was a powerful security control in operating systems, it can limit the potential for harm in a network in two important ways: Segmentation reduces the number of threats, and it limits the amount of damage a single vulnerability can allow.

Assume your network implements electronic commerce for users of the Internet. The fundamental parts of your network may be

 a web server, to handle users' HTTP sessions
 application code, to present your goods and services for purchase
 a database of goods, and perhaps an accompanying inventory to the count of stock on hand and being requested from suppliers
 a database of orders taken If all these activities were to run on one machine, your network would be in trouble: Any compromise or failure of that machine would destroy your entire commerce capability.

A more secure design uses multiple segments, as shown in Figure 7-19. Suppose one piece of hardware is to be a web server box exposed to access by the general public. To reduce the risk of attack from outside the system, that box should not also have other, more sensitive, functions on it, such as user authentication or access to a sensitive data repository. Separate segments and servers corresponding to the principles of least privilege and encapsulation reduce the potential harm should any subsystem be compromised.

Figure 7-19. Segmented Architecture.



Separate access is another way to segment the network. For example, suppose a network is being used for three purposes: using the "live" production system, testing the next production version, and developing subsequent systems. If the network is well segmented, external users should be able to access only the live system, testers should access only the test system, and developers should access only the development system. Segmentation permits these three populations to coexist without risking that, for instance, a developer will inadvertently change the production system.

Redundancy

Another key architectural control is redundancy: allowing a function to be performed on more than one node, to avoid "putting all the eggs in one basket." For example, the design of Figure 7-19 has only one web server; lose it and all connectivity is lost. A better design would have two servers, using what is called failover mode. In failover mode the servers communicate with each other periodically, each determining if the other is still active. If one fails, the other takes over processing for both of them. Although performance is cut approximately in half when a failure occurs, at least some processing is being done.

Single Points of Failure

Ideally, the architecture should make the network immune to failure. In fact, the architecture should at least make sure that the system tolerates failure in an acceptable way (such as slowing down but not stopping processing, or recovering and restarting incomplete transactions). One way to evaluate the network architecture's tolerance of failure is to look for single points of failure. That is, we should ask if there is a single point in the network that, if it were to fail, could deny access to all or a significant part of the network. So, for example, a single database in one location is vulnerable to all the failures that could affect that location. Good network design eliminates single points of failure. Distributing the database placing copies of it on different network segments, perhaps even in different physical locations can reduce the risk of serious harm from a failure at any one point. There is often substantial overhead in implementing such a design; for example, the independent databases must be synchronized. But usually we can deal with the failure-tolerant features more easily than with the harm caused by a failed single link.

Architecture plays a role in implementing many other controls. We point out architectural features as we introduce other controls throughout the remainder of this chapter.

Mobile Agents

Mobile code and hostile agents are potential methods of attack, as described earlier in this chapter. However, they can also be forces for good. Good agents might look for unsecured wireless access, software vulnerabilities, or embedded malicious code. Schneider and Zhou [SCH05] investigate distributed trust, through a corps of communicating, state-sharing agents. The idea is straightforward: Just as with soldiers, you know some agents will be stopped and others will be subverted by the enemy, but some agents will remain intact. The corps can recover from Byzantine failures [LAM82]. Schneider and Zhou propose a design in which no one agent is critical to the overall success but the overall group can be trusted.

Encryption

Encryption is probably the most important and versatile tool for a network security expert. We have seen in earlier chapters that encryption is powerful for providing privacy, authenticity, integrity, and limited access to data. Because networks often involve even greater risks, they often secure data with encryption, perhaps in combination with other controls.

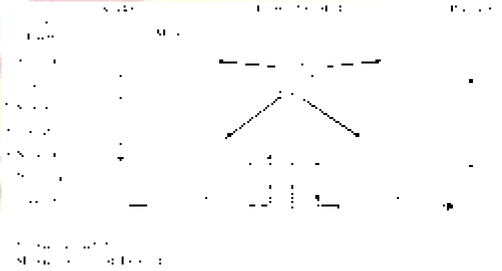
Before we begin to study the use of encryption to counter network security threats, let us consider these points. First, remember that encryption is not a panacea or silver bullet. A flawed system design with encryption is still a flawed system design. Second, notice that encryption protects only what is encrypted (which should be obvious but isn't). Data are exposed between a user's fingertips and the encryption process before they are transmitted, and they are exposed again once they have been decrypted on the remote end. The best encryption cannot protect against a malicious Trojan horse that intercepts data before the point of encryption. Finally, encryption is no more secure than its key management. If an attacker can guess or deduce a weak encryption key, the game is over. People who do not understand encryption sometimes mistake it for fairy dust to sprinkle on a system for magic protection. This book would not be needed if such fairy dust existed.

In network applications, encryption can be applied either between two hosts (called link encryption) or between two applications (called end-to-end encryption). We consider each below. With either form of encryption, key distribution is always a problem. Encryption keys must be delivered to the sender and receiver in a secure manner. In this section, we also investigate techniques for safe key distribution in networks. Finally, we study a cryptographic facility for a network computing environment.

Link Encryption

In link encryption, data are encrypted just before the system places them on the physical communications link. In this case, encryption occurs at layer 1 or 2 in the OSI model. (A similar situation occurs with TCP/IP protocols.) Similarly, decryption occurs just as the communication arrives at and enters the receiving computer. A model of link encryption is shown in Figure 7-20.

Figure 7-20. Link Encryption.



Encryption protects the message in transit between two computers, but the message is in plaintext inside the hosts. (A message in plaintext is said to be "in the clear.") Notice that because the encryption is added at the bottom protocol layer, the message is exposed in all other layers of the sender and receiver. If we have good physical security, we may not be too concerned about this exposure; the exposure occurs on the sender's or receiver's host or workstation, protected by alarms or locked doors, for example. Nevertheless, you should notice that the message is exposed in two layers of all intermediate hosts through which the message may pass. This exposure occurs because routing and addressing are not read at the bottom layer, but only at higher layers. The message is in the clear in the intermediate hosts, and one of these hosts may not be especially trustworthy.

Link encryption is invisible to the user. The encryption becomes a transmission service performed by a low-level network protocol layer, just like message routing or transmission error detection. Figure 7-21 shows a typical link encrypted message, with the shaded fields encrypted. Because some of the data link header and trailer is applied before the block is encrypted, part of each of those blocks is shaded. As the message M is handled at each layer, header and control information is added on the sending side and removed on the receiving side. Hardware encryption devices operate quickly and reliably; in this case, link encryption is invisible to the operating system as well as to the operator.

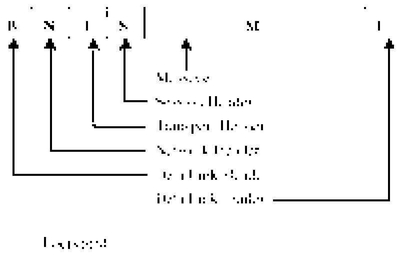


Figure 7-21. Message Under Link Encryption.

Link encryption is especially appropriate when the transmission line is the point of greatest vulnerability. If all hosts on a network are reasonably secure but the communications medium is shared with other users or is not secure, link encryption is an easy control to use.

End-to-End Encryption

As its name implies, end-to-end encryption provides security from one end of a transmission to the other. The encryption can be applied by a hardware device between the user and the host. Alternatively, the encryption can be done by software running on the host computer. In either case, the encryption is performed at the highest levels (layer 7, application, or perhaps at layer 6, presentation) of the OSI model. A model of end-to-end encryption is shown in Figure 7-22.

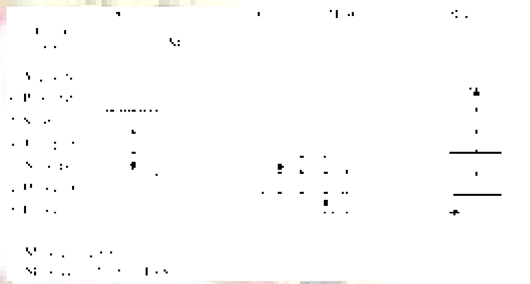


Figure 7-22. End-to-End Encryption.

Since the encryption precedes all the routing and transmission processing of the layer, the message is transmitted in encrypted form throughout the network. The encryption addresses potential flaws in lower layers in the transfer model. If a lower layer should fail to preserve security and reveal data it has received, the data's confidentiality is not endangered. Figure 7-23 shows a typical message with end-to-end encryption, again with the encrypted field shaded.

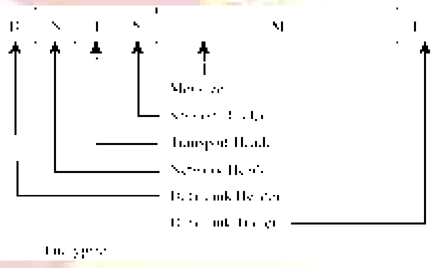


Figure 7-23. End-to-End Encrypted Message.

When end-to-end encryption is used, messages sent through several hosts are protected. The data content of the message is still encrypted, as shown in Figure 7-24, and the message is encrypted (protected against disclosure) while in transit. Therefore, even though a message must pass through potentially insecure nodes (such as C through G) on the path between A and B, the message is protected against disclosure while in transit.

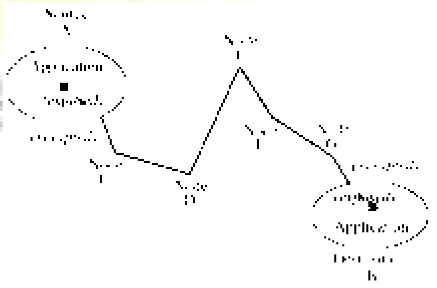


Figure 7-24. Encrypted Message Passing Through a Host.

Comparison of Encryption Methods Simply encrypting a message is not absolute assurance that it will not be revealed during or after transmission. In many instances, however, the strength of encryption is adequate protection, considering the likelihood of the interceptor's breaking the encryption and the timeliness of the message. As with many aspects of security, we must balance the strength of protection with the likelihood of attack. (You will learn more about managing these risks in Chapter 8.)

With link encryption, encryption is invoked for all transmissions along a particular link. Typically, a given host has only one link into a network, meaning that all network traffic initiated on that host will be encrypted by that host. But this encryption scheme implies that every other host receiving these communications must also have a cryptographic facility to decrypt the messages. Furthermore, all hosts must share keys. A message may pass through one or more intermediate hosts on the way to its final destination. If the message is encrypted along some links of a network but not others, then part of the advantage of encryption is lost. Therefore, link encryption is usually performed on all links of a network if it is performed at all.

By contrast, end-to-end encryption is applied to "logical links," which are channels between two processes, at a level well above the physical path. Since the intermediate hosts along a transmission path do not need to encrypt or decrypt a message, they have no need for cryptographic facilities. Thus, encryption is used only for those messages and applications for which it is needed. Furthermore, the encryption can be done with software, so we can apply it selectively, one application at a time or even to one message within a given application.

The selective advantage of end-to-end encryption is also a disadvantage regarding encryption keys. Under end-to-end encryption, there is a virtual cryptographic channel between each pair of users. To provide proper security, each pair of users should share a unique cryptographic key. The number of keys required is thus equal to the number of pairs of users, which is $n * (n - 1)/2$ for n users. This number increases rapidly as the number of users increases. However, this count assumes that single key encryption is used. With a public key system, only one pair of keys is needed per recipient.

As shown in Table 7-5, link encryption is faster, easier for the user, and uses fewer keys. End-to-end encryption is more flexible, can be used selectively, is done at the user level, and can be integrated with the application. Neither form is right for all situations.

Table 7-5. Comparison of Link and End-to-End Encryption.

Link Encryption	End-to-End Encryption
Security within hosts	Data exposed in sending host
Data encrypted in sending host	Data exposed in intermediate nodes
Data encrypted in intermediate nodes	Role of user
Applied by sending host	Applied by sending process
Invisible to user	User applies encryption
Host maintains encryption	User must find algorithm
One facility for all users	User selects encryption
Typically done in hardware	Either software or hardware implementation
All or no data encrypted	User chooses to encrypt or not,
for each data item	Implementation concerns

Table 7-5. Comparison of Link and End-to-End Encryption.

Link Encryption	End-to-End Encryption
Requires one key per host pair	Requires one key per user pair
Provides node authentication	Provides user authentication

In some cases, both forms of encryption can be applied. A user who does not trust the quality of the link encryption provided by a system can apply end-to-end encryption as well. A system administrator who is concerned about the security of an end-to-end encryption scheme applied by an application program can also install a link encryption device. If both encryptions are relatively fast, this duplication of security has little negative effect.

Virtual Private Networks

Link encryption can be used to give a network's users the sense that they are on a private network, even when it is part of a public network. For this reason, the approach is called a virtual private network (or VPN).

Typically, physical security and administrative security are strong enough to protect transmission inside the perimeter of a network. Thus, the greatest exposure for a user is between the user's workstation or client and the perimeter of the host network or server.

A firewall is an access control device that sits between two networks or two network segments. It filters all traffic between the protected or "inside" network and a less trustworthy or "outside" network or segment. (We examine firewalls in detail later in this chapter.)

Many firewalls can be used to implement a VPN. When a user first establishes a communication with the firewall, the user can request a VPN session with the firewall. The user's client and the firewall negotiate a session encryption key, and the firewall and the client subsequently use that key to encrypt all traffic between the two. In this way, the larger network is restricted only to those given special access by the VPN. In other words, it feels to the user that the network is private, even though it is not. With the VPN, we say that the communication passes through an encrypted tunnel or tunnel. Establishment of a VPN is shown in Figure 7-25.

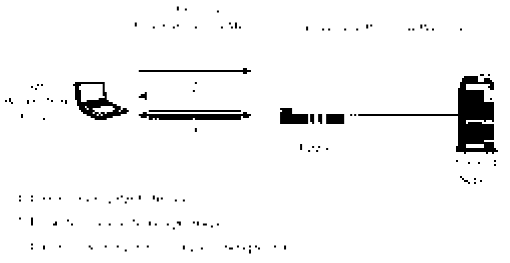


Figure 7-25. Establishing a Virtual Private Network.

Virtual private networks are created when the firewall interacts with an authentication service inside the perimeter. The firewall may pass user authentication data to the authentication server and, upon confirmation of the authenticated identity, the firewall provides the user with appropriate security privileges. For example, a known trusted person, such as an employee or a system administrator, may be allowed to access resources not available to general users. The firewall implements this access control on the basis of the VPN. A VPN with privileged access is shown in Figure 7-26. In that figure, the firewall passes to the internal server the (privileged) identity of User 2.

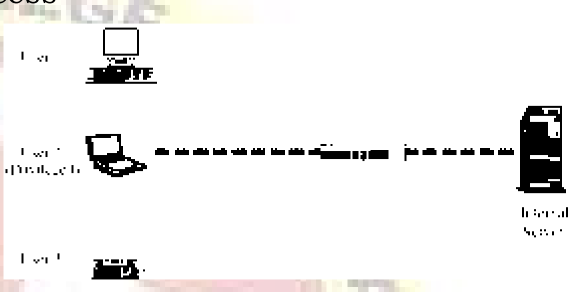


Figure 7-26. VPN to Allow Privileged Access

PKI and Certificates

A public key infrastructure, or PKI, is a process created to enable users to implement public key cryptography, usually in a large (and frequently, distributed) setting. PKI offers each user a set of services, related to identification and access control, as follows:

•. Create certificates associating a user's identity with a (public) cryptographic key
•. Give out certificates from its database
•. Sign certificates, adding its credibility to the authenticity of the certificate
•. Confirm (or deny) that a certificate is valid
•. Invalidate certificates for users who no longer are allowed access or whose private key has been exposed

PKI is often considered to be a standard, but in fact it is a set of policies, products, and procedures that leave some room for interpretation. (Housley and Polk [HOU01b] describe both the technical parts and the procedural issues in developing a PKI.) The policies define the rules under which the cryptographic systems should operate. In particular, the policies specify how to handle keys and valuable information and how to match level of control to level of risk.

The procedures dictate how the keys should be generated, managed, and used. Finally, the products actually implement the policies, and they generate, store, and manage the keys.

PKI sets up entities, called certificate authorities, that implement the PKI policy on certificates. The general idea is that a certificate authority is trusted, so users can delegate the construction, issuance, acceptance, and revocation of certificates to the authority, much as one would use a trusted bouncer to allow only some people to enter a restricted nightclub.

The specific actions of a certificate authority include the following:
 managing public key certificates for their whole life cycle
 issuing certificates by binding a user's or system's identity to a public key with a digital signature
 scheduling expiration dates for certificates
 ensuring that certificates are revoked when necessary by publishing certificate revocation lists

The functions of a certificate authority can be done in-house or by a commercial service or a trusted third party.

PKI also involves a registration authority that acts as an interface between a user and a certificate authority. The registration authority captures and authenticates the identity of a user and then submits a certificate request to the appropriate certificate authority. In this sense, the registration authority is much like the U.S. Postal Service; the postal service acts as an agent of the U.S. State Department to enable U.S. citizens to obtain passports (official U.S. authentication) by providing the appropriate forms, verifying identity, and requesting the actual passport (akin to a certificate) from the appropriate passport-issuing office (the certificate authority). As with passports, the quality of registration authority determines the level of trust that can be placed in the certificates that are issued. PKI fits most naturally in a hierarchically organized, centrally controlled organization, such as a government agency.

PKI efforts are under way in many countries to enable companies and government agencies to implement PKI and interoperate. For example, a Federal PKI Initiative in the United States will eventually allow any U.S. government agency to send secure communication to any other U.S. government agency, when appropriate. The initiative also specifies how commercial PKI-enabled tools should operate, so agencies can buy ready-made PKI products rather than build their own. The European Union has a similar initiative (see www.europepki.org for more information.) Sidebar 7-8 describes the commercial use of PKI in a major U.K. bank. Major PKI solutions vendors include Baltimore Technologies, Northern Telecom/Entrust, and Identrus.

Expand the notion of certificate to a broader characterization of credentials. For instance, a credit card company may be more interested in verifying your financial status than your identity; a PKI scheme may involve a certificate that is based on binding the financial status with a key. The Simple Distributed Security Infrastructure (SDSI) takes this approach, including identity certificates, group membership certificates, and name-binding certificates.

As of this writing, there are drafts of two related standards: ANSI standard X9.45 and the Simple Public Key Infrastructure (SPKI); the latter has only a set of requirements and a certificate format.

PKI is close to but not yet a mature process. Many issues must be resolved, especially since PKI has yet to be implemented commercially on a large scale. Table 7-6 lists several issues to be addressed as we learn more about PKI. However, some things have become clear. First, the certificate authority should be approved and verified by an independent body. The certificate authority's private key should be stored in a tamper-resistant security module.

Then, access to the certificate and registration authorities should be tightly controlled, by means of strong user authentication such as smart cards.

Table 7-6. Issues Relating to PKI.

Issue Questions
Flexibility
How do we implement interoperability and stay consistent with
other PKI implementations?
 Open, standard interfaces?
 Compatible security policies?
How do we register certificates?
 Face-to-face, e-mail, web, network?
 Single or batch (e.g., national identity cards, bank cards)?
Ease of use How do we train people to implement, use, maintain PKI?
How do we configure and integrate PKI?
How do we incorporate new users?
How do we do backup and disaster recovery?
Support for security policy
How does PKI implement an organization's security policy?
Who has which responsibilities?
Scalability How do we add more users?
Add more applications?
Add more certificate authorities?
Add more registration authorities?

Table 7-6. Issues Relating to PKI.

Issue Questions
How do we expand certificate types?
How do we expand registration mechanisms?

The security involved in protecting the certificates involves administrative procedures. For example, more than one operator should be required to authorize certification requests.

Controls should be put in place to detect hackers and prevent them from issuing bogus certificate requests. These controls might include digital signatures and strong encryption.

Finally, a secure audit trail is necessary for reconstructing certificate information should the system fail and for recovering if a hacking attack does indeed corrupt the authentication process.

SSH Encryption

SSH (secure shell) is a pair of protocols (versions 1 and 2), originally defined for Unix but also available under Windows 2000, that provides an authenticated and encrypted path to the shell or operating system command interpreter. Both SSH versions replace Unix utilities such as Telnet, *rlogin*, and *rsh* for remote access. SSH protects against spoofing attacks and modification of data in communication.

The SSH protocol involves negotiation between local and remote sites for encryption algorithm (for example, DES, IDEA, AES) and authentication (including public key and Kerberos).

SSL Encryption

The SSL (Secure Sockets Layer) protocol was originally designed by Netscape to protect communication between a web browser and server. It is also known now as TLS, for transport layer security. SSL interfaces between applications (such as browsers) and the TCP/IP protocols to provide server authentication, optional client authentication, and an encrypted communications channel between client and server. Client and server negotiate a mutually supported suite of encryption for session encryption and hashing; possibilities include triple DES and SHA1, or RC4 with a 128-bit key and MD5.

To use SSL, the client requests an SSL session. The server responds with its public key certificate so that the client can determine the authenticity of the server. The client returns part of a symmetric session key encrypted under the server's public key. Both the server and client compute the session key, and then they switch to encrypted communication, using the shared session key.

The protocol is simple but effective, and it is the most widely used secure communication protocol on the Internet. However, remember that SSL protects only from the client's browser to the server's decryption point (which is often only to the server's firewall or, slightly stronger, to the computer that runs the web application). Data are exposed from the user's keyboard to the browser and throughout the recipient's company. Blue Gem Security has developed a product called LocalSSL that encrypts data after it has been typed until the operating system delivers it to the client's browser, thus thwarting any keylogging Trojan horse that has become implanted in the user's computer to reveal everything the user types.

IPSec

As noted previously, the address space for the Internet is running out. As domain names and equipment proliferate, the original, 30-year-old, 32-bit address structure of the Internet is filling up. A new structure, called IPv6 (version 6 of the IP protocol suite), solves the addressing problem. This restructuring also offered an excellent opportunity for the Internet
Engineering Task Force (IETF) to address serious security requirements.

As a part of the IPv6 suite, the IETF adopted IPSec, or the IP Security Protocol Suite. Designed to address fundamental shortcomings such as being subject to spoofing, eavesdropping, and session hijacking, the IPSec protocol defines a standard means for handling encrypted data. IPSec is implemented at the IP layer, so it affects all layers above it, in particular TCP and UDP. Therefore, IPSec requires no change to the existing large number of TCP and UDP protocols.

IPSec is somewhat similar to SSL, in that it supports authentication and confidentiality in a way that does not necessitate significant change either above it (in applications) or below it (in the TCP protocols). Like SSL, it was designed to be independent of specific cryptographic protocols and to allow the two communicating parties to agree on a mutually supported set of protocols.

The basis of IPSec is what is called a security association, which is essentially the set of security parameters for a secured communication channel. It is roughly comparable to an SSL session. A security association includes

- encryption algorithm and mode (for example, DES in block-chaining mode)
- encryption key
- encryption parameters, such as the initialization vector
- authentication protocol and key
- lifespan of the association, to permit long-running sessions to select a new cryptographic key as often as needed
- address of the opposite end of association
- sensitivity level of protected data (usable for classified data)

A host, such as a network server or a firewall, might have several security associations in effect for concurrent communications with different remote hosts. A security association is selected by a security parameter index (SPI), a data element that is essentially a pointer into a table of security associations.

The fundamental data structures of IPSec are the AH (authentication header) and the ESP (encapsulated security payload). The ESP replaces (includes) the conventional TCP header and data portion of a packet, as shown in Figure 7-27. The physical header and trailer depend on the data link and physical layer communications medium, such as Ethernet.

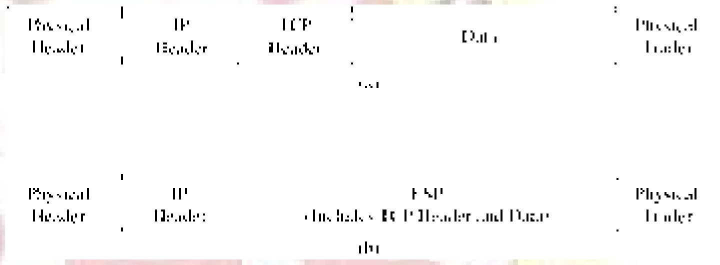


Figure 7-27. Packets: (a) Conventional Packet; (b) IPSec Packet.

The ESP contains both an authenticated portion and an encrypted portion, as shown in Figure 7-28. The sequence number is incremented by one for each packet transmitted to the same address using the same SPI, to preclude packet replay attacks. The payload data is the actual data of the packet. Because some encryption or other security mechanisms require blocks of certain sizes, the padding factor and padding length fields contain padding and the amount of padding to bring the payload data to an appropriate length. The next header indicates the type of payload data. The authentication field is used for authentication of the entire object.

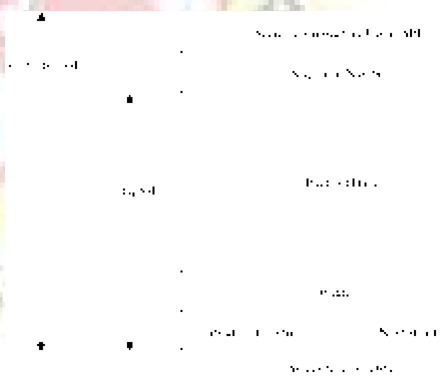


Figure 7-28. Encapsulated Security Packet.

As with most cryptographic applications, the critical element is key management. IPSec addresses this need with ISAKMP or Internet Security Association Key Management Protocol. Like SSL, ISAKMP requires that a distinct key be generated for each security association. The ISAKMP protocol is simple, flexible, and scalable. In IPSec, ISAKMP is implemented through IKE or ISAKMP key exchange. IKE provides a way to agree on and manage protocols, algorithms, and keys. For key exchange between unrelated parties IKE uses the Diffie Hellman scheme (also described in Chapter 2). In Diffie Hellman, each of the two parties, *X* and *Y*, chooses a large prime and sends a number *g* raised to the power of the prime to the other. That is, *X* sends g_x and *Y* sends g_y. They both raise what they receive to the power they kept: *Y* raises g_x to $(g_x)_y$ and *X* raises g_y to $(g_y)_x$, which are both the same;voilà, they share a secret $(g_x)_y = (g_y)_x$. (The computation is slightly more complicated, being done in a finite field *mod*(*n*), so an attacker cannot factor the secret easily.) With their shared secret, the two parties now exchange identities and certificates to authenticate those identities. Finally, they derive a shared cryptographic key and enter a security association.

The key exchange is very efficient: The exchange can be accomplished in two messages, with an optional two more messages for authentication. Because this is a public key method, only two keys are needed for each pair of communicating parties. IKE has sub modes for authentication (initiation) and for establishing new keys in an existing security association.

IPSec can establish cryptographic sessions with many purposes, including VPNs, applications, and lower-level network management (such as routing). The protocols of IPSec have been published and extensively scrutinized. Work on the protocols began in 1992. They were first published in 1995, and they were finalized in 1998 (RFCs 24012409) [KEN98].

Signed Code

As we have seen, someone can place malicious active code on a web site to be downloaded by unsuspecting users. Running with the privilege of whoever downloads it, such active code can do serious damage, from deleting files to sending e-mail messages to fetching Trojan horses to performing subtle and hard-to-detect mischief. Today's trend is to allow applications and updates to be downloaded from central sites, so the risk of downloading something malicious is growing.

A partial not complete approach to reducing this risk is to use signed code. A trustworthy third party appends a digital signature to a piece of code, supposedly connoting more trustworthy code. A signature structure in a PKI helps to validate the signature.

Who might the trustworthy party be? A well-known manufacturer would be recognizable as a code signer. But what of the small and virtually unknown manufacturer of a device driver or a code add-in? If the code vendor is unknown, it does not help that the vendor signs its own code; miscreants can post their own signed code, too.

In March 2001, Verisign announced it had erroneously issued two code-signing certificates under the name of Microsoft Corp. to someone who purported to bebut was nota Microsoft employee. These certificates were in circulation for almost two months before the error was detected. Even after Verisign detected the error and canceled the certificates, someone would know the certificates had been revoked only by checking Verisign's list. Most people would not question a code download signed by Microsoft.

Encrypted E-mail

An electronic mail message is much like the back of a post card. The mail carrier (and everyone in the postal system through whose hands the card passes) can read not just the address but also everything in the message field. To protect the privacy of the message and routing information, we can use encryption to protect the confidentiality of the message and perhaps its integrity.

As we have seen in several other applications, the encryption is the easy part; key management is the more difficult issue. The two dominant approaches to key management are the use of a hierarchical, certificate-based PKI solution for key exchange and the use of a flat, individual-to-individual exchange method. The hierarchical method is called S/MIME and is employed by many commercial mail-handling programs, such as Microsoft Exchange or Eudora.

The individual method is called PGP and is a commercial add-on. We look more carefully at encrypted e-mail in a later section of this chapter.

Content Integrity

Content integrity comes as a bonus with cryptography. No one can change encrypted data in a meaningful way without breaking the encryption. This does not say, however, that encrypted data cannot be modified. Changing even one bit of an encrypted data stream affects the result after decryption, often in a way that seriously alters the resulting plaintext.

We need to consider three potential threats:

 malicious modification that changes content in a meaningful way

 malicious or nonmalicious modification that changes content in a way that is not necessarily meaningful

 nonmalicious modification that changes content in a way that will not be detected

Encryption addresses the first of these threats very effectively. To address the others, we can use other controls.

Error Correcting Codes

We can use error detection and error correction codes to guard against modification in a transmission. The codes work as their names imply: Error detection codes detect when an error has occurred, and error correction codes can actually correct errors without requiring retransmission of the original message. The error code is transmitted along with the original data, so the recipient can recompute the error code and check whether the received result matches the expected value.

The simplest error detection code is a parity check. An extra bit is added to an existing group of data bits depending on their sum or an exclusive OR. The two kinds of parity are called even and odd. With even parity the extra bit is 0 if the sum of the data bits is even and 1 if the sum is odd; that is, the parity bit is set so that the sum of all data bits plus the parity bit is even. Odd parity is the same except the sum is odd. For example, the data stream 01101101 would have an even parity bit of 1 (and an odd parity bit of 0) because 0+1+1+0+1+1+0+1 = 5 + 1 = 6 (or 5 + 0 = 5 for odd parity). A parity bit can reveal the modification of a single bit. However, parity does not detect two-bit errorscases in which two bits in a group are changed. That is, the use of a parity bit relies on the assumption that single-bit errors will occur infrequently, so it is very unlikely that two bits would be changed.

Parity signals only that a bit has been changed; it does not identify which bit has been changed. There are other kinds of error detection codes, such as hash codes and Huffman codes. Some of the more complex codes can detect multiple-bit errors (two or more bits changed in a data group) and may be able to pinpoint which bits have been changed.

Parity and simple error detection and correction codes are used to detect nonmalicious changes in situations in which there may be faulty transmission equipment, communications noise and interference, or other sources of spurious changes to data.

Cryptographic Checksum

Malicious modification must be handled in a way that prevents the attacker from modifying the error detection mechanism as well as the data bits themselves. One way to do this is to use a technique that shrinks and transforms the data, according to the value of the data bits.

To see how such an approach might work, consider an error detection code as a many-to-one transformation. That is, any error detection code reduces a block of data to a smaller digest whose value depends on each bit in the block. The proportion of reduction (that is, the ratio of original size of the block to transformed size) relates to the code's effectiveness in detecting errors. If a code reduces an 8-bit data block to a 1-bit result, then half of the 2_8 input values map to 0 and half to 1, assuming a uniform distribution of outputs. In other words, there are $2_8/2 = 2_7 = 128$ different bit patterns that all produce the same 1-bit result.

The fewer inputs that map to a particular output, the fewer ways the attacker can change an input value without affecting its output. Thus, a 1-bit result is too weak for many applications. If the output is three bits instead of one, then each output result comes from $2_8 /2_3$ or $2_5 = 32$ inputs. The smaller number of inputs to a given output is important for blocking malicious modification.

A cryptographic checksum (sometimes called a message digest) is a cryptographic function that produces a checksum. The cryptography prevents the attacker from changing the data block (the plaintext) and also changing the checksum value (the ciphertext) to match. Two major uses of cryptographic checksums are code tamper protection and message integrity protection in transit. For code protection, a system administrator computes the checksum of each program file on a system and then later computes new checksums and compares the values. Because executable code usually does not change, the administrator can detect unanticipated changes from, for example, malicious code attacks. Similarly, a checksum on data in communication identifies data that have been changed in transmission, maliciously or accidentally.

Strong Authentication

As we have seen in earlier chapters, operating systems and database management systems enforce a security policy that specifies whowhich individuals, groups, subjectscan access which resources and objects. Central to that policy is authentication: knowing and being assured of the accuracy of identities.

Networked environments need authentication, too. In the network case, however, authentication may be more difficult to achieve securely because of the possibility of eavesdropping and wiretapping, which are less common in nonnetworked environments. Also, both ends of a communication may need to be authenticated to each other: Before you send your password across a network, you want to know that you are really communicating with the remote host you expect. Lampson [LAM00] presents the problem of authentication in autonomous, distributed systems; the real problem, he points out, is how to develop trust of network entities with which you have no basis for a relationship. Let us look more closely at authentication methods appropriate for use in networks.

One-Time Password

The wiretap threat implies that a password could be intercepted from a user who enters a password across an unsecured network. A one-time password can guard against wiretapping and spoofing of a remote host.

As the name implies, a one-time password is good for one use only. To see how it works, consider the easiest case, in which the user and host both have access to identical lists of passwords, like the one-time pad for cryptography from Chapter 2. The user would enter the first password for the first login, the next one for the next login, and so forth. As long as the password lists remained secret and as long as no one could guess one password from another, a password obtained through wiretapping would be useless. However, as with the one-time cryptographic pads, humans have trouble maintaining these password lists.

To address this problem, we can use a password token, a device that generates a password that is unpredictable but that can be validated on the receiving end. The simplest form of password token is a synchronous one, such as the SecurID device from RSA Security, Inc.

This device displays a random number, generating a new number every minute. Each user is issued a different device (that generates a different random number sequence). The user reads the number from the device's display and types it in as a one-time password. The computer on the receiving end executes the algorithm to generate the password appropriate for the current minute; if the user's password matches the one computed remotely, the user is authenticated. Because the devices may get out of alignment if one clock runs slightly faster than the other, these devices use fairly natural rules to account for minor drift.

What are the advantages and disadvantages of this approach? First, it is easy to use. It largely counters the possibility of a wiretapper reusing a password. With a strong password-generating algorithm, it is immune to spoofing. However, the system fails if the user loses the generating device or, worse, if the device falls into an attacker's hands. Because a new password is generated only once a minute, there is a small (one-minute) window of vulnerability during which an eavesdropper can reuse an intercepted password.

Challenge Response Systems

To counter the loss and reuse problems, a more sophisticated one-time password scheme uses challenge and response, as we first studied in Chapter 4. A challenge and response device looks like a simple pocket calculator. The user first authenticates to the device, usually by means of a PIN. The remote system sends a random number, called the "challenge," which the user enters into the device. The device responds to that number with another number, which the user then transmits to the system.

The system prompts the user with a new challenge for each use. Thus, this device eliminates the small window of vulnerability in which a user could reuse a time-sensitive authenticator. A generator that falls into the wrong hands is useless without the PIN. However, the user must always have the response generator to log in, and a broken device denies service to the user.

Finally, these devices do not address the possibility of a rogue remote host. Digital Distributed Authentication In the 1980s, Digital Equipment Corporation recognized the problem of needing to authenticate nonhuman entities in a computing system. For example, a process might retrieve a user query, which it then reformats, perhaps limits, and submits to a database manager. Both the database manager and the query processor want to be sure that a particular communication channel is authentic between the two. Neither of these servers is running under the direct control or supervision of a human (although each process was, of course, somehow initiated by a human). Human forms of access control are thus inappropriate.

Digital [GAS89, GAS90] created a simple architecture for this requirement, effective against the following threats:

- *impersonation* of a server by a rogue process, for either of the two servers involved in the authentication
- *interception or modification* of data exchanged between servers

- *replay* of a previous authentication

The architecture assumes that each server has its own private key and that the corresponding public key is available to or held by every other process that might need to establish an authenticated channel. To begin an authenticated communication between server A and server B, A sends a request to B, encrypted under B's public key. B decrypts the request and replies with a message encrypted under A's public key. To avoid replay, A and B can append a random number to the message to be encrypted.

A and B can establish a private channel by one of them choosing an encryption key (for a secret key algorithm) and sending it to the other in the authenticating message. Once the authentication is complete, all communication under that secret key can be assumed to be as secure as was the original dual public key exchange. To protect the privacy of the channel, Gasser recommends a separate cryptographic processor, such as a smart card, so that private keys are never exposed outside the processor.

Two implementation difficulties remain to be solved: (a) How can a potentially large number of public keys be distributed and (b) how can the public keys be distributed in a way that ensures the secure binding of a process with the key? Digital recognized that a key server (perhaps with multiple replications) was necessary to distribute keys. The second difficulty is addressed with certificates and a certification hierarchy, as described in Chapter 2.

Both of these design decisions are to a certain degree implied by the nature of the rest of the protocol. A different approach was taken by Kerberos, as we see in the following sections.

Kerberos

As we introduced in Chapter 4, Kerberos is a system that supports authentication in distributed systems. Originally designed to work with secret key encryption, Kerberos, in its latest version, uses public key technology to support key exchange. The Kerberos system was designed at Massachusetts Institute of Technology [STE88, KOH93].

Kerberos is used for authentication between intelligent processes, such as client-to-server tasks, or a user's workstation to other hosts. Kerberos is based on the idea that a central server provides authenticated tokens, called tickets, to requesting applications. A ticket is an unforgeable, nonreplayable, authenticated object. That is, it is an encrypted data structure naming a user and a service that user is allowed to obtain. It also contains a time value and some control information.

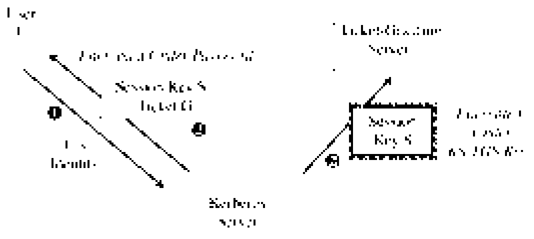
The first step in using Kerberos is to establish a session with the Kerberos server, as shown in Figure 7-29. A user's workstation sends the user's identity to the Kerberos server when a user logs in. The Kerberos server verifies that the user is authorized. The Kerberos server sends two messages:

1. to the user's workstation, a session key S_G for use in communication with the ticket-granting server (G) and a ticket T_G for the ticket-granting server; S_G is encrypted under the user's password: $E(S_G + T_G, pw)$[4]

[4] In Kerberos version 5, only SG is encrypted; in Kerberos version 4, both the session key and the ticket were encrypted when returned to the user.

2. to the ticket-granting server, a copy of the session key S_G and the identity of the user (encrypted under a key shared between the Kerberos server and the ticket-granting server)

Figure 7-29. Initiating a Kerberos Session.

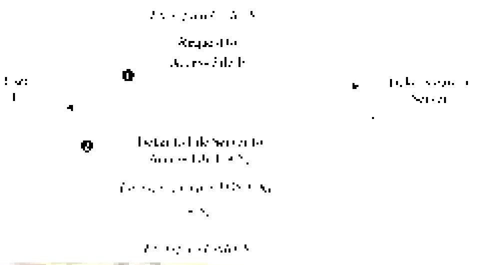


If the workstation can decrypt $E(S_G + T_G, pw)$ by using pw, the password typed by the user, then the user has succeeded in an authentication with the workstation.

Notice that passwords are stored at the Kerberos server, *not* at the workstation, and that the user's password did not have to be passed across the network, even in encrypted form. Holding passwords centrally but not passing them across the network is a security advantage.

Next, the user will want to exercise some other services of the distributed system, such as accessing a file. Using the key S_G provided by the Kerberos server, the user U requests a ticket to access file F from the ticket-granting server. As shown in Figure 7-30, after the ticket-granting server verifies U's access permission, it returns a ticket and a session key. The ticket contains U's authenticated identity (in the ticket U obtained from the Kerberos server), an identification of F (the file to be accessed), the access rights (for example, to read), a session key S_F for the file server to use while communicating this file to U, and an expiration date for the ticket. The ticket is encrypted under a key shared exclusively between the ticket-granting server and the file server. This ticket cannot be read, modified, or forged by the user U (or anyone else). The ticket-granting server must, therefore, also provide U with a copy of S_F, the session key for the file server. Requests for access to other services and servers are handled similarly.

Figure 7-30. Obtaining a Ticket to Access a File.



Kerberos was carefully designed to withstand attacks in distributed environments:

 No passwords communicated on the network. As already described, a user's password is stored only at the Kerberos server. The user's password is not sent from the user's workstation when the user initiates a session. (Obviously, a user's initial password must be sent outside the network, such as in a letter.)

 Cryptographic protection against spoofing. Each access request is mediated by the ticket-granting server, which knows the identity of the requester, based on the authentication performed initially by the Kerberos server and on the fact that the user was able to present a request encrypted under a key that had been encrypted under the user's password.

 Limited period of validity. Each ticket is issued for a limited time period; the ticket contains a timestamp with which a receiving server will determine the ticket's validity.

In this way, certain long-term attacks, such as brute force cryptanalysis, will usually be neutralized because the attacker will not have time to complete the attack.

 Timestamps to prevent replay attacks. Kerberos requires reliable access to a universal clock. Each user's request to a server is stamped with the time of the request. A server receiving a request compares this time to the current time and fulfills the request only if the time is reasonably close to the current time. This time-checking prevents most replay attacks, since the attacker's presentation of the ticket will be delayed too long.

 Mutual authentication. The user of a service can be assured of any server's authenticity by requesting an authenticating response from the server. The user sends a ticket to a server and then sends the server a request encrypted under the session key for that server's service; the ticket and the session key were provided by the ticket-granting server. The server can decrypt the ticket only if it has the unique key it shares with the ticket-granting server. Inside the ticket is the session key, which is the only means the server has of decrypting the user's request. If the server can return to the user a message encrypted under this same session key but containing 1 + the user's timestamp, the server must be authentic. Because of this mutual authentication, a server can provide a unique channel to a user and the user may not need to encrypt communications on that channel to ensure continuous authenticity.

Avoiding encryption saves time in the communication.

Kerberos is not a perfect answer to security problems in distributed systems.

 Kerberos requires continuous availability of a trusted ticket-granting server. Because the ticket-granting server is the basis of access control and authentication, constant access to that server is crucial. Both reliability (hardware or software failure) and performance (capacity and speed) problems must be addressed.

 Authenticity of servers requires a trusted relationship between the ticket-granting server and every server. The ticket-granting server must share a unique encryption key with each "trustworthy" server. The ticket-granting server (or that server's human administrator) must be convinced of the authenticity of that server. In a local environment, this degree of trust is warranted. In a widely distributed environment, an administrator at one site can seldom justify trust in the authenticity of servers at other sites.

 Kerberos requires timely transactions. To prevent replay attacks, Kerberos limits the validity of a ticket. A replay attack could succeed during the period of validity, however. And setting the period fairly is hard: Too long increases the exposure to replay attacks, while too short requires prompt user actions and risks providing the user with a ticket that will not be honored when presented to a server. Similarly, subverting a server's clock allows reuse of an expired ticket.

 A subverted workstation can save and later replay user passwords. This vulnerability exists in any system in which passwords, encryption keys, or other constant, sensitive information is entered in the clear on a workstation that might be subverted.

 Password guessing works. A user's initial ticket is returned under the user's password. An attacker can submit an initial authentication request to the Kerberos server and then try to decrypt the response by guessing at the password.

 Kerberos does not scale well. The architectural model of Kerberos, shown in Figure 7-31, assumes one Kerberos server and one ticket-granting server, plus a collection of other servers, each of which shares a unique key with the ticket-granting server.

Adding a second ticket-granting server, for example, to enhance performance or reliability, would require duplicate keys or a second set for all servers. Duplication increases the risk of exposure and complicates key updates, and second keys more than double the work for each server to act on a ticket.

 Kerberos is a complete solution. All applications must use Kerberos authentication and access control. Currently, few applications use Kerberos authentication, and so integration of Kerberos into an existing environment requires modification of existing applications, which is not feasible.

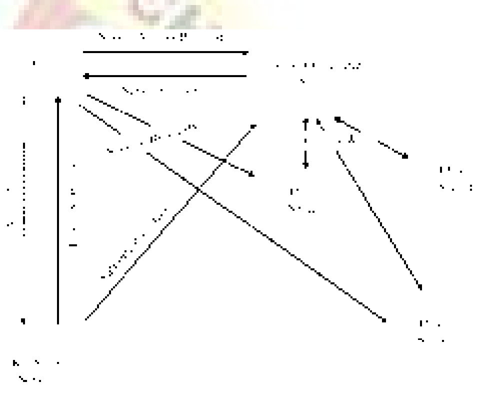


Figure 7-31. Access to Services and Servers in Kerberos.

Access Controls

Authentication deals with the *who* of security policy enforcement; access controls enforce the *what* and *how.*

ACLs on Routers Routers perform the major task of directing network traffic either to subnetworks they control or to other routers for subsequent delivery to other subnetworks. Routers convert external IP addresses into internal MAC addresses of hosts on a local subnetwork.

Suppose a host is being spammed (flooded) with packets from a malicious rogue host. Routers can be configured with access control lists to deny access to particular hosts from particular hosts. So, a router could delete all packets with a source address of the rogue host and a destination address of the target host.

This approach has three problems, however. First, routers in large networks perform a lot of work: They have to handle every packet coming into and going out of the network. Adding ACLs to the router requires the router to compare every packet against the ACLs. One ACL adds work, degrading the router's performance; as more ACLs are added, the router's performance may become unacceptable. The second problem is also an efficiency issue:

Because of the volume of work they perform, routers are designed to perform only essential services. Logging of activity is usually not done on a router because of the volume of traffic and the performance penalty logging would entail. With ACLs, it would be useful to know how many packets were being deleted, to know if a particular ACL could be removed (thereby improving performance). But without logging it is impossible to know whether an ACL is being used. These two problems together imply that ACLs on routers are most effective against specific known threats but that they should not be used indiscriminately.

The final limitation on placing ACLs on routers concerns the nature of the threat. A router inspects only source and destination addresses. An attacker usually does not reveal an actual source address. To reveal the real source address would be equivalent to a bank robber's leaving his home address and a description of where he plans to store the stolen money.

Because someone can easily forge any source address on a UDP datagram, many attacks use UDP protocols with false source addresses so that the attack cannot be blocked easily by a router with an ACL. Router ACLs are useful only if the attacker sends many datagrams with the same forged source address.

In principle, a router is an excellent point of access control because it handles every packet coming into and going out of a subnetwork. In specific situations, primarily for internal subnetworks, ACLs can be used effectively to restrict certain traffic flows, for example, to ensure that only certain hosts (addresses) have access to an internal network management subnetwork. But for large-scale, general traffic screening, routers are less useful than firewalls.

Firewalls

A firewall does the screening that is less appropriate for a router to do. A router's primary function is addressing, whereas a firewall's primary function is filtering. Firewalls can also do auditing. Even more important, firewalls can examine an entire packet's contents, including the data portion, whereas a router is concerned only with source and destination MAC and IP addresses. Because they are an extremely important network security control, we study firewalls in an entire section later in this chapter.

Wireless Security

Because wireless computing is so exposed, it requires measures to protect communications between a computer (called the client) and a wireless base station or access point.

Remembering that all these communications are on predefined radio frequencies, you can expect an eavesdropping attacker to try to intercept and impersonate. Pieces to protect are finding the access point, authenticating the remote computer to the access point, and vice versa, and protecting the communication stream.

SSID

As described earlier in this chapter, the Service Set Identifier or SSID is the identification of an access point; it is a string of up to 32 characters. Obviously the SSIDs need to be unique in a given area to distinguish one wireless network from another. The factory-installed default for early versions of wireless access points was not unique, such as "wireless," "tsunami" or "Linksys" (a brand name); now most factory defaults are a serial number unique to the device.

A client and an access point engage in a handshake to locate each other: Essentially the client says, "I am looking to connect to access point S" and the access point says, "I am access point S; connect to me." The order of these two steps is important. In what is called "open mode," an access point can continually broadcast its appeal, indicating that it is open for the next step in establishing a connection. Open mode is a poor security practice because it advertises the name of an access point to which an attacker might attach. "Closed" or "stealth mode" reverses the order of the protocol: The client must send a signal seeking an access point with a particular SSID before the access point responds to that one query with an invitation to connect.

But closed mode does not prevent knowledge of the SSID. The initial exchange "looking for S,"

"I am S" occurs in the clear and is available to anyone who uses a sniffer to intercept wireless communications in range. Thus, anyone who sniffs the SSID can save the SSID (which is seldom changed in practice) to use later.

WEP

The second step in securing a wireless communication involves use of encryption. The original 802.11 wireless standard relied upon a cryptographic protocol called wired equivalent privacy or WEP. WEP was meant to provide users privacy equivalent to that of a dedicated wire, that is, immunity to most eavesdropping and impersonation attacks. WEP uses an encryption key shared between the client and the access point. To authenticate a user, the access point sends a random number to the client, which the client encrypts using the shared key and returns to the access point. From that point on, the client and access point are authenticated and can communicate using their shared encryption key. Several problems exist with this seemingly simple approach.

First, the WEP standard uses either a 64- or 128-bit encryption key. The user enters the key in any convenient form, usually in hexadecimal or as an alphanumeric string that is converted to a number. Entering 64 or 128 bits in hex requires choosing and then typing 16 or 32 symbols correctly for the client and access point. Not surprisingly, hex strings like C0DE C0DE... (that is a zero between C and D) are common. Passphrases are vulnerable to a dictionary attack.

Even if the key is strong, it really has an effective length of only 40 or 104 bits because of the way it is used in the algorithm. A brute force attack against a 40-bit key succeeds quickly. Even for the 104-bit version, flaws in the RC4 algorithm and its use (see [BOR01, FLU01, and ARB02]) defeat WEP security. Several tools, starting with WEPCrack and AirSnort, allow an attacker to crack a WEP encryption, usually in a few minutes. At a 2005 conference, the FBI demonstrated the ease with which a WEP-secured wireless session can be broken.

For these reasons, in 2001 the IEEE began design of a new authentication and encryption scheme for wireless. Unfortunately, some wireless devices still on the market allow only the false security of WEP.

WPA and WPA2

The alternative to WEP is WiFi Protected Access or WPA, approved in 2003. The IEEE standard 802.11i is now known as WPA2, approved in 2004, and is an extension of WPA. How does WPA improve upon WEP?

First, WEP uses an encryption key that is unchanged until the user enters a new key at the client and access point. Cryptologists hate unchanging encryption keys because a fixed key gives the attacker a large amount of ciphertext to try to analyze and plenty of time in which to analyze it. WPA has a key change approach, called Temporal Key Integrity Program (TKIP), by which the encryption key is changed automatically on each packet.

Second, WEP uses the encryption key as an authenticator, albeit insecurely. WPA employs the extensible authentication protocol (EAP) by which authentication can be done by password, token, certificate, or other mechanism. For small network (home) users, this probably still means a shared secret, which is not ideal. Users are prone to selecting weak keys, such as short numbers or pass phrases subject to a dictionary attack.

The encryption algorithm for WEP is RC4, which has cryptographic flaws both in key length and design [ARB02]. In WEP the initialization vector for RC4 is only 24 bits, a size so small that collisions commonly occur; furthermore, there is no check against initialization vector reuse.

WPA2 adds AES as a possible encryption algorithm (although RC4 is also still supported for compatibility reasons).

WEP includes a 32-bit integrity check separate from the data portion. But because the WEP encryption is subject to cryptanalytic attack [FLU01], the integrity check was also subject, so an attacker could modify content and the corresponding check without having to know the associated encryption key [BOR01]. WPA includes a 64-bit integrity check that is encrypted.

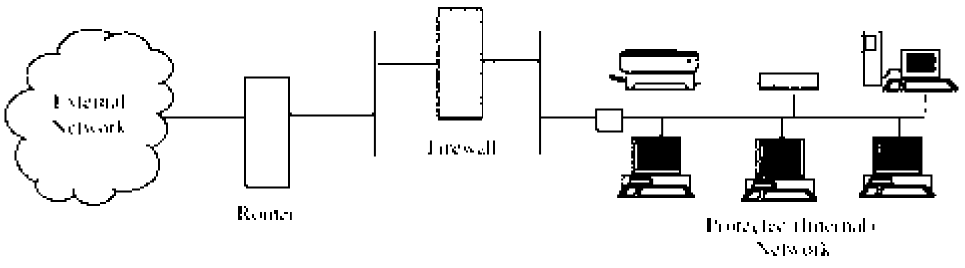
The setup protocol for WPA and WPA2 is much more robust than that for WEP. Setup for WPA involves three protocol steps: authentication, a four-way handshake (to ensure that the client can generate cryptographic keys and to generate and install keys for both encryption and integrity on both ends), and an optional group key handshake (for multicast communication.) A good overview of the WPA protocols is in [LEH05].

WPA and WPA2 address the security deficiencies known in WEP. Arazi et al. [ARA05] make a strong case for public key cryptography in wireless sensor networks, and a similar argument can be made for other wireless applications (although the heavier computation demands of public key encryption is a limiting factor on wireless devices with limited processor capabilities.)

Alarms and Alerts

The logical view of network protection looks like Figure 7-32, in which both a router and a firewall provide layers of protection for the internal network. Now let us add one more layer to this defense.

Figure 7-32. Layered Network Protection.



Monitor what occurs within the network. If an attacker passes through the router and passes through the firewall, an intrusion detection system offers the opportunity to detect the attack at the beginning, in progress, or after it has occurred. Intrusion detection systems activate an alarm, which can take defensive action. We study intrusion detection systems in more detail later in this chapter.

Honeypots

How do you catch a mouse? You set a trap with bait (food the mouse finds attractive) and catch the mouse after it is lured into the trap. You can catch a computer attacker the same way.

In a very interesting book, Cliff Stoll [STO89] details the story of attracting and monitoring the actions of an attacker. Cheswick [CHE90, CHE02] and Bellovin [BEL92c] tell a similar story.

These two cases describe the use of a honeypot: a computer system open to attackers.

You put up a honeypot for several reasons:

- to watch what attackers do, in order to learn about new attacks (so that you can strengthen your defenses against these new attacks)
- to lure an attacker to a place in which you may be able to learn enough to identify and stop the attacker
- to provide an attractive but diversionary playground, hoping that the attacker will leave your real system alone

A honeypot has no special features. It is just a computer system or a network segment loaded with servers and devices and data. It may be protected with a firewall, although you want the attackers to have some access. There may be some monitoring capability, done carefully so that the monitoring is not evident to the attacker.

The two difficult features of a honeypot are putting up a believable, attractive false environment and confining and monitoring the attacker surreptitiously. Spitzner [SPI02, SPI03a] has done extensive work developing and analyzing honeypots. He thinks like the attacker, figuring what the attacker will want to see in an invaded computer, but as McCarty [MCC03] points out, it is always a race between attacker and defender. Spitzner also tries to move much of his data off the target platform so that the attacker will not be aware of the analysis and certainly not be able to modify or erase the data gathered. Raynal [RAY04a. RAY04b] discusses how to analyze the data collected.

Traffic Flow Security

So far, we have looked at controls that cover the most common network threats: cryptography for eavesdropping, authentication methods for impersonation, intrusion detection systems for attacks in progress, architecture for structural flaws. Earlier in this chapter, we listed threats, including a threat of traffic flow inference. If the attacker can detect an exceptional volume of traffic between two points, the attacker may infer the location of an event about to occur.

The countermeasure to traffic flow threats is to disguise the traffic flow. One way to disguise traffic flow, albeit costly and perhaps crude, is to ensure a steady volume of traffic between two points. If traffic between A and B is encrypted so that the attacker can detect only the number of packets flowing, A and B can agree to pass recognizable (to them) but meaningless encrypted traffic. When A has much to communicate to B, there will be few meaningless packets; when communication is light, A will pad the traffic stream with many spurious packets.

A more sophisticated approach to traffic flow security is called onion routing [SYV97]. Consider a message that is covered in multiple layers, like the layers of an onion. A wants to send a message to B but doesn't want anyone in or intercepting traffic on the network to know A is communicating with B. So A takes the message to B, wraps it in a package for D to send to B. Then, A wraps that package in another package for C to send to D. Finally, A sends this package to C. This process is shown in Figure 7-33. The internal wrappings are all encrypted under a key appropriate for the intermediate recipient.

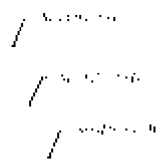


Figure 7-33. Onion Routing.

Receiving the package, C knows it came from A, although C does not know if A is the originator or an intermediate point. C then unwraps the outer layer and sees it should be sent to D. At this point, C cannot know if D is the final recipient or merely an intermediary. C sends the message to D, who unwraps the next layer. D knows neither where the package originally came from nor where its final destination is. D forwards the package to B, its ultimate recipient.

With this scheme, any intermediate recipientsthose other than the original sender and ultimate receiverknow neither where the package originated nor where it will end up. This scheme provides confidentiality of content, source, destination, and routing.

Controls Review

At the end of our earlier discussion on threats in networks, we listed in Table 7-4 many of the vulnerabilities present in networks. Now that we have surveyed the controls available for networks, we repeat that table as Table 7-7, adding a column to show the controls that can protect against each vulnerability. (Note: This table is not exhaustive; other controls can be used against some of the vulnerabilities.)

Table 7-7. Network Vulnerabilities and Controls.

Target Vulnerability Control
Precursors to attack
□ Port scan □ Firewall
□ Intrusion detection system
□ Running as few services as possible
□ Services that reply withonly what is necessary
□ Social engineering
□ Education, user awareness
□ Policies and procedures
□ Systems in which two people must agree to perform certain
security-critical functions
□ Reconnaissance
□ Firewall
□ "Hardened" (self-defensive) operating system and applications
□ Intrusion detection system
□ OS and application
fingerprinting
□ Firewall
□ "Hardened" (self-defensive) applications
□ Programs that reply with only what is necessary
□ Intrusion detection system
Authentication failures
□ Impersonation
□ Strong, one-time authentication

Table 7-7. Network Vulnerabilities and Controls.

Target Vulnerability Control
□ Guessing
□ Strong, one-time authentication
□ Education, user awareness
□ Eavesdropping
□ Strong, one-time authentication
□ Encrypted authentication channel
□ Spoofing
□ Strong, one-time authentication
□ Session hijacking
□ Strong, one-time authentication
□ Encrypted authentication channel
□ Virtual private network
□ Man-in-the-middle attack
□ Strong, one-time authentication
□ Virtual private network
□ Protocol analysis
Programming flaws

- Buffer overflow
- Programming controls
- Intrusion detection system
- Controlled execution environment
- Personal firewall
- Addressing errors
- Programming controls
- Intrusion detection system
- Controlled execution environment
- Personal firewall
- Two-way authentication

Table 7-7. Network Vulnerabilities and Controls.

Target Vulnerability Control

- Parameter modification, time-of-check to time-of-use errors
- Programming controls
- Intrusion detection system
- Controlled execution environment
- Intrusion detection system
- Personal firewall
- Server-side include
- Programming controls
- Personal firewall
- Controlled execution environment
- Intrusion detection system
- Cookie
- Firewall
- Intrusion detection system
- Controlled execution environment
- Personal firewall
- Malicious active code: Java, ActiveX
- Intrusion detection system
- Programming controls
- Signed code
- Malicious code: virus, worm, Trojan horse
- Intrusion detection system
- Signed code
- Controlled execution environment
- Intrusion detection system
- Malicious typed code
- Signed code
- Intrusion detection system
- Controlled execution environment

Confidentiality

- Protocol flaw
- Programming controls
- Controlled execution

Table 7-7. Network Vulnerabilities and Controls.

Target Vulnerability Control environment

- Eavesdropping
- Encryption
- Passive wiretap
- Encryption
- Misdelivery
- Encryption
- Exposure within the network
- End-to-end encryption
- Traffic flow analysis
- Encryption
- Traffic padding

- Onion routing
- Cookie
- Firewall
- Intrusion detection system
- Controlled execution environment

Integrity

- Protocol flaw
- Firewall
- Controlled execution environment
- Intrusion detection system
- Protocol analysis
- Audit
- Active wiretap
- Encryption
- Error detection code
- Audit
- Impersonation
- Firewall
- Strong, one-time authentication
- Encryption
- Error detection code
- Audit

Table 7-7. Network Vulnerabilities and Controls.

Target Vulnerability Control

- Falsification of message
- Firewall
- Encryption
- Strong authentication
- Error detection code
- Audit
- Noise
- Error detection code
- Web site defacement
- Error detection code
- Intrusion detection system
- Controlled execution environment
- Hardened host
- Honeypot
- Audit
- DNS attack
- Firewall
- Intrusion detection system
- Strong authentication for DNS changes
- Audit

Availability

- Protocol flaw
- Firewall
- Redundant architecture
- Transmission or component failure
- Architecture
- Connection flooding, e.g., echo-chargen, ping of death, smurf, syn flood
- Firewall
- Intrusion detection system
- ACL on border router
- Honeypot
- DNS attack
- Firewall
- Intrusion detection system

Table 7-7. Network Vulnerabilities and Controls.

Target Vulnerability Control

- ACL on border router
- Honeypot
- Traffic redirection
- Encryption
- Audit
- Distributed denial of service
- Firewall
- Intrusion detection system
- ACL on border router
- Honeypot

As Table 7-7 shows, network security designers have many successful tools at their disposal. Some of these, such as encryption, access control and authentication, and programming controls, are familiar from previous chapters in this book.

But three are specific to networked settings, and we explore them now in greater depth: firewalls, intrusion detection systems, and encrypted e-mail. Firewalls control traffic flow into and out of protected network segments. Intrusion detection systems monitor traffic within a network to spot potential attacks under way or about to occur. And encrypted email uses encryption to enhance the confidentiality or authenticity of e-mail messages.

Firewalls

A firewall is a device that filters all traffic between a protected or "inside" network and a less trustworthy or "outside" network. Usually a firewall runs on a dedicated device; because it is a single point through which traffic is channeled, performance is important, which means nonfirewall functions should not be done on the same machine. Because a firewall is executable code, an attacker could compromise that code and execute from the firewall's device. Thus, the fewer pieces of code on the device, the fewer tools the attacker would have by compromising the firewall. Firewall code usually runs on a proprietary or carefully minimized operating system.

The purpose of a firewall is to keep "bad" things outside a protected environment. To accomplish that, firewalls implement a security policy that is specifically designed to address what bad things might happen. For example, the policy might be to prevent any access from outside (while still allowing traffic to pass *from* the inside *to* the outside). Alternatively, the policy might permit accesses only from certain places, from certain users, or for certain activities. Part of the challenge of protecting a network with a firewall is determining which security policy meets the needs of the installation.

People in the firewall community (users, developers, and security experts) disagree about how a firewall should work. In particular, the community is divided about a firewall's default behavior. We can describe the two schools of thought as "that which is not expressly forbidden is permitted" (default permit) and "that which is not expressly permitted is forbidden" (default deny). Users, always interested in new features, prefer the former. Security experts, relying on several decades of experience, strongly counsel the latter. An administrator implementing or configuring a firewall must choose one of the two approaches, although the administrator can often broaden the policy by setting the firewall's parameters.

Design of Firewalls

Remember from Chapter 5 that a reference monitor must be

- always invoked
- tamperproof
- small and simple enough for rigorous analysis

A firewall is a special form of reference monitor. By carefully positioning a firewall within a network, we can ensure that all network accesses that we want to control must pass through it. This restriction meets the "always invoked" condition. A firewall is typically well isolated, making it highly immune to modification. Usually a firewall is implemented on a separate computer, with direct connections only to the outside and inside networks. This isolation is expected to meet the "tamperproof" requirement. And firewall designers strongly recommend keeping the functionality of the firewall simple.

Types of Firewalls

Firewalls have a wide range of capabilities. Types of firewalls include

 packet filtering gateways or screening routers
 stateful inspection firewalls
 application proxies
 guards
 personal firewalls

Each type does different things; no one is necessarily "right" and the others "wrong." In this section, we examine each type to see what it is, how it works, and what its strengths and weaknesses are. In general, screening routers tend to implement rather simplistic security policies, whereas guards and proxy gateways have a richer set of choices for security policy.

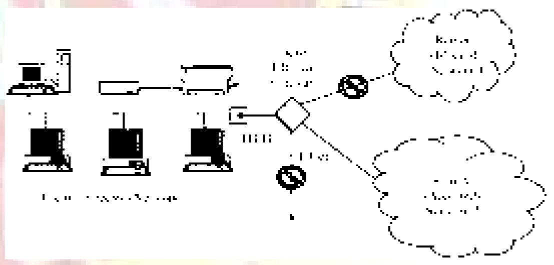
Simplicity in a security policy is not a bad thing; the important question to ask when choosing a type of firewall is what threats an installation needs to counter.

Because a firewall is a type of host, it often is as programmable as a good-quality workstation. While a screening router *can* be fairly primitive, the tendency is to host even routers on complete computers with operating systems because editors and other programming tools assist in configuring and maintaining the router. However, firewall developers are minimalists: They try to eliminate from the firewall all that is not strictly necessary for the firewall's functionality. There is a good reason for this minimal constraint: to give as little assistance as possible to a successful attacker. Thus, firewalls tend not to have user accounts so that, for example, they have no password file to conceal. Indeed, the most desirable firewall is one that runs contentedly in a back room; except for periodic scanning of its audit logs, there is seldom reason to touch it.

Packet Filtering Gateway

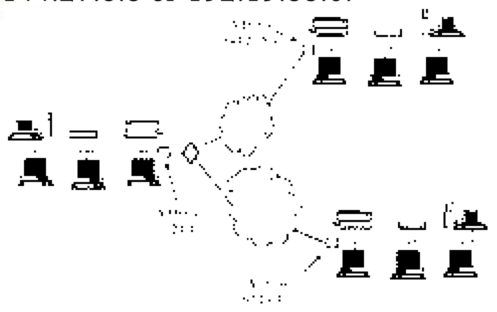
A packet filtering gateway or screening router is the simplest, and in some situations, the most effective type of firewall. A packet filtering gateway controls access to packets on the basis of packet address (source or destination) or specific transport protocol type (such as HTTP web traffic). As described earlier in this chapter, putting ACLs on routers may severely impede their performance. But a separate firewall behind (on the local side) of the router can screen traffic before it gets to the protected network. Figure 7-34 shows a packet filter that blocks access from (or to) addresses in one network; the filter allows HTTP traffic but blocks traffic using the Telnet protocol.

Figure 7-34. Packet Filter Blocking Addresses and Protocols.



For example, suppose an international company has three LANs at three locations throughout the world, as shown in Figure 7-35. In this example, the router has two sides: inside and outside. We say that the local LAN is on the inside of the router, and the two connections to distant LANs through wide area networks are on the outside. The company might want communication *only* among the three LANs of the corporate network. It could use a screening router on the LAN at 100.24.4.0 to allow *in* only communications destined to the host at 100.24.4.0 and to allow *out* only communications addressed either to address 144.27.5.3 or 192.19.33.0.

Figure 7-35. Three Connected LANs.



Packet filters do not "see inside" a packet; they block or accept packets solely on the basis of the IP addresses and ports. Thus, any details in the packet's data field (for example, allowing certain Telnet commands while blocking other services) is beyond the capability of a packet filter.

Packet filters can perform the very important service of ensuring the validity of inside addresses. Inside hosts typically trust other inside hosts for all the reasons described as characteristics of LANs. But the only way an inside host can distinguish another inside host is by the address shown in the source field of a message. Source addresses in packets can be forged, so an inside application might think it was communicating with another host on the inside instead of an outside forger. A packet filter sits between the inside network and the outside net, so it can know if a packet from the outside is forging an inside address, as shown in Figure 7-36. A screening packet filter might be configured to block all packets from the *outside* that claimed their source address was an *inside* address. In this example, the packet filter blocks all packets claiming to come from any address of the form 100.50.25.*x* (but, of course, it permits in any packets with *destination* 100.50.25.*x*).

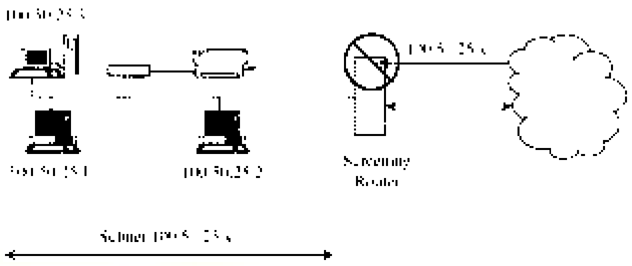


Figure 7-36. Filter Screening Outside Addresses.

The primary disadvantage of packet filtering routers is a combination of simplicity and complexity. The router's inspection is simplistic; to perform sophisticated filtering, the filtering rules set needs to be very detailed. A detailed rules set will be complex and therefore prone to error. For example, blocking all port 23 traffic (Telnet) is simple and straightforward. But if some Telnet traffic is to be allowed, each IP address from which it is allowed must be specified in the rules; in this way, the rule set can become very long.

Stateful Inspection Firewall

Filtering firewalls work on packets one at a time, accepting or rejecting each packet and moving on to the next. They have no concept of "state" or "context" from one packet to the next. A stateful inspection firewall maintains state information from one packet to another in the input stream.

One classic approach used by attackers is to break an attack into multiple packets by forcing some packets to have very short lengths so that a firewall cannot detect the signature of an attack split across two or more packets. (Remember that with the TCP protocols, packets can arrive in any order, and the protocol suite is responsible for reassembling the packet stream in proper order before passing it along to the application.) A stateful inspection firewall would track the sequence of packets and conditions from one packet to another to thwart such an attack.

Application Proxy

Packet filters look only at the headers of packets, not at the data *inside* the packets. Therefore, a packet filter would pass anything to port 25, assuming its screening rules allow inbound connections to that port. But applications are complex and sometimes contain errors.

Worse, applications (such as the e-mail delivery agent) often act on behalf of all users, so they require privileges of all users (for example, to store incoming mail messages so that inside users can read them). A flawed application, running with all users' privileges, can cause much damage.

An application proxy gateway, also called a bastion host, is a firewall that simulates the (proper) effects of an application so that the application receives only requests to act properly. A proxy gateway is a two-headed device: It looks to the inside as if it is the outside (destination) connection, while to the outside it responds just as the insider would.

An application proxy runs pseudo applications. For instance, when electronic mail is transferred to a location, a sending process at one site and a receiving process at the destination communicate by a protocol that establishes the legitimacy of a mail transfer and then actually transfers the mail message. The protocol between sender and destination is carefully defined.

A proxy gateway essentially intrudes in the middle of this protocol exchange, seeming like a destination in communication with the sender that is outside the firewall, and seeming like the sender in communication with the real destination on the inside. The proxy in the middle has the opportunity to screen the mail transfer, ensuring that only acceptable e-mail protocol commands are sent to the destination.

As an example of application proxying, consider the FTP (file transfer) protocol. Specific protocol commands fetch (*get*) files from a remote location, store (*put*) files onto a remote host, list files (*ls*) in a directory on a remote host, and position the process (*cd*) at a particular point in a directory tree on a remote host. Some administrators might want to permit gets but block puts, and to list only certain files or prohibit changing out of a particular directory (so that an outsider could retrieve only files from a prespecified directory). The proxy would simulate both sides of this protocol exchange. For example, the proxy might accept get commands, reject put commands, and filter the local response to a request to list files.

To understand the real purpose of a proxy gateway, let us consider several examples.

 A company wants to set up an online price list so that outsiders can see the products and prices offered. It wants to be sure that (a) no outsider can change the prices or product list and (b) outsiders can access only the price list, not any of the more sensitive files stored inside.

 A school wants to allow its students to retrieve any information from World Wide Web resources on the Internet. To help provide efficient service, the school wants to know what sites have been visited and what files from those sites have been fetched; particularly popular files will be cached locally.

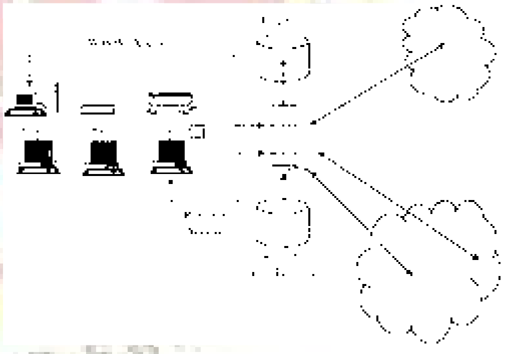
 A government agency wants to respond to queries through a database management system. However, because of inference attacks against databases, the agency wants to restrict queries that return the mean of a set of fewer than five values.

 A company with multiple offices wants to encrypt the data portion of all e-mail to addresses at its other offices. (A corresponding proxy at the remote end will remove the encryption.)

 A company wants to allow dial-in access by its employees, without exposing its company resources to login attacks from remote nonemployees.

Each of these requirements can be met with a proxy. In the first case, the proxy would monitor the file transfer protocol *data* to ensure that only the price list file was accessed, and that file could only be read, not modified. The school's requirement could be met by a logging procedure as part of the web browser. The agency's need could be satisfied by a special-purpose proxy that interacted with the database management system, performing queries but also obtaining the number of values from which the response was computed and adding a random minor error term to results from small sample sizes. The requirement for limited login could be handled by a specially written proxy that required strong user authentication (such as a challengeresponse system), which many operating systems do not require. These functions are shown in Figure 7-37.

Figure 7-37. Actions of Firewall Proxies.



The proxies on the firewall can be tailored to specific requirements, such as logging details about accesses. They can even present a common user interface to what may be dissimilar internal functions. Suppose the internal network has a mixture of operating system types, none of which support strong authentication through a challenge response token. The proxy can demand strong authentication (name, password, and challenge response), validate the challenge response itself, and then pass on only simple name and password authentication details in the form required by a specific internal host's operating system.

The distinction between a proxy and a screening router is that the proxy interprets the protocol stream to an application, to control actions through the firewall on the basis of things visible *within* the protocol, not just on external header data.

Guard A guard is a sophisticated firewall. Like a proxy firewall, it receives protocol data units, interprets them, and passes through the same or different protocol data units that achieve either the same result or a modified result. The guard decides what services to perform on the user's behalf in accordance with its available knowledge, such as whatever it can reliably know of the (outside) user's identity, previous interactions, and so forth. The degree of control a guard can provide is limited only by what is computable. But guards and proxy firewalls are similar enough that the distinction between them is sometimes fuzzy. That is, we can add functionality to a proxy firewall until it starts to look a lot like a guard.

Guard activities can be quite sophisticated, as illustrated in the following examples:

 A university wants to allow its students to use e-mail up to a limit of so many messages or so many characters of e-mail in the last so many days. Although this result could be achieved by modifying e-mail handlers, it is more easily done by monitoring the common point through which all e-mail flows, the mail transfer protocol.
 A school wants its students to be able to access the World Wide Web but, because of the slow speed of its connection to the web, it will allow only so many characters per downloaded image (that is, allowing text mode and simple graphics, but disallowing complex graphics, animation, music, or the like).
 A library wants to make available certain documents but, to support fair use of copyrighted matter, it will allow a user to retrieve only the first so many characters of a document. After that amount, the library will require the user to pay a fee that will be forwarded to the author.
 A company wants to allow its employees to fetch files via ftp. However, to prevent introduction of viruses, it will first pass all incoming files through a virus scanner. Even though many of these files will be nonexecutable text or graphics, the company administrator thinks that the expense of scanning them (which should pass) will be negligible.

Each of these scenarios can be implemented as a modified proxy. Because the proxy decision is based on some quality of the communication data, we call the proxy a guard. Since the security policy implemented by the guard is somewhat more complex than the action of a proxy, the guard's code is also more complex and therefore more exposed to error. Simpler firewalls have fewer possible ways to fail or be subverted.

Personal Firewalls

Firewalls typically protect a (sub)network of multiple hosts. University students and employees in offices are behind a real firewall. Increasingly, home users, individual workers, and small businesses use cable modems or DSL connections with unlimited, always-on access. These people need a firewall, but a separate firewall computer to protect a single workstation can seem too complex and expensive. These people need a firewall's capabilities at a lower price.

A personal firewall is an application program that runs on a workstation to block unwanted traffic, usually from the network. A personal firewall can complement the work of a conventional firewall by screening the kind of data a single host will accept, or it can compensate for the lack of a regular firewall, as in a private DSL or cable modem connection.

Just as a network firewall screens incoming and outgoing traffic for that network, a personal firewall screens traffic on a single workstation. A workstation could be vulnerable to malicious code or malicious active agents (ActiveX controls or Java applets), leakage of personal data stored on the workstation, and vulnerability scans to identify potential weaknesses.

Commercial implementations of personal firewalls include Norton Personal Firewall from Symantec, McAfee Personal Firewall, and Zone Alarm from Zone Labs (now owned by CheckPoint).

The personal firewall is configured to enforce some policy. For example, the user may decide that certain sites, such as computers on the company network, are highly trustworthy, but most other sites are not. The user defines a policy permitting download of code, unrestricted data sharing, and management access from the corporate segment, but not from other sites.

Personal firewalls can also generate logs of accesses, which can be useful to examine in case something harmful does slip through the firewall.

Combining a virus scanner with a personal firewall is both effective and efficient. Typically, users forget to run virus scanners daily, but they do remember to run them occasionally, such as sometime during the week. However, leaving the virus scanner execution to the user's memory means that the scanner detects a problem only after the factsuch as when a virus has been downloaded in an e-mail attachment. With the combination of a virus scanner and a personal firewall, the firewall directs all incoming e-mail to the virus scanner, which examines every attachment the moment it reaches the target host and before it is opened.

A personal firewall runs on the very computer it is trying to protect. Thus, a clever attacker is likely to attempt an undetected attack that would disable or reconfigure the firewall for the future. Still, especially for cable modem, DSL, and other "always on" connections, the static workstation is a visible and vulnerable target for an ever-present attack community. A personal firewall can provide reasonable protection to clients that are not behind a network firewall.

Comparison of Firewall Types

We can summarize the differences among the several types of firewalls we have studied in depth. The comparisons are shown in Table 7-8.

Table 7-8. Comparison of Firewall Types.

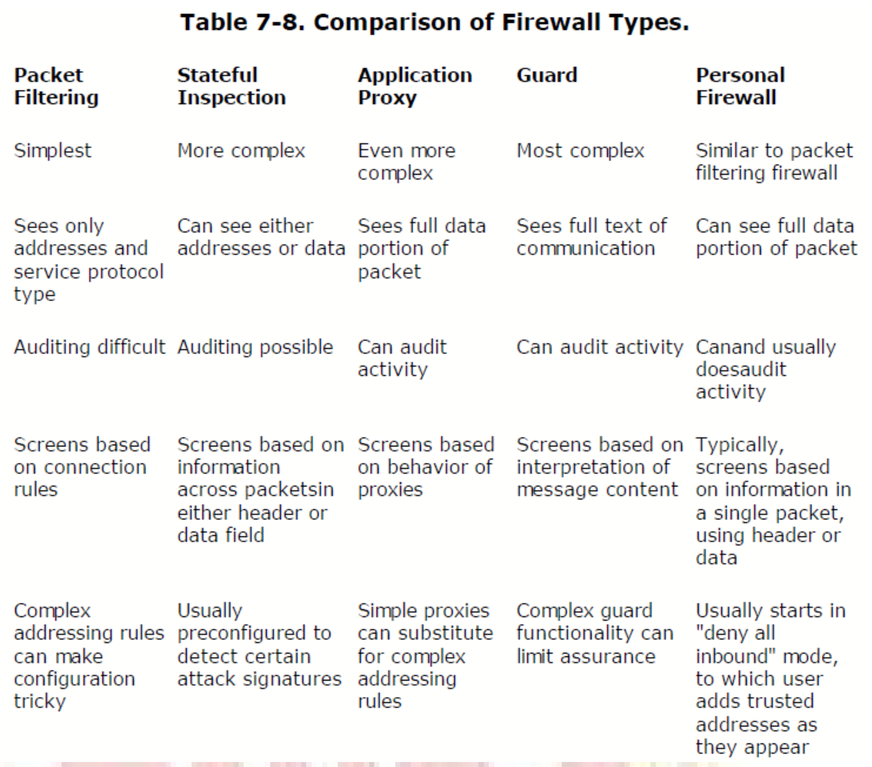


Table 7-8. Comparison of Firewall Types.

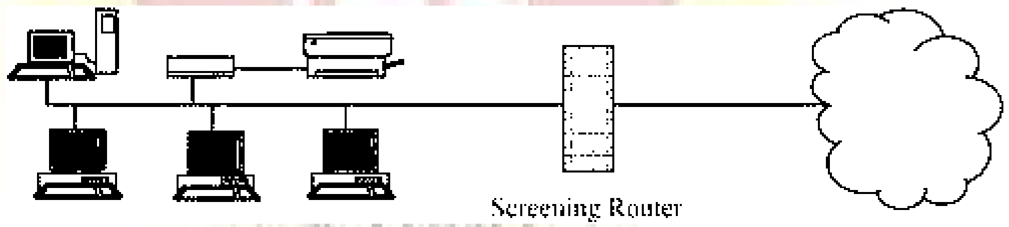
Packet Filtering	**Stateful Inspection**	**Application Proxy**	**Guard**	**Personal Firewall**
Simplest	More complex	Even more complex	Most complex	Similar to packet filtering firewall
Sees only addresses and service protocol type	Can see either addresses or data	Sees full data portion of packet	Sees full text of communication	Can see full data portion of packet
Auditing difficult	Auditing possible	Can audit activity	Can audit activity	Canand usually doesaudit activity
Screens based on connection rules	Screens based on information across packetsin either header or data field	Screens based on behavior of proxies	Screens based on interpretation of message content	Typically, screens based on information in a single packet, using header or data
Complex addressing rules can make configuration tricky	Usually preconfigured to detect certain attack signatures	Simple proxies can substitute for complex addressing rules	Complex guard functionality can limit assurance	Usually starts in "deny all inbound" mode, to which user adds trusted addresses as they appear

Example Firewall Configurations

Let us look at several examples to understand how to use firewalls. We present situations designed to show how a firewall complements a sensible security policy and architecture.

The simplest use of a firewall is shown in Figure 7-38. This environment has a screening router positioned between the internal LAN and the outside network connection. In many cases, this installation is adequate when we need only screen the address of a router.

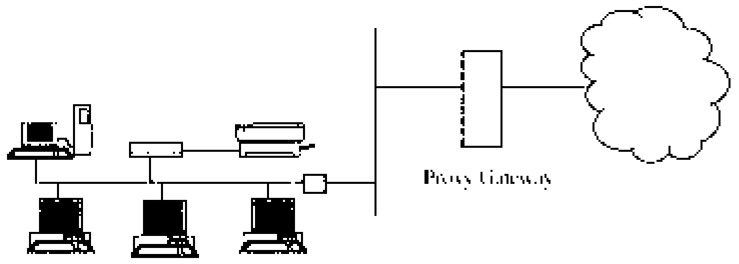
Figure 7-38. Firewall with Screening Router.



However, to use a proxy machine, this organization is not ideal. Similarly, configuring a router for a complex set of approved or rejected addresses is difficult. If the firewall router is successfully attacked, then all traffic on the LAN to which the firewall is connected is visible.

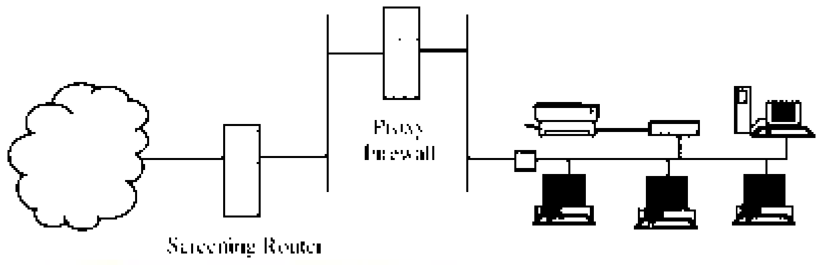
To reduce this exposure, a proxy firewall is often installed on its own LAN, as shown in Figure 7-39. In this way the only traffic visible on that LAN is the traffic going into and out of the firewall.

Figure 7-39. Firewall on Separate LAN.



For even more protection, we can add a screening router to this configuration, as shown in Figure 7-40. Here, the screening router ensures address correctness to the proxy firewall (so that the proxy firewall cannot be fooled by an outside attacker forging an address from an inside host); the proxy firewall filters traffic according to its proxy rules. Also, if the screening router is subverted, only the traffic to the proxy firewall is visible not any of the sensitive information on the internal protected LAN.

Figure 7-40. Firewall with Proxy and Screening Router.



Although these examples are simplifications, they show the kinds of configurations firewalls protect. Next, we review the kinds of attacks against which firewalls can and cannot protect.

What Firewalls Canand CannotBlock As we have seen, firewalls are not complete solutions to all computer security problems. A firewall protects only the perimeter of its environment against attacks from outsiders who want to execute code or access data on the machines in the protected environment. Keep in mind these points about firewalls.

 Firewalls can protect an environment only if the firewalls control the entire perimeter. That is, firewalls are effective only if no unmediated connections breach the perimeter. If even one inside host connects to an outside address, by a modem for example, the entire inside net is vulnerable through the modem and its host.

 Firewalls do not protect data outside the perimeter; data that have properly passed (outbound) through the firewall are just as exposed as if there were no firewall.

 Firewalls are the most visible part of an installation to the outside, so they are the most attractive target for attack. For this reason, several different layers of protection, called defense in depth, are better than relying on the strength of just a single firewall.

 Firewalls must be correctly configured, that configuration must be updated as the internal and external environment changes, and firewall activity reports must be reviewed periodically for evidence of attempted or successful intrusion.

 Firewalls are targets for penetrators. While a firewall is designed to withstand attack, it is not impenetrable. Designers intentionally keep a firewall small and simple so that even if a penetrator breaks it, the firewall does not have further tools, such as compilers, linkers, loaders, and the like, to continue an attack.

 Firewalls exercise only minor control over the content admitted to the inside, meaning that inaccurate data or malicious code must be controlled by other means inside the perimeter.

Firewalls are important tools in protecting an environment connected to a network. However, the environment must be viewed as a whole, all possible exposures must be considered, and the firewall must fit into a larger, comprehensive security strategy. Firewalls alone cannot secure an environment.

Intrusion detection systems

After the perimeter controls, firewall, and authentication and access controls block certain actions, some users are admitted to use a computing system. Most of these controls are preventive: They block known bad things from happening. Many studies (for example, see [DUR99]) have shown that most computer security incidents are caused by insiders, people who would not be blocked by a firewall. And insiders require access with significant privileges to do their daily jobs. The vast majority of harm from insiders is not malicious; it is honest people making honest mistakes. Then, too, there are the potential malicious outsiders who have somehow passed the screens of firewalls and access controls. Prevention, although necessary, is not a complete computer security control; detection during an incident copes with harm that cannot be prevented in advance. Halme and Bauer [HAL95] survey the range of controls to address intrusions.

Intrusion detection systems complement these preventive controls as the next line of defense. An intrusion detection system (IDS) is a device, typically another separate computer, that monitors activity to identify malicious or suspicious events. Kemmerer and Vigna [KEM02] survey the history of IDSs. An IDS is a sensor, like a smoke detector, that raises an alarm if specific things occur. A model of an IDS is shown in Figure 7-41. The components in the figure are the four basic elements of an intrusion detection system, based on the Common Intrusion Detection Framework of [STA96]. An IDS receives raw inputs from sensors. It saves those inputs, analyzes them, and takes some controlling action.

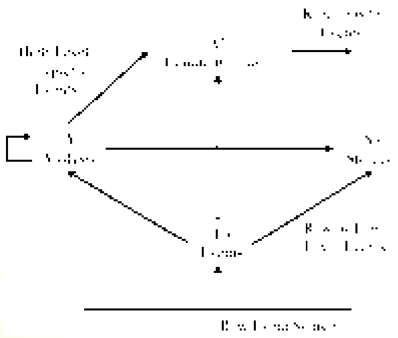


Figure 7-41. Common Components of an Intrusion Detection Framework.

IDSs perform a variety of functions:

- monitoring users and system activity
- auditing system configuration for vulnerabilities and misconfigurations
- assessing the integrity of critical system and data files
- recognizing known attack patterns in system activity
- identifying abnormal activity through statistical analysis
- managing audit trails and highlighting user violation of policy or normal activity
- correcting system configuration errors
- installing and operating traps to record information about intruders

No one IDS performs all of these functions. Let us look more closely at the kinds of IDSs and their use in providing security.

Types of IDSs

The two general types of intrusion detection systems are signature based and heuristic Signature-based intrusion detection systems perform simple pattern-matching and report situations that match a pattern corresponding to a known attack type. Heuristic intrusion detection systems, also known as anomaly based, build a model of acceptable behavior and flag exceptions to that model; for the future, the administrator can mark a flagged behavior as acceptable so that the heuristic IDS will now treat that previously unclassified behavior as acceptable.

Intrusion detection devices can be network based or host based. A network-based IDS is a stand-alone device attached to the network to monitor traffic throughout that network; a host-based IDS runs on a single workstation or client or host, to protect that one host.

Early intrusion detection systems (for example, [DEN87b, LUN90a, FOX90, LIE89]) worked after the fact, by reviewing logs of system activity to spot potential misuses that had occurred. The administrator could review the results of the IDS to find and fix weaknesses in the system. Now, however, intrusion detection systems operate in real time (or near real time), watching activity and raising alarms in time for the administrator to take protective action.

Signature-Based Intrusion Detection

A simple signature for a known attack type might describe a series of TCP SYN packets sent to many different ports in succession and at times close to one another, as would be the case for a port scan. An intrusion detection system would probably find nothing unusual in the first SYN, say, to port 80, and then another (from the same source address) to port 25. But as more and more ports receive SYN packets, especially ports that are not open, this pattern reflects a possible port scan. Similarly, some implementations of the protocol stack fail if they receive an ICMP packet with a data length of 65535 bytes, so such a packet would be a pattern for which to watch.

The problem with signature-based detection is the signatures themselves. An attacker will try to modify a basic attack in such a way that it will not match the known signature of that attack. For example, the attacker may convert lowercase to uppercase letters or convert a symbol such as "blank space" to its character code equivalent %20. The IDS must necessarily work from a canonical form of the data stream in order to recognize that %20 matches a pattern with a blank space. The attacker may insert malformed packets that the IDS will see, to intentionally cause a pattern mismatch; the protocol handler stack will discard the packets because of the malformation. Each of these variations could be detected by an IDS, but more signatures require additional work for the IDS, which reduces performance.

Of course, signature-based IDSs cannot detect a new attack for which a signature is not yet installed in the database. Every attack type starts as a new pattern at some time, and the IDS is helpless to warn of its existence.

Signature-based intrusion detection systems tend to use statistical analysis. This approach uses statistical tools both to obtain sample measurements of key indicators (such as amount of external activity, number of active processes, number of transactions) and to determine whether the collected measurements fit the predetermined attack signatures.

Ideally, signatures should match every instance of an attack, match subtle variations of the attack, but not match traffic that is not part of an attack. However, this goal is grand but unreachable.

Heuristic Intrusion Detection

Because signatures are limited to specific, known attack patterns, another form of intrusion detection becomes useful. Instead of looking for matches, heuristic intrusion detection looks for behavior that is out of the ordinary. The original work in this area (for example, [TEN90]) focused on the individual, trying to find characteristics of that person that might be helpful in understanding normal and abnormal behavior. For example, one user might always start the day by reading e-mail, write many documents using a word processor, and occasionally back up files. These actions would be normal. This user does not seem to use many administrator utilities. If that person tried to access sensitive system management utilities, this new behavior might be a clue that someone else was acting under the user's identity.

If we think of a compromised system in use, it starts clean, with no intrusion, and it ends dirty, fully compromised. There may be no point in the trace of use in which the system changed from clean to dirty; it was more likely that little dirty events occurred, occasionally at first and then increasing as the system became more deeply compromised. Any one of those events might be acceptable by itself, but the accumulation of them and the order and speed at which they occurred could have been signals that something unacceptable was happening. The inference engine of an intrusion detection system performs continuous analysis of the system, raising an alert when the system's dirtiness exceeds a threshold.

Inference engines work in two ways. Some, called state-based intrusion detection systems, see the system going through changes of overall state or configuration. They try to detect when the system has veered into unsafe modes. Others try to map current activity onto a model of unacceptable activity and raise an alarm when the activity resembles the model.

These are called model-based intrusion detection systems. This approach has been extended to networks in [MUK94]. Later work (for example, [FOR96, LIN99]) sought to build a dynamic model of behavior, to accommodate variation and evolution in a person's actions over time. The technique compares real activity with a known representation of normality.

Alternatively, intrusion detection can work from a model of known bad activity. For example, except for a few utilities (login, change password, create user), any other attempt to access a password file is suspect. This form of intrusion detection is known as misuse intrusion detection. In this work, the real activity is compared against a known suspicious area.

All heuristic intrusion detection activity is classified in one of three categories: good/benign, suspicious, or unknown. Over time, specific kinds of actions can move from one of these categories to another, corresponding to the IDS's learning whether certain actions are acceptable or not.

As with pattern-matching, heuristic intrusion detection is limited by the amount of information the system has seen (to classify actions into the right category) and how well the current actions fit into one of these categories.

Stealth Mode

An IDS is a network device (or, in the case of a host-based IDS, a program running on a network device). Any network device is potentially vulnerable to network attacks. How useful would an IDS be if it itself were deluged with a denial-of-service attack? If an attacker succeeded in logging in to a system within the protected network, wouldn't trying to disable the IDS be the next step?

To counter those problems, most IDSs run in stealth mode, whereby an IDS has two network interfaces: one for the network (or network segment) being monitored and the other to generate alerts and perhaps other administrative needs. The IDS uses the monitored interface as input only; it *never* sends packets out through that interface. Often, the interface is configured so that the device has no published address through the monitored interface; that is, a router cannot route anything to that address directly, because the router does not know such a device exists. It is the perfect passive wiretap. If the IDS needs to generate an alert, it uses only the alarm interface on a completely separate control network. Such an architecture is shown in Figure 7-42.

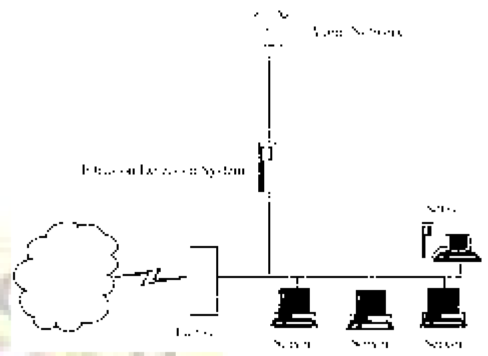


Figure 7-42. Stealth Mode IDS Connected to Two Networks.

Other IDS Types

Some security engineers consider other devices to be IDSs as well. For instance, to detect unacceptable code modification, programs can compare the active version of a software code with a saved version of a digest of that code. The *tripwire* program [KIM98] is the most well known software (or static data) comparison program. You run *tripwire* on a new system, and it generates a hash value for each file; then you save these hash values in a secure place (offline, so that no intruder can modify them while modifying a system file). If you later suspect your system may have been compromised, you rerun *tripwire*, providing it the saved hash values. It recomputes the hash values and reports any mismatches, which would indicate files that were changed.

System vulnerability scanners, such as *ISS Scanner* or *Nessus*, can be run against a network. They check for known vulnerabilities and report flaws found. As we have seen, a honeypot is a faux environment intended to lure an attacker. It can be considered an IDS, in the sense that the honeypot may record an intruder's actions and even attempt to trace who the attacker is from actions, packet data, or connections.

Goals for Intrusion Detection Systems

The two styles of intrusion detection pattern matching and heuristic represent different approaches, each of which has advantages and disadvantages. Actual IDS products often blend the two approaches.

Ideally, an IDS should be fast, simple, and accurate, while at the same time being complete. It should detect all attacks with little performance penalty. An IDS could use some or all of the following design approaches:

- Filter on packet headers
- Filter on packet content
- Maintain connection state
- Use complex, multipacket signatures
- Use minimal number of signatures with maximum effect
- Filter in real time, online
- Hide its presence
- Use optimal sliding time window size to match signatures

Responding to Alarms

Whatever the type, an intrusion detection system raises an alarm when it finds a match. The alarm can range from something modest, such as writing a note in an audit log, to something significant, such as paging the system security administrator. Particular implementations allow the user to determine what action the system should take on what events.

What are possible responses? The range is unlimited and can be anything the administrator can imagine (and program). In general, responses fall into three major categories (any or all of which can be used in a single response):

- Monitor, collect data, perhaps increase amount of data collected
- Protect, act to reduce exposure
- Call a human

Monitoring is appropriate for an attack of modest (initial) impact. Perhaps the real goal is to watch the intruder, to see what resources are being accessed or what attempted attacks are tried. Another monitoring possibility is to record all traffic from a given source for future analysis. This approach should be invisible to the attacker. Protecting can mean increasing access controls and even making a resource unavailable (for example, shutting off a network connection or making a file unavailable). The system can even sever the network connection the attacker is using. In contrast to monitoring, protecting may be very visible to the attacker. Finally, calling a human allows individual discrimination. The IDS can take an initial defensive action immediately while also generating an alert to a human who may take seconds, minutes, or longer to respond.

False Results

Intrusion detection systems are not perfect, and mistakes are their biggest problem. Although an IDS might detect an intruder correctly most of the time, it may stumble in two different ways: by raising an alarm for something that is not really an attack (called a false positive, or type I error in the statistical community) or not raising an alarm for a real attack (a false negative, or type II error). Too many false positives means the administrator will be less confident of the IDS's warnings, perhaps leading to a real alarm's being ignored. But false negatives mean that real attacks are passing the IDS without action. We say that the degree of false positives and false negatives represents the sensitivity of the system. Most IDS implementations allow the administrator to tune the system's sensitivity, to strike an acceptable balance between false positives and negatives.

IDS Strengths and Limitations

Intrusion detection systems are evolving products. Research began in the mid-1980s and products had appeared by the mid-1990s. However, this area continues to change as new research influences the design of products.

On the upside, IDSs detect an ever-growing number of serious problems. And as we learn more about problems, we can add their signatures to the IDS model. Thus, over time, IDSs continue to improve. At the same time, they are becoming cheaper and easier to administer.

On the downside, avoiding an IDS is a first priority for successful attackers. An IDS that is not well defended is useless. Fortunately, stealth mode IDSs are difficult even to find on an internal network, let alone to compromise.

IDSs look for known weaknesses, whether through patterns of known attacks or models of normal behavior. Similar IDSs may have identical vulnerabilities, and their selection criteria may miss similar attacks. Knowing how to evade a particular model of IDS is an important piece of intelligence passed within the attacker community. Of course, once manufacturers become aware of a shortcoming in their products, they try to fix it. Fortunately, commercial IDSs are pretty good at identifying attacks.

Another IDS limitation is its sensitivity, which is difficult to measure and adjust. IDSs will never be perfect, so finding the proper balance is critical.

A final limitation is not of IDSs *per se*, but is one of use. An IDS does not run itself; someone has to monitor its track record and respond to its alarms. An administrator is foolish to buy and install an IDS and then ignore it.

In general, IDSs are excellent additions to a network's security. Firewalls block traffic to particular ports or addresses; they also constrain certain protocols to limit their impact. But by definition, firewalls have to allow some traffic to enter a protected area. Watching what that traffic actually does inside the protected area is an IDS's job, which it does quite well.

Secure e-mail

The final control we consider in depth is secure e-mail. Think about how much you use e-mail and how much you rely on the accuracy of its contents. How would you react if you received a message from your instructor saying that because you had done so well in your course so far, you were excused from doing any further work in it? What if that message were a joke from a classmate? We rely on e-mail's confidentiality and integrity for sensitive and important communications, even though ordinary e-mail has almost no confidentiality or integrity. In this section we investigate how to add confidentiality and integrity protection to ordinary e-mail.

Security for E-mail

E-mail is vital for today's commerce, as well a convenient medium for communications among ordinary users. But, as we noted earlier, e-mail is very public, exposed at every point from the sender's workstation to the recipient's screen. Just as you would not put sensitive or private thoughts on a postcard, you must also acknowledge that e-mail messages are exposed and available for others to read.

Sometimes we would like e-mail to be more secure. To define and implement a more secure form, we begin by examining the exposures of ordinary e-mail.

Threats to E-mail

Consider threats to electronic mail:

- message interception (confidentiality)
- message interception (blocked delivery)
- message interception and subsequent replay
- message content modification
- message origin modification
- message content forgery by outsider
- message origin forgery by outsider
- message content forgery by recipient
- message origin forgery by recipient
- denial of message transmission

Confidentiality and content forgery are often handled by encryption. Encryption can also help in a defense against replay, although we would also have to use a protocol in which each message contains something unique that is encrypted. Symmetric encryption cannot protect against forgery by a recipient, since both sender and recipient share a common key; however, public key schemes can let a recipient decrypt but not encrypt. Because of lack of control over the middle points of a network, senders or receivers generally cannot protect against blocked delivery.

Requirements and Solutions

If we were to make a list of the requirements for secure e-mail, our wish list would include the following protections.

- *message confidentiality* (the message is not exposed en route to the receiver)
- *message integrity* (what the receiver sees is what was sent)
- *sender authenticity* (the receiver is confident who the sender was)
- *nonrepudiation* (the sender cannot deny having sent the message)

Not all these qualities are needed for every message, but an ideal secure e-mail package would allow these capabilities to be invoked selectively.

Designs

The standard for encrypted e-mail was developed by the Internet Society, through its architecture board (IAB) and research (IRTF) and engineering (IETF) task forces. The encrypted e-mail protocols are documented as an Internet standard in documents 1421, 1422, 1423, and 1424 [LIN93, KEN93, BAL93, KAL93a]. This standard is actually the third refinement of the original specification.

One of the design goals for encrypted e-mail was allowing security-enhanced messages to travel as ordinary messages through the existing Internet e-mail system. This requirement ensures that the large existing e-mail network would not require change to accommodate security. Thus, all protection occurs within the body of a message.

Confidentiality

Because the protection has several aspects, we begin our description of them by looking first at how to provide confidentiality enhancements. The sender chooses a (random) symmetric algorithm encryption key. Then, the sender encrypts a copy of the entire message to be transmitted, including FROM:, TO:, SUBJECT:, and DATE: headers. Next, the sender prepends plaintext headers. For key management, the sender encrypts the message key under the recipient's public key, and attaches that to the message as well. The process of creating an encrypted e-mail message is shown in Figure 7-43.

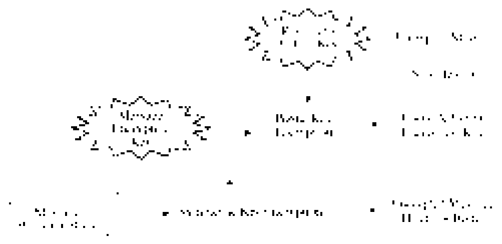


Figure 7-43. Overview of Encrypted E-mail Processing.

Encryption can potentially yield any string as output. Many e-mail handlers expect that message traffic will not contain characters other than the normal printable characters.

Network e-mail handlers use unprintable characters as control signals in the traffic stream. To avoid problems in transmission, encrypted e-mail converts the entire ciphertext message to printable characters. An example of an encrypted e-mail message is shown in Figure 7-44.
Notice the three portions: an external (plaintext) header, a section by which the message encryption key can be transferred, and the encrypted message itself. (The encryption is shown with shading.)

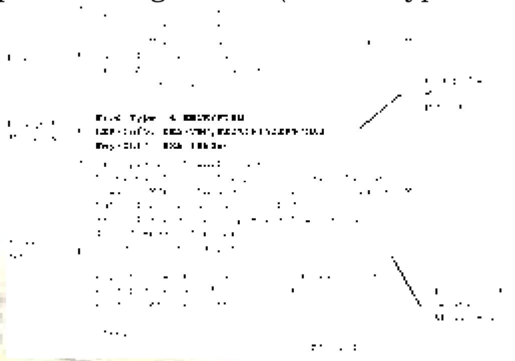


Figure 7-44. Encrypted E-mailSecured Message.

The encrypted e-mail standard works most easily as just described, using both symmetric and asymmetric encryption. The standard is also defined for symmetric encryption only: To use symmetric encryption, the sender and receiver must have previously established a shared secret encryption key. The processing type ("Proc-Type") field tells what privacy enhancement services have been applied. In the data exchange key field ("DEK-Info"), the kind of key exchange (symmetric or asymmetric) is shown. The key exchange ("Key-Info") field contains the message encryption key, encrypted under this shared encryption key. The field also identifies the originator (sender) so that the receiver can determine which shared symmetric key was used. If the key exchange technique were to use asymmetric encryption, the key exchange field would contain the message encryption field, encrypted under the recipient's public key. Also included could be the sender's certificate (used for determining authenticity and for generating replies).

The encrypted e-mail standard supports multiple encryption algorithms, using popular algorithms such as DES, triple DES, and AES for message confidentiality, and RSA and DiffieHellman for key exchange.

Other Security Features

In addition to confidentiality, we may want various forms of integrity for secure e-mail. Encrypted e-mail messages always carry a digital signature, so the authenticity and nonrepudiability of the sender is assured. The integrity is also assured because of a hash function (called a message integrity check, or MIC) in the digital signature. Optionally, encrypted e-mail messages can be encrypted for confidentiality.

Notice in Figure 7-44 that the header inside the message (in the encrypted portion) differs from that outside. A sender's identity or the actual subject of a message can be concealed within the encrypted portion.

The encrypted e-mail processing can integrate with ordinary e-mail packages, so a person can send both enhanced and nonenhanced messages, as shown in Figure 7-45. If the sender decides to add enhancements, an extra bit of encrypted e-mail processing is invoked on the sender's end; the receiver must also remove the enhancements. But without enhancements, messages flow through the mail handlers as usual.

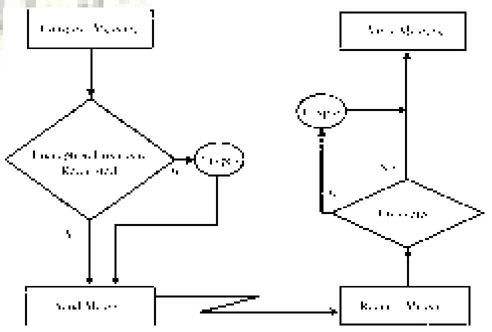


Figure 7-45. Encrypted E-mail Processing in Message Transmission.

S/MIME (discussed later in this section) can accommodate the exchange of other than just text messages: support for voice, graphics, video, and other kinds of complex message parts.

Encryption for Secure E-mail

The major problem with encrypted e-mail is key management. The certificate scheme described in Chapter 2 is excellent for exchanging keys and for associating an identity with a public encryption key. The difficulty with certificates is building the hierarchy. Many organizations have hierarchical structures. The encrypted e-mail dilemma is moving beyond the single organization to an interorganizational hierarchy. Precisely because of the problem of imposing a hierarchy on a nonhierarchical world, PGP was developed as a simpler form of encrypted e-mail.

Encrypted e-mail provides strong end-to-end security for electronic mail. Triple DES, AES, and RSA cryptography are quite strong, especially if RSA is used with a long bit key (1024 bits or more). The vulnerabilities remaining with encrypted e-mail come from the points not covered: the endpoints. An attacker with access could subvert a sender's or receiver's machine, modifying the code that does the privacy enhancements or arranging to leak a cryptographic key.

Example Secure E-mail Systems

Encrypted e-mail programs are available from many sources. Several universities (including Cambridge University in England and The University of Michigan in the United States) and companies (BBN, RSA-DSI, and Trusted Information Systems) have developed either prototype or commercial versions of encrypted e-mail.

PGP

PGP stands for Pretty Good Privacy. It was invented by Phil Zimmerman in 1991. Originally a free package, it became a commercial product after being bought by Network Associates in 1996. A freeware version is still available. PGP is widely available, both in commercial versions and freeware, and it is heavily used by individuals exchanging private e-mail.

PGP addresses the key distribution problem with what is called a "ring of trust" or a user's " keyring." One user directly gives a public key to another, or the second user fetches the first's public key from a server. Some people include their PGP public keys at the bottom of e-mail messages. And one person can give a second person's key to a third (and a fourth, and so on). Thus, the key association problem becomes one of caveat emptor: "Let the buyer beware." If I am reasonably confident that an e-mail message really comes from you and has not been tampered with, I will use your attached public key. If I trust you, I may also trust the keys you give me for other people. The model breaks down intellectually when you give me all the keys you received from people, who in turn gave you all the keys they got from still other people, who gave them all their keys, and so forth.

You sign each key you give me. The keys you give me may also have been signed by other people. I decide to trust the veracity of a key-and-identity combination, based on who signed the key.

PGP does not mandate a policy for establishing trust. Rather, each user is free to decide how much to trust each key received.

The PGP processing performs some or all of the following actions, depending on whether confidentiality, integrity, authenticity, or some combination of these is selected:

- Create a random session key for a symmetric algorithm.
- Encrypt the message, using the session key (for message confidentiality).
- Encrypt the session key under the recipient's public key.
- Generate a message digest or hash of the message; sign the hash by encrypting it with the sender's private key (for message integrity and authenticity).
- Attach the encrypted session key to the encrypted message and digest.
- Transmit the message to the recipient.

The recipient reverses these steps to retrieve and validate the message content.

S/MIME

An Internet standard governs how e-mail is sent and received. The general MIME specification defines the format and handling of e-mail attachments. S/MIME (Secure Multipurpose Internet Mail Extensions) is the Internet standard for secure e-mail attachments.

S/MIME is very much like PGP and its predecessors, PEM (Privacy-Enhanced Mail) and RIPEM. The Internet standards documents defining S/MIME (version 3) are described in [HOU99] and [RAM99]. S/MIME has been adopted in commercial e-mail packages, such as Eudora and Microsoft Outlook.

The principal difference between S/MIME and PGP is the method of key exchange. Basic PGP depends on each user's exchanging keys with all potential recipients and establishing a ring of trusted recipients; it also requires establishing a degree of trust in the authenticity of the keys for those recipients. S/MIME uses hierarchically validated certificates, usually represented in X.509 format, for key exchange. Thus, with S/MIME, the sender and recipient do not need to have exchanged keys in advance as long as they have a common certifier they both trust.

S/MIME works with a variety of cryptographic algorithms, such as DES, AES, and RC2 for symmetric encryption. S/MIME performs security transformations very similar to those for PGP. PGP was originally designed for plaintext messages, but S/MIME handles (secures) all sorts of attachments, such as data files (for example, spreadsheets, graphics, presentations, movies, and sound).

Because it is integrated into many commercial e-mail packages, S/MIME is likely to dominate the secure e-mail market.

Networks and cryptography

Example protocols:

PEM

Privacy-Enhanced Mail

Privacy-Enhanced Mail (**PEM**) is a de facto file format for storing and sending cryptographic keys, certificates, and other data, based on a set of 1993 IETF standards defining "privacy-enhanced mail." While the original standards were never broadly adopted and were supplanted by PGP andS/MIME, the textual encoding they defined became very popular. The PEM format was eventually formalized by the IETF in RFC 7468.

Many cryptography standards use ASN.1 to define their data structures, and Distinguished Encoding Rules (DER) to serialize those structures.[2] Because DER produces binary output, it can be challenging to transmit the resulting files through systems, like electronic mail, that only support ASCII.

The PEM format solves this problem by encoding the binary data using base64. PEM also defines a one-line header, consisting of "-----BEGIN ", a label, and "-----", and a one-line footer, consisting of "-----END ", a label, and "-----". The label determines the type of message encoded. Common labels include "CERTIFICATE", "CERTIFICATE REQUEST", "PRIVATE KEY" and "X509 CRL". PEM data is commonly stored in files with a ".pem" suffix, a ".cer" or ".crt" suffix (for certificates), or a ".key" suffix (for public or private keys).[3] The label inside a PEM file represents the type of the data more accurately than the file suffix, since many different types of data can be saved in a ".pem" file.
A PEM file may contain multiple instances. For instance, an operating system might provide a file containing a list of trustedCA certificates, or a web server might be configured with a "chain" file containing an end-entity certificate plus a list of intermediate certificates.

```
-----BEGIN PRIVATE KEY-----
MIIJRAIBADANBgkqhkiG9w0BAQEFAASCCS4wggkqAgEAAoICAQD2GlhF9HuoPwiF
S8lh0lHCwVGlVq0Jqtmp7ieyVOZ0mbU6T2KCDwkL3mWWSiVZc+cjh3EOsXtyzuiq
C1nsynrlSQuU3/pTkKVRWJCL51KXe7Rf+NPjSzqDdyowwQubWH42MMYbdLvjKRp9
Yje8yjHDE6N1OSJyp5TCN+74qLT/xqJOLsBQBEJoQESVUM5xIuEJk8epHwNaP4kP
SiYan5lTqXl0pOwNwyxG/kfRUdhdLI6VhaVzOqG3BT/HBYmu1Tk5CanqLeK8g5yw
pVfhKc/HIlbjTDjBXWfqUqTOOJaLkJxpJgcHS2FvtVzuc2VIbJoaOZqs10SIoqCZ
HuNVMvsFIeM2T2VB+XozoQrboP057wnUr5cvglpFZo7bWvDcbwZs8wXG9u0k1xxo
kolCtIM/FKwdZqgWBfKvmrQFZR5jUos0yaRVilTKMGKRCZvB2B242Z/JjhWabwjO
DyytbCADVUwyc4u7ZpJySDmAw9WiLmcGc2Z4E8qEeS/ejiVvBc5hE06CgaENdkTl
TpKunhZc0nfdOJvAxENfwT+D+SNN3oh6m8/thNWFsj6pd4uIc77s6WI9pfmxC8MC
G4NqJp2L0TDxN4e7iDCnQuWIWVMTSNGlIKRqsSnTXvdPbLsqD3+CcRkTe7oKu1Mv
jhVUqo1GQpLivx6GmCGDPOfjud4/qwIDAQABAoICAGBYtgBFE3gtnS9aGS/zv9CI
EGezRDEJswck1mOpe0NgvaNjWsRiEH+WBJ1Oz8pyZqXxK2C+OP8cY9fWc9ERAAXt
j9wrx3uZoC71hjlPSXVGl7oO1x0f4XuAVoMZwGqDvX2HoLc0/y2SdO1rWzjMy6h8
cry8rnDGjKVwclzk519PK1GCWgW5dSoNJwxxwzBloEY4lazvGEfnocfblolLvCIV
shfpTQSiSR5OBF29NgBcJkIBPWGcLZ1SL8LBt0I79ZgP4XZtmluFBv3c7UeYL+Dc
37lCqOGCqOECk8SsQLvI1IMRwSSo7S9niEdN4/PzmGz16kkKSalYSZlIj2VWfRCj
wS6xWD4Sfez9Fx/s8tfPEv15KXtoIDKn20h/m7szlfSNYTi15ZdKPPW9SHKCol6C
1kDgHg31eQtf3Nn6ukrBeHsyZr1FdkYwBSkRy+VIl6Kj5FJXUR1hWEvaLi1tplGP
I6NwSA6ArouaAGTVyS3VpHZw8hyGpBiJ4ZYIMcISyXyMNEM3kNcvbKnb+bME88/6
p/od4gb5kCUWEui34Nh7VtxY1n4g8HYJXWC07W0Xxz1/bvfAH7trPLHEyaoZ8kjz
Lj4TZLwx7BplUMaOsXEFFb+lgzwhZL0Nd9F4zPT3WeGUTYJMukRMGumS86qDbLof
7W08V+3MxKtfXgcIgKD5AoIBAQD/RoIchybjiTggPCZgCkRSx/HmjvtRZukwF9vk
LSjuhaIDxY/x+eyLstubjn3Wkhjx/DcTxV+efs/QeOneerj0n6ft8H0yQcgfTvmP
y6Kucx5lvSy4JH3ftKGuUu9X2erxKYaYIPWPMiPqQojECB1B1Bjsp8Xx0E4WioNK
WYWkQmrVvaxUuYa1JV3cdREcrbgObj84qbz15NprfOqLUJMBQ2qqvkiJ3yAiW72i
Nl1N6vf/TUGRCu8eCyMepDoLZI6apG0szqnAGPXW1v5m6/GxKFbuZCgR+73Dypsx
qbmQ7F4r2QKrZuPnsae68l8MmfawbEYb+yObuzSaJuHGc7JvAoIBAQD2zSvoKiHi
G1VVPkoXcZwe1rBn6zFZ+SZFtiTNR486xE0PvaJ/tCiPsqDO4xBZG+RwkOldAPbi
```

```
oNsFgbOw+3mjQ2r8HGpgeuZ3GIy9e48olGP1b+fcvf0JFLkAHZr1iOeJDR+ewddT
MzQmY3V1RWf13DvZDl0dT8Kaddz1UDlv9YN1iwPva7Rew4BLmIqmpcSo+m7qgG3f
wmHKvV+ABuj/8GyCMogUkc+ZAw8XfnnzRGdQHDJhRJIvaS0b6IxVun3m4mRbJ93r
poB8o15mL2DTjOGlN1qIQym6shuLZR2OV1hz1e0NEFwxsw4EbzR8cVCVBNhtKrBg
bBfHDJNKvDSFAoIBAQCSmuXCiIP3DSlkqebIJV1TxWzRorAG8fleG3W3t4/YXHrN
e7rNl2EYYeHplo4CmukkoFrpSeM+XUnwSmDV2tk59C83YXQlVs4d3PdKdAK7+XUt
rNVv89UuksiAGzVF2OsrEVQxLkyYro0dzpRJMPfMhIjD8R6nX6BZJbd9DajIyzQc
CNBd4CbTBeC/6aOoOsSH1R0N572T4pjmPlldJSsoAJXT9XAnbjNASDvCW0J+q0E+
KBpNuF/Xe0HyKRME0/1qJqBkqVOn9S3K8rIsXbjyq14xjufMXC2Bsmwqu3TNHFKB
ECKOWYrt14Oiw+t+izW34JbrHvI2T+9H0Ki9lliVAoIBAQDjwAZqoqbN/ydKGMdK
xx6pHrl/zHyoaNE2t5VSklzMgGYUxoz0iA5+PPtEsClf8etnLXMMzl0GWiaD+GMY
SZjAXvCVYquQCRowgUkvepxreDSeQ/yVqgWdoa/vOWLMfuAbiy7I1FyefLv9SP8V
j00Wh3v7G9Q1vmr5GxcikjvO46PCjty8zomOgZ8dI9GfY44N2b0NTiMWwEx6STOd
88KEnRulMnh9cuk+bKI6rg5fvZoRRVQAisTUV5y3CmymmAijTfwKWsniMq6TVjdA
2SvjTTjVvDUhVclmbgIZKuCRgG0xyBKPYa+SdYfT88NdqzwPqH8IjsJg2J9Aoowy
BWf5AoIBAQD8zybogaMZBBAHbnEObVzHfJS6+g3nqNEL/pKePc7oh2e7wLIOe9zq
3E7DYKJOnC4CulqxK6r7cE2H8dEbdbyMc2u9CqsM4kpLL7aaLWjG3H+MADbgDnWN
lE+wjylfewl+y99tElBHxtsfWXf4AdM9eofirjY5nlDCuym44XG1T/MnJETKyobC
vUmP0OzwCADl/pzVNkbeUyNZVTd9Y34f0FyxWelM5y/MSKCmLBBmS6FXB58nrlQY
psGUNwWXrARgiInCeQkvN3toQrXOyQ5Df3MwrTAUIy0Nec7MrUEcdjrE0Mks3HhH
hMnpHOOGnVBZdVNxlZ9utshYrhRTfEnn
-----END PRIVATE KEY-----
```

Privacy-enhanced mail

The PEM format was first developed in the privacy-enhanced mail series of RFCs: RFC 1421, RFC 1422, RFC 1423, and RFC 1424. These standards assumed prior deployment of a hierarchical public key infrastructure (PKI) with a single root. Such a PKI was never deployed, due to operational cost and legal liability concerns. These standards were eventually obsoleted by PGP and S/MIME, competing e-mail encryption standards. The initiative to develop Privacy Enhanced Mail began in 1985 on behalf of the PSRG (Privacy and Security Research Group) also known as the Internet Research Task Force.

SSL

SSL Encryption

The SSL (Secure Sockets Layer) protocol was originally designed by Netscape to protect communication between a web browser and server. It is also known now as TLS, for transport layer security. SSL interfaces between applications (such as browsers) and the TCP/IP protocols to provide server authentication, optional client authentication, and an encrypted communications channel between client and server. Client and server negotiate a mutually supported suite of encryption for session encryption and hashing; possibilities include triple DES and SHA1, or RC4 with a 128-bit key and MD5.

To use SSL, the client requests an SSL session. The server responds with its public key certificate so that the client can determine the authenticity of the server. The client returns part of a symmetric session key encrypted under the server's public key. Both the server and client compute the session key, and then they switch to encrypted communication, using the shared session key.

The protocol is simple but effective, and it is the most widely used secure communication protocol on the Internet. However, remember that SSL protects only from the client's browser to the server's decryption point (which is often only to the server's firewall or, slightly stronger, to the computer that runs the web application). Data are exposed from the user's keyboard to the browser and throughout the recipient's company. Blue Gem Security has developed a product called LocalSSL that encrypts data after it has been typed until the operating system delivers it to the client's browser, thus thwarting any keylogging Trojan horse that has become implanted in the user's computer to reveal everything the user types.

Ipsec.

As noted previously, the address space for the Internet is running out. As domain names and equipment proliferate, the original, 30-year-old, 32-bit address structure of the

Internet is filling up. A new structure, called IPv6 (version 6 of the IP protocol suite), solves the addressing problem. This restructuring also offered an excellent opportunity for the Internet
Engineering Task Force (IETF) to address serious security requirements.

As a part of the IPv6 suite, the IETF adopted IPSec, or the IP Security Protocol Suite. Designed to address fundamental shortcomings such as being subject to spoofing, eavesdropping, and session hijacking, the IPSec protocol defines a standard means for handling encrypted data. IPSec is implemented at the IP layer, so it affects all layers above it, in particular TCP and UDP. Therefore, IPSec requires no change to the existing large number of TCP and UDP protocols.

IPSec is somewhat similar to SSL, in that it supports authentication and confidentiality in a way that does not necessitate significant change either above it (in applications) or below it (in the TCP protocols). Like SSL, it was designed to be independent of specific cryptographic protocols and to allow the two communicating parties to agree on a mutually supported set of protocols.

The basis of IPSec is what is called a security association, which is essentially the set of security parameters for a secured communication channel. It is roughly comparable to an SSL session. A security association includes

- encryption algorithm and mode (for example, DES in block-chaining mode)
- encryption key
- encryption parameters, such as the initialization vector
- authentication protocol and key
- lifespan of the association, to permit long-running sessions to select a new cryptographic key as often as needed
- address of the opposite end of association
- sensitivity level of protected data (usable for classified data)

A host, such as a network server or a firewall, might have several security associations in effect for concurrent communications with different remote hosts. A security association is selected by a security parameter index (SPI), a data element that is essentially a pointer into a table of security associations.

The fundamental data structures of IPSec are the AH (authentication header) and the ESP (encapsulated security payload). The ESP replaces (includes) the conventional TCP header and data portion of a packet, as shown in Figure 7-27. The physical header and trailer depend on the data link and physical layer communications medium, such as Ethernet.

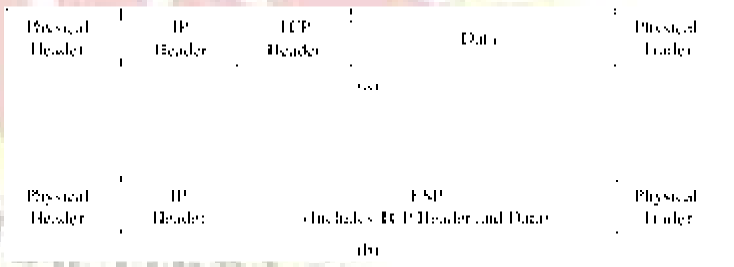


Figure 7-27. Packets: (a) Conventional Packet; (b) IPSec Packet.

The ESP contains both an authenticated portion and an encrypted portion, as shown in Figure 7-28. The sequence number is incremented by one for each packet transmitted to the same address using the same SPI, to preclude packet replay attacks. The payload data is the actual data of the packet. Because some encryption or other security mechanisms require blocks of certain sizes, the padding factor and padding length fields contain padding and the amount of padding to bring the payload data to an appropriate length. The next header indicates the type of payload data. The authentication field is used for authentication of the entire object.

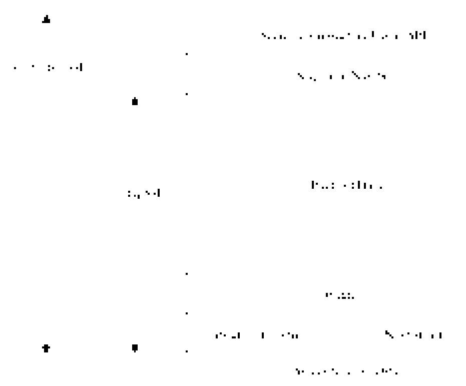


Figure 7-28. Encapsulated Security Packet.

As with most cryptographic applications, the critical element is key management. IPSec addresses this need with ISAKMP or Internet Security Association Key Management Protocol. Like SSL, ISAKMP requires that a distinct key be generated for each security association. The

ISAKMP protocol is simple, flexible, and scalable. In IPSec, ISAKMP is implemented through IKE or ISAKMP key exchange. IKE provides a way to agree on and manage protocols, algorithms, and keys. For key exchange between unrelated parties IKE uses the Diffie Hellman scheme (also described in Chapter 2). In Diffie Hellman, each of the two parties, X and Y, chooses a large prime and sends a number g raised to the power of the prime to the other. That is, X sends g_x and Y sends g_y. They both raise what they receive to the power they kept: Y raises g_x to $(g_x)_y$ and X raises g_y to $(g_y)_x$, which are both the same;voilà, they share a secret $(g_x)_y = (g_y)_x$. (The computation is slightly more complicated, being done in a finite field *mod*(*n*), so an attacker cannot factor the secret easily.) With their shared secret, the two parties now exchange identities and certificates to authenticate those identities. Finally, they derive a shared cryptographic key and enter a security association.

The key exchange is very efficient: The exchange can be accomplished in two messages, with an optional two more messages for authentication. Because this is a public key method, only two keys are needed for each pair of communicating parties. IKE has sub modes for authentication (initiation) and for establishing new keys in an existing security association.

IPSec can establish cryptographic sessions with many purposes, including VPNs, applications, and lower-level network management (such as routing). The protocols of IPSec have been published and extensively scrutinized. Work on the protocols began in 1992. They were first published in 1995, and they were finalized in 1998 (RFCs 24012409) [KEN98].

Unit 5 :

Administrating Security:

In reading this book you may have concluded by now that security is achieved through technology. You may think that the important activities in security are picking the right IDS, configuring your firewall properly, encrypting your wireless link, and deciding whether fingerprint readers are better than retina scanners. These are important matters. But not all of security is addressed by technology. Focusing on the firewall alone is like choosing a car by the shape of the headlight. Before you get to the headlights, there are some more fundamental questions to answer, such as how you intend to use the car, how much you can afford, and whether you have other transportation choices.

Security is a combination of technical, administrative, and physical controls, as we first pointed out in Chapter 1. So far, we have considered technical controls almost exclusively.

But stop and think for a moment: What good is a firewall if there is no power to run it? How effective is a public key infrastructure if someone can walk off with the certificate server? And why have elaborate access control mechanisms if your employee mails a sensitive document to a competitor? The administrative and physical controls may be less glamorous than the technical ones, but they are surely as important.

In this chapter we complete our study of security controls by considering administrative and physical aspects. We look at four related areas:

- *Planning.* What advance preparation and study lets us know that our implementation meets our security needs for today and tomorrow?
- *Risk analysis.* How do we weigh the benefits of controls against their costs, and how do we justify any controls?
- Policy. How do we establish a framework to see that our computer security needs continue to be met?
- *Physical control.* What aspects of the computing environment have an impact on security?

These four areas are just as important to achieving security as are the latest firewall or coding practice.

Security planning

Years ago, when most computing was done on mainframe computers, data processing centers were responsible for protection. Responsibility for security rested neither with the programmers nor the users but instead with the computing centers themselves. These centers developed expertise in security, and they implemented many protection activities in the background, without users having to be conscious of protection needs and practices.

Since the early 1980s, the introduction of personal computers and the general ubiquity of computing have changed the way many of us work and interact with computers. In particular, a significant amount of the responsibility for security has shifted to the user and away from the computing center. But many users are unaware of (or choose to ignore) this responsibility, so they do not deal with the risks posed or do not implement simple measures to prevent or mitigate problems.

Unfortunately, there are many common examples of this neglect. Moreover, it is exacerbated by the seemingly hidden nature of important data: Things we would protect if they were on paper are ignored when they are stored electronically. For example, a person who carefully locks up paper copies of company confidential records overnight may leave running a personal computer or terminal on an assistant's or manager's desk. In this situation, a curious or malicious person walking past can retrieve confidential memoranda and data. Similarly, the data on laptops and workstations are often more easily available than on older, more isolated systems. For instance, the large and cumbersome disk packs and tapes from a few years ago have been replaced by media such as diskettes, zip disks, and CDs, which hold a similar volume of data but fit easily in a pocket or briefcase. Moreover, we all recognize that a box of CDs or diskettes may contain many times more data than a printed report. But since the report is an apparent, visible exposure and the CD or diskette is not, we leave the computer media in plain view, easy to borrow or steal.

In all cases, whether the user initiates some computing action or simply interacts with an active application, every application has confidentiality, integrity, and availability requirements that relate to the data, programs, and computing machinery. In these situations, users suffer from lack of sensitivity: They often do not appreciate the security risks associated with using computers.

For these reasons, every organization using computers to create and store valuable assets should perform thorough and effective security planning. A security plan is a document that describes how an organization will address its security needs. The plan is subject to periodic review and revision as the organization's security needs change.

A good security plan is an official record of current security practices, plus a blueprint for orderly change to improve those practices. By following the plan, developers and users can measure the effect of proposed changes, leading eventually to further improvements. The impact of the security plan is important, too. A carefully written plan, supported by management, notifies employees that security is important to management (and therefore to everyone). Thus, the security plan has to have the appropriate content and produce the desired effects.

In this section we study how to define and implement a security plan. We focus on three aspects of writing a security plan: what it should contain, who writes it, and how to obtain support for it. Then, we address two specific cases of security plans: business continuity plans, to ensure that an organization continues to function in spite of a computer security incident, and incident response plans, to organize activity to deal with the crisis of an incident.

Contents of a Security Plan

A security plan identifies and organizes the security activities for a computing system. The plan is both a description of the current situation and a plan for improvement. Every security plan must address seven issues.

1. *policy*, indicating the goals of a computer security effort and the willingness of the people involved to work to achieve those goals
2. *current state*, describing the status of security at the time of the plan
3. requirements, recommending ways to meet the security goals
4. *recommended controls*, mapping controls to the vulnerabilities identified in the policy and requirements
5. *accountability*, describing who is responsible for each security activity
6. *timetable*, identifying when different security functions are to be done
7. *continuing attention*, specifying a structure for periodically updating the security plan

There are many approaches to creating and updating a security plan. Some organizations have a formal, defined security planning process, much as they might have a defined and accepted development or maintenance process. Others look to security professionals for guidance on how to perform security planning. For example, Sidebar 8-1 describes a security planning methodology suggested by the U.S. Software Engineering

Institute and made available on its web site. But every security plan contains the same basic material, no matter the format. The following sections expand on the seven parts of a security plan.

1. Policy

A security plan must state the organization's policy on security. A security policy is a high-level statement of purpose and intent. Initially, you might think that all policies would be the same: to prevent security breaches. But in fact the policy is one of the most difficult sections to write well. As we discuss later in this chapter, there are tradeoffs among the strength of the security, the cost, the inconvenience to users, and more.

For example, we must decide whether to implement very stringent and possibly unpopular controls that prevent all security problems or simply mitigate the effects of security breaches once they happen. For this reason, the policy statement must answer three essential questions:

• *Who* should be allowed access?
• To what system and organizational *resources* should access be allowed?
• What *types* of access should each user be allowed for each resource?

The policy statement should specify the following:

 The organization's *goals* on security. For example, should the system protect data from leakage to outsiders, protect against loss of data due to physical disaster, protect the data's integrity, or protect against loss of business when computing resources fail?
What is the higher priority: serving customers or securing data?
 Where the *responsibility* for security lies. For example, should the responsibility rest with a small computer security group, with each employee, or with relevant managers?
 The organization's *commitment* to security. For example, who provides security support for staff, and where does security fit into the organization's structure?

2. Current Security Status

To be able to plan for security, an organization must understand the vulnerabilities to which it may be exposed. The organization can determine the vulnerabilities by performing a risk analysis: a careful investigation of the system, its environment, and the things that might go wrong. The risk analysis forms the basis for describing the current status of security. The status can be expressed as a listing of organizational assets, the security threats to the assets, and the controls in place to protect the assets. We look at risk analysis in more detail later in this chapter.

The status portion of the plan also defines the limits of responsibility for security. It describes not only which assets are to be protected but also who is responsible for protecting them.

The plan may note that some groups may be excluded from responsibility; for example, joint ventures with other organizations may designate one organization to provide security for all member organizations. The plan also defines the boundaries of responsibility, especially when networks are involved. For instance, the plan should clarify who provides the security for a network router or for a leased line to a remote site.

Even though the security plan should be thorough, there will necessarily be vulnerabilities that are not considered. These vulnerabilities are not always the result of ignorance or naïveté; rather, they can arise from the addition of new equipment or data as the system evolves.

They can also result from new situations, such as when a system is used in ways not anticipated by its designers. The security plan should detail the process to be followed when someone identifies a new vulnerability. In particular, instructions should explain how to integrate controls for that vulnerability into the existing security procedures.

3. Requirements

The heart of the security plan is its set of security requirements: functional or performance demands placed on a system to ensure a desired level of security. The requirements are usually derived from organizational needs. Sometimes these needs include the need to conform to specific security requirements imposed from outside, such as by a government agency or a commercial standard.

Pfleeger [PFL91] points out that we must distinguish the requirements from constraints and controls. A constraint is an aspect of the security policy that constrains, circumscribes, or directs the implementation of the requirements. As we learned in Chapter 1, a control is an action, device, procedure, or technique that removes or reduces a vulnerability. To see the difference between requirements, constraints, and controls, consider the six "requirements" of the U.S. Department of Defense's TCSEC, introduced in Chapter 5. These six items are listed in Table 8-1.

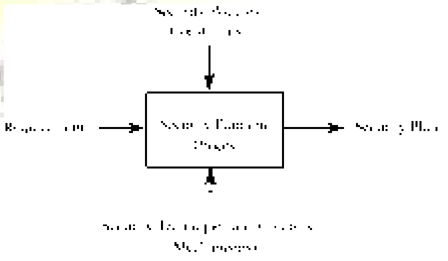
Table 8-1. The Six aRequirementsa of the TCSEC.

1. Security policy	-There must be an explicit and well-defined security policy enforced by the system.
2. Identification	-Every subject must be uniquely and convincingly identified. Identification is necessary so that subject/object access can Be checked.
3. Marking	-Every object must be associated with a label that indicates its Security level. The association must be done so that the label is available for comparison each time an access to the object is requested.
4. Accountability	-The system must maintain complete, secure records of actions that affect security. Such actions include introducing new users to the system, assigning or changing the security level of a subject or an object, and denying access attempts.
5. Assurance	-The computing system must contain mechanisms that enforce security, and it must be possible to evaluate the effectiveness of these mechanisms.
6. Continuous protection	-The mechanisms that implement security must be protected Against unauthorized change.

Given our definitions of requirement, constraint, and control, it is easy to see that the first "requirement" of the TCSEC is really a constraint: the security policy. The second and third "requirements" describe mechanisms for enforcing security, not descriptions of required behaviors. That is, the second and third "requirements" describe explicit implementations, not a general characteristic or property that the system must have. However, the fourth, fifth, and sixth TCSEC "requirements" are indeed true requirements. They state that the system must have certain characteristics, but they do not enforce a particular implementation.

These distinctions are important because the requirements explain *what* should be accomplished, not *how*. That is, the requirements should always leave the implementation details to the designers, whenever possible. For example, rather than writing a requirement that certain data records should require passwords for access (an implementation decision), a security planner should state only that access to the data records should be restricted (and note to whom the access should be restricted). This more flexible requirement allows the designers to decide among several other access controls (such as access control lists) and to balance the security requirements with other system requirements, such as performance and reliability. Figure 8-1 illustrates how the different aspects of system analysis support the security planning process.

Figure 8-1. Inputs to the Security Plan.



As with the general software development process, the security planning process must allow customers or users to specify desired functions, independent of the implementation. The requirements should address all aspects of security: confidentiality, integrity, and availability.

They should also be reviewed to make sure that they are of appropriate quality. In particular,
we should make sure that the requirements have these characteristics:

- *Correctness:* Are the requirements understandable? Are they stated without error?
- *Consistency:* Are there any conflicting or ambiguous requirements?
- *Completeness:* Are all possible situations addressed by the requirements?
- *Realism:* Is it possible to implement what the requirements mandate?
- *Need:* Are the requirements unnecessarily restrictive?

 Verifiability: Can tests be written to demonstrate conclusively and objectively that the requirements have been met? Can the system or its functionality be measured in some way that will assess the degree to which the requirements are met?
 Traceability: Can each requirement be traced to the functions and data related to it so that changes in a requirement can lead to easy reevaluation?

The requirements may then be constrained by budget, schedule, performance, policies, governmental regulations, and more. Given the requirements and constraints, the developers then choose appropriate controls.

4. Recommended Controls

The security requirements lay out the system's needs in terms of what should be protected. The security plan must also recommend what controls should be incorporated into the system to meet those requirements. Throughout this book you have seen many examples of controls, so we need not review them here. As we see later in this chapter, we can use risk analysis to create a map from vulnerabilities to controls. The mapping tells us how the system will meet the security requirements. That is, the recommended controls address implementation issues:
how the system will be designed and developed to meet stated security requirements.

5. Responsibility for Implementation

A section of the security plan should identify which people are responsible for implementing the security requirements. This documentation assists those who must coordinate their individual responsibilities with those of other developers. At the same time, the plan makes explicit who is accountable should some requirement not be met or some vulnerability not be addressed. That is, the plan notes who is responsible for implementing controls when a new vulnerability is discovered or a new kind of asset is introduced. (But see Sidebar 8-2 on who is responsible.)

People building, using, and maintaining the system play many roles. Each role can take some responsibility for one or more aspects of security. Consider, for example, the groups listed here.

 Personal computer users may be responsible for the security of their own machines. Alternatively, the security plan may designate one person or group to be coordinator of personal computer security.
 Project leaders may be responsible for the security of data and computations.
 Managers may be responsible for seeing that the people they supervise implement security measures.
 Database administrators may be responsible for the access to and integrity of data in their databases.
 Information officers may be responsible for overseeing the creation and use of data; these officers may also be responsible for retention and proper disposal of data.
 Personnel staff members may be responsible for security involving employees, for example, screening potential employees for trustworthiness and arranging security training programs.

6. Timetable

A comprehensive security plan cannot be executed instantly. The security plan includes a timetable that shows how and when the elements of the plan will be performed. These dates also give milestones so that management can track the progress of implementation.

If the implementation is to be a phased development (that is, the system will be implemented partially at first, and then changed functionality or performance will be added in later releases), the plan should also describe how the security requirements will be implemented over time. Even when overall development is not phased, it may be desirable to implement the security aspects of the system over time. For example, if the controls are expensive or complicated, they may be acquired and implemented gradually. Similarly, procedural controls may require staff training to ensure that everyone understands and accepts the reason for the control.

The plan should specify the order in which the controls are to be implemented so that the most serious exposures are covered as soon as possible. A timetable also gives milestones by which to judge the progress of the security program.

Furthermore, the plan must be extensible. Conditions will change: New equipment will be acquired, new degrees and modes of connectivity will be requested, and new threats will be identified. The plan must include a procedure for change and growth, so that the security aspects of changes are considered as a part of preparing for the change, not for adding security after the change has been made. The plan should also contain a schedule for periodic review. Even though there may have been no obvious, major growth, most organizations experience modest change every day. At some point the cumulative impact of the change is enough to require the plan to be modified.

7. Continuing Attention

Good intentions are not enough when it comes to security. We must not only take care in defining requirements and controls, but we must also find ways for evaluating a system's security to be sure that the system is as secure as we intend it to be. Thus, the security plan must call for reviewing the security situation periodically. As users, data, and equipment change, new exposures may develop. In addition, the current means of control may become obsolete or ineffective (such as when faster processor times enable attackers to break an encryption algorithm). The inventory of objects and the list of controls should periodically be scrutinized and updated, and risk analysis performed anew. The security plan should set times for these periodic reviews, based either on calendar time (such as, review the plan every nine months) or on the nature of system changes (such as, review the plan after every major system release).

Security Planning Team Members

Who performs the security analysis, recommends a security program, and writes the security plan? As with any such comprehensive task, these activities are likely to be performed by a committee that represents all the interests involved. The size of the committee depends on the size and complexity of the computing organization and the degree of its commitment to security. Organizational behavior studies suggest that the optimum size for a working committee is between five and nine members. Sometimes a larger committee may serve as an oversight body to review and comment on the products of a smaller working committee.

Alternatively, a large committee might designate subcommittees to address various sections of the plan.

The membership of a computer security planning team must somehow relate to the different aspects of computer security described in this book. Security in operating systems and networks requires the cooperation of the systems administration staff. Program security measures can be understood and recommended by applications programmers. Physical security controls are implemented by those responsible for general physical security, both against human attacks and natural disasters. Finally, because controls affect system users, the plan should incorporate users' views, especially with regard to usability and the general desirability of controls.

Thus, no matter how it is organized, a security planning team should represent each of the following groups.

- computer hardware group
- system administrators
- systems programmers
- applications programmers
- data entry personnel
- physical security personnel
- representative users

In some cases, a group can be adequately represented by someone who is consulted at appropriate times, rather than a committee member from each possible constituency being enlisted.

Assuring Commitment to a Security Plan

After the plan is written, it must be accepted and its recommendations carried out. Acceptance by the organization is key; a plan that has no organizational commitment is simply a plan that collects dust on the shelf. Commitment to the plan means that security functions will be implemented and security activities carried out. Three groups of people must contribute to making the plan a success.

- The planning team must be sensitive to the needs of each group affected by the plan.

 Those affected by the security recommendations must understand what the plan means for the way they will use the system and perform their business activities. In particular, they must see how what they do can affect other users and other systems.
 Management must be committed to using and enforcing the security aspects of the system.

Education and publicity can help people understand and accept a security plan. Acceptance involves not only the letter but also the spirit of the security controls. Recall from Chapter 4 the employee who went through 24 password changes at a time to get back to a favorite password, in a system that prevented use of any of the 23 most recently used passwords.

Clearly, the employee either did not understand or did not agree with the reason for restrictions on passwords. If people understand the need for recommended controls and accept them as sensible, they will use the controls properly and effectively. If people think the controls are bothersome, capricious, or counterproductive, they will work to avoid or subvert them.

Management commitment is obtained through understanding. But this understanding is not just a function of what makes sense technologically; it also involves knowing the cause and the potential effects of lack of security. Managers must also weigh tradeoffs in terms of convenience and cost. The plan must present a picture of how cost effective the controls are, especially when compared to potential losses if security is breached without the controls.

Thus, proper presentation of the plan is essential, in terms that relate to management as well as technical concerns.

Remember that some managers are not computing specialists. Instead, the system supports a manager who is an expert in some other business function, such as banking, medical technology, or sports. In such cases, the security plan must present security risks in language that the managers understand. It is important to avoid technical jargon and to educate the readers about the nature of the perceived security risks in the context of the business the system supports. Sometimes outside experts can bridge the gap between the managers' business and security.

Management is often reticent to allocate funds for controls until the value of those controls is explained. As we note in the next section, the results of a risk analysis can help communicate the financial tradeoffs and benefits of implementing controls. By describing vulnerabilities in financial terms and in the context of ordinary business activities (such as leaking data to a competitor or an outsider), security planners can help managers understand the need for controls.

The plans we have just discussed are part of normal business. They address how a business handles computer security needs. Similar plans might address how to increase sales or improve product quality, so these planning activities should be a natural part of management.

Next we turn to two particular kinds of business plans that address specific security problems: coping with and controlling activity during security incidents.

Business Continuity Plans

Small companies working on a low profit margin can literally be put out of business by a computer incident. Large, financially sound businesses can weather a modest incident that interrupts their use of computers for a while, although it is painful to them.

But even rich companies do not want to spend money unnecessarily. The analysis is sometimes as simple as *no computers means no customers means no sales means no profit.* Government agencies, educational institutions, and nonprofit organizations also have limited budgets, which they want to use to further their needs. They may not have a direct profit motive, but being able to meet the needs of their customersthe public, students, and constituents partially determines how well they will fare in the future. All kinds of organizations must plan for ways to cope with emergency situations.

A business continuity plan[1] documents how a business will continue to function during a computer security incident. An ordinary security plan covers computer security during normal times and deals with protecting against a wide range of vulnerabilities from the usual sources.

A business continuity plan deals with situations having two characteristics:

[1] The standard terminology is a business continuity plan, an even though such a plan is needed by and applies to an university's abusiness of educating students or a government's business of serving the public.

 catastrophic situations, in which all or a major part of a computing capability is suddenly unavailable

 long duration, in which the outage is expected to last for so long that business will suffer

There are many situations in which a business continuity plan would be helpful. Here are some examples that typify what you might find in reading your daily newspaper:

 A fire destroys a company's entire network.

 A seemingly permanent failure of a critical software component renders the computing system unusable.

 A business must deal with the abrupt failure of its supplier of electricity, telecommunications, network access, or other critical service.

 A flood prevents the essential network support staff from getting to the operations center.

As you can see, these examples are likely to recur, and each disables a vital function You may also have noticed how often "the computer" is blamed for an inability to provide a service or product. For instance, the clerk in a shop is unable to use the cash register because "the computer is down." You may have a CD in your hand, plus exactly the cash to pay for it. But the clerk will not take your money and send you on your way. Often, computer service is restored shortly. But sometimes it is not. Once we were delayed for over an hour in an airport because of an electrical storm that caused a power failure and disabled the airlines' computers. Although our tickets showed clearly our reservations on a particular flight, the airline agents refused to let anyone board because they could not assign seats. As the computer remained down, the agents were frantic[2] because the technology was delaying the flight and, more importantly, disrupting hundreds of connections.

[2] The obvious, at least to us, idea of telling passengers to asit in any seata seemed to be against airline policy.

The key to coping with such disasters is advance planning and preparation, identifying activities that will keep a business viable when the computing technology is disabled. The steps in business continuity planning are these:

 Assess the business impact of a crisis.

 Develop a strategy to control impact.

 Develop and implement a plan for the strategy

Assess Business Impact

To assess the impact of a failure on your business, you begin by asking two key questions:

• What are the *essential assets*? What are the things that will prevent the business from doing business? Answers are typically of the form "the network," "the customer reservations database," or "the system controlling traffic lights."

• What could *disrupt use* of these assets? The vulnerability is more important than the threat agent. For example, whether destroyed by a fire or zapped in an electrical storm, the network is nevertheless down. Answers might be "failure," "corrupted," or "loss of power."

You probably will find only a handful of key assets when doing this analysis. Do not overlook people and the things they need for support, such as documentation and communications equipment. Another way to think about your assets is to ask yourself, "What is the minimum set of things or activities needed to keep business operational, at least to some degree?" If a manual system would compensate for a failed computer system, albeit inefficiently, you may want to consider building such a manual system as a potential critical asset. Think of the airline unable to assign seats from a chart of the cabin.

Later in this chapter we study risk analysis, a comprehensive examination of assets, vulnerabilities, and controls. For business continuity planning we do not need a full risk analysis. Instead, we focus on only those things that are critical to continued operation. We also look at larger classes of objects, such as "the network," whose loss or compromise can have catastrophic effect.

Develop Strategy

The continuity strategy investigates how the key assets can be safeguarded. In some cases, a backup copy of data or redundant hardware or an alternative manual process is good enough. Sometimes, the most reasonable answer is reduced capacity. For example, a planner might conclude that if the call center in London fails, the business can divert all calls to Tokyo. It is possible, though, that the staff in Tokyo cannot handle the full load of the London traffic; this situation may result in irritated or even lost customers, but at least some business can be transacted.

Ideally, you would like to continue business with no loss. But with catastrophic failures, usually only a portion of the business function can be preserved. In this case, you must develop a strategy appropriate for your business and customers. For instance, you can decide whether it is better to preserve half of function A and half of B, or most of A and none of B. You also must consider the time frame in which business is done. Some catastrophes last longer than others. For example, rebuilding after a fire is a long process and implies a long time in disaster mode. Your strategy may have several steps, each dependent on how long the business is disabled. Thus, you may take one action in response to a one-hour outage, and another if the outage might last a day or longer.

Because you are planning in advance, you have the luxury of being able to think about possible circumstances and evaluate alternatives. For instance, you may realize that if the Tokyo site takes on work for the disabled London site, there will be a significant difference in time zones. It may be better to divert morning calls to Tokyo and afternoon ones to Dallas, to avoid asking Tokyo workers to work extra hours.

The result of a strategy analysis is a selection of the best actions, organized by circumstances. The strategy can then be used as the basis for your business continuity plan.

Develop Plan

The business continuity plan specifies several important things:

- who is in charge when an incident occurs
- what to do
- who does it

The plan justifies making advance arrangements, such as acquiring redundant equipment, arranging for data backups, and stockpiling supplies, before the catastrophe. The plan also justifies advance training so that people know how they should react. In a catastrophe there will be confusion; you do not want to add confused people to the already severe problem.

The person in charge declares the state of emergency and instructs people to follow the procedures documented in the plan. The person in charge also declares when the emergency is over and conditions can revert to normal.

Thus, the business continuity planning addresses how to maintain some degree of critical business activity in spite of a catastrophe. Its focus is on keeping the business viable. It is based on the asset survey, which focuses on only a few critical assets and serious vulnerabilities that could threaten operation for a long or undetermined period of time.

The focus of the business continuity plan is to keep the business going while someone else addresses the crisis. That is, the business continuity plan does not include calling the fire department or evacuating the building, important though those steps are. The focus of a business continuity plan is the *business* and how to keep it functioning to the degree possible in the situation. Handling the emergency is someone else's problem.

Now we turn to a different plan that deals specifically with computer crises.

Incident Response Plans

An incident response plan tells the staff how to deal with a security incident. In contrast to the business continuity plan, the goal of incident response is handling the current security incident, without regard for the business issues. The security incident may at the same time be a business catastrophe, as addressed by the business continuity plan. But as a specific security event, it might be less than catastrophic (that is, it may not interrupt business severely) but could be a serious breach of security, such as a hacker attack or a case of internal fraud. An incident could be a single event, a series of events, or an ongoing problem.

An incident response plan should

- define what constitutes an *incident*
- identify who is responsible for *taking charge* of the situation
- describe the plan of *action*

The plan usually has three phases: advance planning, triage, and running the incident. A fourth phase, review, is useful after the situation abates so that this incident can lead to improvement for future incidents.

Advance Planning

As with all planning functions, advance planning works best because people can think logically, unhurried, and without pressure. What constitutes an incident may be vague. We cannot know the details of an incident in advance. Typical characteristics include harm or risk of harm to computer systems, data, or processing; initial uncertainty as to the extent of damage; and similar uncertainty as to the source or method of the incident. For example, you can see that the file is missing or the home page has been defaced, but you do not know how or by whom or what other damage there may be.

In organizations that have not done incident planning, chaos may develop at this point. Someone calls the network manager. Someone sends e-mail to the help desk. Someone calls the FBI, the CERT, the newspapers, or the fire department. People start to investigate on their own, without coordinating with the relevant staff in other departments, agencies, or businesses. And there is a lot of conversation, rumor, and misinformation: more heat than light.

With an incident response plan in place, everybody is trained in advance to call the designated leader. There is an established list of people to call, in order, in case the first person is unavailable. The leader decides what to do next, and he or she begins by determining if this is a real incident or a false alarm. Indeed, natural events sometimes look like incidents, and the facts of the situation should be established first. If the leader decides this may be a real incident, he or she invokes the response team.

Response Team

The response team is the set of people charged with responding to the incident. The response team may include

 director: person in charge of the incident, who decides what actions to take and when to terminate the response. The director is typically a management employee.

 lead technician: person who directs and coordinates the response. The lead technician decides where to focus attention, analyzes situation data, documents the incident and how it was handled, and calls for other technical people to assist with the analysis.

 advisor(s): legal, human resources, or public relations staff members as appropriate. In a small incident a single person can handle more than one of these roles. Nevertheless, it is important that a single person be in charge, a single person who directs the response work, a single point of contact for "insiders" (employees, users), and a single point of contact for "the public."

To develop policy and identify a response team, you need to consider certain matters.

 Legal issues: An incident has legal ramifications. In some countries, computer intrusions are illegal, so law enforcement officials must be involved in the investigation. In other places, you have discretion in deciding whether to ask law enforcement to participate.

In addition to criminal action, you may be able to bring a civil case. Both kinds of legal action have serious implications for the response. For example, evidence must be gathered and maintained in specific ways in order to be usable in court. Similarly, laws may limit what you can do against the alleged attacker: Cutting off a connection is probably acceptable, but launching a retaliatory denial-of-service attack may not be.

 Preserving evidence: The most common reaction in an incident is to assume the cause was internal or accidental. For instance, you may surmise that the hardware has failed or that the software isn't working correctly. The staff may be directed to change the configuration, reload the software, reboot the system, or similarly attempt to resolve the problem by adjusting the software. Unfortunately, each of these acts can irreparably distort or destroy evidence. When dealing with a possible incident, do as little as possible before "dusting for fingerprints."

 Records: It may be difficult to remember what you have already done: Have you already reloaded a particular file? What steps got you to the prompt asking for the new DNS server's address? If you call in an outside forensic investigator or the police, you will need to tell exactly what you have already done.

 Public relations: In handling an incident your organization should speak with one voice.

You risk sending confusing messages if too many people speak. It is especially important that only one person speak publicly if legal action may be taken. An unguarded comment may tip off the attacker or have a negative effect on the case.

You can simply say that an incident occurred, tell briefly and generally what it was,

and state that the incident is now under control and normal operation is resuming.
After the Incident Is Resolved

Eventually, the incident response team closes the case. At this point it will hold a review after the incident to consider two things:

- *Is any security control action to be taken?* Did an intruder compromise a system because security patches were not up-to-date; if so, should there be a procedure to ensure that patches are applied when they become available? Was access obtained because of a poorly chosen password; if so, should there be a campaign to educate users on how to strong passwords? If there were control failures, what should be done to prevent similar attacks in the future?
- *Did the incident response plan work?* Did everyone know whom to notify? Did the team have needed resources? Was the response fast enough? What should be done differently next time?

The incident response plan ensures that incidents are handled promptly, efficiently, and with minimal harm.

Risk analysis

Good, effective security planning includes a careful risk analysis. A risk is a potential problem that the system or its users may experience. We distinguish a risk from other project events by looking for three things, as suggested by Rook [ROO93]:

1. *A loss associated with an event.* The event must generate a negative effect: compromised security, lost time, diminished quality, lost money, lost control, lost understanding, and so on. This loss is called the risk impact.
2. *The likelihood that the event will occur.* The probability of occurrence associated with each risk is measured from 0 (impossible) to 1 (certain). When the risk probability is 1, we say we have a problem.
3. *The degree to which we can change the outcome.* We must determine what, if anything, we can do to avoid the impact or at least reduce its effects. Risk control involves a set of actions to reduce or eliminate the risk. Many of the security controls we describe in this book are examples of risk control.

We usually want to weigh the pros and cons of different actions we can take to address each risk. To that end, we can quantify the effects of a risk by multiplying the risk impact by the risk probability, yielding the risk exposure. For example, if the likelihood of virus attack is 0.3 and the cost to clean up the affected files is $10,000, then the risk exposure is $3,000. So we can use a calculation like this one to decide that a virus checker is worth an investment of $100, since it will prevent a much larger potential loss. Clearly, risk probabilities can change over time, so it is important to track them and plan for events accordingly.

Risk is inevitable in life: Crossing the street is risky but that does not keep us from doing it. We can identify, limit, avoid, or transfer risk but we can seldom eliminate it. In general, we have three strategies for dealing with risk:

1. *avoiding* the risk, by changing requirements for security or other system characteristics
2. *transferring* the risk, by allocating the risk to other systems, people, organizations, or assets; or by buying insurance to cover any financial loss should the risk become a reality
3. *assuming* the risk, by accepting it, controlling it with available resources, and preparing to deal with the loss if it occurs

Thus, costs are associated not only with the risk's potential impact but also with reducing it. Risk leverage is the difference in risk exposure divided by the cost of reducing the risk. In other words, risk leverage is If the leverage value of a proposed action is not high enough, then we look for alternative but less costly actions or more effective reduction techniques.

$$\frac{(\text{risk exposure before reduction}) - (\text{risk exposure after reduction})}{(\text{cost of risk reduction})}$$

Risk analysis is the process of examining a system and its operational context to determine possible exposures and the potential harm they can cause. Thus, the first step in a risk analysis is to identify and list all exposures in the computing system of interest. Then, for each exposure, we identify possible controls and their costs. The last step is a cost benefit analysis: Does it cost less to implement a control or to accept the expected cost of the loss?

In the remainder of this section, we describe risk analysis, present examples of risk analysis methods, and discuss some of the drawbacks to performing risk analysis.

The Nature of Risk

In our everyday lives, we take risks. In crossing the road, eating oysters, or playing the lottery, we take the chance that our actions may result in some negative result such as being injured, getting sick, or losing money. Consciously or unconsciously, we weigh the benefits of taking the action with the possible losses that might result. Just because there is a risk to a certain act we do not necessarily avoid it; we may look both ways before crossing the street, but we do cross it. In building and using computing systems, we must take a more organized and careful approach to assessing our risks. Many of the systems we build and use can have a dramatic impact on life and health if they fail. For this reason, risk analysis is an essential part of security planning.

We cannot guarantee that our systems will be risk free; that is why our security plans must address actions needed should an unexpected risk become a problem. And some risks are simply part of doing business; for example, as we have seen, we must plan for disaster recovery, even though we take many steps to avoid disasters in the first place.

When we acknowledge that a significant problem cannot be prevented, we can use controls to reduce the seriousness of a threat. For example, you can back up files on your computer as a defense against the possible failure of a file storage device. But as our computing systems become more complex and more distributed, complete risk analysis becomes more difficult and time consuming and more essential.

Steps of a Risk Analysis

Risk analysis is performed in many different contexts; for example, environmental and health risks are analyzed for activities such as building dams, disposing of nuclear waste, or changing a manufacturing process. Risk analysis for security is adapted from more general management practices, placing special emphasis on the kinds of problems likely to arise from security issues. By following well-defined steps, we can analyze the security risks in a computing system.

The basic steps of risk analysis are listed below.

1. Identify assets.
2. Determine vulnerabilities.
3. Estimate likelihood of exploitation.
4. Compute expected annual loss.
5. Survey applicable controls and their costs.
6. Project annual savings of control.

Sidebar 8-3 illustrates how different organizations take slightly different approaches, but the basic activities are still the same. These steps are described in detail in the following sections.

Step 1: Identify Assets

Before we can identify vulnerabilities, we must first decide what we need to protect. Thus, the first step of a risk analysis is to identify the assets of the computing system. The assets can be considered in categories, as listed below. The first three categories are the assets identified in Chapter 1 and described throughout this book. The remaining items are not strictly a part of a computing system but are important to its proper functioning.

- *hardware*: processors, boards, keyboards, monitors, terminals, microcomputers, workstations, tape drives, printers, disks, disk drives, cables, connections, communications controllers, and communications media
- *software*: source programs, object programs, purchased programs, in-house programs, utility programs, operating systems, systems programs (such as compilers), and maintenance diagnostic programs
- *data*: data used during execution, stored data on various media, printed data, archival data, update logs, and audit records
- *people:* skills needed to run the computing system or specific programs
- *documentation*: on programs, hardware, systems, administrative procedures, and the entire system
- *supplies*: paper, forms, laser cartridges, magnetic media, and printer fluid

It is essential to tailor this list to your own situation. No two organizations will have the same assets to protect, and something that is valuable in one organization may not be as valuable to another. For example, if a project has one key designer, then that designer is an essential asset; on the other hand, if a similar project has ten designers, any of whom could do the project's design, then each designer is not as essential because there are nine easily available replacements. Thus, you must add to the list of assets the other people, processes, and things that must be protected.

For example, RAND Corporation's Vulnerability Assessment and Mitigation (VAM) methodology [ANT02] includes additional assets, such as

 the enabling infrastructure
 the building or vehicle in which the system will reside
 the power, water, air, and other environmental conditions necessary for proper functioning
 human and social assets, such as policies, procedures, and training

The VAM methodology is a process supported by a tool to help people identify assets, vulnerabilities, and countermeasures. We use other aspects of VAM as an example technique in later risk analysis steps.

In a sense, the list of assets is an inventory of the system, including intangibles and human resource items. For security purposes, this inventory is more comprehensive than the traditional inventory of hardware and software often performed for configuration management or accounting purposes. The point is to identify all assets necessary for the system to be usable.

Step 2: Determine Vulnerabilities

The next step in risk analysis is to determine the vulnerabilities of these assets. This step requires imagination; we want to predict what damage might occur to the assets and from what sources. We can enhance our imaginative skills by developing a clear idea of the nature of vulnerabilities. This nature derives from the need to ensure the three basic goals of computer security: confidentiality, integrity, and availability. Thus, a vulnerability is any situation that could cause loss of confidentiality, integrity, and availability. We want to use an organized approach to considering situations that could cause these losses for a particular object.

Software engineering offers us several techniques for investigating possible problems. Hazard analysis, described in Sidebar 8-4, explores failures that may occur and faults that may cause them. These techniques have been used successfully in analyzing safety-critical systems.

However, additional techniques are tailored specifically to security concerns; we address those techniques in this and following sections.

To organize the way we consider threats and assets, we can use a matrix such as the one shown in Table 8-2. One vulnerability can affect more than one asset or cause more than one type of loss. The table is a guide to stimulate thinking, but its format is not rigid.

Table 8-2. Assets and Security Properties.

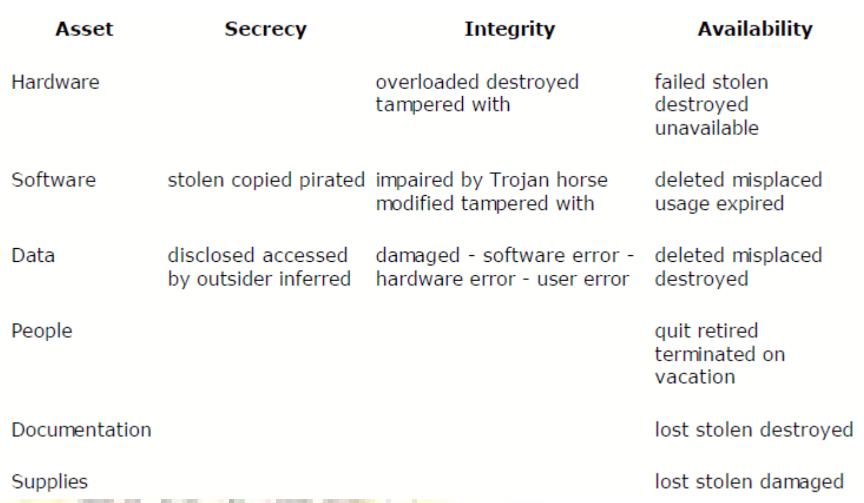
Asset	Confidentiality	Integrity	Availability
Hardware			
Software			
Data			
People			
Documentation			
Supplies			

In thinking about the contents of each matrix entry, we can ask the following questions.

• What are the effects of unintentional errors? Consider typing the wrong command, entering the wrong data, using the wrong data item, discarding the wrong listing, and disposing of output insecurely.
• What are the effects of willfully malicious insiders? Consider disgruntled employees, bribery, and curious browsers.
• What are the effects of outsiders? Consider network access, dial-in access, hackers, people walking through the building, and people sifting through the trash.
• What are the effects of natural and physical disasters? Consider fires, storms, floods, power outages, and component failures.

Table 8-3 is a version of the previous table with some of the entries filled in. It shows that certain general problems can affect the assets of a computing system. In a given installation, it is necessary to determine what can happen to specific hardware, software, data items, and other assets.

Table 8-3. Assets and Attacks.



Asset	Secrecy	Integrity	Availability
Hardware		overloaded destroyed tampered with	failed stolen destroyed unavailable
Software	stolen copied pirated	impaired by Trojan horse modified tampered with	deleted misplaced usage expired
Data	disclosed accessed by outsider inferred	damaged - software error - hardware error - user error	deleted misplaced destroyed
People			quit retired terminated on vacation
Documentation			lost stolen destroyed
Supplies			lost stolen damaged

Some organizations use other approaches to determining vulnerabilities and assessing their
importance. For example, Sidebar 8-5 describes the U.S. Navy's approach to vulnerability evaluation.

Alas, there is no simple checklist or easy procedure to list all vulnerabilities. But from the earlier chapters of this book you have seen many examples of vulnerabilities to assets, and your mind has been trained to think of harm that can occur. Tools can help us conceive of vulnerabilities by providing a structured way to think. For example, RAND's VAM methodology suggests that assets have certain properties that make them vulnerable. The properties exist in three categories: aspects of the design or architecture, aspects of behavior, and general attributes. Table 8-4 lists these properties in more detail. Notice that the properties apply to
many kinds of systems and at various places within a given system.

Table 8-4. Attributes Contributing to Vulnerabilities.

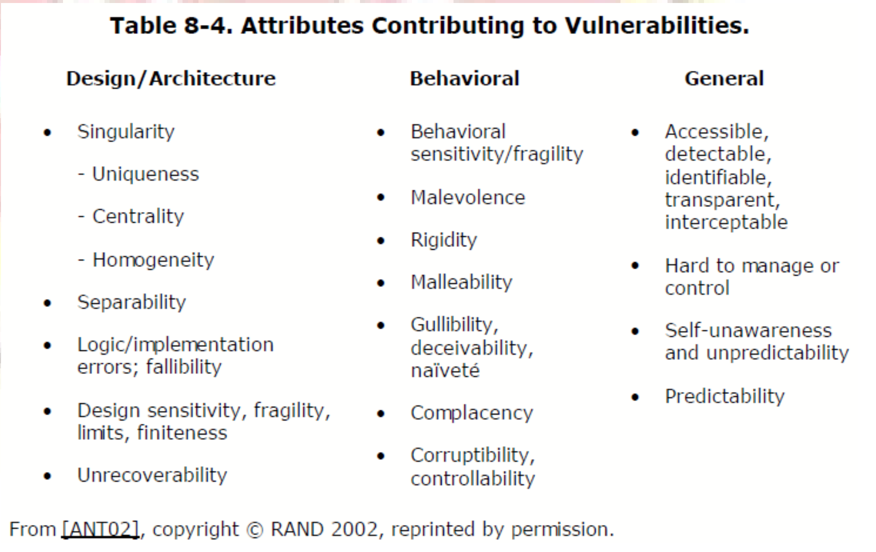


Table 8-4. Attributes Contributing to Vulnerabilities.

Design/Architecture	Behavioral	General
• Singularity	• Behavioral sensitivity/fragility	• Accessible, detectable, identifiable, transparent, interceptable
- Uniqueness	• Malevolence	• Hard to manage or control
- Centrality	• Rigidity	• Self-unawareness and unpredictability
- Homogeneity	• Malleability	• Predictability
• Separability	• Gullibility, deceivability, naïveté	
• Logic/implementation errors; fallibility	• Complacency	
• Design sensitivity, fragility, limits, finiteness	• Corruptibility, controllability	
• Unrecoverability		

From [ANT02], copyright © RAND 2002, reprinted by permission.

These attributes can be used to build a matrix, each of whose entries may suggest one or more vulnerabilities. An example of such a matrix is shown in Figure 8-2. Using that matrix for example, the design attribute *limits, finiteness* applied to a *cyber object*, a *software program* could lead you to suspect buffer overflow vulnerabilities, or *uniqueness* for a *hardware object* could signal a single point of failure. To use this methodology you would work through the matrix, thinking of each contributing attribute on each asset class to derive the set of vulnerabilities.

Figure 8-2. Vulnerabilities Suggested by Attributes and Objects. (From [ANT02], copyright © RAND 2002, reprinted by permission.)

Antón et al. [ANT02] point out that it is not enough to fill in the matrix cells. We must also consider combinations of situations that might enable certain vulnerabilities. For example, as Figure 8-3 shows, at least six attributes can allow a successful attack by Trojan horse. The homogeneity of the design or architecture may encourage an attacker to place a Trojan horse in a well-understood location. The horse may be loaded by a gullible user who downloads a seemingly benign file. To do this, the attacker must have some control over users and their machines; in general, this is a manifestation of the accessibility of systems, especially on the Internet, and the lack of user awareness when a remote site sends data to an unsuspecting system.

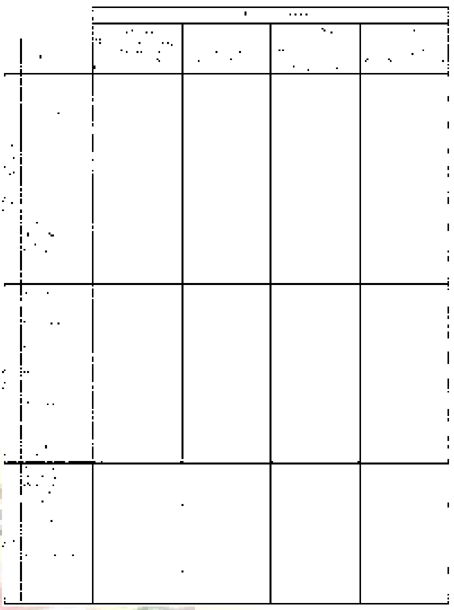
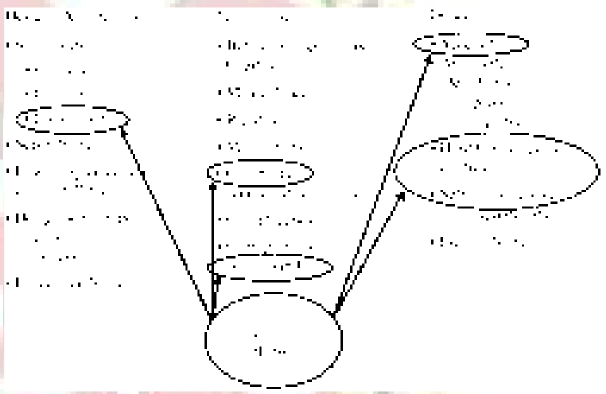


Figure 8-3. Vulnerabilities Enabling a Trojan Horse Attack. (From [ANT02], copyright © RAND 2002, reprinted by permission.)



Step 3: Estimate Likelihood of Exploitation

The third step in conducting a risk analysis is determining how often each exposure is likely to be exploited. Likelihood of occurrence relates to the stringency of the existing controls and the likelihood that someone or something will evade the existing controls. Sidebar 8-6 describes several approaches to computing the probability that an event will occur: classical, frequency, and subjective. Each approach has its advantages and disadvantages, and we must choose the approach that best suits the situation (and its available information).

In security, it is often not possible to directly evaluate an event's probability by using classical techniques. However, we can try to apply frequency probability by using observed data for a specific system. Local failure rates are fairly easy to record, and we can identify which failures resulted in security breaches or created new vulnerabilities. In particular, operating systems can track data on hardware failures, failed login attempts, numbers of accesses, and changes in the sizes of data files.

Another alternative is to estimate the number of occurrences in a given time period. We can ask an analyst familiar with the system to approximate the number of times a described event occurred in the last year, for example. Although the count is not exact (because the analyst is unlikely to have complete information), the analyst's knowledge of the system and its usage may yield reasonable estimates.

Of course, the two methods described depend on the fact that a system is already built and has been in use for some period of time. In many cases, and especially for proposed systems, the usage data are not available. In this case, we may ask an analyst to estimate likelihood by reviewing a table based on a similar system; this approach is incorporated in several formal security

Table 8-5. Ratings of Likelihood.

Frequency	**Rating**
More than once a day	10
Once a day	9
Once every three days	8
Once a week	7
Once in two weeks	6
Once a month	5
Once every four months	4
Once a year	3
Once every three years	2
Less than once in three years	1

risk processes. For example, the analyst may be asked to choose one of the ratings shown in Table 8-5. Completing this analysis depends on the rater's professional expertise.

The table provides the rater with a framework within which to consider each likelihood. Differences between close ratings are not very significant. A rater should be able to distinguish between something that happens once a year and once a month.

The Delphi approach is a subjective probability technique originally devised by RAND [HAL67] to deal with public policy decisions. It assumes that experts can make informed estimates based on their experience; the method brings a group of experts to consensus. The first step in using Delphi is to provide each of several experts with information describing the situation surrounding the event under consideration. For example, the experts may be told about the software and hardware architecture, conditions of use, and expertise of users. Then, each expert individually estimates the likelihood of the event. The estimates are collected, reproduced, and distributed to all experts. The individual estimates are listed anonymously, and the experts are usually given some statistical information, such as mean or median. The experts are then asked whether they wish to modify their individual estimates in light of values their colleagues have supplied. If the revised values are reasonably consistent, the process ends with the group's reaching consensus. If the values are inconsistent, additional rounds of revision may occur until consensus is reached.

Step 4: Compute Expected Loss

By this time, we have gained an understanding of the assets we value, their possible vulnerabilities, and the likelihood that the vulnerabilities will be exploited. Next, we must determine the likely loss if the exploitation does indeed occur. As with likelihood of occurrence, this value is difficult to determine. Some costs, such as the cost to replace a hardware item, are easy to obtain. The cost to replace a piece of software can be approximated reasonably well from the initial cost to buy it (or specify, design, and write it). However, we must take care to include hidden costs in our calculations. For instance, there is a cost to others of not having a piece of hardware or software. Similarly, there are costs in restoring a system to its previous state, reinstalling software, or deriving a piece of information. These costs are substantially harder to measure.

In addition, there may be hidden costs that involve legal fees if certain events take place. For example, some data require protection for legal reasons. Personal data, such as police records, tax information, census data, and medical information, are so sensitive that there are criminal penalties for releasing the data to unauthorized people. Other data are company confidential; their release may give competitors an edge on new products or on likely changes to the stock price. Some financial data, especially when they reflect an adverse event, could seriously affect public confidence in a bank, an insurance company, or a stock brokerage. It is difficult to determine the cost of releasing these data.

If a computing system, a piece of software, or a key person is unavailable, causing aparticular computing task to be delayed, there may be serious consequences. If a program that prints paychecks is delayed, employees' confidence in the company may be shaken, or some employees may face penalties from not being able to pay their own bills. If customers cannot make transactions because the computer is down, they may choose to take their business to a competitor. For some time-critical services involving human lives, such as a hospital's life-support systems or a space station's guidance systems, the costs of failure are infinitely high.

Thus, we must analyze the ramifications of a computer security failure. The following questions can prompt us to think about issues of explicit and hidden cost related to security.

The answers may not produce precise cost figures, but they will help identify the sources of various types of costs.

• What are the legal obligations for preserving the confidentiality or integrity of a given data item?

• What business requirements and agreements cover the situation? Does the organization have to pay a penalty if it cannot provide a service?

• Could release of a data item cause harm to a person or organization? Would there be the possibility of legal action if harm were done?

• Could unauthorized access to a data item cause the loss of future business opportunity? Might it give a competitor an unfair advantage? What would be the estimated loss in revenue?
• What is the psychological effect of lack of computer service? Embarrassment? Loss of credibility? Loss of business? How many customers would be affected? What is their value as customers?
• What is the value of access to data or programs? Could this computation be deferred? Could this computation be performed elsewhere? How much would it cost to have a third party do the computing elsewhere?
• What is the value to someone else of having access to data or programs? How much would a competitor be willing to pay for access?
• What other problems would arise from loss of data? Could the data be replaced or reconstructed? With what amount of work?

These are not easy costs to evaluate. Nevertheless, they are needed to develop a thorough understanding of the risks. Furthermore, the vulnerabilities in computer security are often considerably higher than managers expect. Realistic estimates of potential harm can raise concern and suggest places in which attention to security is especially needed.

Step 5: Survey and Select New Controls

By this point in our risk analysis, we understand the system's vulnerabilities and the likelihood of exploitation. We turn next to an analysis of the controls to see which ones address the risks we have identified. We want to match each vulnerability with at least one appropriate security technique, as shown in Figure 8-4. Once we do that, we can use our expected loss estimates to help us decide which controls, alone or in concert, are the most cost effective for a given situation. Notice that vulnerabilities E and F are countered by primary techniques 2 and 4, respectively. The secondary control techniques 2 and 3 for vulnerability F are good defense in depth. The fact that there is no secondary control for vulnerability E is a minor concern. But vulnerability T is a serious caution, because it has no control whatsoever.

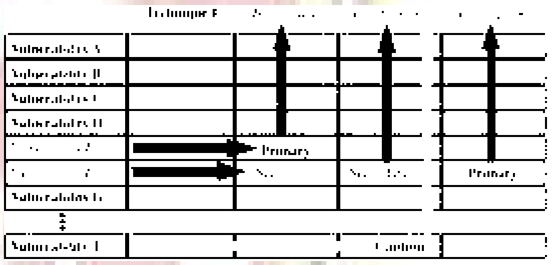


Figure 8-4. Mapping Control Techniques to Vulnerabilities. (Adapted from [ANT02], copyright © RAND 2002, reprinted by permission.)

For example, consider the risk of losing data. This loss could be addressed by several of the controls we have discussed in previous chapters: periodic backups, redundant data storage, access controls to prevent unauthorized deletion, physical security to keep someone from stealing a disk, or program development standards to limit the effect of programs on the data.

We must determine the effectiveness of each control in a given situation; for instance, using physical security in a building already equipped with guards and limited access may be more effective than sophisticated software-based controls.

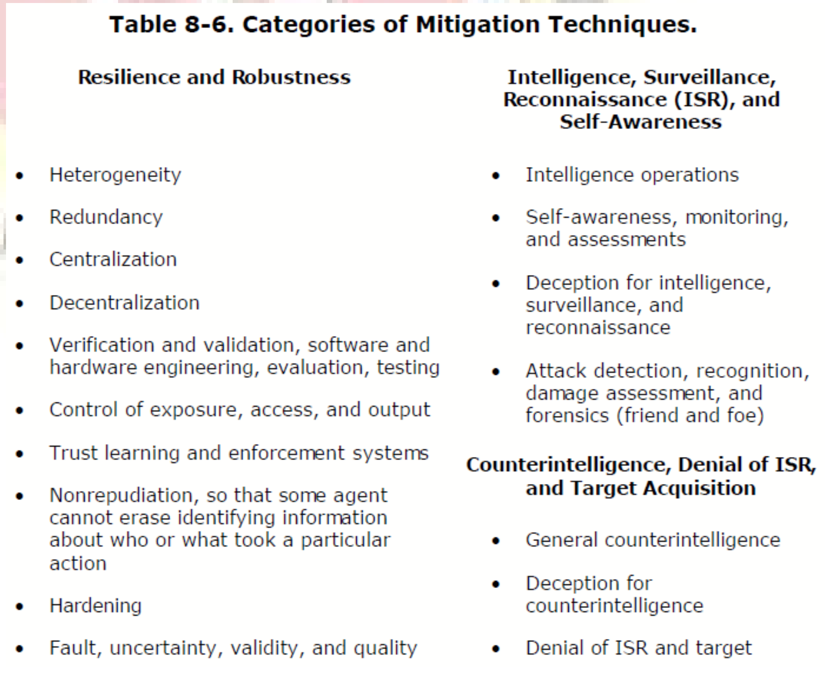


Table 8-6. Categories of Mitigation Techniques.

Resilience and Robustness

- Heterogeneity
- Redundancy
- Centralization
- Decentralization
- Verification and validation, software and hardware engineering, evaluation, testing
- Control of exposure, access, and output
- Trust learning and enforcement systems
- Nonrepudiation, so that some agent cannot erase identifying information about who or what took a particular action
- Hardening
- Fault, uncertainty, validity, and quality

Intelligence, Surveillance, Reconnaissance (ISR), and Self-Awareness

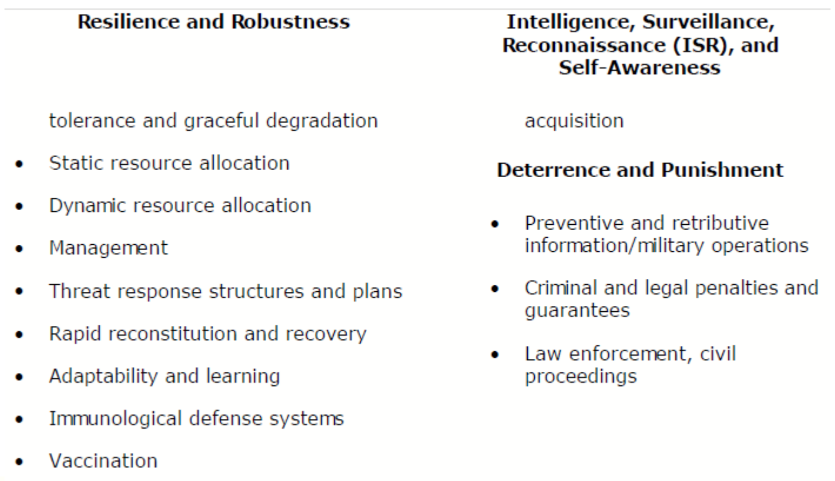
- Intelligence operations
- Self-awareness, monitoring, and assessments
- Deception for intelligence, surveillance, and reconnaissance
- Attack detection, recognition, damage assessment, and forensics (friend and foe)

Counterintelligence, Denial of ISR, and Target Acquisition

- General counterintelligence
- Deception for counterintelligence
- Denial of ISR and target

What Criteria Are Used for Selecting Controls?

We can also think of controls at a different level. Table 8-6 lists a selection of strategies presented in the VAM methodology; we can use the list to mitigate the effects of a vulnerability. This method reflects a systems approach and also the military defense environment for which VAM was developed.



Resilience and Robustness	Intelligence, Surveillance, Reconnaissance (ISR), and Self-Awareness
tolerance and graceful degradation	acquisition
• Static resource allocation	**Deterrence and Punishment**
• Dynamic resource allocation	• Preventive and retributive information/military operations
• Management	• Criminal and legal penalties and guarantees
• Threat response structures and plans	• Law enforcement, civil proceedings
• Rapid reconstitution and recovery	
• Adaptability and learning	
• Immunological defense systems	
• Vaccination	

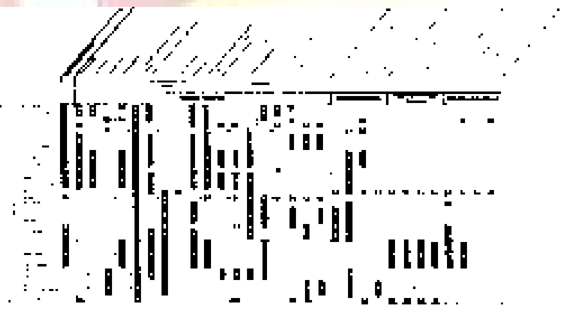
Table 8-6. Categories of Mitigation Techniques.

Table 8-6. Categories of Mitigation Techniques.

From [ANT02], copyright © RAND 2002, reprinted by permission. VAM characterizes controls in terms of four high-level aspects: resilience and robustness; intelligence, surveillance, reconnaissance (ISR), and self-awareness; counterintelligence, denial of ISR, and target acquisition; and deterrence and punishment. Notice that many of these controls are technical but embrace the entire system architecture. For example, heterogeneity is a control that can be implemented only when the system is designed so that it is composed of dissimilar pieces, such as operating systems of different brands. Similarly, redundancy and decentralization are architectural elements, too. Some people think of controls as specific pieces of hardware and software, such as firewalls and virus checkers. But in fact, this broader list takes a software engineering approach to security: Make the system sturdy from the beginning, rather than trying only to patch holes with security-specific, self-contained subsystems.

The VAM methodology takes this table one step further, using it to compare vulnerabilities to possible controls. The matrix shown in Figure 8-5 lists attributes leading to vulnerabilities (as seen in Table 8-4) along the left side, and the controls of Table 8-6 along the top. Thus, each cell of the matrix corresponds to whether a particular control addresses a given vulnerability.

Figure 8-5. Matrix of Vulnerabilities and Controls. (From [ANT02], copyright © RAND 2002, reprinted by permission.)



How Do Controls Affect What They Control? Controls have positive and negative effects: Encryption, for example, protects confidentiality, but it also takes time and introduces key management issues. Thus, when selecting controls, you have to consider the full impact.

The creators of VAM recognized that sometimes attributes enhance security and other times detract from it. For example, heterogeneity may be useful as a control in preventing the proliferation of the same kind of logic error throughout a system. But heterogeneity can also make the system's design harder to understand and, therefore, harder to maintain; the result can be a fragile design that is easy for an attacker to cause to fail. For this reason, VAM has included a rating scheme to reflect the relationship depicted by each cell of the matrix. A cell relating a vulnerability to a security technique contains a number from 2 to 2, according to this scheme:

 2 means that the control mitigates the vulnerability significantly and should be a prime candidate for addressing it.

 1 means that the control mitigates the vulnerability somewhat, but not as well as one labeled 2, so it should be a secondary candidate for addressing it.
 0 means that the vulnerability may have beneficial side effects that enhance some aspect of security. (Example: homogeneity can facilitate both static and dynamic resource allocation. It can also facilitate rapid recovery and reconstitution.)
 -1 means that the control worsens the vulnerability somewhat or incurs new vulnerabilities.
 -2 means that the control worsens the vulnerability significantly or incurs new vulnerabilities.

The VAM rating scheme is depicted in Figure 8-6; the full explanation of each row name, column name and rating can be found in [ANT02]. The matrix is used to support decisions about controls in the following way. We begin with the rows of the matrix, each of which corresponds to a vulnerability. We follow the row across to look for instances in which a cell is labeled with a 2 (or a 1, if there are no 2s). Then we follow the column up to its heading, to see which security techniques (the column labels) are strong controls for this vulnerability. For example, the matrix indicates that heterogeneity, redundancy, and decentralization are good controls for design sensitivity or fragility. Next, we notice that both heterogeneity and decentralization are also labeled with a -1 in that cell, indicating that by using them, we may enable other vulnerabilities. For instance, heterogeneity can enable several systems to complement each other but can make the overall system harder to maintain. Similarly, decentralization makes it more difficult for an attacker to exploit fragilities, but at the same time it can make the system more fragile due to a need for coordination. In this way, we can look at the implications of using each control to address known vulnerabilities.

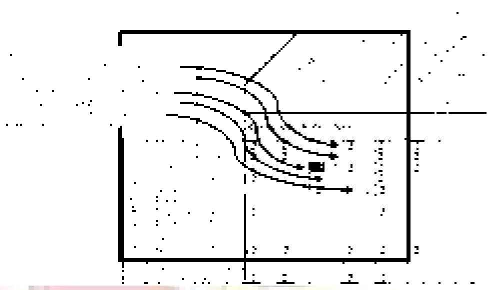


Figure 8-6. Valuation of Security Techniques. (From [ANT02], copyright © RAND 2002, reprinted by permission.)

Which Controls Are Best?

By now, we have noted a large number of primary and secondary controls to use against our identified vulnerabilities. We need a way to determine the most appropriate controls for a given situation. VAM offers us a refinement process based on three roles: operational, design, and policy. That is, if we are interested in security from the perspective of someone who will be using or managing the system, we take the operational perspective. If instead we view security from an implementation point of view, we take the developer's role. And if we view the system in the larger context of how it provides information processing to relevant organizations, we adopt the policy point of view. VAM provides tables, such as the one shown in Figure 8-7, to identify the relevance of each control to each perspective.

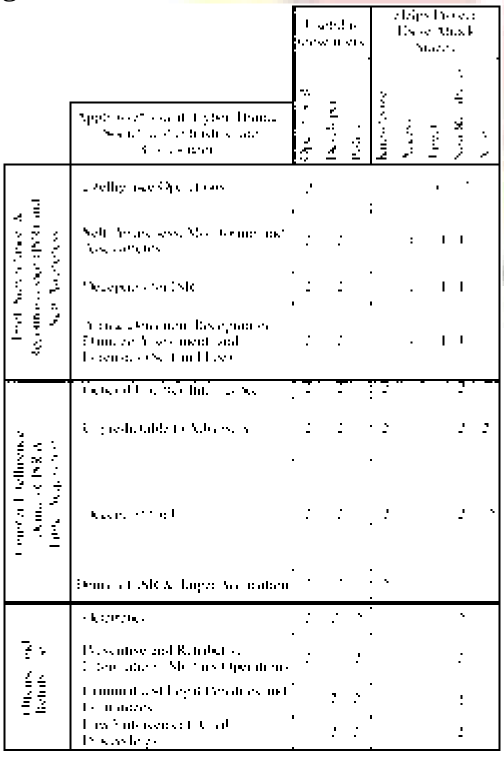

[illegible]

Figure 8-7. Relevance of Certain Security Techniques to Roles and Attack Components. (From [ANT02], copyright © RAND 2002, reprinted by permission.)

In this matrix, the rows represent security controls, and the columns serve two functions. The first three columns represent the three perspectives for evaluating the relevance of the control: operational, developer, and policy. The second five columns note at what stage of an attack the control is most useful: allowing an attacker to have knowledge about the system, enabling access to the system, providing a target for attack, enabling non retribution, and assessing the extent to which an attack has been successful. In this matrix, the 1s and 2s labeling the cells have a different meaning from the previous matrix. Here, a 1 indicates that the control is weakly relevant to the perspective or attack stage, and a 2 indicates that it is strongly relevant.

Finally, VAM presents a matrix to illustrate the relationships among the attack stages and the vulnerable objects in a system. For example, an attacker can gain knowledge about a system not only by obtaining source code and doing reverse engineering but also by using organizational charts and social engineering.

The VAM approach is comprehensive and effective, supported by a software tool to walk an analyst through the stages of identifying vulnerabilities, selecting controls, and refining choices. [ANT02] contains tables and charts that explain the rating system and the relationships among tables; we have presented some of those tables and charts, courtesy of Antón et al., because they offer good examples that introduce you to the details of selecting controls. Sometimes, however, you can do a much less rigorous analysis by simply listing the possible controls, assessing the strengths and weaknesses of each, and choosing the one(s) that seem to be most appropriate.

Step 6: Project Savings

By this point in our risk analysis, we have identified controls that address each vulnerability in our list. The next step is to determine whether the costs outweigh the benefits of preventing or mitigating the risks. Recall that we multiply the risk probability by the risk impact to determine the risk exposure. The risk impact is the loss that we might experience if the risk were to turn into a real problem. There are techniques to help us determine the risk exposure.

The effective cost of a given control is the actual cost of the control (such as purchase price, installation costs, and training costs) minus any expected loss from using the control (such as administrative or maintenance costs). Thus, the true cost of a control may be positive if the control is expensive to administer or introduces new risk in another area of the system. Or the cost can even be negative if the reduction in risk is greater than the cost of the control.

For example, suppose a department has determined that some users have gained unauthorized access to the computing system. It is feared that the intruders might intercept or even modify sensitive data on the system. One approach to addressing this problem is to install a more secure data access control program. Even though the cost of the access control software is high ($25,000), its cost is easily justified when compared to its value, as shown in Table 8-7. Because the entire cost of the package is charged in the first year, even greater benefits are expected for subsequent years.

Table 8-7. Justification of Access Control Software.

Item	Amount
Risks: disclosure of company confidential data, computation based on incorrect data	
Cost to reconstruct correct d ata :	$1,000,000
@ 10% likelihood per year	$100,000
Effectiveness of access control software: 60% -	60,000
Cost of access control software	+25,000
Expected annual costs due to loss and controls (100,000 60,000 + 25,000)	$65,000
Savings (100,000 65,000)	$35,000

Another company uses a common carrier to link to a network for certain computing applications. The company has identified the risks of unauthorized access to data and computing facilities through the network. These risks can be eliminated by replacement of remote network access with the requirement to access the system only from a machine operated on the company premises. The machine is not owned; a new one would have to be acquired. The economics of this example are not promising, as shown in Table 8-8.

Table 8-8. Cost/Benefit Analysis for Replacing Network Access.

Item	Amount
Risk: unauthorized access and use	
Access to unauthorized data and programs	$100,000 @
2% likelihood per year	$2,000
Unauthorized use of computing facilities	$10,000 @

40% likelihood per year	4,000
Expected annual loss (2,000 + 4,000)	6,000
Effectiveness of network control: 100% -	6,000
Control cost:	
Hardware (50,000 amortized over 5 years) +	10,000
Software (20,000 amortized over 5 years) +	4,000
Support personnel (each year) +	40,000
Annual cost	54,000
Expected annual loss (6,000 6,000 + 54,000)	$54,000
Savings (6,000 54,000) -	$48,000

To supplement this tabular analysis, we can use a graphical depiction to contrast the economics involved in choosing among several strategies. For example, suppose we are considering the use of regression testing after making an upgrade to fix a security flaw. Regression testing means applying tests to verify that all remaining functions are unaffected by the change. It can be an expensive process, especially for large systems that implement many functions. (This example is taken from Pfleeger and Atlee [PFL06a].)

To help us make our decision, we draw a diagram such as that in Figure 8-8. We want to compare the risk impact of doing regression testing with not doing it. Thus, the upper part of the diagram shows the risks in doing regression testing, and the lower part the risks of not doing regression testing. In each of the two cases, one of three things can happen: We find a critical fault, there is a critical fault but we miss finding it, or there are no critical faults to be found. For each possibility, we first calculate the probability of an unwanted outcome, P(UO).

Then, we associate a loss with that unwanted outcome, L(UO). Thus, in our example, if we do regression testing and miss a critical fault lurking in the system (a probability of 0.05), the loss could be $30 million. Multiplying the two, we find the risk exposure for that strategy to be $1.5 million. As you can see from the calculations in the figure, it is far safer to do the regression testing than to skip it.

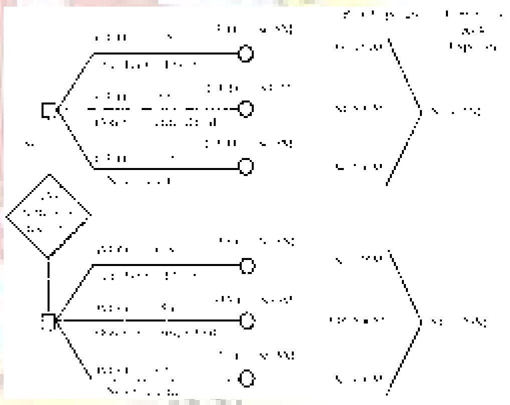


Figure 8-8. Risk Calculation for Regression Testing.

As shown in these examples, risk analysis can be used to evaluate the true costs of proposed controls. In this way, risk analysis can be used as a planning tool. The effectiveness of different controls can be compared on paper before actual investments are made. Risk analysis can thus be used repeatedly, to select an optimum set of controls.

Arguments For and Against Risk Analysis

Risk analysis is a well-known planning tool, used often by auditors, accountants, and managers. In many situations, such as obtaining approval for new drugs, new power plants, and new medical devices, a risk analysis is required by law in many countries. There are many good reasons to perform a risk analysis in preparation for creating a security plan.

- *Improve awareness.* Discussing issues of security can raise the general level of interest and concern among developers and users. Especially when the user population has little expertise in computing, the risk analysis can educate users about the role security plays in protecting functions and data that are essential to user operations and products.
- *Relate security mission to management objectives.* Security is often perceived as a financial drain for no gain. Management does not always see that security helps balance harm and control costs.
- *Identify assets, vulnerabilities, and controls.* Some organizations are unaware of their computing assets, their value to the organization, and the vulnerabilities associated with those assets. A systematic analysis produces a comprehensive list of assets, valuations, and risks.

 Improve basis for decisions. A security manager can present an argument such as "I think we need a firewall here" or "I think we should use token-based authentication instead of passwords." Risk analysis augments the manager's judgment as a basis for the decision.
 Justify expenditures for security. Some security mechanisms appear to be very expensive and without obvious benefit. A risk analysis can help identify instances where it is worth the expense to implement a major security mechanism. Justification is often derived from examining the much larger risks of *not* spending for security.

However, despite the advantages of risk analysis, there are several arguments against using it to support decision making.
 False sense of precision and confidence. The heart of risk analysis is the use of empirical data to generate estimates of risk impact, risk probability, and risk exposure.
The danger is that these numbers will give us a false sense of precision, thereby giving rise to an undeserved confidence in the numbers. However, in many cases the numbers themselves are much less important than their relative sizes. Whether an expected loss is $100,000 or $150,000 is relatively unimportant. It is much more significant that the expected loss is far above the $10,000 or $20,000 budget allocated for implementing a particular control. Moreover, anytime a risk analysis generates a large potential loss, the system deserves further scrutiny to see if the root cause of the risk can be addressed.
 Hard to perform. Enumerating assets, vulnerabilities, and controls requires creative thinking. Assessing loss frequencies and impact can be difficult and subjective. A large risk analysis will have many things to consider. Risk analysis can be restricted to certain assets or vulnerabilities, however.
 Immutability. It is typical on many software projects to view processes like risk analysis as an irritating fact of lifea step to be taken in a hurry so that the developers can get on with the more interesting jobs related to designing, building, and testing the system. For this reason, risk analyses, like contingency plans and five-year plans, have a tendency to be filed and promptly forgotten. But if an organization takes security seriously, it will view the risk analysis as a living document, updating it at least annually or in conjunction with major system upgrades.
 Lack of accuracy. Risk analysis is not always accurate, for many reasons. First, we may not be able to calculate the risk probability with any accuracy, especially when we have no past history of similar situations. Second, even if we know the likelihood, we cannot always estimate the risk impact very well. The risk management literature is replete with papers about describing the scenario, showing that presenting the same situation in two different ways to two equivalent groups of people can yield two radically different estimates of impact. And third, we may not be able to anticipate all the possible risks. For example, bridge builders did not know about the risks introduced by torque from high winds until the Tacoma Narrows Bridge twisted in the wind and collapsed. After studying the colossal failure of this bridge and discovering the cause, engineers made mandatory the inclusion of torque in their simulation parameters. Similarly, we may not know enough about software, security, or the context in which the system is to be used, so there may be gaps in our risk analysis that cause it to be inaccurate.

This lack of accuracy is often cited as a deficiency of risk analysis. But this lack is a red herring. Risk analysis is useful as a planning tool, to compare and contrast options. We may not be able to predict events accurately, but we can use risk analysis to weigh the tradeoffs between one action and another. When risk analysis is used in security planning, it highlights which security expenditures are likely to be most cost effective. This investigative basis is important for choosing among controls when money available for security is limited. And our risk analysis should improve as we build more systems, evaluate their security, and have a larger experience base from which to draw our estimates.

A risk analysis has many advantages as part of security plan or as a tool for less formal security decision making. It ranges from very subjective and imprecise to highly quantitative.

It is useful for generating and documenting thoughts about likely threats and possible countermeasures. Finally, it supports rational decision making about security controls.

Next we turn to another aspect of security planningdeveloping security policies.

Organizational security policies

A key element of any organization's security planning is an effective security policy. A security policy must answer three questions: *who* can access *which resources* in *what manner*?

A security policy is a high-level management document to inform all users of the goals of and constraints on using a system. A policy document is written in broad enough terms that it does not change frequently. The information security policy is the foundation upon which all protection efforts are built. It should be a visible representation of priorities of the entire organization, definitively stating underlying assumptions that drive security activities. The policy should articulate senior management's decisions regarding security as well as asserting management's commitment to security. To be effective, the policy must be understood by everyone as the product of a directive from an authoritative and influential person at the top of the organization.

People sometimes issue other documents, called procedures or guidelines, to define how the policy translates into specific actions and controls. In this section, we examine how to write a useful and effective security policy.

Purpose

Security policies are used for several purposes, including the following:

- recognizing sensitive information assets
- clarifying security responsibilities
- promoting awareness for existing employees
- guiding new employees

Audience

A security policy addresses several different audiences with different expectations. That is each group users, owners, and beneficiaries uses the security policy in important but different ways.

Users

Users legitimately expect a certain degree of confidentiality, integrity, and continuous availability in the computing resources provided to them. Although the degree varies with the situation, a security policy should reaffirm a commitment to this requirement for service.

Users also need to know and appreciate what is considered acceptable use of their computers, data, and programs. For users, a security policy should define acceptable use.

Owners

Each piece of computing equipment is owned by someone, and the owner may not be a system user. An owner provides the equipment to users for a purpose, such as to further education, support commerce, or enhance productivity. A security policy should also reflect the expectations and needs of owners.

Beneficiaries

A business has paying customers or clients; they are beneficiaries of the products and services offered by that business. At the same time, the general public may benefit in several ways: as a source of employment or by provision of infrastructure. For example, you may not be a client of BellSouth, but when you place a telephone call from London to Atlanta, you benefit from BellSouth's telecommunications infrastructure. In the same way, the government has customers: the citizens of its country, and "guests" who have visas enabling entry for various purposes and times. A university's customers include its students and faculty; other beneficiaries include the immediate community (which can take advantage of lectures and concerts on campus) and often the world population (enriched by the results of research and service).

To varying degrees, these beneficiaries depend, directly or indirectly, on the existence of or access to computers, their data and programs, and their computational power. For this set of beneficiaries, continuity and integrity of computing are very important. In addition, beneficiaries value confidentiality and correctness of the data involved. Thus, the interests of beneficiaries of a system must be reflected in the system's security policy.

Balance Among All Parties

A security policy must relate to the needs of users, owners, and beneficiaries. Unfortunately, the needs of these groups may conflict. A beneficiary might require immediate access to data, but owners or users might not want to bear the expense or inconvenience of providing access at all hours. Continuous availability may be a goal for users, but that goal is inconsistent with a need to perform preventive or emergency maintenance. Thus, the security policy must balance the priorities of all affected communities.

Contents

A security policy must identify its audiences: the beneficiaries, users, and owners. The policy should describe the nature of each audience and their security goals. Several other sections are required, including the purpose of the computing system, the resources needing protection, and the nature of the protection to be supplied. We discuss each one in turn.

Purpose

The policy should state the purpose of the organization's security functions, reflecting the requirements of beneficiaries, users, and owners. For example, the policy may state that the system will "protect customers' confidentiality or preserve a trust relationship," "ensure continual usability," or "maintain profitability." There are typically three to five goals, such as:

- Promote efficient business operation.
- Facilitate sharing of information throughout the organization.
- Safeguard business and personal information.
- Ensure that accurate information is available to support business processes.
- Ensure a safe and productive place to work.
- Comply with applicable laws and regulations.

The security goals should be related to the overall goal or nature of the organization. It is important that the system's purpose be stated clearly and completely because subsequent sections of the policy will relate back to these goals, making the policy a goal-driven product.

Protected Resources

A risk analysis will have identified the assets that are to be protected. These assets should be listed in the policy, in the sense that the policy lays out which items it addresses. For example, will the policy apply to all computers or only to those on the network? Will it apply to all data or only to client or management data? Will security be provided to all programs or only the ones that interact with customers? If the degree of protection varies from one service, product, or data type to another, the policy should state the differences. For example, data that uniquely identify clients may be protected more carefully than the names of cities in which clients reside.

Nature of the Protection

The asset list tells us *what* should be protected. The policy should also indicate *who* should have access to the protected items. It may also indicate *how* that access will be ensured and *how* unauthorized people will be denied access. All the mechanisms described in this book are at your disposal in deciding which controls should protect which objects. In particular, the security policy should state what degree of protection should be provided to which kinds of resources.

Characteristics of a Good Security Policy

If a security policy is written poorly, it cannot guide the developers and users in providing appropriate security mechanisms to protect important assets. Certain characteristics make a security policy a good one.

Coverage

A security policy must be comprehensive: It must either apply to or explicitly exclude all possible situations. Furthermore, a security policy may not be updated as each new situation arises, so it must be general enough to apply naturally to new cases that occur as the system is used in unusual or unexpected ways.

Durability

A security policy must grow and adapt well. In large measure, it will survive the system's growth and expansion without change. If written in a flexible way, the existing policy will be applicable to new situations. However, there are times when the policy must change (such as when government regulations mandate new security constraints), so the policy must be changeable when it needs to be.

An important key to durability is keeping the policy free from ties to specific data or protection mechanisms that almost certainly will change. For example, an initial version of a security policy might require a ten-character password for anyone needing access to data on the Sun workstation in room 110. But when that workstation is replaced or moved, the policy's guidance becomes useless. It is preferable to describe assets needing protection in terms of their function and characteristics, rather than in terms of specific implementation. For example, the policy on Sun workstations could be reworded to mandate strong authentication for access to sensitive student grades or customers' proprietary data. Better still, we can separate the elements of the policy, having one policy statement for student grades and another for customers' proprietary data. Similarly, we may want to define one policy that applies to preserving the confidentiality of relationships, and another protecting the use of the system through strong authentication.

Realism

The policy must be realistic. That is, it must be possible to implement the stated security requirements with existing technology. Moreover, the implementation must be beneficial in terms of time, cost, and convenience; the policy should not recommend a control that works but prevents the system or its users from performing their activities and functions. Sidebar 8-7 points out that sometimes the policy writers are seduced by what is fashionable in security at the time of writing. It is important to make economically worthwhile investments in security, just as for any other careful business investment.

Usefulness

An obscure or incomplete security policy will not be implemented properly, if at all. The policy must be written in language that can be read, understood, and followed by anyone who must implement it or is affected by it. For this reason, the policy should be succinct, clear, and direct.

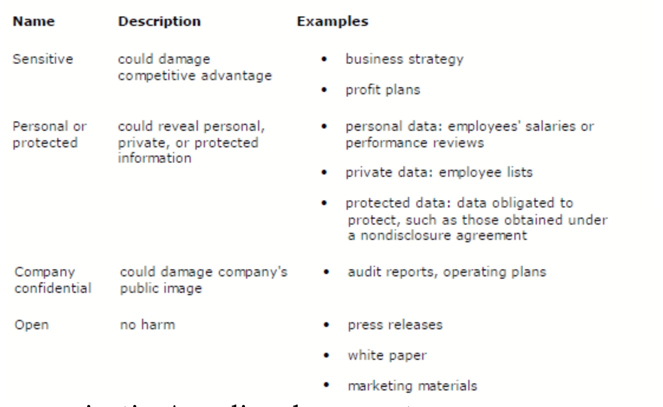
Examples

To understand the nature of security policies, we study a few examples to illustrate some of the points just presented.

Data Sensitivity Policy

Our first example is from an organization that decided to classify all its data resources into four levels, based on how severe might be the effect if a resource were damaged. These levels are listed in Table 8-9. Then, the required protection was based on the resource's level. Finally, the organization analyzed its threats, their possible severities, and countermeasures, and their effectiveness, within each of the four levels.

Table 8-9. Example: Defined Levels of Data Sensitivity.



Name	Description	Examples
Sensitive	could damage competitive advantage	• business strategy • profit plans
Personal or protected	could reveal personal, private, or protected information	• personal data: employees' salaries or performance reviews • private data: employee lists • protected data: data obligated to protect, such as those obtained under a nondisclosure agreement
Company confidential	could damage company's public image	• audit reports, operating plans
Open	no harm	• press releases • white paper • marketing materials

Although the phrases describing the degree of damage are open to interpretation, the intent of these levels is clear: All information assets are to be classified as sensitive, personal, confidential, or open, and protection requirements for these four types are detailed in the remainder of the organization's policy document.

Government Agency IT Security Policy

The U.S. Department of Energy (DOE), like many government units, has established its own security policy. The following excerpt is from the policy on protecting classified material, although the form is appropriate for many unclassified uses as well.

It is the policy of DOE that classified information and classified ADP [automatic data processing] systems shall be protected from unauthorized access (including the enforcement of need-to-know protections), alteration, disclosure, destruction, penetration, denial of service, subversion of security measures, or improper use as a result of espionage, criminal, fraudulent, negligent, abusive, or other improper actions. The DOE shall use all reasonable measures to protect ADP systems that process, store, transfer, or provide access to classified information, to include but not limited to the following: physical security, personnel security, telecommunications security, administrative security, and hardware and software security measures. This order establishes this policy and defines responsibilities for the development, implementation, and periodic evaluation of the DOE program.

The policy then continues for several more pages to list specific responsibilities for specific people.

The cited paragraph is comprehensive, covering practically every possible source (espionage, crime, fraud, etc.) of practically every possible harm (unauthorized access, alteration, destruction, etc.), and practically every possible kind of control (physical, personnel, etc.).

The generality of the header paragraph is complemented by subsequent paragraphs giving specific responsibilities:

- "Each data owner shall determine and declare the required protection level of information . . ."
- "Each security officer shall . . . perform a risk assessment to identify and document specific . . . assets, . . . threats, . . . and vulnerability . . ."
- "Each manager shall...establish procedures to ensure that systems are continuously monitored...to detect security infractions . . ." and so on.

Internet Security Policy

The Internet does not have a governing security policy per se, because it is a federation of users. Nevertheless, the Internet Society drafted a security policy for its members [PET91].

The policy contains the following interesting portions.

- Users are individually responsible for understanding and respecting the security policies of the systems (computers and networks) they are using. Users are individually accountable for their own behavior.
- Users have a responsibility to employ available security mechanisms and procedures for protecting their own data. They also have a responsibility for assisting in the protection of the systems they use.
- Computer and network service providers are responsible for maintaining the security of the systems they operate. They are further responsible for notifying users of their security policies and any changes to these policies.
- Vendors and system developers are responsible for providing systems which are sound and which embody adequate security controls.
- Users, service providers, and hardware and software vendors are responsible for cooperating to provide security.
- Technical improvements in Internet security protocols should be sought on a continuing basis. At the same time, personnel developing new protocols, hardware or software for the Internet are expected to include security considerations as part of the design and development process.

These statements clearly state to whom they apply and for what each party is responsible.

Policy Issue Example: Government E-mail

Organizations develop computer security policies along the lines just described. Generally the policies lead to the familiar assets, vulnerabilities, and controls. But sometimes you have to start with existing policies which may be formal documents or informal understandings and consider how they apply in new situations. Is this action consistent with the goals of the policy and therefore acceptable? Applying policies can be like being a judge. As security professionals, we often focus on security policy without remembering the context in which we are making policy decisions. In this section, we look at a real-life issue to see how security policy fits into the broader scope of issues the security must address.

The U.S. government has proposed using network technologies to enhance its ability to interact with American citizens. Some people think that by employing functions such as electronic mail and World Wide Web access, the government could make more information available to citizens more quickly and at the same time be more responsive to citizens' needs.

It is also hoped that costs would be reduced, a winning proposition for government and taxpayers alike. This proposal has clear security implications. Indeed, having read this far in this book, you can probably list dozens of security issues that must be addressed to make this proposal work. The technology to design, build, and support this type of function exists, and the requirements, design, and implementation can easily be done from a technological point of view. But what about the other issues involved in building such a system? Neu et al. [NEU98] point out that the technology must be viewed in the larger institutional, organizational, and administrative contexts.

Much of what the government wants to do is already done. Many federal agencies have web sites providing large amounts of information to citizens, such as regulations, reports, and forms. This type of information is equally accessible to anyone who needs it. But other information exchange is more personalized: submitting completed tax forms, filing required paperwork for licenses and benefits, and asking specific questions about an individual's records, for example. Clearly the last type suggests stringent requirements relating to confidentiality, authentication, and integrity.

Neu et al. mention several security policy issues that must be addressed before such a system could be implemented. These include the following:

 How do the commercial firms' security policies meet the government's security needs?

 To enable secure communication, the government will likely want to use public key encryption. As we noted in Chapter 2, a certificate authority associates a public key with a particular user, establishing the user's identity. But for the government communication system, we must also know who has authority to access information and services and to initiate transactions. The processes required to perform identification are likely to be different from those performing authorization. In particular, identification may require direct interaction with a user, whereas authorization may require links among large databases.

 A citizen may have more than one identity. For example, Jane Doe may be the same person as Mrs. Nathaniel Simmons, who is also the same person as the Trustee for the Estate of Mr. Robert Jones. In turn, each of these identities may have multiple authorities. How will the identification authorities interact with the authorization ones to enable these situations?

 Sometimes the authorization does not need to be tied to a specific identity. For example, a government agency may need to know only that an individual is capable of paying for a service, much as a credit card company provides a credit rating. How will the authorization be able to release the minimum amount of information possible about an individual?

 How will certificate authorities have a high degree of confidence in their identification of individuals?

 How will certificate authorities deal with the need to view certain documents, such as birth certificates and passports, in person? This condition may mean that certificate authorities may be required to have local offices around the country.

 Should there be a single certificate authority or many? A single provider can minimize the need for multiple keys and might save money by streamlining operations. But a single provider can also monitor all of a citizen's transactions, inviting abuse.

These issues are not trivial. Their solutions, not at all obvious, build on the concepts presented in this book. But they do so in a way that is not just technological. We can easily build a PKI to provide certificates to anyone we want. But how do we connect two certificates, connoting that the digital identities actually belong to the same person? In the real world you can be anonymous by purchasing something with cash; how can you be anonymous digitally?

But in addition to the security issues, there are also broader issues of management, responsibility, and law. Neu et al. note that, even when the technical issues are resolved, we have still to answer these questions:

• What happens if a certificate authority makes a mistake, either by identifying or authorizing the wrong person or by assigning keys to an impostor? What are the legal and financial implications of such an error? What if the error is made even though the certificate authority followed government guidelines?

• How will citizens create, record, and protect their keys? If smart cards are used to store keys, does that card become a national identity card?

• What legal protections are available to electronic transactions? For example, in the United States today, it is illegal to intercept someone's surface mail, but it is not illegal to intercept someone's electronic mail.

• How do we prove that official electronic communications, such as a summons or subpoena, have been read? Will a citizen be responsible for regularly checking e-mail for official documents?
• If law enforcement officials need to access encrypted electronic communications, how will they be able to perform the decryption? Will there be a method by which they can obtain the key? Does this require the citizen to participate?
• What levels of protection are required for electronic documents? For instance, should medical records have the same level of protection as tax returns or driving violations? How do these levels apply across the different states that have very different laws? How does the protection address international law?
• How will every citizen be provided with an electronic mail address? What happens when an e-mail address changes? What security standards will apply to e-mail boxes and service providers?
• How will the government ensure equal access to electronic government services? Should the government provide help and training to first-time users?
• How will electronic communication be phased in to the current mix of paper and telephone communication?

These questions are not challenges to the technical side of computer security. But they are very much a part of the administrative side. It is not sufficient to know all the latest encryption algorithms; you also have to know how the use of computer security mechanisms fits into the broader context of how they are used and what they support. This example is included to introduce you to the procedural, administrative, policy, and privacy issues that a computer security administrator must consider. These questions highlight the degree to which security planning and policy must fit in with the larger policy issues that we, as individuals, organizations, and societies, must address. For this reason, in the next chapter we turn to the legal and ethical considerations of computer security.

But before we move to those concerns, we must cover one more topic involved in administering security: physical security. Protecting computing systems from physical harm is no less important than protecting data from modification in transit through a network. In the next section we briefly survey physical security vulnerabilities and controls.

Physical security

Much of this book has focused on technical issues in security and their technical solutions firewalls, encryption techniques, and more. But many threats to security involve human or natural disasters, events that should also be addressed in the security plan. For this reason, in this section we consider how to cope with the nontechnical things that can go wrong.

There are two pieces to the process of dealing with nontechnical problems: preventing things that can be prevented and recovering from the things that cannot be prevented. Physical security is the term used to describe protection needed outside the computer system.

Typical physical security controls include guards, locks, and fences to deter direct attacks. In addition, there are other kinds of protection against less direct disasters, such as floods and power outages; these, too, are part of physical security. As we will see, many physical security measures can be provided simply by good common sense, a characteristic that Mark Twain noted "is a most uncommon virtue."

Natural Disasters

Computers are subject to the same natural disasters that can occur to homes, stores, and automobiles. They can be flooded, burned, melted, hit by falling objects, and destroyed by earthquakes, storms, and tornadoes. Additionally, computers are sensitive to their operating environment, so excessive heat or inadequate power is also a threat. It is impossible to prevent natural disasters, but through careful planning it is possible to reduce the damage they inflict. Some measures can be taken to reduce their impact. Because many of these perils cannot be prevented or predicted, controls focus on limiting possible damage and recovering quickly from a disaster. Issues to be considered include the need for offsite backups, the cost of replacing equipment, the speed with which equipment can be replaced, the need for available computing power, and the cost or difficulty of replacing data and programs.

Flood

Water from a natural flood comes from ground level, rising gradually, and bringing with it mud and debris. Often, there is time for an orderly shutdown of the computing system; at worst, the organization loses some of the processing in progress. At other times, such as when a dam breaks, a water pipe bursts, or the roof collapses in a storm, a sudden flood can overwhelm the system and its users before anything can be saved. Water can come from above, below, or the side. The machinery may be destroyed or damaged by mud and water, but most computing systems are insured and replaceable by the manufacturer. Managers of unique or irreplaceable equipment who recognize the added risk sometimes purchase or lease duplicate redundant hardware systems to ensure against disruption of service.

Even when the hardware can be replaced, we must be concerned about the stored data and programs. The system administrator may choose to label storage media in a way that makes it easy to identify the most important data. For example, green, yellow, and red labels may show which disks are the most sensitive, so that all red disks are moved from the data center during a storm. Similarly, large plastic bags and waterproof tape can be kept near important equipment and media; they are used to protect the hardware and storage media in case of a burst pipe or other sudden flood.

The real issue is protecting data and preserving the ability to compute. The only way to ensure the safety of data is to store backup copies in one or more safe locations.

Fire

Fire is more serious than water; often there is not as much time to react, and human lives are more likely to be in immediate danger. To ensure that system personnel can react quickly, every user and manager should have a plan for shutting down the system in an orderly manner. Such a process takes only a few minutes but can make recovery much easier. This plan should include individual responsibilities for all people: some to halt the system, others to protect crucial media, others to close doors on media cabinets. Provision should be made for secondary responsibilities, so that onsite staff can perform duties for those who are not in the office.

Water is traditionally used to put out fires, but it is not a good idea for use in computer rooms. In fact, more destruction can be the result of sprinklers than of the fires themselves. A fire sensor usually activates many sprinklers, dousing an entire room, even when the fire is merely some ignited paper in a wastebasket and of no threat to the computing system. Many computing centers use carbon dioxide extinguishers or an automatic system that sprays a gas such as Halon to smother a fire but leave no residue. Unfortunately, these gas systems work by displacing the oxygen in the room, choking the fire but leaving humans unable to breathe.

Consequently, when these protection devices are activated, humans must leave, disabling efforts to protect media. The best defense for situations like these is careful placement of the computing facility. A windowless location with fire-resistant access doors and nonflammable full-height walls can prevent some fires from spreading from adjacent areas to the computing room. With a fire and smoke-resistant facility, personnel merely shut down the system and leave, perhaps carrying out the most important media.

Fire prevention is quite effective, especially because most computer goods are not especially flammable. Advance planning, reinforced with simulation drills, can help make good use of the small amount of time available before evacuation is necessary.

Other Natural Disasters

Computers are subject to storms, earthquakes, volcanoes, and similar events. Although not natural disasters, building collapse, explosion, and damage from falling objects can be considered in the same category. These kinds of catastrophes are difficult to predict or estimate.

But we know these catastrophes will occur. Security managers cope with them in several ways:

- developing contingency plans so that people know how to react in emergencies and business can continue
- insuring physical assets computers, buildings, devices, supplies against harm
- preserving sensitive data by maintaining copies in physically separated locations

Power Loss

Computers need their foodelectricityand they require a constant, pure supply of it. With a direct power loss, all computation ceases immediately. Because of possible damage to media by sudden loss of power, many disk drives monitor the power level and quickly retract the recording head if power fails. For certain time-critical applications, loss of service from the system is intolerable; in these cases, alternative complete power supplies must be instantly available.

Uninterruptible Power Supply

One protection against power loss is an uninterruptible power supply. This device stores energy during normal operation so that it can return the backup energy if power fails. One form of uninterruptible power supply uses batteries that are continually charged when the power is on but which then provide power when electricity fails. However, size, heat, flammability, and low output can be problems with batteries.

Some uninterruptible power supplies use massive wheels that are kept in continuous motion when electricity is available. When the power fails, the inertia in the wheels operates generators to produce more power. Size and limited duration of energy output are problems with this variety of power supply. Both forms of power supplies are intended to provide power for a limited time, just long enough to allow the current state of the computation to be saved so that no computation is lost.

Surge Suppressor

Another problem with power is its "cleanness." Although most people are unaware of it, a variation of 10 percent from the stated voltage of a line is considered acceptable, and some power lines vary even more. A particular power line may always be 10 percent high or low.

In many places, lights dim momentarily when a large appliance, such as an air conditioner, begins operation. When a large motor starts, it draws an exceptionally large amount of current, which reduces the flow to other devices on the line. When a motor stops, the sudden termination of draw can send a temporary surge along the line. Similarly, lightning strikes may send a momentary large pulse. Thus, instead of being constant, the power delivered along any electric line shows many brief fluctuations, called drops, spikes, and surges. A drop is a momentary reduction in voltage, and a spike or surge is a rise. For computing equipment, a drop is less serious than a surge. Most electrical equipment is tolerant of rather large fluctuations of current.

These variations can be destructive to sensitive electronic equipment, however. Simple devices called "surge suppressors" filter spikes from an electric line, blocking fluctuations that would affect computers. These devices cost from $20 to $100; they should be installed on every computer, printer, or other connected component. More sensitive models are typically used on larger systems.

As mentioned previously, a lightning strike can send a surge through a power line. To increase protection, personal computer users usually unplug their machines when they are not in use, as well as during electrical storms. Another possible source of destruction is lightning striking a telephone line. Because the power surge can travel along the phone line and into the computer or peripherals, the phone line should be disconnected from the modem during storms. These simple measures may save much work as well as valuable equipment.

Human Vandals

Because computers and their media are sensitive to a variety of disruptions, a vandal can destroy hardware, software, and data. Human attackers may be disgruntled employees, bored operators, saboteurs, people seeking excitement, or unwitting bumblers. If physical access is easy to obtain, crude attacks using axes or bricks can be very effective. One man recently shot a computer that he claimed had been in the shop for repairs many times without success.

Physical attacks by unskilled vandals are often easy to prevent; a guard can stop someone approaching a computer installation with a threatening or dangerous object. When physical access is difficult, more subtle attacks can be tried, resulting in quite serious damage. People with only some sophisticated knowledge of a system can short-circuit a computer with a car key or disable a disk drive with a paper clip. These items are not likely to attract attention until the attack is completed.

Unauthorized Access and Use

Films and newspaper reports exaggerate the ease of gaining access to a computing system. Still, as distributed computing systems become more prevalent, protecting the system from outside access becomes more difficult and more important. Interception is a form of unauthorized access; the attacker intercepts data and either breaks confidentiality or prevents the data from being read or used by others. In this context, interception is a passive attack. But we must also be concerned about active interception, in the sense that the attacker can change or insert data before allowing it to continue to its destination.

Theft

It is hard to steal a large mainframe computer. Not only is carrying it away difficult, but finding a willing buyer and arranging installation and maintenance also require special assistance.

However, printed reports, tapes, or disks can be carried easily. If done well, the loss may not be detected for some time. Personal computers, laptops, and personal digital assistants (PDAs, such as Palms or Blackberries) are designed to be small and portable. Diskettes and tape backup cartridges are easily carried in a shirt pocket or briefcase. Computers and media that are easy to carry are also easy to conceal.

We can take one of three approaches to preventing theft: preventing access, preventing portability, or detecting exit.

Preventing Access

The surest way to prevent theft is to keep the thief away from the equipment. However, thieves can be either insiders or outsiders. Therefore, access control devices are needed both to prevent access by unauthorized individuals and to record access by those authorized. A record of accesses can help identify who committed a theft.

The oldest access control is a guard, not in the database management system sense we discussed in Chapter 6 but rather in the sense of a human being stationed at the door to control access to a room or to equipment. Guards offer traditional protection; their role is well understood, and the protection they offer is adequate in many situations. However, guards must be on duty continuously in order to be effective; providing breaks implies at least four guards for a 24-hour operation, with extras for vacation and illness. A guard must personally recognize someone or recognize an access token, such as a badge. People can lose or forget badges; terminated employees and forged badges are also problems. Unless the guard makes a record of everyone who has entered a facility, there is no way to know who (employee or visitor) has had access in case a problem is discovered.

The second oldest access control is a lock. This device is even easier, cheaper, and simpler to manage than a guard. However, it too provides no record of who has had access, and difficulties arise when keys are lost or duplicated. At computer facilities, it is inconvenient to fumble for a key when your hands are filled with tapes or disks, which might be ruined if dropped. There is also the possibility of piggybacking: a person walks through the door that someone else has just unlocked. Still, guards and locks provide simple, effective security for access to facilities such as computer rooms.

More exotic access control devices employ cards with radio transmitters, magnetic stripe cards (similar to 24-hour bank cards), and smart cards with chips containing electronic circuitry that makes them difficult to duplicate. Because each of these devices interfaces with a computer, it is easy for the computer to capture identity information, generating a list of who entered and left the facility, when, and by which routes. Some of these devices operate by proximity, so that a person can carry the device in a pocket or clipped to a collar; the person obtains easy access even when hands are full. Because these devices are computer controlled, it is easy to invalidate an access authority when someone quits or reports the access token lost or stolen.

The nature of the application or service determines how strict the access control needs to be. Working in concert with computer-based authentication techniques, the access controls can be part of defense in depth using multiple mechanisms to provide security.

Preventing Portability

Portability is a mixed blessing. We can now carry around in our pockets devices that provide as much computing power as mainframes did twenty years ago. Portability is in fact a necessity in devices such as PDAs and mobile phones. And we do not want to permanently affix our personal computers to our desks, in case they need to be removed for repair or replacement. Thus, we need to find ways to enable portability without promoting theft.

One antitheft device is a pad connected to cable, similar to those used to secure bicycles. The pad is glued to the desktop with extremely strong adhesive. The cables loop around the equipment and are locked in place. Releasing the lock permits the equipment to be moved. An alternative is to couple the base of the equipment to a secure pad, in much the same way that televisions are locked in place in hotel rooms. Yet a third possibility is a large, lockable cabinet in which the personal computer and its peripherals are kept when they are not in use.

Some people argue that cables, pads, and cabinets are unsightly and, worse, they make the equipment inconvenient to use. Another alternative is to use movement-activated alarm devices when the equipment is not in use. Small alarms are available that can be locked to a laptop or PDA. When movement is detected, a loud, annoying whine or whistle warns that the equipment has been disturbed.

Such an alarm is especially useful when laptops must be left in meeting or presentation rooms overnight or during a break. Used in concert with guards, the alarms can offer reasonable protection at reasonable cost.

Detecting Theft

For some devices, protection is more important than detection. We want to keep someone from stealing certain systems or information at all costs. But for other devices, it may be enough to detect that an attempt has been made to access or steal hardware or software.

For example, chaining down a disk makes it unusable. Instead, we try to detect when someone tries to leave a protected area with the disk or other protected object. In these cases, the protection mechanism should be small and unobtrusive.

One such mechanism is similar to the protection used by many libraries, bookstores, or department stores. Each sensitive object is marked with a special label. Although the label looks like a normal pressure-sensitive one, its presence can be detected by a machine at the exit door if the label has not been disabled by an authorized party, such as a librarian or salesclerk. Similar security code tags are available for vehicles, people, machinery, and documents.

Some tags are enabled by radio transmitters. When the detector sounds an alarm, someone must apprehend the person trying to leave with the marked object.

Interception of Sensitive Information

When disposing of a draft copy of a confidential report containing its sales strategies for the next five years, a company wants to be especially sure that the report is not reconstructable by one of its competitors. When the report exists only as hard copy, destroying the report is straightforward, usually accomplished by shredding or burning. But when the report exists digitally, destruction is more problematic. There may be many copies of the report in digital and paper form and in many locations (including on the computer and on storage media).

There may also be copies in backups and archived in e-mail files. In this section, we look at several ways to dispose of sensitive information.

Shredding

Shredders have existed for a long time, as devices used by banks, government agencies, and others organizations to dispose of large amounts of confidential data. Although most of the shredded data is on paper, shredders can also be used for destroying printer ribbons and some types of disks and tapes. Shredders work by converting their input to thin strips or pulp, with enough volume to make it infeasible for most people to try to reconstruct the original from its many pieces. When data are extremely sensitive, some organizations burn the shredded output for added protection.

Overwriting Magnetic Data

Magnetic media present a special problem for those trying to protect the contents. When data are stored on magnetic disks, the ERASE or DELETE functions often simply change a directory pointer to free up space on the disk. As a result, the sensitive data are still recorded on the medium, and they can be recovered by analysis of the directory. A more secure way to destroy data on magnetic devices is to overwrite the data several times, using a different pattern each time. This process removes enough magnetic residue to prevent most people from reconstructing the original file. However, "cleaning" a disk in this fashion takes time.

Moreover, a person using highly specialized equipment might be able to identify each separate message, much like the process of peeling off layers of wallpaper to reveal the wall beneath.

Degaussing

Degaussers destroy magnetic fields. Passing a disk or other magnetic medium through a degausser generates a magnetic flux so forceful that all magnetic charges are instantly realigned, thereby fusing all the separate layers. A degausser is a fast way to cleanse a magnetic medium, although there is still question as to whether it is adequate for use in the most sensitive of applications. (Media that have had the same pattern for a long time, such as a disk saved for archival purposes, may retain traces of the original pattern even after it has been overwritten many times or degaussed.) For most users, a degausser is a fast way to neutralize a disk or tape, permitting it to be reused by others.

Protecting Against Emanation: Tempest

Computer screens emit signals that can be detected from a distance. In fact, any components, including printers, disk drives, and processors, can emit information. Tempest is a U.S. government program under which computer equipment is certified as emission-free (that is, no detectable emissions). There are two approaches for preparing a device for Tempest certification: enclosing the device and modifying the emanations.

The obvious solution to preventing emanations is to trap the signals before they can be picked up. Enclosing a device in a conductive case, such as copper, diffuses all the waves by conducting them throughout the case. Copper is a good conductor, and the waves travel much better through copper than through the air outside the case, so the emissions are rendered harmless.

This solution works very well with cable, which is then enclosed in a solid, emanation-proof shield. Typically, the shielded cable is left exposed so that it is easy to inspect visually for any signs of tapping or other tampering. The shielding must be complete. That is, it does little good to shield a length of cable but not also shield the junction box at which that cable is connected to a component. The line to the component and the component itself must be shielded, too.

The shield must enclose the device completely. If top, bottom, and three sides are shielded, emanations are prevented only in those directions. However, a solid copper shield is useless in front of a computer screen. Covering the screen with a fine copper mesh in an intricate pattern carries the emanation safely away. This approach solves the emanation problem while still maintaining the screen's usability.

Entire computer rooms or even whole buildings can be shielded in copper so that large computers inside do not leak sensitive emanations. Although it seems appealing to shield the room or building instead of each component, the scheme has significant drawbacks. A shielded room is inconvenient because it is impossible to expand the room easily as needs change. The shielding must be done carefully, because any puncture is a possible point of emanation.

Furthermore, continuous metal pathways, such as water pipes or heating ducts, act as antennas to convey the emanations away from their source.

Emanations can also be designed in such a way that they cannot be retrieved. This process is similar to generating noise in an attempt to jam or block a radio signal. With this approach, the emanations of a piece of equipment must be modified by addition of spurious signals.

Additional processors are added to Tempest equipment specifically to generate signals that fool an interceptor. The exact Tempest modification methods are classified. As might be expected, Tempest-enclosed components are larger and heavier than their unprotected counterparts. Tempest testing is a rigorous program of the U.S. Department of Defense. Once a product has been approved, even a minor design modification, such as changing from one manufacturer's power supply to an equivalent one from another manufacturer, invalidates the Tempest approval. Therefore, these components are costly, ranging in price from 10 percent to 300 percent more than similar non-Tempest products. They are most appropriate in situations in which the data to be confined are of great value, such as top-level government information. Other groups with less dramatic needs can use other less rigorous shielding.

Contingency Planning

The key to successful recovery is adequate preparation. Seldom does a crisis destroy irreplaceable equipment; most computing systems personal computers to mainframes are standard, off-the-shelf systems that can be easily replaced. Data and locally developed programs are more vulnerable because they cannot be quickly substituted from another source. Let us look more closely at what to do after a crisis occurs.

Backup

In many computing systems, some data items change frequently, whereas others seldom change. For example, a database of bank account balances changes daily, but a file of depositors' names and addresses changes much less often. Also the number of changes in a given period of time is different for these two files. These variations in number and extent of change relate to the amount of data necessary to reconstruct these files in the event of a loss.

A backup is a copy of all or a part of a file to assist in reestablishing a lost file. In professional computing systems, periodic backups are usually performed automatically, often at night when system usage is low. Everything on the system is copied, including system files, user files, scratch files, and directories, so that the system can be regenerated after a crisis. This type of backup is called a complete backup. Complete backups are done at regular intervals, usually weekly or daily, depending on the criticality of the information or service provided by the system.

Major installations may perform revolving backups, in which the last several backups are kept. Each time a backup is done, the oldest backup is replaced with the newest one. There are two reasons to perform revolving backups: to avoid problems with corrupted media (so that all is not lost if one of the disks is bad) and to allow users or developers to retrieve old versions of a file. Another form of backup is a selective backup, in which only files that have been changed (or created) since the last backup are saved. In this case, fewer files must be saved, so the backup can be done more quickly. A selective backup combined with an earlier complete backup gives the effect of a complete backup in the time needed for only a selective backup. The selective backup is subject to the configuration management techniques described in Chapter 3.

For each type of backup, we need the means to move from the backup forward to the point of failure. That is, we need a way to restore the system in the event of failure. In critical transaction systems, we address this need by keeping a complete record of changes since the last backup. Sometimes, the system state is captured by a combination of computer- and paper-based recording media. For example, if a system handles bank teller operations, the individual tellers duplicate their processing on paper records the deposit and withdrawal slips that accompany your bank transactions; if the system fails, the staff restores the latest backup version and reapplies all changes from the collected paper copies. Or the banking system creates a paper journal, which is a log of transactions printed just as each transaction completes.

Personal computer users often do not appreciate the need for regular backups. Even minor crises, such as a failed piece of hardware, can seriously affect personal computer users. With a backup, users can simply change to a similar machine and continue work.

Offsite Backup

A backup copy is useless if it is destroyed in the crisis, too. Many major computing installations rent warehouse space some distance from the computing system, far enough away that a crisis is not likely to affect the offsite location at the same time. As a backup is completed, it is transported to the backup site. Keeping a backup version separate from the actual system reduces the risk of its loss. Similarly, the paper trail is also stored somewhere other than at the main computing facility.

Personal computer users concerned with integrity can take home a copy of important disks as protection or send a copy to a friend in another city. If both secrecy and integrity are important, a bank vault, or even a secure storage place in another part of the same building can be used. The worst place to store a backup copy is where it usually is stored: right next to the machine.

Networked Storage

With today's extensive use of networking, using the network to implement backups is a good idea. Storage providers sell space in which you can store data; think of these services as big network-attached disk drives. You rent space just as you would consume electricity: You pay for what you use. The storage provider needs to provide only enough total space to cover everyone's needs, and it is easy to monitor usage patterns and increase capacity as combined needs rise.

Networked storage is perfect for backups of critical data because you can choose a storage provider whose physical storage is not close to your processing. In this way, physical harm to your system will not affect your backup. You do not need to manage tapes or other media and physically transport them offsite.

Cold Site

Depending on the nature of the computation, it may be important to be able to recover from a crisis and resume computation quickly. A bank, for example, might be able to tolerate a four-hour loss of computing facilities during a fire, but it could not tolerate a ten-month period to rebuild a destroyed facility, acquire new equipment, and resume operation.

Most computer manufacturers have several spare machines of most models that can be delivered to any location within 24 hours in the event of a real crisis. Sometimes the machine will come straight from assembly; other times the system will have been in use at a local office. Machinery is seldom the hard part of the problem. Rather, the hard part is deciding where to put the equipment in order to begin a temporary operation.

A cold site or shell is a facility with power and cooling available, in which a computing system can be installed to begin immediate operation. Some companies maintain their own cold sites, and other cold sites can be leased from disaster recovery companies. These sites usually come with cabling, fire prevention equipment, separate office space, telephone access, and other features. Typically, a computing center can have equipment installed and resume operation from a cold site within a week of a disaster.

Hot Site

If the application is more critical or if the equipment needs are more specialized, a hot site may be more appropriate. A hot site is a computer facility with an installed and ready-to-run computing system. The system has peripherals, telecommunications lines, power supply, and even personnel ready to operate on short notice. Some companies maintain their own; other companies subscribe to a service that has available one or more locations with installed and running computers. To activate a hot site, it is necessary only to load software and data from offsite backup copies.

Numerous services offer hot sites equipped with every popular brand and model of system. They provide diagnostic and system technicians, connected communications lines, and an operations staff. The hot site staff also assists with relocation by arranging transportation and housing, obtaining needed blank forms, and acquiring office space.

Because these hot sites serve as backups for many customers, most of whom will not need the service, the annual cost to any one customer is fairly low. The cost structure is like insurance: The likelihood of an auto accident is low, so the premium is reasonable, even for a policy that covers the complete replacement cost of an expensive car. Notice, however, that the first step in being able to use a service of this type is a complete and timely backup.

Physical Security Recap

By no means have we covered all of physical security in this brief introduction. Professionals become experts at individual aspects, such as fire control or power provision. However, this section should have made you aware of the major issues in physical security. We have to protect the facility against many sorts of disasters, from weather to chemical spills and vehicle crashes to explosions. It is impossible to predict what will occur or when. The physical security manager has to consider all assets and a wide range of harm.

Malicious humans seeking physical access are a different category of threat agent. With them, you can consider motive or objective: is it theft of equipment, disruption of processing, interception of data, or access to service? Fences, guards, solid walls, and locks will deter or prevent most human attacks. But you always need to ask where weaknesses remain; a solid wall has a weakness in every door and window.

The primary physical controls are strength and duplication. Strength means overlapping controls implementing a defense-in-depth approach so that if one control fails, the next one will protect. People who built ancient castles practiced this philosophy with moats, walls, drawbridges, and arrow slits. Duplication means eliminating single points of failure. Redundant copies of data protect against harm to one copy from any cause. Spare hardware components protect against failures.

Privacy

Computers did not invent or even cause privacy issues; we had those long before computers and probably even before written language. But computers' high-speed processing and data storage and transmission capabilities made possible data collection and correlation that affect privacy. Because privacy is part of confidentiality, it is an aspect of computer security.

Privacy is a human right, although people can legitimately disagree over when or to what extent privacy is deserved; this disagreement may have cultural, historical, or personal roots.

Laws and ethics, which we study in Chapter 11, can set the baseline for and enforce expectations of privacy. But inherently, the right to privacy depends on the situation and the affected parties. And just as confidentiality, integrity, and availability can conflict, so too can privacy and other aspects of security. We won't take a position on when a right to privacy should be enforceable because that is outside the scope of this book. You might characterize the presentation of this chapter as "assuming a particular right to privacy exists, what are its implications in computing and information technology?" We as citizens help decide the contours of privacy rights; we as computer security experts implement those decisions in computer systems.

Privacy is also a broad topic, affected by computing but not just a security topic. We don't want to try to survey all possible privacy issues in this chapter, just those inextricably linked to computer security.

In this chapter we look at the meaning of information privacy. We examine identification and authentication, two familiar aspects of computing that have significant privacy implications.

We study how privacy relates to the Internet, specifically in e-mail and web access. Finally, we investigate some emerging computer-based technologies for which privacy is important.

10.1. Privacy Concepts

In this section we examine privacy, first from its general or common usage and then as it applies in technological situations.

Aspects of Information Privacy

Information privacy has three aspects: sensitive data, affected parties, and controlled disclosure. In fact, these aspects are similar to the three elements of access control from Chapter 5: subject, object, and access rights.
We examine these three in turn.
Controlled Disclosure

What is privacy? A good working definition is that privacy is the right to control who knows certain aspects about you, your communications, and your activities. In other words, you voluntarily choose who can know things about you and what those things are. People ask you for your telephone number: your auto mechanic, a clerk in a store, your tax authority, a new business contact, or a cute person in a bar. You consider why the person wants the number and decide whether to give it out. But the key point is *you* decide. So privacy is something over which you have considerable influence.

You do not have complete control, however. Once you give your number to someone else, your control is diminished because it depends in part on what someone else does. As soon as you give out your number, you transfer authority and control to someone else. You may say "don't give my number to anyone else," "use discretion," or "I am sensitive about my privacy," but you do not control the other person. You have to trust the other person to comply with your wishes, whether you state them explicitly or not. This problem is similar to the propagation problem of computer security: Anyone who has access to an object can copy, transfer, or propagate that object or its content to others without restriction.
Sensitive Data

Someone asks you for your shoe size; you might answer, "I'm a very private person and cannot imagine why you would want to know such an intimate detail" or you could say "10C"; some people find that data more sensitive than others. We know things people usually consider sensitive, such as financial status, certain health data, unsavory events in their past, and the like, so if you learn something you consider sensitive about someone, you will keep it quiet. But most of us are not too sensitive about our shoe size, so we don't normally protect that if we learn it about someone else. Of course, if a friend told me not to pass that along, I wouldn't. It is not up to me to question why someone else considers something private.

Here are examples (in no particular order) of data many people consider private.

- identity, the ownership of private data and the ability to control its disclosure
- finances, credit, bank details
- legal matters
- medical conditions, drug use, DNA, genetic predisposition to illnesses
- voting, opinions, membership in advocacy organizations
- preferences: religion, sexuality
- biometrics, physical characteristics, polygraph results, fingerprints
- diaries, poems, correspondence, recorded thoughts
- privileged communications with professionals such as lawyers, accountants, doctors, counselors, and clergy
- performance: school records, employment ratings
- activities: reading habits, web browsing, music, art, videos
- air travel data, general travel data, a person's location (present and past)
- communications: mail, e-mail, telephone calls, spam
- history: "youthful indiscretions," past events
- illegal activities, criminal records

Privacy is also affected by who you are. When you are in a room of people you don't know, perhaps at a reception, someone may come up to you and say "So you are the man who baked that beautiful cake over there; I really appreciate your skills as a pastry chef." It feels kind of nice to get that kind of recognition. Conversely, a friend was frequently on local television; she far preferred having dinner at home instead of going to a restaurant because she had grown tired of people rushing up to her saying "you're [Olga], I see you all the time on TV." Public personalities cherish the aspects of privacy they retain. World champion athletes cannot avoid having their results made public, whereas you might not want everyone to know how poorly you finished in the last event. Culture also influences what people consider sensitive.

In general, a person's privacy expectations depend on context: who is affected and what the prevailing norm of privacy is.

Affected Subject

This brings us to another point about privacy: Individuals, groups, companies, organizations, and governments all have data they consider sensitive. So far we have described privacy from the standpoint of a person. Companies may have data they consider private or sensitive: product plans, key customers, profit margins, and newly discovered technologies. For organizations such as companies, privacy usually relates to gaining and maintaining an edge over the competition. Other organizations, for example, schools, hospitals, or charities, may need to protect personal data on their students, patients, or donors, or they may want to control negative news, and so forth. Governments consider military and diplomatic matters sensitive, but they also recognize a responsibility to keep confidential data they collect from citizens, such as tax information. We may use terms like subject or owner to cover privacy issues affecting people, groups, and the like.

Privacy is an aspect of confidentiality. As we have learned throughout this book, the three security goals of confidentiality, integrity, and availability conflict, and confidentiality frequently conflicts with availability. If you choose not to have your telephone number published in a directory, that also means some people will not be able to reach you by telephone.

Summary

To summarize, here are some points about privacy:

- Privacy is controlled disclosure: The subject chooses what personal data to give out and to whom.
- After disclosing something, a subject relinquishes much control to the receiver.
- What data are sensitive is at the discretion of the subject; people consider different things sensitive. Why a person considers something sensitive is less important than that it is.
- Individuals, informal groups, and formal organizations all have things they consider private.
- Privacy has a cost; choosing not to give out certain data may limit other benefits.

In the next section we consider some examples of data that some people consider private.

Computer-Related Privacy Problems

You may notice that many of the kinds of sensitive data and many of the points about privacy have nothing to do with computers. You are exactly right: These sensitivities and issues predate computers. Computers and networks have only affected the feasibility of some unwanted disclosures. Public records offices have long been open for people to study the data held there, but the storage capacity and speed of computers have given us the ability to a mass, search, and correlate. Search engines have given us the ability to find one data item out of billions, the equivalent of finding one sheet of paper out of a warehouse full of boxes of papers. Furthermore, the openness of networks and the portability of technology (such as laptops, PDAs, cell phones, and memory devices) have greatly increased the risk of disclosures affecting privacy.

Rezgui et al. [REZ03] list eight dimensions of privacy (specifically as it relates to the web, although the definitions carry over naturally to other types of computing).

- *Information collection*: Data are collected only with knowledge and explicit consent.
- *Information usage*: Data are used only for certain specified purposes.
- *Information retention*: Data are retained for only a set period of time.
- *Information disclosure*: Data are disclosed to only an authorized set of people.
- *Information security*: Appropriate mechanisms are used to ensure the protection of the data.
- *Access control*: All modes of access to all forms of collected data are controlled.
- *Monitoring*: Logs are maintained showing all accesses to data.
- *Policy changes*: Less restrictive policies are never applied after-the-fact to already obtained data.

Here are the privacy issues that have come about through use of computers.

Data Collection

As we have previously said, advances in computer storage make it possible to hold and manipulate huge numbers of records. Disks on ordinary consumer PCs are measured in gigabytes (10^9 bytes), and commercial storage capacities often measure in terabytes (10^{12} bytes). In 2006, EMC Corporation announced a storage product whose capacity exceeds one peta byte (10^{15} bytes). (For perspective on these numbers, scientists estimate the capacity of the human brain to be between one terabyte and one petabyte.) Indiana University plans to acquire a supercomputer with one petabyte of storage, and the San Diego Supercomputer Center has online storage of one petabyte and offline archives of seven petabytes. Estimates of Google's stored data are also in the petabyte range. We have both devices to store massive amounts of data and the data to fill those devices. Whereas physical space limited storing (and locating) massive amounts of printed data, electronic data take relatively little space.

We never throw away data; we just move it to slower secondary media or buy more storage.

No Informed Consent

Where do all these bytes come from? Although some are from public and commercial sources (newspapers, web pages, digital audio, and video recordings) and others are from intentional data transfers (tax returns, a statement to the police after an accident, readers' survey forms, school papers), still others are collected without announcement. Telephone companies record the date, time, duration, source, and destination of each telephone call. ISPs track sites visited. Some sites keep the IP address of each visitor to the site (although an IP address is usually not unique to a specific individual). The user is not necessarily aware of this third category of data collection and thus cannot be said to have given informed consent.

Loss of Control

We realize that others may keep data we give them. When you order merchandise online, you know you have just released your name, probably some address and payment data, and the items you purchased. Or when you use a customer appreciation card at a store, you know the store can associate your identity with the things you buy. Having acquired your data, a merchant can redistribute it to anyone. The fact that you booked one brand of hotel room through a travel agent could be sold to other hotels. If you frequently telephone someone in one city and have taken several plane trips to that city, local stores, restaurants, or tourist attractions in that city might want your name. You have little control over dissemination (or redissemination) of your data.

We do not always appreciate the ramifications of lost control. Suppose in a moment of anger you dash off a strong note to someone. Although 100 years ago you would have written the note on paper and 50 years ago you would have voiced the comment by telephone, now you post the message to a blog. Next suppose you have a change of heart and you want to retract your angry note. Let us consider how you would deal with these three forms of the communication. For the written note, you write a letter of apology, your recipient tears up your first note, and no trace remains. In the second case you telephone to apologize and all that remains is a memory. As for the blog, you delete your posting. However, several other people might have seen your original posting and copied it to blogs or other web sites that you do not control. Search engines might have found the original or copies. And other people might have picked up your words and circulated them in e-mail. Thus, with letters and phone calls, we can usually obliterate something we want to retract. But once something is out of your control on the web, it may never be deleted.

This example concerned something you wrote. A similar situation concerns something written about you. Someone else has posted something on the web that is personal about you and you want it removed. Even if the poster agrees, you may not be able to remove all its traces.

Finally, some people are finding they reveal more than they should on sites like myspace.com.

Prospective employees are being turned down for jobs because of things they have written.

The web is a great historical archive, but because of archives, caches, and mirror sites, things posted on the web may never go away.

A second issue of loss of control concerns data exposure. Suppose a company holds data about you and that company's records are exposed in a computer attack. The company may not be responsible for preventing harm to you, compensating you if you are harmed, or even informing you of the event.

Ownership of the Data

In the cases just described, customer details are being marketed. Information about you is being sold and you have no control; nor do you get to share in the profit. Even before computers customer data were valuable. Mailing lists and customer lists were company assets that were safeguarded against access by the competition. Sometimes companies rented their mailing lists when there was not a conflict with a competitor. But in those cases, the subject of the data, the name on the list, did not own the right to be on the list or not. With computers the volume and sources of data have increased significantly, but the subject still has no rights.

These issues loss of control, no informed consent, no ownership of datahave significant privacy implications. The way we address these kinds of issues is with policies, written statements of practice that inform all affected parties of their rights. In the next section we investigate privacy policies for computing.

10.2. Privacy Principles and Policies

In the United States, interest in privacy and computer databases dates back at least to the early 1970s. (It is worth noting that the U.S. Watergate burglary occurred in 1972. Shortly after, reports surfaced that Nixon maintained an enemies list and had used IRS records as a means of combating adversaries. Thus people in the United States were sensitive about privacy at that time. Public concern for privacy has varied over the years.) In the early 1970s, a committee developed privacy principles that have affected U.S. laws and regulations and that also set the path for privacy legislation in other countries. We study the recommendations of that committee in the next section.

Fair Information Policies

In 1973 Willis Ware of the RAND Corporation chaired a committee to advise the Secretary of the U.S. Department of Human Services on privacy issues. The report (summarized in [WAR73a]) proposes a set of principles of fair information practice.

 Collection limitation. Data should be obtained lawfully and fairly.

 Data quality. Data should be relevant to their purposes, accurate, complete, and up-to-date.

 Purpose specification. The purposes for which data will be used should be identified and the data destroyed if no longer necessary to serve that purpose.

 Use limitation. Use for purposes other than those specified is authorized only with consent of the data subject or by authority of law.
 Security safeguards. Procedures to guard against loss, corruption, destruction, or misuse of data should be established.
 Openness. It should be possible to acquire information about the collection, storage, and use of personal data systems.
 Individual participation. The data subject normally has a right to access and to challenge data relating to her.
 Accountability. A data controller should be designated and accountable for complying with the measures to give effect to the principles.
These principles describe the rights of individuals, not requirements on collectors; that is, the principles do not require protection of the data collected.

Ware [WAR73b] raises the problem of linking data in multiple files and of overusing keys, such as social security numbers, that were never intended to be used to link records. And although he saw that society was moving toward a universal identity number, he feared that movement would be without plan (and hence without control). He was right, even though he could not have foreseen the amount of data exchanged 30 years later.

Turn and Ware [TUR75] consider protecting the data themselves, recognizing that collections of data will be attractive targets for unauthorized access attacks. They suggest four ways to protect stored data:
 Reduce exposure by limiting the amount of data maintained, asking for only what is necessary and using random samples instead of complete surveys.
 Reduce data sensitivity by interchanging data items or adding subtle errors to the data (and warning recipients that the data have been altered).
 Anonymize the data by removing or modifying identifying data items.
 Encrypt the data.

You will see these four approaches mentioned again because they are the standard techniques available for protecting the privacy of data.

U.S. Privacy Laws

Ware and his committee expected these principles to apply to all collections of personal data on individuals. Unfortunately, that is not the way the legislation developed. The Ware committee report led to the 1974 Privacy Act (5 USC 552a), which embodies most of these principles, although that law applies only to data maintained by the U.S. government.

The Privacy Act is a broad law, covering all data collected by the government. It is the strongest U.S. privacy law because of its breadth: It applies to all personal data held anywhere in the government.

The United States subsequently passed laws protecting data collected and held by other organizations, but these laws apply piecemeal, by individual data type. Consumer credit is addressed in the Fair Credit Reporting Act, healthcare information in the Health Insurance Portability and Accountability Act (HIPAA), financial service organizations in the Gramm Leach Bliley Act (GLBA), children's web access in the Children's Online Privacy Protection Act (COPPA), and student records in the Federal Educational Rights and Privacy Act. Not surprisingly these separate laws are inconsistent in protecting privacy.

Laws and regulations do help in some aspects of privacy protection. Antón et al. investigated the impact of the HIPAA law by analyzing companies' posted privacy policies before and after the privacy provisions of the law became effective [ANT06]. They found the following in policies posted after HIPAA:
 Statements on data transfer (to other organizations) were more explicit than before HIPAA.
 Consumers still had little control over the disclosure or dissemination of their data.
 Statements were longer and more complex, making them harder for consumers to understand.
 Even within the same industry branch (such as drug companies), statements varied substantially, making it hard for consumers to compare policies.
 Statements were unique to specific web pages, meaning they covered more precisely the content and function of a particular page.

A problem with many laws is that the target areas of the laws still overlap: Which law (if any) would require privacy protection of a university student's health center bills paid by credit card? The laws have different protection and handling requirements, so it is important to determine which law applies to a single piece of data. Also, gaps between laws are not covered. As new technologies (such as computers, the Internet, or cell phones) are developed, either existing privacy laws have to be reinterpreted by the courts to apply to the new technologies or new laws have to be passed, which takes time.

Sometimes the privacy provisions of a law are a second purpose, somewhat disguised by the first purpose of the law. As an example, the primary purpose of HIPAA was to ensure that people who left or were terminated from one job had health insurance to cover them until they got another job; the privacy aspects were far less prominent as the law was being developed.

Controls on U.S. Government Web Sites

Because privacy is ambiguous, privacy policies are an important way to both define the concept in a particular setting and specify what should or will be done about it. The Federal Trade Commission (FTC) has jurisdiction over web sites, including those of the federal government, that solicit potentially private data. In 2000 [FTC00], the FTC set requirements for privacy policy for government web sites. Because government web sites are covered by the Privacy Act, it was easy for the FTC to require privacy protection. The FTC determined that in order to obey the Privacy Act, government web sites would have to address five privacy factors.

- *Notice*. Data collectors must disclose their information practices before collecting personal information from consumers.
- *Choice*. Consumers must be given a choice as to whether and how personal information collected from them may be used.
- *Access*. Consumers should be able to view and contest the accuracy and completeness of data collected about them.
- *Security*. Data collectors must take reasonable steps to ensure that information collected from consumers is accurate and secure from unauthorized use.
- *Enforcement*. A reliable mechanism must be in place to impose sanctions for noncompliance with these fair information practices.

In 2002, the U.S. Congress enacted the e-Government Act of 2002 requiring that federal government agencies post privacy policies on their web sites. Those policies must disclose

- the information that is to be collected
- the reason the information is being collected
- the intended use by the agency of the information
- the entities with whom the information will be shared
- the notice or opportunities for consent that would be provided to individuals regarding what information is collected and how that information is shared
- the way in which the information will be secured
- the rights of the individual under the Privacy Act and other laws relevant to the protection of the privacy of an individual

These two acts apply only to web sites; data collected by other means (for example, by filing forms) are handled differently, usually on a case-by-case or agency-by-agency basis. The requirements reflected in the e-Government Act focus on the type of data (data supplied to the government through a web site) and not on the general notion of privacy.

Controls on Commercial Web Sites

The e-Government Act places strong controls on government data collection through web sites. As we described, privacy outside the government is protected by law in some areas, such as credit, banking, education, and healthcare. But there is no counterpart to the e-Government Act for private companies.

No Deceptive Practices

The Federal Trade Commission has the authority to prosecute companies that engage in deceptive trade or unfair business practices. If a company advertises in a false or misleading way, the FTC can sue. The FTC has used that approach on web privacy: If a company advertises a false privacy protection that is, if the company says it will protect privacy in some way but does not do so the FTC considers that false advertising and can take legal action.

Because of the FTC, privacy notices at the bottom of web sites do have meaning. This practice leads to a bizarre situation, however. A company is allowed to collect personal information and pass it in any form to anyone, as long as the company's privacy policy said it would do so, or at least the policy did not say it would not do so. Vowing to maintain privacy and intentionally not doing so is an illegal deceptive practice. Stating an intention to share data with marketing firms or "other third parties" makes such sharing acceptable, even though the third parties could be anyone.

Examples of Deceptive Practices

The FTC settled a prosecution in 2005 against Cart Manager International, a firm that runs familiar web shopping cart software to collect items of an order, obtain the purchaser's name and address, and determine shipping and payment details. This software runs as an application under other well-known retail merchants' web sites to handle order processing. Some of these other retailers had privacy statements on their web sites saying, in effect, that they would not sell or distribute customers' data, but Cart Manager did sell the data it collected. The FTC held that the relationship to Cart Manager was invisible to users, and so the policy from the online merchants applied also to Cart Manager.

In another case, Antón [ANT04] analyzed the privacy policy posted on the web site of Jet Blue airlines and found it misleading. Jet Blue stated that it would not disclose passenger data to third parties. It then released passenger data, "in response to a special request from the Department of Defense" to Torch Concepts, which in turn passed it to the Defense Department to use to test passenger screening algorithms for airline security. The data in question involved credit card information: Clearly the only reason for Jet Blue to have collected those data from passengers was to process charges for airline tickets. The analysis by Antón is interesting for two reasons:

First, Jet Blue violated its own policy.

Second, the Department of Defense may have circumvented the e-Government Act by acquiring from a private company data it would not have been able to collect as a government entity. The purpose for which the data were originally collected was ordinary business and accounting activities of Jet Blue. Using those same records to screen for terrorists was outside the scope of the original data collection.

Commercial sites have no standard of content comparable to the FTC recommendation from the e-Government Act. Some companies display solid and detailed privacy statements that they must obey. On the other hand, you may find no statement at all, which gives the company the greatest flexibility because it is impossible to lie when saying nothing. Cranor [CRA03] makes some recommendations for useful web privacy policies.

Non-U.S. Privacy Principles

In 1981, the Council of Europe (an international body of 46 European countries, founded in 1949) adopted Convention 108 for the protection of individuals with regard to the automatic processing of personal data, and in 1995, the European Union (E.U.) adopted Directive 95/46/EC on the processing of personal data. Directive 95/46/EC, often called the European Privacy Directive, requires that rights of privacy of individuals be maintained and that data about them be

- processed fairly and lawfully
- collected for specified, explicit and legitimate purposes and not further processed in a way incompatible with those purposes (unless appropriate safeguards protect privacy)
- adequate, relevant, and not excessive in relation to the purposes for which they are collected and/or further processed
- accurate and, where necessary, kept up to date; every reasonable step must be taken to ensure that inaccurate or incomplete data having regard for the purposes for which they were collected or for which they are further processed, are erased or rectified
- kept in a form that permits identification of data subjects for no longer than is necessary for the purposes for which the data were collected or for which they are further processed

In addition, individuals have the right to access data collected about them, to correct inaccurate or incomplete data, and to have those corrections sent to those who have received the data. The report adds three more principles to the Fair Information Policies.

- *Special protection for sensitive data*. There should be greater restrictions on data collection and processing that involves "sensitive data." Under the E.U. data protection directive, information is sensitive if it involves "racial or ethnic origin, political opinions, religious beliefs, philosophical or ethical persuasion . . . [or] health or sexual life."

 Data transfer. This principle explicitly restricts authorized users of personal information from transferring that information to third parties without the permission of the data subject.

 Independent oversight. Entities that process personal data should not only be accountable but should also be subject to independent oversight. In the case of the government, this requires oversight by an office or department that is separate and independent from the unit engaged in the data processing. Under the data protection directive, the independent overseer must have the authority to audit data processing systems, investigate complaints brought by individuals, and enforce sanctions for noncompliance.

(This is a very brief summary of the much longer law. See the original Directive for more detail.) These requirements apply to governments, businesses, and other organizations that collect personal data. Since the 1995 directive, the European Union has extended coverage to telecommunications systems and made other changes to adapt to advances in technology.

In addition to European countries and the United States, other countries, such as Japan, Australia, and Canada, have passed laws protecting the privacy of personal data about individuals.

Different laws in different jurisdictions will inevitably clash. Relations between the European Union and the United States have been strained over privacy because the E.U. law forbids sharing data with companies or governments in countries whose privacy laws are not as strong as those of the E.U. (The United States and the European Union have agreed to a set of "safe harbor" principles that let U.S. companies trade with European countries in spite of not meeting all European privacy laws.) In Sidebar 10-1 you can see how these different laws can affect commerce and, ultimately, diplomatic relations.

Anonymity, Multiple Identities

One way to preserve privacy is to guard our identity. Not every context requires us to reveal our identity, so some people wear a form of electronic mask.

Anonymity

A person may want to do some things anonymously. For example, a rock star buying a beach house might want to avoid unwanted attention from neighbors, or someone posting to a dating list might want to view replies before making a date.

Mulligan [MUL99] lists several reasons people prefer anonymous activity on the web. Some people like the anonymity of the web because it reduces fears of discrimination. Fairness in housing, employment, and association are easier to ensure when the basis for potential discrimination is hidden. Also, people researching what they consider a private matter, such as a health issue or sexual orientation, are more likely to seek first information from what they consider an anonymous source, turning to a human when they have found out more about their situation.

Anonymity creates problems, too. How does an anonymous person pay for something? A trusted third party (for example, a real estate agent or a lawyer) can complete the sale and preserve anonymity. But then you need a third party and the third party knows who you are.

Chaum [CHA81, CHA82, CHA85] studied this problem and devised a set of protocols by which such payments could occur without revealing the buyer to the seller.

Multiple IdentitiesLinked or Not

Most people already have multiple identities. To your bank you might be the holder of account 123456, to your motor vehicles bureau you might be the holder of driver's license number 234567, and to your credit card company you might be the holder of card 345678. For their purposes, these numbers are your identity; the fact that each may (or may not) be held in your name is irrelevant. The name does become important if it is used as a way to link these records. How many people share your name? Can (or should) it serve as a key value to link these separate databases? We ignore the complication of misspellings and multiple valid forms (with and without middle initials, with full middle name, with one of two middle names if you have them, and so forth).

Suppose you changed your name legally but never changed the name on your credit card; then your name could not be used as a key on which to link. Another possible link field is address. However, trying to use an address on which to link presents another risk: Perhaps a criminal lived in your house before you bought it. You should not have to defend your reputation because of a previous occupant. Now we need to match on date, too, so we connect only people who actually lived in a house at the same time. Then we need to address the problem of group houses or roommates of convenience, and so forth. As computer scientists, we know we can program all these possibilities, but that requires careful and time-consuming consideration of the potential problems before designing the solution. We can also see the potential for misuse and inaccuracy.

Linking identities correctly to create dossiers and break anonymity creates privacy risks, but linking them incorrectly creates much more serious risks for the use of the data and the privacy of affected people. If we think carefully we can determine many of the ways such a system would fail, but that approach is potentially expensive and time consuming. The temptation to act quickly but inaccurately will also affect privacy.

Pseudonymity

Sometimes, full anonymity is not wanted. A person may want to order flower bulbs but not be placed on a dozen mailing lists for gardening supplies. But the person does want to be able to place similar orders again, asking for the same color tulips as before. This situation calls for pseudonyms, unique identifiers that can be used to link records in a server's database but that cannot be used to trace back to a real identity.

Multiple identities can also be convenient, for example, having a professional e-mail account and a social one. Similarly, disposable identities (that you use for a while and then stop using) can be convenient. When you sign up for something and you know your e-mail address will be sold many times, you might get a new e-mail address to use until the spam and other unsolicited e-mail are oppressive, and then you discard the address. These uses are called pseudonymity. Seigneur and Jensen [SEI03] discuss the use of e-mail aliases to maintain privacy. These ways protect our privacy because we do not have to divulge what we consider sensitive data. But they also show we need a form of privacy protection that is unavailable.

The Swiss bank account was a classic example of a pseudonym. Each customer had only a number to access the account. Presumably anyone with that number could perform any transaction on the account. (Obviously there were additional protections against guessing.)

While such accounts were in use (their use was discontinued in the early 1990s because of their having been used to hold ill-gotten Nazi gains from World War II), Swiss banks had an outstanding reputation for maintaining the anonymity of the depositors.

Some people register pseudonyms with e-mail providers so that they have anonymous drop boxes for e-mail. Others use pseudonyms in chat rooms or with online dating services. We consider pseudonyms later in this chapter when we study privacy for e-mail.

Government and Privacy

The government gathers and stores data on citizens, residents, and visitors. Government facilitates and regulates commerce and other kinds of personal activities such as healthcare, employment, education, and banking. In those roles the government is both an enabler or regulator of privacy and a user of private data. Government use of private data should be controlled. In this section we consider some of the implications of government access to private data.

Authentication

Government plays a complex role in personal authentication. Many government agencies (such as the motor vehicles bureau) use identifiers to perform their work. Authentication documents (such as passports and insurance cards) often come from the government. The government may also regulate the businesses that use identification and authentication keys. And sometimes the government obtains data based on those keys from others (for example, the U.S. government planned to buy credit reports from private companies to help with screening airline passenger lists for terrorists). In these multiple roles, the government may misuse data and violate privacy rights.

Data Access Risks

Recognizing that there were risks in government access to personal data, the Secretary of Defense appointed a committee to investigate private data collection. The Technology and Privacy Advisory Committee, chaired by Newton Minow, former chair of the

Federal Communications Commission, produced its report in 2004 [TAP04]. Although their charge had been to review privacy and data collection within the Department of Defense, they found it impossible to separate the DoD from the rest of government, so they made recommendations for both the Department of Defense and the federal government as a whole.

They recognized risks when the government started to acquire data from other parties:

 data errors: ranging from transcription errors to incorrect analysis
 inaccurate linking: two or more correct data items but incorrectly linked on a presumed common element
 difference of form and content: precision, accuracy, format, and semantic errors
 purposely wrong: collected from a source that intentionally gives incorrect data, such as a forged identity card or a false address given to mislead
 false positive: an incorrect or out-of-date conclusion that the government does not have data to verify or reject, for example, delinquency in paying state taxes
 mission creep: data acquired for one purpose leading to a broader use because the data will support that mission
 poorly protected: data of questionable integrity because of the way it has been managed and handled

These risks apply to all branches of government, and most of them apply to private collection and use of data.

Steps to Protect Against Privacy Loss

The committee recommended several steps the government can take to help safeguard private data.

 Data minimization. Obtain the least data necessary for the task. For example, if the goal is to study the spread of a disease, only the condition, date, and vague location (city or county) may suffice; the name or contact information of the patient may be unnecessary.
 Data anonymization. Where possible, replace identifying information with untraceable codes (such as a record number); but make sure those codes cannot be linked to another database that reveals sensitive data.
 Audit trail. Record who has accessed data and when, both to help identify responsible parties in the event of a breach and to document the extent of damage.
 Security and controlled access. Adequately protect and control access to sensitive data.
 Training. Ensure people accessing data understand what to protect and how to do so.
 Quality. Take into account the purpose for which data were collected, how they were stored, their age, and similar factors to determine the usefulness of the data.
 Restricted usage. Different from controlling access, review all proposed uses of the data to determine if those uses are consistent with the purpose for which the data were collected and the manner in which they were handled (validated, stored, controlled).
 Data left in place. If possible, leave data in place with the original owner. This step helps guard against possible misuses of the data from expanded mission just because the data are available.
 Policy. Establish a clear policy for data privacy. Do not encourage violation of privacy policies.

These steps would help significantly to ensure protection of privacy.

Identity Theft

As the name implies, identity theft is taking another person's identity. Use of another person's credit card is fraud; taking out a new credit card in that person's name is identity theft. Identity theft has risen as a problem from a relatively rare issue in the 1970s. In 2005, the U.S. Federal Trade Commission received over 250,000 complaints of identity theft [FTC06].

Most cases of identity theft become apparent in a month or two when fraudulent bills start coming in. By that time the thief has made a profit and has dropped this identity, moving on to a new victim.

Having relatively few unique keys facilitates identity theft: A thief who gets one key can use that to get a second, and those two to get a third. Each key gives access to more data and resources. Few companies or agencies are set up to ask truly discriminating authentication questions (such as the grocery store at which you frequently shop or the city to which you recently bought an airplane ticket or third digit on line four of your last tax return). Because there are few authentication keys, we are often asked to give the same key (such as mother's maiden name) out to many people, some of whom might be part-time accomplices in identity theft.

10.3. Authentication and Privacy

In Chapter 4 we studied authentication, which we described as a means of proving or verifying a previously given identity. We also discussed various authentication technologies, which are subject to false accept (false positive) and false reject (false negative) limitations. A social problem occurs when we confuse authentication with identification.
We know that passwords are a poor discriminator. You would not expect all users of a system to have chosen different passwords. All we need is for the IDpassword pair to be unique. On the other end of the spectrum, fingerprints and the blood vessel pattern in the retina of the eye are unique: given a fingerprint or retina pattern we expect to get but one identity that corresponds or to find no match in the database. That assumes we work with a good image. If the fingerprint is blurred or incomplete (not a complete contact or on a partly unsuitable surface), we might get several possible matches. If the possible matches are A, B, and C and the question is whether the print belongs to B, it is probably acceptable to allow the access on the grounds that the identity was among a small set of probable matches. Other authenticators are less sophisticated still. Hand geometry or the appearance of a face does not discriminate so well. Face recognition, in particular, is highly dependent on the quality of the facial image: Evaluating a photograph of one person staring directly into a camera is very different from trying to work with one face in the picture of a crowd.

Two different purposes are at work here, although the two are sometimes confused. For authentication we have an identity and some authentication data, and we ask if the authentication data match the pattern for the given identity. For identification, we have only the authentication data and we ask which identity corresponds to the authenticator. The second is a much harder question to answer than the first. For the first, we can say the pattern matches some percentage of the characteristics of our stored template, and based on the percentage, we declare a match or no match. For the second question, we do not know if the subject is even in the database. So even if we find several potential matches at various percentages, we do not know if there might be an even better match with a template not in our database.

What Authentication Means

We use the term authentication to mean three different things [KEN03]: We authenticate an individual, identity, or attribute. An individual is a unique person. Authenticating an individual is what we do when we allow a person to enter a controlled room: We want only that human being to be allowed to enter. An identity is a character string or similar descriptor, but it does not necessarily correspond to a single person, nor does each person have only one name. We authenticate an *identity* when we acknowledge that whoever (or whatever) is trying to log in as *admin* has presented an authenticator valid for that account. Similarly, authenticating an identity in a chat room as SuzyQ does not say anything about the person using that identifier:

It might be a 16-year-old girl or a pair of middle-aged male police detectives, who at other times use the identity Frere Jacques.

Finally, we authenticate an *attribute* if we verify that a person has that attribute. An attribute is a characteristic. Here's an example of authenticating an attribute. Some places require one to be 21 or older in order to drink alcohol. A club's doorkeeper verifies a person's age and stamps the person's hand to show that the patron is over 21. Note that to decide, the doorkeeper may have looked at an identity card listing the person's birth date, so the doorkeeper knew the person's exact age to be 24 years, 6 months, 3 days, or the doorkeeper might be authorized to look at someone's face and decide if the person looks so far beyond 21 that there is no need to verify. The stamp authenticator signifies only that the person possesses the attribute of being 21 or over.

In computing applications we frequently authenticate individuals, identities, and attributes. Privacy issues arise when we confuse these different authentications and what they mean.

For example, the U.S. social security number was never intended to be an identifier, but now it often serves as an identifier, an authenticator, a database key, or all of these.

When one data value serves two or more uses, a person acquiring it for one purpose can use it for another. Relating an identity to a person is tricky. In Chapter 7 we tell the story of rootkits, malicious software by which an unauthorized person can acquire supervisory control of a computer. Suppose the police arrest Ionut for chewing gum in public and seize his computer. By examining the computer the police find evidence connecting that computer to an espionage case. The police show incriminating e-mail messages from Ionut on Ionut's computer and charge him. In his defense, Ionut points to a rootkit on his computer. He acknowledges that his computer may have been used in the espionage, but he denies that he was personally involved. The police have, he says, drawn an unjustifiable connection between Ionut's identity in the e-mail and Ionut the person. The rootkit is a plausible explanation for how some other person acted under the identity of Ionut. This example shows why we must carefully distinguish individual, identity, and attribute authentication.

We examine the privacy implications of authentication in the next section.

Individual Authentication

There are relatively few ways of identifying an individual. When we are born, for most of us our birth is registered at a government records office, and we (probably our parents) receive a birth certificate. A few years later our parents enroll us in school, and they have to present the birth certificate, which then may lead to receiving a school identity card. We submit the birth certificate and a photo to get a passport or a national identity card. We receive many other authentication numbers and cards throughout life.

The whole process starts with a birth certificate issued to (the parents of) a baby, whose physical description (height, weight, even hair color) will change significantly in just months.

Birth certificates may contain the baby's fingerprints, but matching a poorly taken fingerprint of a newborn baby to that of an adult is challenging at best. (For additional identity authentication problems, see Sidebar 10-2.)

Fortunately, in most settings it is acceptable to settle for weak authentication for individuals:

A friend who has known you since childhood, a schoolteacher, neighbors, and coworkers can support a claim of identity.

Identity Authentication

We all use many different identities. When you buy something with a credit card, you do so under the identity of the credit card holder. In some places you can pay road tolls with a radio frequency device in your car, so the sensor authenticates you as the holder of a particular toll device. You may have a meal plan that you can access by means of a card, so the cashier authenticates you as the owner of that card. You check into a hotel and get a magnetic stripe card instead of a key, and the door to your room authenticates you as a valid resident for the next three nights. If you think about your day, you will probably find 10 to 20 different ways some identity of you has been authenticated.

From a privacy standpoint, there may or may not be ways to connect all these different identities. A credit card links to the name and address of the card payer, who may be you, your spouse, or anyone else willing to pay your expenses. Your auto toll device links to the name and perhaps address of whoever is paying the tolls: you, the car's owner, or an employer. When you make a telephone call, there is an authentication to the account holder of the telephone, and so forth.

Sometimes we do not want an action associated with an identity. For example, an anonymous tip or "whistle-blower's" telephone line is a means of providing anonymous tips of illegal or inappropriate activity. If you know your boss is cheating the company, confronting your boss might not be a good career-enhancing move. You probably don't even want there to be a record that would allow your boss to determine who reported the fraud. So you report it anonymously. You might take the precaution of calling from a public phone so there would be no way to trace the person who called. In that case, you are purposely taking steps so that no common identifier could link you to the report.

Because of the accumulation of data, however, linking may be possible. As you leave your office to go to a public phone, there is a record of the badge you swiped at the door. A surveillance camera shows you standing at the public phone. The record of the coffee shop has a timestamp showing when you bought your coffee (using your customer loyalty card) before returning to your office. The time of these details matches the time of the anonymous tip by telephone. In the abstract these data items do not stand out from millions of others. But someone probing a few minutes around the time of the tip can construct those links. In this example, linking would be done by hand. Ever-improving technology permits more parallels like these to be drawn by computers from seemingly unrelated and uninteresting datapoints.

Therefore, to preserve our privacy we may thwart attempts to link records. A friend gives a fictitious name when signing up for customer loyalty cards at stores. Another friend makes dinner reservations under a pseudonym. In one store they always ask for my telephone number when I buy something, even if I pay cash. Records clerks do not make the rules, so it is futile asking them why they need my number. If all they want is a number, I gladly give them one; it just doesn't happen to correspond to me.

Anonymized Records

Part of privacy is linkages: Some person is named Erin, some person has the medical condition diabetes; neither of those facts is sensitive. The linkage that Erin has diabetes becomes sensitive.

Medical researchers want to study populations to determine incidence of diseases, common factors, trends, and patterns. To preserve privacy, researchers often deal with anonymized records, records from which identifying information has been removed. If those records can be reconnected to the identifying information, privacy suffers. If, for example, names have been removed from records but telephone numbers remain, a researcher can use a different database of telephone numbers to determine the patient, or at least the name assigned to the telephone. Removing enough information to prevent identification is difficult and can also limit the research possibilities.

As described in Chapter 6, Ross Anderson was asked to study a major database being prepared for citizens of Iceland. The database would have brought together several healthcare databases for the benefit of researchers and healthcare professionals. Anderson's analysis was that even though the records had been anonymized, it was still possible to relate specific records to individual people [AND98a, JON00]. Even though there were significant privacy difficulties, Iceland went ahead with plans to build the combined database.

In one of the most stunning analyses on deriving identities, Sweeney [SWE01] reports that 87 percent of the population of the United States is likely to be identified by the combination of 5-digit zip code, gender, and date of birth. That statistic is amazing when you consider that close to 10,000 U.S. residents must share any birthday or that the average population in any 5-digit zip code area is 30,000. Sweeney backs up her statistical analysis with a real-life study. In 1997 she analyzed the voter rolls of Cambridge, Massachusetts, a city of about 50,000 people, one of whom was the then current governor. She took him as an example and found that only six people had his birth date, only three of those were men, and he was the only one of those three living in his 5-digit zip code. As a public figure, he had published his date of birth in his campaign literature, but birth dates are sometimes available from public records. Similar work on deriving identities from anonymized records [SWE04, MAL02] showed how likely one is to deduce an identity from other easily obtained data.

Sweeney's work demonstrates compellingly how difficult it is to anonymize data effectively. Many medical records are coded with at least gender and date of birth, and those records are often thought to be releasable for anonymous research purposes. Furthermore, medical researchers may want a zip code to relate medical conditions to geography and demography.

Few people would think that adding zip codes would lead to such high rates of breach of privacy.

Conclusions

As we have just seen, identification and authentication are two different activities that are easy to confuse. Part of the confusion arises because people do not clearly distinguish the underlying concepts. The confusion is also the result of using one data item for more than one purpose.

Authentication depends on something that confirms a property. In life few sound authenticators exist, so we tend to overuse those we do have: an identification number, birth date, or family name. But, as we described, those authenticators are also used as database keys, with negative consequences to privacy.

We have also studied cases in which we do not want to be identified. Anonymity and pseudonymity are useful in certain contexts. But data collection and correlation, on a scale made possible only with computers, can defeat anonymity and pseudonymity.

As we computer professionals introduce new computer capabilities, we need to encourage a public debate on the related privacy issues.

In the next section we study data mining, a data retrieval process involving the linking of databases.

10.4. Data Mining

In Chapter 6 we described the process and some of the security and privacy issues of data mining. Here we consider how to maintain privacy in the context of data mining. Private sector data mining is a lucrative and rapidly growing industry. The more data collected, the more opportunities for learning from various aggregations. Determining trends, market preferences, and characteristics may be good because they lead to an efficient and effective market. But people become sensitive if the private information becomes known without permission.

Government Data Mining

Especially troubling to some people is the prospect of government data mining. We believe we can stop excesses and intrusive behavior of private companies by the courts, unwanted publicity, or other forms of pressure. It is much more difficult to stop the government. In many examples governments or rulers have taken retribution against citizens deemed to be enemies, and some of those examples come from presumably responsible democracies. Much government data collection and analysis occurs without publicity; some programs are just not announced and others are intentionally kept secret. Thus, citizens have a fear of what unchecked government can do. Citizens' fears are increased because data mining is not perfect or exact, and as many people know, correcting erroneous data held by the government is next to impossible.

Privacy-Preserving Data Mining

Because data mining does threaten privacy, researchers have looked into ways to protect privacy during data mining operations. A naïve and ineffective approach is trying to remove all identifying information from databases being mined. Sometimes, however, the identifying information is precisely the goal of data mining. More importantly, as the preceding example from Sweeney showed, identification may be possible even when the overt identifying information is removed from a database.

Data mining has two approaches correlation and aggregation. We examine techniques to preserve privacy with each of those approaches.

Privacy for Correlation

Correlation involves joining databases on common fields. As in a previous example, the facts that someone is named Erin and someone has diabetes have privacy significance only if the link between Erin and diabetes exists. Privacy preservation for correlation attempts to control that linkage.

Vaidya and Clifton [VAI04] discuss data perturbation as a way to prevent privacy-endangering correlation. As a simplistic example, assume two databases contain only three records, as shown in Table 10-1. The ID field linking these databases makes it easy to see that Erin has diabetes.

Table 10-1. Example for Data Perturbation.

Name	ID
Erin	1
Aarti	2
Geoff	3

ID	Condition
1	diabetes
2	none
3	measles

One form of data perturbation involves swapping data fields to prevent linking of records. Swapping the values Erin and Geoff (but *not* the ID values) breaks the linkage of Erin to diabetes. Other properties of the databases are preserved: Three patients have actual names and three conditions accurately describe the patients. Swapping all data values can prevent useful analysis, but limited swapping balances privacy and accuracy. With our example of swapping just Erin and Geoff, you still know that one of the participants has diabetes, but you cannot know if Geoff (who now has ID=1) has been swapped or not. Because you cannot know if a value has been swapped, you cannot assume any such correlation you derive is true.

Our example of three data points is, of course, too small for a realistic data mining application, but we constructed it just to show how value swapping would be done. Consider a more realistic example on larger databases. Instead of names we might have addresses, and the purpose of the data mining would be to determine if there is a correlation between a neighborhood and an illness, such as measles. Swapping all addresses would defeat the ability to draw any correct conclusions regarding neighborhood. Swapping a small but significant number of addresses would introduce uncertainty to preserve privacy. Some measles patients might be swapped out of the high-incidence neighborhoods, but other measles patients would also be swapped in. If the neighborhood has a higher incidence than the general population, random swapping would cause more losses than gains, thereby reducing the strength of the correlation. After value swapping an already weak correlation might become so weak as to be statistically insignificant. But a previously strong correlation would still be significant, just not as strong.

Thus value-swapping is a technique that can help to achieve some degrees of privacy and accuracy under data mining.

Privacy for Aggregation

Aggregation need not directly threaten privacy. As demonstrated in Chapter 6, an aggregate (such as sum, median, or count) often depends on so many data items that the sensitivity of any single contributing item is hidden. Government statistics show this well: Census data, labor statistics, and school results show trends and patterns for groups (such as a neighborhood or school district) but do not violate the privacy of any single person.

As we explained in Chapter 6, inference and aggregation attacks work better nearer the ends of the distribution. If there are very few or very many points in a database subset, a small number of equations may disclose private data. The mean of one data value is that value exactly. With three data values, the means of each pair yield three equations in three unknowns, which you know can be solved easily with linear algebra. A similar approach works for very large subsets, such as (*n*-3) values. Mid-sized subsets preserve privacy quite well. So privacy is maintained with the rule of *n* items, over *k* percent, as described in Chapter 6.

Data perturbation works for aggregation, as well. With perturbation you add a small positive or negative error term to each data value. Agrawal and Srikant [AGR00] show that given the distribution of data after perturbation and given the distribution of added errors, it is possible to determine the distribution (*not* the values) of the underlying data. The underlying distribution is often what researchers want. This result demonstrates that data perturbation can help protect privacy without sacrificing the accuracy of results.

Vaidya and Clifton [VAI04) also describe a method by which databases can be partitioned to preserve privacy. Our trivial example in Table 10-1 could be an example of a database that was partitioned vertically to separate the sensitive association of name and condition.

Summary of Data Mining and Privacy

As we have described in this section, data mining and privacy are not mutually exclusive: We can derive results from data mining without sacrificing privacy. True, some accuracy is lost with perturbation. A counterargument is that the weakening of confidence in conclusions mostseriously affects weak results; strong conclusions become only marginally less strong.

Additional research will likely produce additional techniques for preserving privacy during data mining operations. We *can* derive results without sacrificing privacy, but privacy will not exist automatically. The techniques described here must be applied by people who understand and respect privacy implications. Left unchecked, data mining has the potential to undermine privacy. Security professionals need to continue to press for privacy in data mining applications.

10.5. Privacy on the Web

The Internet is perhaps the greatest threat to privacy. As Chapter 7 says, an advantage of the Internet, which is also a disadvantage, is anonymity. A user can visit web sites, send messages, and interact with applications without revealing an identity. At least that is what we would like to think. Unfortunately, because of things like cookies, ad-ware, spybots, and malicious code, the anonymity is superficial and largely one-sided. Sophisticated web applications can know a lot about a user, but the user knows relatively little about the application.

The topic is clearly of great interest: a recent Google search returned 7 billion hits for the phrase "web privacy." In this section we investigate some of the ways a user's privacy is lost on the Internet.

Understanding the Online Environment

The Internet is like a nightmare of a big, unregulated bazaar. Every word you speak can be heard by many others. And the merchants' tents are not what they seem: the spice merchant actually runs a gambling den, and the kind woman selling scarves is really three pirate brothers and a tiger. You reach into your pocket for money only to find that your wallet has been emptied. Then the police tell you that they would love to help but, sadly, no laws apply.

Caveat emptor in excelsis. We have previously described the anonymity of the web. It is difficult for two unrelated parties to authenticate each other. Internet authentication most often confirms the user's identity, not the server's, so the user is unsure that the web site is legitimate. This uncertainty makes it difficult to give informed consent to release of private data: How can consent be informed if you don't know to whom you are giving consent?

Payments on the Web

Customers of online merchants have to be able to pay for purchases. Basically, there are two approaches: the customer presents a credit card to the merchant or the customer arranges payment through an online payment system such as PayPal.

Credit Card Payments

With a credit card, the user enters the credit card number, a special number printed on the card (presumably to demonstrate that the user actually possesses the card), the expiration date of the card (to ensure that the card is currently active), and the billing address of the credit card (presumably to protect against theft of credit card). These protections are all on the side of the merchant: They demonstrate that the merchant made a best effort to determine that the credit card use was legitimate. There is no protection to the customer that the merchant will secure these data. Once the customer has given this information to one merchant, that same information is all that would be required for another merchant to accept a sale charged to the same card.

Furthermore, these pieces of information provide numerous static keys by which to correlate databases. As we have seen, names can be difficult to work with because of the risk of misspelling, variation in presentation, truncation, and the like. Credit card numbers make excellent keys because they can be presented in only one way and there is even a trivial check digit to ensure that the card number is a valid sequence.

Because of problems with stolen credit card numbers, there has been some consideration of disposable credit cards: cards you could use for one transaction or for a fixed short period of time. That way, if a card number is stolen or intercepted, it could not be reused. Furthermore, having multiple card numbers limits the ability to use a credit card number as a key to compromise privacy.

Payment Schemes

The other way to make web payments is with an online payment scheme, such as PayPal (which is now a subsidiary of the eBay auction site). You pay PayPal a sum of money and you receive an account number and a PIN. You can then log in to the PayPal central site, give an e-mail address and amount to be paid, and PayPal transfers that amount. Because it is not regulated under the same banking laws as credit cards, PayPal offers less consumer protection than does a credit card. However, the privacy advantage is that the user's credit card or financial details are known only to PayPal, thus reducing the risk of their being stolen. Similar schemes use cell phones.

Site and Portal Registrations

Registering to use a site is now common. Often the registration is free; you just choose a user ID and password. Newspapers and web portals (such as Yahoo or MSN) are especially fond of this technique. The explanation they give sounds soothing: They will enhance your browsing experience (whatever *that* means) and be able to offer content to

people throughout the world. In reality, the sites want to obtain customer demographics that they can then sell to marketers or show to advertisers to warrant their advertising.
People have trouble remembering numerous IDs so they tend to default to simple ones, often variations on their names. And because people have trouble remembering IDs, the sites are making it easier: Many now ask you to use your e-mail address as your ID. The problem with using the same ID at many sites is that it now becomes a database key on which previously separate databases from different sites can be merged. Even worse, because the ID or e-mail address is often closely related to the individual's real name, this link also connects a person's identity with the other collected data. So now, a data aggregator can infer that V. Putin browsed the *New York Times* looking for articles on vodka and longevity and then bought 200 shares of stock in a Russian distillery.

You can, of course, try to remember many different IDs. Or you can choose a disposable persona, register for a free e-mail account under a name like xxxyyy, and never use the account for anything except these mandatory free registrations. And it often seems that when there is a need, there arises a service. See www.bugmenot.com for a service that will supply a random anonymous ID and password for sites that require a registration.

Whose Page Is This?

The reason for registrations has little to do with the newspaper or the portal; it has to do with advertisers, the people who pay so the web content can be provided. The web offers much more detailed tracking possibilities than other media. If you see a billboard for a candy bar in the morning and that same advertisement remains in your mind until lunch time and you buy that same candy bar at lunch, the advertiser is very happy: The advertising money has paid off. But the advertiser has no way to know whether you saw an ad (and if so which one).

There are some coarse measures: After an ad campaign if sales go up, the campaign probably had some effect. But advertisers would really like a closer cause-and-effect relationship. Then the web arrived.

Third-Party Ads

You log in to Yahoo Sports and you might see advertisements for mortgages, banking, auto loans, maybe some sports magazines or a cable television offer, and a fast food chain. You click one of the links and you either go directly to a "buy here now" form or you get to print a special coupon worth something on your purchase in person. Web advertising is much more connected to the purchaser: You see the ad, you click on it, and they know the ad did its job by attracting your attention. (With a highway billboard they never know if you watch it or traffic.) When you click through and buy, the ad has really paid off. When you click through and print a coupon that you later present, a tracking number on the coupon lets them connect to advertising on a particular web site. From the advertiser's point of view, the immediate feedback is great.

But each of these activities can be tracked and connected. Is it anyone's business that you like basketball and are looking into a second mortgage? Remember that from your having logged in to the portal site, they already have an identity that may link to your actual name.

Contests and Offers

We cannot resist anything free. We will sign up for a chance to win a large prize, even if we have only a minuscule chance of succeeding. Advertisers know that. So contests and special offers are a good chance to get people to divulge private details. Another thing advertisers know is that people are enthusiastic at the moment but enthusiasm and attention wane quickly.

A typical promotion offers you a free month of a service. You just sign up, give a credit card number, which won't be charged until next month, and you get a month's use of the service for free. As soon as you sign up, the credit card number and your name become keys by which to link other data. You came via a web access, so there may be a link history from the forwarding site.

Precautions for Web Surfing

In this section we discuss cookies and web bugs, two technologies that are frequently used to monitor a user's activities without the user's knowledge.

Cookies

Cookies are files of data set by a web site. They are really a cheap way to transfer a storage need from a web site to a user. A portal such as Yahoo allows a user to customize the look of the web page. Sadie wants the news headlines, the weather, and her e-mail, with a bright background; Norman wants stock market results, news about current movies playing in his area, and interesting things that happened on this day in history, displayed on a gentle pastel background. Yahoo could keep all this preference information in its database so that it could easily customize pages it sends to these two users. But Netscape realized that the burden could be shifted to the user. The web protocol is basically stateless, meaning that the browser displays whatever it is given, regardless of anything that has happened previously.

A cookie is a text file stored on the user's computer and passed by the user's browser to the web site when the user goes to that site. Thus, preferences for Sadie or Norman are stored on their own computers and passed back to Yahoo to help Yahoo form and deliver a web page according to Sadie's or Norman's preferences. A cookie contains six fields: name, value, expiration date, path on the server to which it is to be delivered, domain of the server to which it is to be delivered, and whether a secure connection (SSL) is required in order for the cookie to be delivered. A site can set as many cookies as it wants and can store any value (up to 4,096 bytes) it wants. Some sites use cookies to avoid a customer's having to log in on each visit to a site; these cookies contain the user's ID and password. A cookie could contain a credit card number, the customer name and shipping address, the date of the last visit to the site, the number of items purchased or the dollar volume of purchases. Obviously for sensitive information, such as credit card number or even name and address, the site should encrypt or otherwise protect the data in the cookie. It is up to the site what kind of protection it wants to apply to its cookies. The user never knows if or how data are protected.

The path and domain fields protect against one site's being able to access another's cookies. Almost. As we show in the next section, one company can cooperate with another to share the data in its cookies.

Third-Party Cookies

When you visit a site, its server asks your browser to save a cookie. When you visit that site again your browser passes that cookie back. The general flow is from a server to your browser and later back to the place from which the cookie came. A web page can also contain cookies for another organization. Because these cookies are for organizations other than the web page's owner, they are called third-party cookies.

DoubleClick has built a network of over 1,500 web sites delivering content: news, sports, food, finance, travel, and so forth. These companies agree to share data with DoubleClick.

Web servers contain pages with invisible ads from DoubleClick, so whenever that page is loaded, DoubleClick is invoked, receives the full invoking URL (which may also indicate other ads to be loaded), and is allowed to read and set cookies for itself. So, in essence, DoubleClick knows where you have been, where you are going, and what other ads are placed. But because it gets to read and write its cookies, it can record all this information for future use.

Here are some examples of things a third-party cookie can do.

- Count the number of times this browser has viewed a particular web page.
- Track the pages a visitor views, within a site or across different sites.
- Count the number of times a particular ad has appeared.
- Match visits to a site with displays of an ad for that site.
- Match a purchase to an ad a person viewed before making the purchase.
- Record and report search strings from a search engine.

Of course, all these counting and matching activities produce statistics that the cookie's site can also send back to the central site any time the bug is activated. And these collected data are also available to send to any other partners of the cookie.

Let us assume you are going to a personal investing page which, being financed by ads, contains spaces for ads from four stockbrokers. Let us also assume eight possible brokers could fill these four ad slots. When the page is loaded, DoubleClick retrieves its cookie, sees that you have been to that page before, and also sees that you clicked on broker B5 sometime in the past; then DoubleClick will probably engineer it so that B5 is one of the four brokers displayed to you this time. Also assume DoubleClick sees that you have previously looked at ads for very expensive cars and jewelry. Then full-priced brokers, not discount brokerages, are likely to be chosen for the other three slots. DoubleClick says that part of its service is to present ads that are the most likely to be of interest to the customer, which is in everybody's best interest.

But this strategy also lets DoubleClick build a rich dossier of your web surfing habits. If you visit online gambling sites and then visit a money-lending site, DoubleClick knows. If you purchase herbal remedies for high blood pressure and then visit a health insurance site, DoubleClick knows. DoubleClick knows what personal information you have previously supplied on web forms, such as political affiliation, sexual matters, religion, financial or medical status, or identity information. Even without your supplying private data, merely opening a web page for one political party could put you on that party's solicitation list and the other parties' enemies lists. All this activity goes under the general name of online profiling. Each of these pieces of data is available to the individual firm presenting the web page; DoubleClick collects and redistributes these separate data items as a package.

Presumably all browsing is anonymous. But as we have shown previously, login IDs, e-mail addresses, and retained shipping or billing details can all lead to matching a person with this dossier, so it is no longer an unnamed string of cookies. In 1999, DoubleClick bought Abacus, another company maintaining a marketing database. Abacus collects personal shopping data from catalog merchants, so with that acquisition, DoubleClick gained a way to link personal names and addresses that had previously been only patterns of a machine, not a person.

Cookies associate with a machine, not a user. (For older versions of Windows this is true; for Unix and Windows NT, 2000, and XP, cookies are separated by login ID.) If all members of a family share one machine or if a guest borrows the machine, the apparent connections will be specious. The second problem of the logic concerns the correctness of conclusions drawn:

Because the cookies associate actions on a browser, their results are incomplete if a person uses two or more browsers or accounts or machines. As in many other aspects of privacy, when the user does not know what data have been collected, the user cannot know the data's validity.

Web Bugs: Is There an Exterminator?

The preceding discussion of DoubleClick had a passing reference to an invisible image. Such an image is called a clear GIF, 1 x 1 GIF, or web bug. It is an image file 1 pixel by 1 pixel, so it is far too small to detect by normal sight. To the web browser, an image is an image, regardless of size; the browser will ask for a file from the given address.

The distinction between a cookie and a bug is enormous. A cookie is a tracking device, transferred between the user's machine and the server. A web bug is an invisible image that invites or invokes a process. That process can come from any location. A typical advertising web page might have 20 web bugs, inviting 20 other sites to drop images, code, or other bugs onto the user's machine. All this occurs without the user's direct knowledge or certainly control.

Unfortunately, extermination is not so simple as prohibiting images smaller than the eye can see, because many web pages use such images innocently to help align content. Or some specialized visual applications may actually use collections of minute images for a valid purpose. The answer is not to restrict the image but to restrict the collection and dissemination of data.

Spyware

Cookies are tracking objects, little notes that show where a person has been or what a person has done. The only information they can gather is what you give them by entering data or selecting an object on a web page. As we see in the next section, spyware is far more powerfuland potentially dangerous.

Cookies are passive files and, as we have seen, the data they can capture is limited. They cannot, for example, read a computer's registry, peruse an e-mail outbox, or capture the file directory structure. Spyware is active code that can do all these things that cookies cannot, generally anything a program can do because that is what they are: programs.

Spyware is code designed to spy on a user, collecting data (including anything the user types). In this section we describe different types of spyware.

Keystroke Loggers and Spyware

We have previously referred to keystroke loggers, programs that reside in a computer and record every key pressed. Sophisticated loggers discriminate, recording only web sites visited or, even more serious, only the keystrokes entered at a particular web site (for example, the login ID and password to a banking site.)

A keystroke logger is the computer equivalent of a telephone wiretap. It is a program that records every key typed. As you can well imagine, keystroke loggers can seriously compromise privacy by obtaining passwords, bank account numbers, contact names, and web search arguments.

Spyware is the more general term that includes keystroke loggers and also programs that surreptitiously record user activity and system data, although not necessarily at the level of each individual keystroke. A form of spyware, known as adware (to be described shortly) records these data and transmits them to an analysis center to present ads that will be interesting to the user. The objectives of general spyware can extend to identity theft and other criminal activity.

In addition to the privacy impact, keystroke loggers and spyware sometimes adversely affect a computing system. Not always written or tested carefully, spyware can interfere with other legitimate programs. Also, machines infected with spyware often have several different pieces of spyware, which can conflict with each other, causing a serious impact on performance.

Another common characteristic of many kinds of spyware is the difficulty of removing it. For one spyware product, Altnet, removal involves at least twelve steps, including locating files in numerous system folders [CDT03].

Hijackers

Another category of spyware is software that hijacks a program installed for a different purpose. For example, file-sharing software is typically used to share copies of music or movie files. Services such as KaZaa and Morpheus allow users to offer part of their stored files to other users. According to the Center for Democracy in Technology [CDT03], when a user installed KaZaa, a second program, Altnet, was also installed. The documentation for Altnet said it would make available unused computing power on the user's machine to unspecified business partners. The license for Altnet grants Altnet the right to access and use unused computing power and storage. An ABC News program in 2006 [ABC06] reports on taxpayers whose tax returns were found on the Internet after the taxpayers used a file-sharing program.

The privacy issue for a service such as Altnet is that even if a user authorizes use of spare computing power or sharing of files or other resources, there may be no control over access to other sensitive data on the user's computer.

Adware

Adware displays selected ads in pop-up windows or in the main browser window. The ads are selected according to the user's characteristics, which the browser or an added program gathers by monitoring the user's computing use and reporting the information to a home base.

Adware is usually installed as part of another piece of software without notice. Buried in the lengthy user's license of the other software is reference to "software x and its extension," so the user arguably gives permission for the installation of the adware. File-sharing software is acommon target of adware, but so too are download managers that retrieve large files in several streams at once for faster downloads. And products purporting to be security tools, such as antivirus agents, have been known to harbor adware.

Writers of adware software are paid to get their clients' ads in front of users, which they do with pop-up windows, ads that cover a legitimate ad, or ads that occupy the entire screen surface. More subtly, adware can reorder search engine results so that clients' products get higher placement or replace others' products entirely.

180Solutions is a company that generates pop-up ads in response to sites visited. It distributes software to be installed on a user's computer to generate the pop-ups and collect data to inform 180Solutions of which ads to display. The user may inadvertently install the software as part of another package; in fact, 180Solutions pays a network of 1,000 third parties for each installation of its software on a user's computer. Some of those third parties may have acted aggressively and installed the software by exploiting a vulnerability on the user's computer [SAN05]. A similar product is Gator or Claria or GAIN from the Gator Corporation. Gator claims its software is installed on some 35 million computers. The software is designed to pop up advertising at times when the user might be receptive, for example, popping up a car rental ad right after the user closed an online travel web site page.

There is little analysis of what these applications collect. Rumors have it that they search for name, address, and other personal identification information. The software privacy notice from Gain's web site lists many kinds of information it may collect:

Gain Privacy Statement

1. WHAT INFORMATION DOES GAIN COLLECT?

GAIN Is Designed to Collect and Use Only Anonymous Information. GAIN collects and stores on its servers anonymous information about your web surfing and computer use. This includes information on how you use the web (including the URL addresses of the web pages you view and how long you view them), non-personally identifiable information you provide on web pages and forms (including the Internet searches you conduct), your response to online ads, what software is on the computer (but no information about the usage or data files associated with the software), system settings, and information about how you use GAIN-Supported Software. For more information about the data we collect, click: www.gainpublishing.com/rdr/73/datause.html.

"What software is on the computer" and "system settings" seem to cover a wide range of possibilities.

Drive-By Installation

Few users will voluntarily install malicious code on their machines. Authors of spyware have overcome suspicions to get the user to install their software. We have already discussed dual-purpose software and software installed as part of another installation.

A drive-by installation is a means of tricking a user into installing software. We are familiar with the pop-up installation box for a new piece of software, saying "your browser is about to install *x* from *y*. Do you accept this installation? Yes / No." In the drive-by installation, a front piece of the software has already been downloaded as part of the web page. The front piece may paste a different image over the installation box, it may intercept the results from the yes / no boxes and convert them to yes, or it may paste a small image over the installation box obliterating "*x* from *y*" and replace it with "an important security update from *your browsermanufacturer*." The point is to perform the installation by concealing from the user the real code being installed.

Shopping on the Internet

The web offers the best prices because many merchants compete for your business, right? Not necessarily so. And spyware is partly to blame.

Consider two cases: You own a store selling hardware. One of your customers, Viva, is extremely faithful: She has come to you for years; she wouldn't think of going anywhere else. Viva is also quite well off; she regularly buys expensive items and tends to buy quickly. Joan is a new customer. You know she has been to other hardware stores but so far she hasn't bought much from you. Joan is struggling with a large family, large mortgage, and small savings. Both come in on the same day to buy a hammer, which you normally sell for $20. What price do you offer each? Many people say you should give Viva a good price because of her loyalty. Others say her loyalty gives you room to make some profit. And she can certainly afford it. As for Joan, is she likely to become a steady customer? If she has been to other places, does she shop by price for everything? If you win her with good prices, might you convince her to stay? Or come back another time? Hardware stores do not go through this analysis: a $20 hammer is priced at $20 today, tomorrow, and next week, for everyone, unless it's on sale.

Not true online. Remember, online you do not see the price on the shelf; you see only the price quoted to you on the page showing the hammer. Unless someone sitting at a nearby computer is looking at the same hammers, you wouldn't know if someone else got a price offer other than $20.

According to a study done by Turow et al. [TUR05] of the Annenberg Public Policy Center of the University of Pennsylvania School of Communications, price discrimination occurs and is likely to expand as merchants gather more information about us. The most widely cited example is Amazon.com, which priced a DVD at 30 percent, 35 percent, and 40 percent off list price concurrently to different customers. One customer reported deleting his Amazon.com tracking cookie and having the price on the web site *drop* from $26.00 to

$22.00 because the web site thought he was a new customer instead of a returning customer. Apparently customer loyalty is worth less than finding a new target.

The Turow study involved an interview of 1,500 U.S. adults on web pricing and buying issues.

Among the significant findings were these:

 53 percent correctly thought most online merchants did not give them the right to correct incorrect information obtained about them.

 50 percent correctly thought most online merchants did not give them the chance to erase information collected about them.

 38 percent correctly thought it was legal for an online merchant to charge different people different prices at the same time of day.

 36 percent correctly thought it was legal for a supermarket to sell buying habit data.

 32 percent correctly thought a price-shopping travel service such as Orbitz or Expedia did not have to present the lowest price found as one of the choices for a trip.

 29 percent correctly thought a video store was not forbidden to sell information on what videos a customer has rented.

A fair market occurs when seller and buyer have complete knowledge: If both can see and agree with the basis for a decision, each knows the other party is playing fairly. The Internet has few rules, however. Loss of Internet privacy causes the balance of knowledge power to shift strongly to the merchant's side.

10.6. E-Mail Security

E-mail is exposed as it travels through the web. Furthermore, the privacy of an e-mail message can be compromised on the sender's or receiver's side, without warning. Consider the differences between e-mail and regular letters. Regular mail is handled by a postal system that by law is forbidden to look inside letters. A letter is sealed inside an opaque envelope, making it almost impossible for an outsider to see the contents. The physical envelope is tamper-evident, meaning it shows if someone opens it. A sender can drop a letter in any mailbox, making the sending of a letter anonymous. For these reasons, we have a high expectation of privacy with regular mail. (At certain times in history, for example during a war or under an autocratic ruler, mail was inspected regularly. In those cases, citizens knew their mail was not private.)

In this section we look at the reality of privacy for e-mail.

Where Does E-Mail Go, and Who Can Access It?

We cover e-mail and privacy-enhanced e-mail in Chapter 7. In this section we look only at the mechanics of transmitting e-mail with attention to privacy impacts. E-mail is conceptually a point-to-point communication. If Janet sends e-mail to Scott, Janet's computer establishes a virtual connection with Scott, the computers synchronize, and the message is transferred by SMTP (simple mail transfer protocol). However, Scott may not be online at the moment Janet wants to send her message, so the message to Scott is stored for him on a server (called a POP or post office protocol server). The next time Scott is online, he downloads that message from the server. In the point-to-point communication, Janet's message is private; in the server version, it is potentially exposed while sitting on the server.

Janet may be part of a large organization (such as a company or university), so she may not have a direct outbound connection herself; instead, her mail is routed through a server, too, where the message's privacy is in jeopardy. A further complication is aliases and forwarding agents that add more midpoints to this description. Also, Internet routing can make many hops out of a conceptual point-to-point model.

What started as a simple case can easily have at least five parties: (a) Janet and her computer, (b) Janet's organization's SMTP server, (c) Janet's organization's ISP, (d) Scott's POP server, and (e) Scott and his computer. For now, we are most interested in the three middle parties: (b), (c), and (d). Any of them can log the fact that it was sent or can even keep a copy of the message.

Interception of E-mail

E-mail is subject to the same interception risks as other web traffic: While in transit on the Internet, e-mail is open for any interceptor to read. In Chapter 7 we described techniques for encrypting e-mail. In particular, S/MIME and PGP are two widely used e-mail protection programs. S/MIME and PGP are available for popular mail handlers such as Outlook, Outlook Express, Eudora, Apple Mail, Netscape Communicator, and others. These products protect e-mail from the client's workstation through mail agents, across the Internet, and to the recipient's workstation. That protection is considered end-to-end, meaning from the sender to the recipient. Encrypted e-mail protection is subject to the strength of the encryption and the security of the encryption protocol.

A virtual private network, also described in Chapter 7, can protect data on the connection between a client's workstation and some edge point, usually a router or firewall, at the organization to which the client belongs. For a corporate or government employee or a university student, communication is protected just up to the edge of the corporate, government, or university network. Thus, with a virtual private network, e-mail is protected only from the sender to the sender's office, not even up to the sender's mail agent, and certainly not to the recipient.

Some organizations routinely copy all e-mail sent from their computers. Purposes for these copies include using the e-mail as evidence in legal affairs and monitoring the e-mail for inappropriate content.

Monitoring E-Mail

Companies and government agencies can legitimately monitor their employees' e-mail use. Schools and libraries can monitor the computer use of patrons. Network administrators and ISPs can monitor traffic for normal business purposes, such as to measure traffic patterns or to detect spam. Organizations must advise users of this monitoring, but the notice can be a small notice in a personnel handbook or in the fine print of a service contract. Organizations can use the monitoring data for any legal purpose, for example, to investigate leaks, to manage resources, or to track user behavior.

Network users should have no expectation of privacy in their e-mail or general computer use.

Anonymous E-mail and Remailers

We have described anonymity in other settings; there are reasons for anonymous e-mail, as well.

As with telephone calls, employees sending tips or complaining to management may want to do so anonymously. For example, consumers may want to contact commercial Establishments to register a complaint, inquire about products, or request information without getting on a mailing list or becoming a target for spam. Or people beginning a personal relationship may want to pass along some information without giving away their identities.

These are some of the reasons people want to be able to send anonymous e-mail. Free e-mail addresses are readily available from Yahoo, Microsoft Hotmail, and many other places. People can treat these addresses as disposable: Obtain one, use it for a while, and discard it (by ceasing to use it).

Simple Remailers

Another solution is a remailer. A remailer is a trusted third party to whom you send an e-mail message and indicate to whom you want it sent. The remailer strips off the sender's name and address, assigns an anonymous pseudonym as the sender, and forwards the message to the designated recipient. The third party keeps a record of the correspondence between pseudonyms and real names and addresses. If the recipient replies, the remailer removes the recipient's name and address, applies a different anonymous pseudonym, and forwards the message to the original sender. Such a remailer knows both sender and receiver, so it provides pseudonymity, not anonymity.

Mixmaster Remailers

A more complicated design is needed to overcome the problem that the remailer knows who are the real sender and receiver. This approach is similar to the concept of onion routing described in Chapter 7. The basic tool is a set of cooperating hosts that agree to forward mail. Each host publishes its own public encryption key.

The sender creates a message and selects several of the cooperating hosts. The sender designates the ultimate recipient (call it node *n*) and places a destination note with the content. The sender then chooses one of the cooperating hosts (call it node *n*-1), encrypts the package with the public key of node (*n*-1) and places a destination note showing node (*n*) with the encrypted package. The sender chooses another node (*n*-2), encrypts, and adds a destination note for (*n*-1). The sender thus builds a multilayered package, with the message inside; each layer adds another layer of encryption and another destination.

Each remailer node knows only from where it received the package and to whom to send it next. Only the first remailer knows the true recipient, and only the last remailer knows the final recipient. Therefore, no remailer can compromise the relationship between sender and receiver.

Although this strategy is sound, the overhead involved indicates that this approach should be used only when anonymity is very important.

Spoofing and Spamming

E-mail has very little authenticity protection. Nothing in the SMTP protocol checks to verify that the listed sender (the From: address) is accurate or even legitimate. Spoofing the source address of an e-mail message is not difficult. This limitation facilitates the sending of spam because it is impossible to trace the real sender of a spam message. Sometimes the apparent sender will be someone who knows the recipient or someone on a common mailing list with the recipient. Spoofing such an apparent sender is intended to lend credibility to the spam message.

Phishing is a form of spam in which the sender attempts to convince the sender to reveal personal data, such as banking details. The sender enhances the credibility of a phishing message by spoofing a convincing source address, or using a deceptive domain name These kinds of e-mail messages entice gullible users to reveal sensitive personal data. Because of limited regulation of the Internet, very little can be done to control these threats. User awareness is the best defense.

Summary

E-mail is exposed from sender to receiver, and there are numerous points for interception. Unless the e-mail is encrypted, there is little to prevent its access along the way. For businesses, governments, schools, and other organizations, network administrators and managers may read any e-mail messages sent.

10.7. Impacts on Emerging Technologies

In this section we look at the privacy implications of three emerging technologies. Nothing inherent in the technologies affects privacy, but the applications for the technologies have risk. The first is a broadcast technology that can be used for tracking objects or people.

Second is a group of technologies to facilitate elections. The final technology is a new method for voice-grade telephone calls.

RFID

Radio frequency identification or RFID is a technology that uses small, low-power wireless radio transmitters called RFID tags. The devices can be as small as a grain of sand and they cost just pennies apiece. Tags are tuned to a particular frequency and each has a unique ID number. When a tag receives its signal, it sends its ID number signal in response. Many tags have no power supply of their own and receive their power to send a signal from the very act of receiving a signal. Thus, these devices are passive until they receive a signal from an interrogating reader.

The distance at which they can receive and broadcast a receivable signal varies from roughly five centimeters at the least powerful end to several meters at the most powerful end. Some transmitters have their own power supply (battery) and can transmit over an even greater distance. Probably as receivers get better, the reception distance will increase.
Current uses of RFID tags include

- toll plaza payments
- transit system fare cards
- stock or inventory labels
- passports and identity cards

Two applications of RFID tags are of special interest from a privacy standpoint, as we show in the next sections.

Consumer Products

Assume you have bought a new shirt. If the manufacturer has embedded an RFID tag in the shirt, the tag will assist the merchant in processing your sale, just as barcodes do today. But barcodes on merchandise identify only a manufacturer's product, such as an L.L Bean green plaid flannel shirt, size M. The RFID tag can identify not only the product but also the batch and shipment; that is, the tag's value designates a specific shirt. The unique ID in the shirt helps the merchant keep track of stock, knowing that this shirt was from a shipment that has been on the sales display for 90 days. The tag also lets the manufacturer determine precisely when and where it was produced, which could be important if you returned the shirt because of a defect.

As you walk down the street, your shirt will respond to any receiver within range that broadcasts its signal. With low-power tags using today's technology, you would have to pass quite close to the receiver for it to obtain your signal, a few centimeters at most. Some scientists think this reception will be extended in the future, and others think the technology exists today for high-power readers to pick up the signal a meter away. If the distance is a few centimeters, you would almost have to brush up against the receiver in order for it to track the tag in your shirt; at a meter, someone could have a reader at the edge of the sidewalk as you walk past.

Your shirt, shoes, pen, wallet, credit card, mobile phone, media player, and candy bar wrapper might each have an RFID tag. Any one of these would allow surreptitious tracking; the others provide redundancy. Tracking scenarios once found only in science fiction are now close to reality.

One privacy interest is the accumulation of readings as you go about your business. If a city were fitted with readers on every street corner, it would be possible to assemble a complete profile of your meanderings; timestamps would show when you stopped for a while between two receivers. Thus, it is imaginable and probably feasible to develop a system that could track all your movements.

The other privacy concern is what these tags say about you: One tag from an employee ID might reveal for whom you work, another from a medicine bottle might disclose a medical condition, and still another from an expensive key fob might suggest your finances. Currently you can conceal objects like your employee ID in your pocket; with RFID technology you may have to be more careful to block invisible radio signals. RFID Tags for Individuals

Tagging a shirt is a matter of chance. If you buy the right kind of shirt you will have a tag that lets you be monitored. But if you buy an untagged shirt, or find and cut out the tag, or disable the tag, or decide not to wear a shirt, you cannot be tracked.

Some people choose to be identifiable, regardless of what they wear. Some people with an unusual medical condition have already had an RFID tag permanently implanted in their arm.

This way, even if a patient is brought unconscious to a hospital, the doctors can scan for a tag, receive the person's unique number, and look up the person's medical record by that number. A similar approach is being used to permit animals to cross quarantine borders or to uniquely identify animals such as valuable racehorses.

In these examples, individuals voluntarily allow the tags to be implanted. But remember that once the tags are implanted, they will respond to any appropriate receiver, so our example of walking down the street still holds.

RFID advocates hasten to point out that the technology does not currently permit reading the simplest tags at a distance and that receivers are so expensive that it would be prohibitive to build a network capable of tracking someone's every movement. As we point out in cryptography and reiterate in software, you should not base your security just on what is technically possible or economiclly feasible today.

Security and Privacy Issues

We have already described two of RFID's major privacy issues: the ability to track individuals wherever they go and the ability to discern sensitive data about people. The related issue is one of correctness. The reading sensor may malfunction or the software processing IDs may fail; both cases lead to mistaken identity. How do you challenge that you were not someplace when the receiver shows you were? Another possible failure is forgery of an RFID tag. Here again the sensor would pick up a reading of a tag associated with you. The only way you could prove you were not near the sensor is to have an alibi, supporting where you actually were.

Juels [JUE05] presents several privacy-restoring approaches to RFID use. Among the ideas he proposes are blasting (disabling a tag), blocking (shielding a tag to block its access by a reader), reprogramming (so a tag emits a different number), and encrypting (so the output is selectively available).

RFID technology is still very young, but its use is growing rapidly. As with similarly sensitive technologies, protecting privacy will be easier before the uses proliferate.

Electronic Voting

Voting is another area in which privacy is important. We want votes to be private, but at the same time we want a way to demonstrate that all collected votes are authentic. With careful control of paper ballots, we can largely satisfy both those requirements, but the efficiency of such systems is poor. We would like to use computerized voting systems to improve efficiency without sacrificing privacy or accuracy. In this section we consider the privacy aspects of computerized voting.

Computer Voting

Citizens want to vote anonymously. Although anonymity is easy to achieve with paper ballots (ignoring the possibility of fingerprint tracing or secretly marked ballots) and fairly easy to achieve with machines (assuming usage protocols preclude associating the order in which people voted with a voting log from the machine), it is more difficult with computers.

Properties essential to a fair election were enumerated by Shamos [SHA93].

- Each voter's choices must be kept secret.
- Each voter may vote only once and only for allowed offices.
- The voting system must be tamperproof, and the election officials must be prevented from allowing it to be tampered with.
- All votes must be reported accurately.
- The voting system must be available for use throughout the election period.
- An audit trail must be kept to detect irregularities in voting, but without disclosing how any individual voted.

These conditions are challenging in ordinary paper- and machine-based elections; they are even harder to meet in computer-based elections. Privacy of a vote is essential; in some repressive countries, voting for the wrong candidate can be fatal. But public confidence in the validity of the outcome is critical, so there is a similarly strong need to be able to validate the accuracy of the collection and reporting of votes. These two requirements are close to contradictory.

DeMillo and Merritt [DEM83] devised protocols for computerized voting. Hoffman [HOF00] studied the use of computers at polling places to implement casting of votes. Rubin [RUB00] concludes: "Given the current state of insecurity of hosts and the vulnerability of the Internet to manipulation and denial-of-service attacks, there is no way that a public election of any significance involving remote electronic voting could be carried out securely." But Tony Blair, British prime minister, announced in July 2002 that in the British 2006 general election, citizens would vote in any of four ways: online (by Internet) from a work or home location, by mail, by touch-tone telephone, or at polling places through online terminals. All the counts of the elections would be done electronically. In 2002, Brazil used a computer network to automate voting in its national election (in which voting was mandatory).

Privacy and the Process

Counting ballots is only one step in the election process; building and maintaining the list of eligible voters, recording who has voted (and keeping one person from voting twice), supporting absentee ballots, assisting voters at the wrong polling place, and transmitting election results to election headquarters are other important steps. Each of these has obvious privacy implications. For example, in some political cultures, it may be desirable to maintain privacy of who has voted (to prevent retaliation against people who did not vote for a powerful candidate). Similarly, as we know from other security studies, it is important to protect the privacy of votes in transmission to election headquarters.

The Computer Science and Telecommunications Board of the National Academy of Science [NRC05] studied electronic voting. Its purpose was to raise questions to ensure they are considered in the debate about electronic voting. The privacy questions they asked concerned individual privacy in voter registration, the privacy of individual voters, and public confidence in the process.

Rubin [RUB02], Schneier [SCH04b], and Bennet [BEN04], among others, have studied electronic voting. Rubin raises the question of Internet voting, which has an obvious benefit of easy access for a segment of the population (and a corresponding weakness of more difficult access for people who do not have Internet access or who are not comfortable with computing technology). But given the very weak privacy protections we have already seen for the Internet, the privacy aspects of such a proposal require a careful look.

VoIP and Skype

Privacy aspects of traditional telephony were fairly well understood: Telephone companies were regulated monopolies that needed to preserve the confidentiality of their clients' communications. Exceptions occur under statutorially defined circumstances for law enforcement purposes and in emergencies. Furthermore, the technology was relatively resistant to eavesdropping, with the greatest exposure at the endpoints.

Cellular telephony and Internet-based phone service have significantly changed that situation. Voice over IP (VoIP) is a protocol for transmission of voice-grade telephone traffic over the Internet. The major VoIP carrier is Skype. (VoIP rhymes with "boy" plus P, and Skype rhymes with "hype.") You use a telephone handset or microphone and speaker connected to your computer. To call from London to Rio, for example, you would invoke the VoIP application, giving it the telephone number in Rio. A local office in Rio would call the number in Rio and patch that call to its Internet servers. (The process is even easier if both endpoints use VoIP.)

The advantage of VoIP is cost: For people who already have a fixed-price broadband Internet connection, adding VoIP need only cover the costs of the local connection on the remote end and a fee for software. But as we have seen in other Internet applications, privacy is sacrificed. Even if the voice traffic is solidly encrypted, the source and destination of the phone call will be somewhat exposed through packet headers.

Conclusions on Emerging Technologies

Each of these areas is a technology in its very early stages. The promise for each is great. Privacy issues will not be considered unless they are raised forcefully. Our experience with security has shown that if we consider security early in a system's life, wider options are available for security. The other thing experience has repeatedly shown is that adding security to a nearly complete system is between very difficult and impossible. For both reasons, privacy and security analysis should occur along with the technology and application development.

For all three technologies, however, there seems to be financial pressure to create devices and deal with use issues later. This is exactly the wrong way to go about designing any system. Unfortunately, people seem to be starting with the technology and working backward to systems that would use that technology. The approach should be the other way around:

Specify the necessary requirements, including privacy considerations, and develop a system to implement those requirements reliably.

Legal and Ethical Issues in Computer Security

In this chapter we study human controls applicable to computer security: the legal system and ethics. The legal system has adapted quite well to computer technology by reusing some old forms of legal protection (copyrights and patents) and creating laws where no adequate ones existed (malicious access). Still, the courts are not a perfect form of protection for computer resources, for two reasons. First, the courts tend to be reactive instead of proactive. That is, we have to wait for a transgression to occur and then adjudicate it, rather than try to prevent it in the first place. Second, fixing a problem through the courts can be time consuming (sometimes taking years) and expensive; the latter characteristic prevents all but the wealthy from addressing most security issues.

On the other hand, ethics has not had to change, because ethics is more situational and personal than the law. For example, the privacy of personal information is becoming an important part of computer security. And although technically this issue is just an aspect of confidentiality, practically it has a long history in both law and ethics. The purpose of this chapter is to round out our study of protection for computing systems by understanding the context in which security is assessed and applied.

Not always are conflicts resolved pleasantly. Some people will think that they have been treated unfairly, and some people do indeed act unfairly. In some countries, a citizen reacts to a wrongful act by going to court. The courts are seen as the ultimate arbiters and enforcers of fairness. But, as most lawyers will tell you, the courts' definition of *fair* may not coincide with yours. Even if you could be sure the courts would side with you, a legal battle can be emotionally draining. Our purpose in this section is not only to understand how the legal system helps protect computer security but also to know how and when to use the legal system wisely.

Law and computer security are related in several ways. First, international, national, state, and city laws can affect privacy and secrecy. These statutes often apply to the rights of individuals to keep personal matters private. Second, laws regulate the use, development, and ownership of data and programs. Patents, copyrights, and trade secrets are legal devices to protect the rights of developers and owners of programs and data. Similarly, one aspect of computer security is controlling access to programs and data; that access control is supported by these mechanisms of the law. Third, laws affect actions that can be taken to protect the secrecy, integrity, and availability of computer information and service. These basic concerns in computer security are both strengthened and constrained by applicable laws. Thus, legal means interact with other controls to establish computer security.

However, the law does not always provide an adequate control. When computer systems are concerned, the law is slowly evolving because the issues are similar to but not the same as those for property rights. Computers are new, compared to houses, land, horses, or money.

As a consequence, the place of computer systems in law is not yet firmly established. As statutes are written and cases decided, the roles of computers and the people, data, and processes involved are becoming more defined in the law. However, laws do not yet address all improper acts committed with computers. Finally, some judges, lawyers, and police officers do not understand computing, so they cannot determine how computing relates to other, more established, parts of the law.

The laws dealing with computer security affect programmers, designers, users, and maintainers of computing systems and computerized data banks. These laws protect, but they also regulate the behavior of people who use computers. Furthermore, computer professionals are among the best-qualified advocates for changing old laws and creating new ones regarding computers. Before recommending change, however, professionals must understand the current state of computers and the law. Therefore, we have three motivations for studying the legal section of this chapter:

- to know what protection the law provides for computers and data
- to appreciate laws that protect the rights of others with respect to computers, programs, and data
- to understand existing laws as a basis for recommending new laws to protect computers, data, and people

The next few sections address the following aspects of protection of the security of computers.

- *Protecting computing systems against criminals.* Computer criminals violate the principles of confidentiality, integrity, and availability for computer systems. Preventing the violation is better than prosecuting it after the fact. However, if other controls fail, legal action may be necessary. In this section we study several representative laws to determine what acts are punishable under the law.
- *Protecting code and data.* Copyrights, patents, and trade secrets are all forms of legal protection that can be applied to programs and, sometimes, data. However, we must understand the fundamental differences between the kind of protection these three provide and the methods of obtaining that protection.
- *Protecting programmers' and employers' rights.* The law protects both programmers and people who employ programmers. Generally, programmers have only limited legal rights to access programs they have written while employed. This section contains a survey of the rights of employees and employers regarding programs written for pay.
- *Protecting users of programs.* When you buy a program, you expect it to work properly. If it doesn't, you want the legal system to protect your rights as a consumer.

This section surveys the legal recourse you have to address faulty programs. Computer law is complex and emerging rather rapidly as it tries to keep up with the rapid technological advances in and enabled by computing. We present the fundamentals in this book not in their full detail as you would expect by someone with a law degree, but as a situational analysis to heighten the awareness of those who are not lawyers but who must deal with the law's implications. You should consult a lawyer who understands and specializes in computer law in order to apply the material of this section to any specific case. And, as most lawyers will advise, ensuring legal protection by doing things correctly from the beginning is far easier and cheaper than hiring a lawyer to sort out a web of conflict after things have gone wrong.

Protecting programs and data-

Suppose Martha wrote a computer program to play a video game. She invited some friends over to play the game and gave them copies so that they could play at home. Steve took a copy and rewrote parts of Martha's program to improve the quality of the screen display. After Steve shared the changes with her, Martha incorporated them into her program. Now Martha's friends have convinced her that the program is good enough to sell, so she wants to advertise and offer the game for sale by mail. She wants to know what legal protection she can apply to protect her software.

Copyrights, patents, and trade secrets are legal devices that can protect computers, programs, and data. However, in some cases, precise steps must be taken to protect the work before anyone else is allowed access to it. In this section, we explain how each of these forms of protection was originally designed to be used and how each is currently used in computing. We focus primarily on U.S. law, to provide examples of intent and consequence. Readers from other countries or doing business in other countries should consult lawyers in those countries to determine the specific differences and similarities.

Copyrights

In the United States, the basis of copyright protection is presented in the U.S. Constitution. The body of legislation supporting constitutional provisions contains laws that elaborate on or expand the constitutional protections. Relevant statutes include the U.S. copyright law of 1978, which was updated in 1998 as the Digital Millennium Copyright Act (DMCA) specifically to deal with computers and other electronic media such as digital video and music. The 1998 changes brought U.S. copyright law into general conformance with the World Intellectual Property Organization treaty of 1996, an international copyright standard to which 95 countries adhere.

Copyrights are designed to protect the expression of ideas. Thus, a copyright applies to a creative work, such as a story, photograph, song, or pencil sketch. The right to copy an *expression* of an idea is protected by a copyright. Ideas themselves, the law alleges, are free; anyone with a bright mind can think up anything anyone else can, at least in theory. The intention of a copyright is to allow regular and free exchange of ideas.

The author of a book translates ideas into words on paper. The paper embodies the expression of those ideas and is the author's livelihood. That is, an author hopes to earn a living by presenting ideas in such an appealing manner that others will pay to read them. (The same protection applies to pieces of music, plays, films, and works of art, each of which is a personal expression of ideas.) The law protects an individual's right to earn a living, while recognizing that exchanging ideas supports the intellectual growth of society. The copyright says that a particular *way* of expressing an idea belongs to the author. For example, in music, there may be two or three copyrights related to a single creation: A composer can copyright a song, an arranger can copyright an arrangement of that song, and an artist can copyright a specific performance of that arrangement of that song. The price you pay for a ticket to a concert includes compensation for all three creative expressions.

Copyright gives the author the *exclusive* right to make copies of the expression and sell them to the public. That is, only the author (or booksellers or others working as the author's agents) can sell copies of the author's book.

Definition of Intellectual Property

The U.S. copyright law (§102) states that a copyright can be registered for "original works of authorship fixed in any tangible medium of expression,...from which they can be perceived, reproduced, or otherwise communicated, either directly or with the aid of a machine or device." Again, the copyright does *not* cover the *idea* being expressed. "In no case does copyright protection for an original work of authorship extend to any idea." The copyright must apply to an *original* work, and it must be in some *tangible* medium of expression.

Only the originator of the expression is entitled to copyright; if an expression has no determinable originator, copyright cannot be granted. Certain works are considered to be in the public domain, owned by the public, by no one in particular. Works of the U.S. government and many other governments are considered to be in the public domain and therefore not subject to copyright. Works generally known, such as the phrase "top o' the mornin' to ye," or the song "Happy Birthday to You," or a recipe for tuna noodle casserole, are also so widely known that it would be very difficult for someone to trace originality and claim a copyright. Finally, copyright lasts for only a limited period of time, so certain very old works, such as the plays of Shakespeare, are in the public domain, their possibility of copyright having expired.

The copyrighted expression must also be in some tangible medium. A story or art work must be written, printed, painted, recorded (on a physical medium such as a plastic record), stored on a magnetic medium (such as a disk or tape), or fixed in some other way. Furthermore, the purpose of the copyright is to promote distribution of the work; therefore, the work must be distributed, even if a fee is charged for a copy.

Originality of Work

The work being copyrighted must be original to the author. As noted previously, some expressions in the public domain are not subject to copyright. A work can be copyrighted even if it contains some public domain material, as long as there is some originality, too. The author does not even have to identify what is public and what is original.

For example, a music historian could copyright a collection of folksongs even if some are in the public domain. To be subject to copyright, something in or *about* the collection has to be original. The historian might argue that collecting the songs, selecting which ones to include, and putting them in order was the original part. In this case, the copyright law would not protect the folksongs (which would be in the public domain) but would instead protect that specific selection and organization. Someone selling a sheet of paper on which just one of the songs was written would likely not be found to have infringed on the copyright of the historian. Dictionaries can be copyrighted in this way, too; the authors do not claim to own the words, just their expression as a particular dictionary.

Fair Use of Material

The copyright law indicates that the copyrighted object is subject to fair use. A purchaser has the right to use the product in the manner for which it was intended and in a way that does not interfere with the author's rights. Specifically, the law allows "fair use of a copyrighted work, including such use by reproduction in copies… for purposes such as criticism, comment, news reporting, teaching (including multiple copies for classroom use), scholarship or research." The purpose and effect of the use on the potential market for or value of the work affect the decision of what constitutes fair use. For example, fair use allows making a backup copy of copyrighted software you acquired legally: Your backup copy protects your use against system failures but it doesn't affect the author because you have no need for nor do you want use of two copies at once. The copyright law usually upholds the author's right to a fair return for the work, while encouraging others to use the underlying ideas. Unfair use of a copyrighted item is called piracy.

The invention of the photocopier made it more difficult to enforce fair use. You can argue it is fair use to make a copy of the Tuscany section of a travel book to carry with you and throw away during your holiday so you don't have to carry the whole book with you. Today many commercial copy shops will copy a portion sometimes an entire chapter of a book or a single article out of a journal but refuse to copy an entire volume, citing fair use. With photocopiers, the quality of the copy degrades with each copy, as you know if you have ever tried to read a copy of a copy of a copy of a paper.

The copyright law also has the concept of a first sale: after having bought a copyrighted object, the new owner can give away or resell the object. That is, the copyright owner is entitled to control the first sale of the object. This concept works fine for books: An author is compensated when a bookstore sells a book, but the author earns no additional revenue if the book is later resold at a secondhand store.

Requirements for Registering a Copyright

The copyright is easy to obtain, and mistakes in securing a copyright can be corrected. The first step of registration is notice. Any potential user must be made aware that the work is copyrighted. Each copy must be marked with the copyright symbol ©, the word *Copyright*, the year, and the author's name. (At one time, these items were followed by

All rights reserved to preserve the copyright in certain South American countries. Adding the phrase now is unnecessary but harmless.)

The order of the elements can be changed, and either © or *Copyright* can be omitted (but not both). Each copy distributed must be so marked, although the law will forgive failure to mark copies if a reasonable attempt is made to recall and mark any ones distributed without a mark.

The copyright must also be officially filed. In the United States a form is completed and submitted to the Copyright Office, along with a nominal fee and a copy of the work. Actually, the Copyright Office requires only the first 25 and the last 25 pages of the work, to help it justify a claim in the event of a court case. The filing must be done within three months after the first distribution of the work. The law allows filing up to five years late, but no infringements before the time of filing can be prosecuted.

A U.S. copyright now lasts for 70 years beyond the death of the last surviving author or, if the item was copyrighted by a company or organization, for 95 years after the date of publication. The international standard is 50 years after the death of the last author or 50 years from publication.

Copyright Infringement

The holder of the copyright must go to court to prove that someone has infringed on the copyright. The infringement must be substantial, and it must be copying, not independent work. In theory, two people might write identically the same song independently, neither knowing the other. These two people would *both* be entitled to copyright protection for their work. Neither would have infringed on the other, and both would have the right to distribute their work for a fee. Again, copyright is most easily understood for written works of fiction because it is extremely unlikely that two people would express an idea with the same or similar wording.

The independence of nonfiction works is not nearly so clear. Consider, for example, an arithmetic book. Long division can be explained in only so many ways, so two independent books could use similar wording for that explanation. The number of possible alternative examples is limited, so that two authors might independently choose to write the same simple example. However, it is far less likely that two textbook authors would have the same pattern of presentation and the same examples from beginning to end.

Copyrights for Computer Software

The original copyright law envisioned protection for things such as books, songs, and photographs. People can rather easily detect when these items are copied. The separation between public domain and creativity is fairly clear. And the distinction between an idea (feeling, emotion) and its expression is pretty obvious. Works of nonfiction understandably have less leeway for independent expression. Because of programming language constraints and speed and size efficiency, computer programs have less leeway still.

Can a computer program be copyrighted? Yes. The 1976 copyright law was amended in 1980 to include an explicit definition of computer software. However, copyright protection may not be an especially desirable form of protection for computer works. To see why, consider the algorithm used in a given program. The algorithm is the idea, and the statements of the programming language are the expression of the idea. Therefore, protection is allowed for the program statements themselves, but not for the algorithmic concept: copying the code intact is prohibited, but reimplementing the algorithm is permitted. Remember that one purpose of copyright is to promote the dissemination of ideas The algorithm, which is the idea embodied in the computer program, is to be shared.
A second problem with copyright protection for computer works is the requirement that the work be published. A program may be published by distribution of copies of its object code, for example, on a disk. However, if the source code is not distributed, it has not been published.

An alleged infringer cannot have violated a copyright on source code if the source code was never published.

Copyrights for Digital Objects

The Digital Millennium Copyright Act (DMCA) of 1998 clarified some issues of digital objects (such as music files, graphics images, data in a database, and also computer programs), but it left others unclear.

Among the provisions of the DMCA are these:

 Digital objects *can be* subject to copyright.
 It is a crime to circumvent or disable antipiracy functionality built into an object.
 It is a crime to manufacture, sell, or distribute devices that disable antipiracy functionality or that copy digital objects.
 However, these devices can be used (and manufactured, sold, or distributed) for research and educational purposes.
 It is acceptable to make a backup copy of a digital object as a protection against hardware or software failure or to store copies in an archive.
 Libraries can make up to three copies of a digital object for lending to other libraries.

So, a user can make reasonable copies of an object in the normal course of its use and as a protection against system failures. If a system is regularly backed up and so a digital object (such as a software program) is copied onto many backups, that is not a violation of copyright.

The uncertainty comes in deciding what is considered to be a device to counter piracy. A disassembler or decompiler could support piracy or could be used to study and enhance a program. Someone who decompiles an executable program, studies it to infer its method, and then modifies, compiles, and sells the result is misusing the decompiler. But the distinction is hard to enforce, in part because the usage depends on intent and context. It is as if there were a law saying it is legal to sell a knife to cut vegetables but not to harm people. Knives do not know their uses; the users determine intent and context.

Consider a music CD that you buy for the obvious reason: to listen to again and again. You want to listen to the music on your MP3 player, a reasonable fair use. But the CD is copy protected, so you cannot download the music to your computer to transfer it to your MP3 player. You have been prohibited from reasonable fair use. Furthermore, if you try to do anything to circumvent the antipiracy protection, you violate the antipiracy provision, nor can you buy a tool or program that would let you download your own music to your own MP3 player, because such a tool would violate that provision.

Reaction to the Digital Millennium Copyright Act has not been uniformly favorable. (See, for example, [MAN98, EFF06].) Some say it limits computer security research. Worse, others point out it can be used to prevent exactly the free interchange of ideas that copyright was intended to promote. In 2001 a Princeton University professor, Edward Felten, and students presented a paper on cryptanalysis of the digital watermarking techniques used to protect digital music files from being copied. They had been pressured not to present in the preceding April by music industry groups who threatened legal action under the DMCA.

Digital objects are more problematic than paper ones because they can be copied exactly. Unlike fifth-generation photocopies, each digital copy of a digital object can be identical to the original.

Copyright protects the right of a creator to profit from a copy of an object, even if no money changes hands. The Napster situation (see Sidebar 11-1) is an interesting case, closely related to computer data. It clearly distinguishes between an object and a copy of that object.

An emerging principle is that software, like music, is acquired in a style more like rental than purchase. You purchase not a piece of software, but the right to use it. Clarifying this position, the U.S. No Electronic Theft (NET) Act of 1997 makes it a criminal offense to reproduce or distribute copyrighted works, such as software or digital recordings, even without charge.

The area of copyright protection applied to computer works continues to evolve and is subject to much interpretation by the courts. Therefore, it is not certain what aspects of a computer work are subject to copyright. Courts have ruled that a computer menu design can be copyrighted but that "look and feel" (such as the Microsoft Windows user interface) cannot.
But is not the menu design part of the look and feel?

Although copyright protection can be applied to computer works, the copyright concept was conceived before the electronic age, and thus the protection may be less than what we desire. Copyrights do not address all the critical computing system elements that require protection. For example, a programmer might want to protect an algorithm, not the way that algorithm was expressed in a particular programming language. Unfortunately, it may be difficult to obtain copyright protection for an algorithm, at least as copyright law is currently interpreted. Because the copyright laws are evolving, we must also take care when copyrights are used as excuses, as we see in Sidebar 11-2.

Patents

Patents are unlike copyrights in that they protect inventions, tangible objects, or ways to make them, not works of the mind. The distinction between patents and copyrights is that patents were intended to apply to the results of science, technology, and engineering, whereas copyrights were meant to cover works in the arts, literature, and written scholarship.

A patent can protect a "new and useful process, machine, manufacture, or composition of matter." The U.S. law excludes "newly discovered laws of nature... [and] mental processes."

Thus "2+2=4" is not a proper subject for a patent because it is a law of nature. Similarly, that expression is in the public domain and would thus be unsuitable for a copyright. Finally, you can argue that mathematics is purely mental, just ideas. Nobody has ever seen or touched a two two horses, yes, but not just a two. A patent is designed to protect the device or process for *carrying out* an idea, not the idea itself.

Requirement of Novelty

If two composers happen to compose the same song independently at different times, copyright law would allow both of them to have copyright. If two inventors devise the same invention, the patent goes to the person who invented it first, regardless of who first filed the patent. A patent can be valid only for something that is truly novel or unique, so there can be only one patent for a given invention.

An object patented must also be nonobvious. If an invention would be obvious to a person ordinarily skilled in the field, it cannot be patented. The law states that a patent *cannot* be obtained "if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains." For example, a piece of cardboard to be used as a bookmark would not be a likely candidate for a patent because the idea of a piece of cardboard would be obvious to almost any reader.

Procedure for Registering a Patent

One registers a copyright by filing a brief form, marking a copyright notice on the creative work, and distributing the work. The whole process takes less than an hour. To obtain a patent, an inventor must convince the U.S. Patent and Trademark Office that the invention deserves a patent. For a fee, a patent attorney will research the patents already issued for similar inventions. This search accomplishes two things. First, it determines that the invention to be patented has not already been patented (and, presumably, has not been previously invented). Second, the search can help identify similar things that have been patented. These similarities can be useful when describing the unique features of the invention that make it worthy of patent protection. The Patent Office compares an application to those of all other similar patented inventions and decides whether the application covers something truly novel and nonobvious. If the office decides the invention is novel, a patent is granted.

Typically, an inventor writes a patent application listing many claims of originality, from very general to very specific. The Patent Office may disallow some of the more general claims while upholding some of the more specific ones. The patent is valid for all the upheld claims. The patent applicant reveals what is novel about the invention in sufficient detail to allow the Patent Office and the courts to judge novelty; that degree of detail may also tell the world how the invention works, thereby opening the possibility of infringement.

The patent owner uses the patented invention by producing products or by licensing others to produce them. Patented objects are sometimes marked with a patent number to warn others that the technology is patented. The patent holder hopes this warning will prevent others from infringing.

Patent Infringement

A patent holder *must* oppose all infringement. With a copyright, the holder can choose which cases to prosecute, ignoring small infringements and waiting for serious infractions where the infringement is great enough to ensure success in court or to justify the cost of the court case. However, failing to sue a patent infringement even a small one or one the patent holder does not know aboutcan mean losing the patent rights entirely. But, unlike copyright infringement, a patent holder does not have to prove that the infringer copied the invention; a patent infringement occurs even if someone independently invents the same thing, without knowledge of the patented invention.

Every infringement must be prosecuted. Prosecution is expensive and time consuming, but even worse, suing for patent infringement could cause the patent *holder* to lose the patent.

Someone charged with infringement can argue all of the following points as a defense against the charge of infringement.

 This isn't infringement. The alleged infringer will claim that the two inventions are sufficiently different that no infringement occurred.

 The patent is invalid. If a prior infringement was not opposed, the patent rights may no longer be valid.

 The invention is not novel. In this case, the supposed infringer will try to persuade the judge that the Patent Office acted incorrectly in granting a patent and that the invention is nothing worthy of patent.

 The infringer invented the object first. If so, the accused infringer, and not the original patent holder, is entitled to the patent.

The first defense does not damage a patent, although it can limit the novelty of the invention. However, the other three defenses can destroy patent rights. Worse, all four defenses can be used every time a patent holder sues someone for infringement. Finally, obtaining and defending a patent can incur substantial legal fees. Patent protection is most appropriate for large companies with substantial research and development (and legal) staffs.

Applicability of Patents to Computer Objects

The Patent Office has not encouraged patents of computer software. For a long time, computer programs were seen as the representation of an algorithm, and an algorithm was a fact of nature, which is not subject to patent. An early software patent case, *Gottschalk v. Benson*, involved a request to patent a process for converting decimal numbers into binary. The Supreme Court rejected the claim, saying it seemed to attempt to patent an abstract idea, in short, an algorithm. But the underlying algorithm is precisely what most software developers would like to protect.

In 1981, two cases (*Diamond v. Bradley* and *Diamond v. Diehr*) won patents for a process that used computer software, a well-known algorithm, temperature sensors, and a computer to calculate the time to cure rubber seals. The court upheld the right to a patent because the claim was not for the software or the algorithm alone, but for the process that happened to use the software as one of its steps. An unfortunate inference is that using the software without using the other patented steps of the process would not be infringement. Since 1981 the patent law has expanded to include computer software, recognizing that algorithms, like processes and formulas, are inventions. The Patent Office has issued thousands of software patents since these cases. But because of the time and expense involved in obtaining and maintaining a patent, this form of protection may be unacceptable for a small-scale software writer.

Trade Secrets

A trade secret is unlike a patent or copyright in that it must be kept a *secret.* The information has value only as a secret, and an infringer is one who divulges the secret. Once divulged, the information usually cannot be made secret again.

Characteristics of Trade Secrets

A trade secret is information that gives one company a competitive edge over others. For example, the formula for a soft drink is a trade secret, as is a mailing list of customers or information about a product due to be announced in a few months.

The distinguishing characteristic of a trade secret is that it must always be kept secret. Employees and outsiders who have access to the secret must be required not to divulge the secret. The owner must take precautions to protect the secret, such as storing it in a safe, encrypting it in a computer file, or making employees sign a statement that they will not disclose the secret.

If someone obtains a trade secret improperly and profits from it, the owner can recover profits, damages, lost revenues, and legal costs. The court will do whatever it can to return the holder to the same competitive position it had while the information was secret and may award damages to compensate for lost sales. However, trade secret protection evaporates incase of independent discovery. If someone else happens to discover the secret independently, there is no infringement and trade secret rights are gone.

Reverse Engineering

Another way trade secret protection can vanish is by reverse engineering. Suppose a secret is the way to pack tissues in a cardboard box to make one pop up as another is pulled out.

Anyone can cut open the box and study the process. Therefore, the trade secret is easily discovered. In reverse engineering, one studies a finished object to determine how it is manufactured or how it works.

Through reverse engineering someone might discover how a telephone is built; the design of the telephone is obvious from the components and how they are connected. Therefore, a patent is the appropriate way to protect an invention such as a telephone. However, something like a soft drink is not just the combination of its ingredients. Making a soft drink may involve time, temperature, presence of oxygen or other gases, and similar factors that could not be learned from a straight chemical decomposition of the product. The recipe of a soft drink is a closely guarded trade secret. Trade secret protection works best when the secret is not apparent in the product.

Applicability to Computer Objects

Trade secret protection applies very well to computer software. The underlying algorithm of a computer program is novel, but its novelty depends on nobody else's knowing it. Trade secret protection allows distribution of the *result* of a secret (the executable program) while still keeping the program design hidden. Trade secret protection does not cover copying a product (specifically a computer program), so it cannot protect against a pirate who sells copies of someone else's program without permission. However, trade secret protection makes it illegal to steal a secret algorithm and use it in another product.

The difficulty with computer programs is that reverse engineering works. Decompiler and disassembler programs can produce a source version of an executable program. Of course, this source does not contain the descriptive variable names or the comments to explain the code, but it is an accurate version that someone else can study, reuse, or extend.

Difficulty of Enforcement

Trade secret protection is of no help when someone infers a program's design by studying its output or, worse yet, decoding the object code. Both of these are legitimate (that is, legal) activities, and both cause trade secret protection to disappear.

The confidentiality of a trade secret must be ensured with adequate safeguards. If source code is distributed loosely or if the owner fails to impress on people (such as employees) the importance of keeping the secret, any prosecution of infringement will be weakened.

Employment contracts typically include a clause stating that the employee will not divulge any trade secrets received from the company, even after leaving a job. Additional protection, such as marking copies of sensitive documents or controlling access to computer files of secret information, may be necessary to impress people with the importance of secrecy.

Protection for Computer Objects

The previous sections have described three forms of protection: the copyright, patent, and trade secret laws. Each of these provides a different form of protection to sensitive things. In this section we consider different kinds of computer objects and describe which forms of protection are most appropriate for each kind. Table 11-1 shows how these three forms of protection compare in several significant ways.

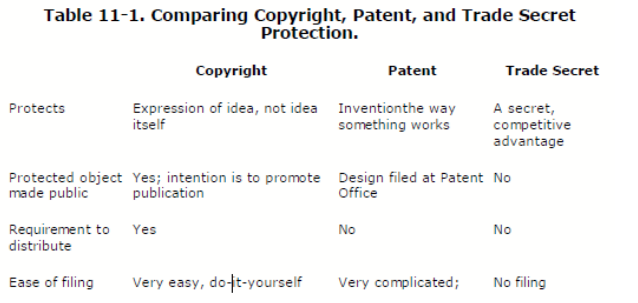
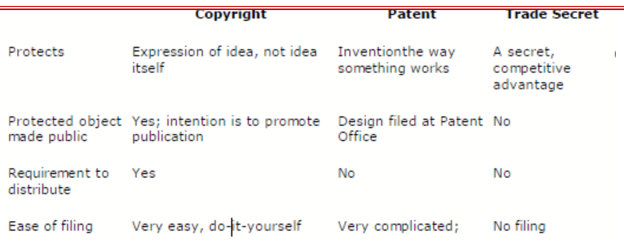


Table 11-1. Comparing Copyright, Patent, and Trade Secret Protection.

	Copyright	Patent	Trade Secret
Protects	Expression of idea, not idea itself	Inventionthe way something works	A secret, competitive advantage
Protected object made public	Yes; intention is to promote publication	Design filed at Patent Office	No
Requirement to distribute	Yes	No	No
Ease of filing	Very easy, do-it-yourself	Very complicated;	No filing

Computer artifacts are new and constantly changing, and they are not yet fully appreciated
by the legal system based on centuries of precedent. Perhaps in a few years the

Table 11-1. Comparing Copyright, Patent, and Trade Secret Protection.



	Copyright	Patent	Trade Secret
Protects	Expression of idea, not idea itself	Inventionthe way something works	A secret, competitive advantage
Protected object made public	Yes; intention is to promote publication	Design filed at Patent Office	No
Requirement to distribute	Yes	No	No
Ease of filing	Very easy, do-it-yourself	Very complicated;	No filing

issue of what protection is most appropriate for a given computer object will be more clear-cut.

Possibly a new form of protection or a new use of an old form will apply specifically to computer objects. For example, the European Union has already enacted model legislation for copyright protection of computer software. However, one of its goals was to promote software that builds on what others have done. Thus, the E.U. specifically exempted a product's interface specification from copyright and permitted others to derive the interface to allow development of new products that could connect via that interface.

Until the law provides protection that truly fits computer goods, here are some guidelines for using the law to protect computer objects.

Protecting Hardware

Hardware, such as chips, disk drives, or floppy disk media, can be patented. The medium itself can be patented, and someone who invents a new process for manufacturing it can obtain a second patent.

Protecting Firmware

The situation is a little less clear with regard to microcode. Certainly, the physical devices on which microcode is stored can be patented. Also, a special-purpose chip that can do only one specific task (such as a floating-point arithmetic accelerator) can probably be patented.

However, the data (instructions, algorithms, microcode, programs) contained in the devices are probably not patentable.

Can they be copyrighted? Are these the expression of an idea in a form that promotes dissemination of the idea? Probably not. And assuming that these devices were copyrighted, what would be the definition of a copy that infringed on the copyright? Worse, would the manufacturer really want to register a copy of the internal algorithm with the Copyright

Office? Copyright protection is probably inappropriate for computer firmware. Trade secret protection seems appropriate for the code embedded in a chip. Given enough time, we can reverse-engineer and infer the code from the behavior of the chip. The behavior of the chip does not reveal what algorithm is used to produce that behavior. The original algorithm may have better (or worse) performance (speed, size, fault tolerance) that would not be obvious from reverse engineering.

For example, Apple Computer is enforcing its right to copyright protection for an operating system embedded in firmware. The courts have affirmed that computer software *is* an appropriate subject for copyright protection and that protection should be no less valid when the software is in a chip rather than in a conventional program.

Protecting Object Code Software

Object code is usually copied so that it can be distributed for profit. The code is a work of creativity, and most people agree that object code distribution is an acceptable medium of publication. Thus, copyright protection seems appropriate.

A copyright application is usually accompanied by a copy of the object being protected. With a book or piece of music (printed or recorded), it is easy to provide a copy. The Copyright Office has not yet decided what is an appropriate medium in which to accept object code. A binary listing of the object code will be taken, but the Copyright Office does so without acknowledging the listing to be acceptable or sufficient. The Office will accept a source code listing. Some people argue that a source code listing is not equivalent to an object code listing, in the same way that a French translation of a novel is different from its original language version. It is not clear *in the courts* that registering a source code version provides copyright protection to object code. However, someone should not be able to take the object code of a system, rearrange the order of the individual routines, and say that the result is a new system. Without the original source listings, it would be very difficult to compare two binary files and determine that one was the functional equivalent of the other simply through rearrangement.

Several court cases will be needed to establish acceptable ways of filing object code for copyright protection. Furthermore, these cases will have to develop legal precedents to define the equivalence of two pieces of computer code.

Protecting Source Code Software

Software developers selling to the mass market are reticent to distribute their source code. The code can be treated as a trade secret, although some lawyers also encourage that it be copyrighted. (These two forms of protection are possibly mutually exclusive, although registering a copyright will not hurt.)

Recall that the Copyright Office requires registering at least the first 25 and the last 25 pages of a written document. These pages are filed with the Library of Congress, where they are available for public inspection. This registration is intended to assist the courts in determining which work was registered for copyright protection. However, because they are available for anybody to see, they are not secret, and copyright registration can expose the secrecy of an ingenious algorithm. A copyright protects the right to distribute copies of the *expression* of an idea, not the idea itself. Therefore, a copyright does not prevent someone from reimplementing an algorithm, expressed through a copyrighted computer program.

As just described, source code may be the most appropriate form in which to register a copyright for a program distributed in object form. It is difficult to register source code with the Copyright Office while still ensuring its secrecy. A long computer program can be rearranged so that the first and last 25 pages do not divulge much of the secret part of a source program. Embedding small errors or identifiable peculiarities in the source (or object) code of a program may be more useful in determining copyright infringement. Again, several court cases must be decided in order to establish procedures for protection of computer programs in either source or object form.

Protecting Documentation

If we think of documentation as a written work of nonfiction (or, perhaps, fiction), copyright protection is effective and appropriate for it. Notice that the documentation is distinct from the program. A program and its documentation must be copyrighted separately. Furthermore, copyright protection of the documentation may win a judgment against someone who illegally copies both a program and its documentation.

In cases where a written law is unclear or is not obviously applicable to a situation, the results of court cases serve to clarify or even extend the words of the law. As more unfair acts involving computer works are perpetrated, lawyers will argue for expanded interpretations of the law. Thus, the meaning and use of the law will continue to evolve through judges' rulings. In a sense, computer technology has advanced much faster than the law has been able to.

Protecting Web Content

Content on the web is media, much the same as a book or photograph, so the most appropriate protection for it is copyright. This copyright would also protect software you write to animate or otherwise affect the display of your web page. And, in theory, if your web page contains malicious code, your copyright covers that, too. As we discussed earlier, a copyrighted work does not have to be exclusively new; it can be a mixture of new work to which you claim copyright and old things to which you do not. You may purchase or use with permission a piece of web art, a widget (such as an applet that shows a spinning globe), or some music. Copyright protects your original works.

Protecting Domain Names and URLs

Domain names, URLs, company names, product names, and commercial symbols are protected by a trademark, which gives exclusive rights of use to the owner of such identifying marks.

Information and law

Source code, object code, and even the "look and feel" of a computer screen are recognizable, if not tangible, objects. The law deals reasonably well, although somewhat belatedly, with these things. But computing is in transition to a new class of object, with new legal protection requirements. Electronic commerce, electronic publishing, electronic voting, electronic banking these are the new challenges to the legal system. In this section we consider some of these new security requirements.

Information as an Object

The shopkeeper used to stock "things" in the store, such as buttons, automobiles, and pounds of sugar. The buyers were customers. When a thing was sold to a customer, the shopkeeper's stock of that thing was reduced by one, and the customer paid for and left with a thing. Sometimes the customer could resell the thing to someone else, for more or less than the customer originally paid.

Other kinds of shops provided services that could be identified as things, for example, a haircut, root canal, or defense for a trial. Some services had a set price (for example, a haircut), although one provider might charge more for that service than another. A "shopkeeper" (hair stylist, dentist, lawyer) essentially sold time. For instance, the price of a haircut generally related to the cost of the stylist's time, and lawyers and accountants charged by the hour for services in which there was no obvious standard item. The value of a service in a free economy was somehow related to its desirability to the buyer and the seller.

For example, the dentist was willing to sell a certain amount of time, reserving the rest of the day for other activities. Like a shopkeeper, once a service provider sold some time or service, it could not be sold again to someone else.

But today we must consider a third category for sale: information. No one would argue against the proposition that information is valuable. Students are tempted to pay others for answers during examinations, and businesses pay for credit reports, client lists, and marketing advice.

But information does not fit the familiar commercial paradigms with which we have dealt for many years. Let us examine why information is different from other commercial things.

Information Is Not Depletable

Unlike tangible things and services, information can be sold again and again without depleting stock or diminishing quality. For example, a credit bureau can sell the same credit report on an individual to an unlimited number of requesting clients. Each client pays for the information in the report. The report may be delivered on some tangible medium, such as paper, but it is the *information*, not the medium, that has the value.

This characteristic separates information from other tangible works, such as books, CDs, or art prints. Each tangible work is a single copy, which can be individually numbered or accounted for. A bookshop can always order more copies of a book if the stock becomes depleted, but it can sell only as many copies as it has.

Information Can Be Replicated

The value of information is what the buyer will pay the seller. But after having bought the information, the buyer can then become a seller and can potentially deprive the original seller of further sales. Because information is not depletable, the buyer can enjoy or use the information and can also sell it many times over, perhaps even making a profit.

Information Has a Minimal Marginal Cost

The marginal cost of an item is the cost to produce another one after having produced some already. If a newspaper sold only one copy on a particular day, that one issue would be prohibitively expensive because it would have to cover the day's cost (salary and benefits) of all the writers, editors, and production staff, as well as a share of the cost of all equipment for its production. These are fixed costs needed to produce a first copy. With this model, the cost of the second and subsequent copies is minuscule, representing basically just the cost of paper and ink to print them. Fortunately, newspapers have very large press runs and daily sales, so the fixed costs are spread evenly across a large number of copies printed. More importantly, publishers have a reasonable idea of how many copies will sell, so they adjust their budgets to make a profit at the expected sales volume, and extra sales simply increase the profit. Also, newspapers budget by the month or quarter or year so that the price of a single issue does not fluctuate based on the number of copies sold of yesterday's edition.

In theory, a purchaser of a copy of a newspaper could print and sell other copies of that copy, although doing so would violate copyright law. Few purchasers do that, for four reasons.

- The newspaper is covered by copyright law.
- The cost of reproduction is too high for the average person to make a profit.
- It is not fair to reproduce the newspaper that way.
- There is usually some quality degradation in making the copy.

Unless the copy is truly equivalent to the original, many people would prefer to buy an authentic issue from the news agent, with clear type, quality photos, actual color, and so forth. The cost of information similarly depends on fixed costs plus costs to reproduce. Typically, the fixed costs are large whereas the cost to reproduce is extremely small, even less than for a newspaper because there is no cost for the raw materials of paper and ink.

However, unlike a newspaper, information is far more feasible for a buyer to resell. A copy of digital information can be perfect, indistinguishable from the original, the same being true for copies of copies of copies of copies.

The Value of Information Is Often Time Dependent

If you knew for certain what the trading price of a share of Microsoft stock would be next week, that information would be extremely valuable because you could make an enormous profit on the stock market. Of course, that price cannot be known today. But suppose you knew that Microsoft was certain to announce something next week that would cause the price to rise or fall. That information would be almost as valuable as knowing the exact price, and it could be known in advance. However, knowing *yesterday's* price for Microsoft stock or knowing that *yesterday* Microsoft announced something that caused the stock price to plummet is almost worthless because it is printed in every major financial newspaper. Thus, the value of information may depend on when you know it.

Information Is Often Transferred Intangibly

A newspaper is a printed artifact. The news agent hands it to a customer, who walks away with it. Both the seller and the buyer realize and acknowledge that something has been acquired. Furthermore, it is evident if the newspaper is seriously damaged; if a serious production flaw appears in the middle, the defect is easy to point out.

But times are changing. Increasingly, information is being delivered as bits across a network instead of being printed on paper. If the bits are visibly flawed (that is, if an error detecting code indicates a transmission error), demonstrating that flaw is easy. However, if the copy of the information is accurate but the underlying information is incorrect, useless, or not as expected, it is difficult to justify a claim that the information is flawed.

Legal Issues Relating to Information

These characteristics of information significantly affect its legal treatment. If we want to understand how information relates to copyright, patent, and trademark laws, we must understand these attributes. We can note first that information has some, limited legal basis for the protection. For example, information can be related to trade secrets, in that information is the stock in trade of the information seller. While the seller has the information, trade secret protection applies naturally to the seller's legitimate ability to profit from information. Thus, the courts recognize that information has value.

However, as shown earlier, a trade secret has value only as long as it remains a secret. For instance, the Coca-Cola Company cannot expect to retain trade secret protection for its formula after it sells that formula. Also, the trade secret is not secure if someone else can derive or infer it.

Other forms of protection are offered by copyrights and patents. As we have seen earlier, neither of these applies perfectly to computer hardware or software, and they apply even less well to information. The pace of change in the legal system is slow, helping to ensure that the changes that do occur are fair and well considered. The deliberate pace of change in the legal system is about to be hit by the supersonic rate of change in the information technology industry. Laws do not, and cannot, control all cyber threats. Let us look at several examples of situations in which information needs are about to place significant demands on the legal system.

Information Commerce

Information is unlike most other goods traded, even though it has value and is the basis of some forms of commerce. The market for information is still young, and so far the legal community has experienced few problems. Nevertheless, several key issues must be resolved.

For example, we have seen that software piracy involves copying information without offering adequate payment to those who deserve to be paid. Several approaches have been tried to ensure that the software developer or publisher receives just compensation for use of the software: copy protection, freeware, and controlled distribution. More recently, software is being delivered as mobile code or applets, supplied electronically as needed. The applet approach gives the author and distributor more control. Each applet can potentially be tracked and charged for, and each applet can destroy itself after use so that nothing remains to be passed for free to someone else. But this scheme requires a great deal of accounting and tracking, increasing the costs of what might otherwise be reasonably priced. Thus, none of the current approaches seem ideal, so a legal remedy will often be needed instead of, or in addition to, the technological ones.

Electronic Publishing

Many newspapers and magazines post a version of their content on the Internet, as do wire services and television news organizations. For example, the British Broadcasting Company (BBC) and the Reuters news services have a significant web presence. We should expect that some news and information will eventually be published and distributed exclusively on the Internet. Indeed, encyclopedias such as the Britannica and Expedia are mainly web-based services now, rather than being delivered as the large number of book volumes they used to occupy. Here again the publisher has a problem ensuring that it receives fair compensation for the work. Cryptography-based technical solutions are under development to address this problem. However, these technical solutions must be supported by a legal structure to enforce their use.

Protecting Data in a Database

Databases are a particular form of software that has posed significant problems for legal interpretation. The courts have had difficulty deciding which protection laws apply to databases. How does one determine that a set of data came from a particular database (so that the database owner can claim some compensation)? Who even owns the data in a database if it is public data, such as names and addresses?

Electronic Commerce

Laws related to trade in goods have evolved literally over centuries. Adequate legal protections exist to cover defective goods, fraudulent payment, and failure to deliver when the goods are tangible and are bought through traditional outlets such as stores and catalogs.

However, the situation becomes less clear when the goods are traded electronically. If you order goods electronically, digital signatures and other cryptographic protocols can provide a technical protection for your "money." However, suppose the information you order is not suitable for use or never arrives or arrives damaged or arrives too late to use. How do you prove conditions of the delivery? For catalog sales, you often have receipts or some paper form of acknowledgment of time, date, and location.

But for digital sales, such verification may not exist or can be easily modified. These legal issues must be resolved as we move into an age of electronic commerce.

Protecting Information

Clearly, current laws are inadequate for protecting the information itself and for protecting electronically based forms of commerce. So how is information to be protected legally? As described, copyrights, patents, and trade secrets cover some, but not all, issues related to information. Nevertheless, the legal system does not allow free traffic in information; some mechanisms can be useful.

Criminal and Civil Law

Statutes are laws that state explicitly that certain actions are illegal. A statute is the result of a legislative process by which a governing body declares that the new law will be in force after a designated time. For example, the parliament may discuss issues related to taxing Internet transactions and pass a law about when relevant taxes must be paid. Often, a violation of a statute will result in a criminal trial, in which the government argues for punishment because an illegal act has harmed the desired nature of society. For example, the government will prosecute a murder case because murder violates a law passed by the government. In the United States, criminal transgressions are severe, and the law requires that the judge or jury find the accused guilty beyond reasonable doubt. For this reason, the evidence must be strong and compelling. The goal of a criminal case is to punish the criminal, usually by depriving him or her of rights in some way (such as putting the criminal in prison or assessing a fine).

Civil law is a different type of law, not requiring such a high standard of proof of guilt. In a civil case, an individual, organization, company, or group claims it has been harmed. The goal of a civil case is restitution: to make the victim "whole" again by repairing the harm. For example, suppose Fred kills John. Because Fred has broken a law against murder, the government will prosecute Fred in criminal court for having broken the law and upsetting the order of society. Abigail, the surviving wife, might be a witness at the criminal trial, hoping to see Fred put in prison. But she may also sue him in civil court for wrongful death, seeking payment to support her surviving children.

Tort Law

Special legal language describes the wrongs treated in a civil case. The language reflects whether a case is based on breaking a law or on violating precedents of behavior that have evolved over time. In other words, sometimes judges may make determinations based on what is reasonable and what has come before, rather than on what is written in legislation. A tort is harm not occurring from violation of a statute or from breach of a contract but instead from being counter to the accumulated body of precedents. Thus, statute law is written by legislators and is interpreted by the courts; tort law is unwritten but evolves through court decisions that become precedents for cases that follow. The basic test of a tort is what a reasonable person would do. Fraud is a common example of tort law in which, basically, one person lies to another, causing harm.

Computer information is perfectly suited to tort law. The court merely has to decide what is reasonable behavior, not whether a statute covers the activity. For example, taking information from someone without permission and selling it to someone else as your own is fraud. The owner of the information can sue you, even though there may be no statute saying that information theft is illegal. That owner has been harmed by being deprived of the revenue you received from selling the information.

Because tort law is written only as a series of court decisions that evolve constantly, prosecution of a tort case can be difficult. If you are involved in a case based on tort law, you and your lawyer are likely to try two approaches: First, you might argue that your case is a clear violation of the norms of society, that it is not what a fair, prudent person would do.

This approach could establish a new tort. Second, you might argue that your case is similar to one or more precedents, perhaps drawing a parallel between a computer program and a work of art. The judge or jury would have to decide whether the comparison was apt. In both of these ways, law can evolve to cover new objects.

Contract Law

A third form of protection for computer objects is contracts. A contract is an agreement between two parties. A contract must involve three things:

- an offer
- an acceptance
- a consideration

One party offers something: "I will write this computer program for you for this amount of money." The second party can accept the offer, reject it, make a counter offer, or simply ignore it. In reaching agreement with a contract, only an acceptance is interesting; the rest is just the history of how agreement was reached. A contract must include consideration of money or other valuables.

The basic idea is that two parties exchange things of value, such as time traded for money or technical knowledge for marketing skills. For example, "I'll wash your car if you feed me dinner" or "Let's trade these two CDs" are offers that define the consideration. It helps for a contract to be in writing, but it does not need to be. A written contract can involve hundreds of pages of terms and conditions qualifying the offer and the consideration.

One final aspect of a contract is its freedom: the two parties have to enter into the contract voluntarily. If I say "sign this contract or I'll break your arm," the contract is not valid, even if leaving your arm intact is a really desirable consideration to you. A contract signed under duress or with fraudulent action is not binding. A contract does not have to be fair, in the sense of equivalent consideration for both parties, as long as both parties freely accept the conditions.

Information is often exchanged under contract. Contracts are ideal for protecting the transfer of information because they can specify any conditions. "You have the right to use but not modify this information," "you have the right to use but not resell this information," or "you have the right to view this information yourself but not allow others to view it" are three potential contract conditions that could protect the commercial interests of an owner of information.

Computer contracts typically involve the development and use of software and computerized data. As we note shortly, there are rules about who has the right to contract for software employers or employees and what are reasonable expectations of software's quality.

If the terms of the contract are fulfilled and the exchange of consideration occurs, everyone is happy. Usually. Difficulties arise when one side thinks the terms have been fulfilled and the other side disagrees.

As with tort law, the most common legal remedy in contract law is money. You agreed to sell me a solid gold necklace and I find it is made of brass. I sue you. Assuming the court agreed with me, it might compel you to deliver a gold necklace to me, but more frequently the court will decide I am entitled to a certain sum of money. In the necklace case, I might argue first to get back the money I originally paid you, and then argue for incidental damages from, for example, the doctor I had to see when your brass necklace turned my skin green, or the embarrassment I felt when a friend pointed to my necklace and shouted "Look at the cheap brass necklacea" I might also argue for punitive damages to punish you and keep you from doing such a disreputable thing again. The court will decide which of my claims are valid and what a reasonable amount of compensation is.

Summary of Protection for Computer Artifacts

This section has presented the highlights of law as it applies to computer hardware, software, and data. Clearly these few pages only skim the surface; the law has countless subtleties.

Still, by now you should have a general idea of the types of protection available for what things and how to use them. The differences between criminal and civil law are summarized in Table 11-2.

Table 11-2. Criminal vs. Civil Law.

Criminal Law	Civil Law
Defined by	
□ Statutes	□ Contracts
□ Common law Cases brought by	
□ Government	□ Government
□ Individuals and companies Wronged party	
□ Society	□ Individuals and companies
Remedy	
□ Jail, fine	□ Damages, typically monetary

Contracts help fill the voids among criminal, civil, and tort law. That is, in the absence of relevant statutes, we first see common tort law develop. But people then enhance these laws by writing contracts with the specific protections they want.

Enforcement of civil law torts or contracts can be expensive because it requires one party to sue the other. The legal system is informally weighted by money. It is attractive to sue a wealthy party who could pay a hefty judgment. And a big company that can afford dozens of top-quality lawyers will more likely prevail in a suit than an average individual.

Rights of employees and employers

Employers hire employees to generate ideas and make products. The protection offered by copyrights, patents, and trade secrets appeals to employers because it applies to the ideas and products. However, the issue of who owns the ideas and products is complex. Ownership is a computer security concern because it relates to the rights of an employer to protect the secrecy and integrity of works produced by the employees. In this section we study the respective rights of employers and employees to their computer products.

Ownership of Products

Suppose Edye works for a computer software company. As part of her job, she develops a program to manage windows for a computer screen display. The program belongs to her company because it paid Edye to write the program: she wrote it as a part of a work assignment. Thus, Edye cannot market this program herself. She could not sell it even if she worked for a non-software-related company but developed the software as part of her job.

Most employees understand this aspect of their responsibilities to their employer. Instead, suppose Edye develops this program in the evenings at home; it is not a part of her job. Then she tries to market the product herself. If Edye works as a programmer, her employer will probably say that Edye profited from training and experience gained on the job; at the very least, Edye probably conceived or thought about the project while at work. Therefore, the employer has an interest in (that is, owns at least part of) the rights to her program. However, the situation changes if Edye's primary job does not involve programming.

If Edye is a television newscaster, her employer may have contributed nothing that relates to her computer product. If her job does not involve programming, she may be free to market any computer product she makes. And if Edye's spare-time program is an application that tracks genealogy, her employer would probably not want rights to her program, since it is far from its area of business. (If you are in such a situation yourself, you should check with your employer to be sure.)

Finally, suppose Edye is not an employee of a company. Rather, she is a consultant who is self-employed and, for a fee, writes customized programs for her clients. Consider her legal position in this situation. She may want to use the basic program design, generalize it somewhat, and market it to others. Edye argues that she thought up, wrote, and tested the program; therefore, it is her work, and she owns it. Her client argues that it paid Edye to develop the program, and it owns the program, just as it would own a bookcase she might be paid to build for the station.

Clearly, these situations differ, and interpreting the laws of ownership is difficult.

Let us consider each type of protection in turn.

Ownership of a Patent

The person who owns a work under patent or copyright law is the inventor; in the examples described earlier, the owner is the programmer or the employer. Under patent law, it is important to know who files the patent application. If an employee lets an employer patent an invention, the employer is deemed to own the patent and therefore the rights to the invention.

The employer also has the right to patent if the employee's job functions included inventing the product. For instance, in a large company a scientist may be hired to do research and development, and the results of this inventive work become the property of the employer.

Even if an employee patents something, the employer can argue for a right to use the invention if the employer contributed some resources (such as computer time or access to a library or database) in developing the invention.

Ownership of a Copyright

Owning a copyright is similar to owning a patent. The author (programmer) is the presumed owner of the work, and the owner has all rights to an object. However, a special situation known as *work for hire* applies to many copyrights for developing software or other products.

Work for Hire

In a work for hire situation, the employer, *not* the employee, is considered the author of a work. Work for hire is not easy to identify and depends in part on the laws of the state in which the employment occurs. The relationship between an employee and employer is considered a work for hire if some or all of the following conditions are true. (The more of these conditions that are true, the more a situation resembles work for hire.)

 The employer has a supervisory relationship, overseeing the manner in which the creative work is done.

 The employer has the right to fire the employee.

 The employer arranges for the work to be done before the work was created (as opposed to the sale of an existing work).

 A written contract between the employer and employee states that the employer has hired the employee to do certain work.

In the situation in which Edye develops a program on her job, her employer will certainly claim a work for hire relationship. Then, the employer owns all copyright rights and should be identified in place of the author on the copyright notice.

Licenses

An alternative to a work for hire arrangement is licensed software. In this situation, the programmer develops and retains full ownership of the software. In return for a fee, the programmer grants to a company a license to use the program. The license can be granted for a definite or unlimited period of time, for one copy or for an unlimited number, to use at one location or many, to use on one machine or all, at specified or unlimited times. This arrangement is highly advantageous to the programmer, just as a work for hire arrangement is highly advantageous to the employer. The choice between work for hire and license is largely what the two parties will agree to.

Trade Secret Protection

A trade secret is different from either a patent or a copyright in that there is no registered inventor or author; there is no registration office for trade secrets. In the event a trade secret is revealed, the owner can prosecute the revealer for damages suffered. But first, ownership must be established because only the owner can be harmed.

A company owns the trade secrets of its business-confidential data. As soon as a secret is developed, the company becomes the owner. For example, as soon as sales figures are accumulated, a company has trade secret right to them, even if the figures are not yet compiled, totaled, summarized, printed, or distributed. As with copyrights, an employer may argue about having contributed to the development of trade secrets. If your trade secret is an improved sorting algorithm and part of your job involves investigating and testing sorting algorithms, your employer will probably claim at least partial ownership of the algorithm you try to market.

Employment Contracts

An employment contract often spells out rights of ownership. But sometimes the software developer and possible employer have no contract. Having a contract is desirable both for employees and employers so that both will understand their rights and responsibilities.

Typically, an employment contract specifies that the employee be hired to work as a programmer exclusively for the benefit of the company. The company states that this is a work for hire situation. The company claims all rights to any programs developed, including all copyright rights and the right to market. The contract may further state that the employee is receiving access to certain trade secrets as a part of employment, and the employee agrees not to reveal those secrets to anyone.

More restrictive contracts (from the employee's perspective) assign to the employer rights to all inventions (patents) and all creative works (copyrights), not just those that follow directly from one's job. For example, suppose an employee is hired as an accountant for an automobile company. While on the job, the employee invents a more efficient way to burn fuel in an automobile engine. The employer would argue that the employee used company time to think about the problem, and therefore the company was entitled to this product. An employment contract transferring all rights of inventions to the employer would strengthen the case even more.

An agreement not to compete is sometimes included in a contract. The employee states that simply having worked for one employer will make the employee very valuable to a competitor.

The employee agrees not to compete by working in the same field for a set period of time after termination. For example, a programmer who has a very high position involving the design of operating systems would understandably be familiar with a large body of operating system design techniques. The employee might memorize the major parts of a proprietary operating system and be able to write a similar one for a competitor in a very short time. To prevent this, the employer might require the employee not to work for a competitor (including working as an independent contractor). Agreements not to compete are not always enforceable in law; in some states the employee's right to earn a living takes precedence over the employer's rights.

Software failures

So far, we have considered programs, algorithms, and data as objects of ownership. But these objects vary in quality, and some of the legal issues involved with them concern the degree to which they function properly or well. In fact, people have legitimate differences of opinion on what constitutes "fair," "good," and "prudent" as these terms relate to computer software and programmers and vendors. The law applies most easily when there is broad consensus. In this section we look closely at the role that quality plays in various legal disputes. At the same time, we also look at the ethical side of software quality, foreshadowing a broader discussion on ethics later in this chapter.

Program development is a human process of design, creation, and testing, involving a great deal of communication and interaction. For these reasons, there will always be errors in the software we produce. We sometimes expect perfect consumer products, such as automobiles or lawn mowers. At other times, we expect products to be "good enough" for use, in that most instances will be acceptable. We do not mind variation in the amount of cheese in our pizza or a slight flaw in the glaze on a ceramic tile. If an instance of a product is not usable, we expect the manufacturer to provide some appropriate remedy, such as repair or replacement. In fact, the way in which these problems are handled can contribute to a vendor's reputation for quality service; on the rare occasions when there is a problem, the vendor will promptly and courteously make amends.

But the situation with software is very different. To be fair, an operating system is a great deal more complex than many consumer products, and more opportunities for failure exist. For this reason, this section addresses three questions:

• What are the legal issues in selling correct and usable software?
• What are the moral or ethical issues in producing correct and usable software?
• What are the moral or ethical issues in finding, reporting, publicizing, and fixing flaws?

In some ways, the legal issues are evolving. Everyone acknowledges that all vendors *should* produce good software, but that does not always happen. The more difficult concerns arise in the development and maintenance communities about what to do when faults are discovered.

Selling Correct Software

Software is a product. It is built with a purpose and an audience in mind, and it is purchased by a consumer with an intended use in an expected context. And the consumer has some expectations of a reasonable level of quality and function. In that sense, buying software is like buying a radio. If you buy a faulty radio, you have certain legal rights relating to your purchase and you can enforce them in court if necessary. You may have three reactions if you find something wrong with the radio: You want your money back, you want a different (not faulty) radio, or you want someone to fix your radio. With software you have the same three possibilities, and we consider each one in turn.

To consider our alternatives with software, we must first investigate the nature of the faulty code. Why was the software bad? One possibility is that it was presented on a defective medium. For example, the CD may have had a flaw and you could not load the software on your computer. In this case, almost any merchant will exchange the faulty copy with a new one with little argument. The second possibility is that the software worked properly, but you don't like it when you try it out. It may not do all it was advertised to do. Or you don't like the "look and feel," or it is slower than you expected it to be, or it works only with European phone numbers, not the phone scheme in your country. The bottom line is that there is some attribute of the software that disappoints you, and you do not want this software.

The final possibility is that the software malfunctions, so you cannot use it with your computer system. Here, too, you do not want the software and hope to return it.

I Want a Refund

If the item were a radio, you would have the opportunity to look at it and listen to it in the shop, to assess its sound quality, measure its size (if it is to fit in a particular space), and inspect it for flaws. Do you have that opportunity with a program? Probably not.

The U.S. Uniform Commercial Code (UCC) governs transactions between buyers and sellers in the United States. Section 2-601 says that "if the goods or the tender of delivery fail in any respect to conform to the contract, the buyer may reject them." You may have had no opportunity to try out the software before purchase, particularly on your computer. Your inspection often could not occur in the store (stores tend to frown on your bringing your own computer, opening their shrink-wrapped software, installing the software on your machine, and checking the features). Even if you could have tried the software in the store, you may not have been able to assess how it works with the other applications with which it must interface. So you take home the software, only to find that it is free from flaws but does not fit your needs. You are entitled to a reasonable period to inspect the software, long enough to try out its features. If you decide within a reasonably short period of time that the product is not for you, you can cite UCC §2-601 to obtain a refund.

More often, though, the reason you want to return the software is because it simply is not of high enough quality. Unfortunately, correctness of software is more difficult to enforce legally.

I Want It to Be Good

Quality demands for mass market software are usually outside the range of legal enforcement for several reasons.

 Mass-market software is seldom totally bad. Certain features may not work, and faults may prevent some features from working as specified or as advertised. But the software works for most of its many users or works most of the time for all of its users.

 The manufacturer has "deep pockets." An individual suing a major manufacturer could find that the manufacturer has a permanent legal staff of dozens of full-time attorneys. The cost to the individual of bringing a suit is prohibitive.

 Legal remedies typically result in monetary awards for damages, not a mandate to fix the faulty software.

 The manufacturer has little incentive to fix small problems. Unless a problem will seriously damage a manufacturer's image or possibly leave the manufacturer open to large damage amounts, there is little justification to fix problems that affect only a small number of users or that do not render the product unfit for general use.

Thus, legal remedies are most appropriate only for a large complaint, such as one from a government or one representing a large class of dissatisfied and vocal users. The "fit for use" provision of the UCC dictates that the product must be usable for its intended purpose; software that doesn't work is clearly not usable. The UCC may help you get your money back, but you may not necessarily end up with working software.

Some manufacturers are very attentive to their customers. When flaws are discovered, the manufacturers promptly investigate the problems and fix serious ones immediately, perhaps holding smaller corrections for a later release. These companies are motivated more by public image or moral obligation than by legal requirement.

Trope [TRO04] proposes a warranty of cyberworthiness. The warranty would state that the manufacturer made a diligent search for security vulnerabilities and had removed all known critical ones. Furthermore, the vendor will continue to search for vulnerabilities after release and, on learning of any critical ones, will contact affected parties with patches and work-arounds. Now, a maker is potentially liable for all possible failings, and a major security-critical flaw could be very costly. Trope's approach limits the exposure to addressing known defects reasonably promptly.

Reporting Software Flaws

Who should publicize flawsthe user or the manufacturer? A user might want the recognition of finding a flaw; delaying the release might let someone else get that credit. A manufacturer might want to ignore a problem or fail to credit the user. And either could say the other was wrong. And how should these flaws be reported? Several different viewpoints exist.

What You Don't Know Can Hurt You

The several variants of Code Red in 2001 sparked a debate about whether we should allow full disclosure of the mechanisms that allow malicious code to enter and thrive in our systems. For example, the first variant of Code Red was relatively benign, but the third and fourth variants were powerful. When the first Code Red variant appeared, it was studied by many security analysts, including those at eEye Digital Security in Aliso Viejo, California. In an effort to pressure vendors and software managers to take seriously the threats they represent, eEye practices full disclosure of what it knows about security flaws.

However, some observers claim that such open sharing of information is precisely what enables hackers to learn about vulnerabilities and then exploit them. Several developers suspect that eEye's openness about Code Red enabled the more powerful variants to be written and disseminated [HUL01].

Scott Culp [CUL01], Microsoft's manager of Windows security, distinguishes between full disclosure and full exposure; he thinks that source code or detailed explanations of a vulnerability's concept should be protected. And many security analysts encourage users and managers to apply patches right away, closing security holes before they can be exploited.

But as we saw in Sidebar 3-5, the patches require resources and may introduce other problems while fixing the initial one. Each software-using organization must analyze and balance the risks and cost of not acting with the risks and costs of acting right away.

The Vendor's Interests

Microsoft argues that producing one patch for each discovered vulnerability is inefficient both for the vendor and the user. The vendor might prefer to bundle several patches into a single service pack or, for noncritical vulnerabilities, to hold them until the next version. So, Microsoft would like to control if or when the report of a vulnerability goes public.

Craig Mundie, Microsoft's Chief Technology Officer, suggests a stronger reason to minimize disclosure of vulnerability information. "Every time we become explicit about a problem that exists in a legacy product, the response to our disclosure is to focus the attack. In essence we end up funneling them to the vulnerability." [FIS02a] Scott Culp argued [CUL01] that " a vendor's responsibility is to its customers, not to a self-described security community." He opposed what he called "information anarchy,... the practice of deliberately publishing explicit, step-by-step instructions for exploiting security vulnerabilities without regard for how the information may be used." But he also acknowledged that the process of developing, distributing, and applying patches is imperfect, and his own company "need[s] to make it easier for users to keep their systems secure."

Users' Interests

David Litchfield, a security researcher noted for locating flaws in vendors' programs, announced in May 2002 that he would no longer automatically wait for a vendor's patch before going public with a vulnerability announcement. Citing "lethargy and an unwillingness to patch security problems as and when they are found," [FIS02b] Litchfield criticized the approach of holding fixes of several vulnerabilities until enough had accumulated to warrant a single service pack. He makes the point that publicized or not, the vulnerabilities still exist. If one reporter has found the problem, so too could any number of malicious attackers. For a vendor to fail to provide timely patches to vulnerabilities of which the vendor is aware leaves the users wide open to attacks of which the user may be unaware.

Litchfield's solution is to put pressure on the vendor. He announced he would give vendors one week's notice of a vulnerability before publicizing the vulnerability but not the details of how to exploit itto the world.

"Responsible" Vulnerability Reporting

Clearly the conflicting interests of vendors and users must meet at some compromise position. (For an example of how vulnerability disclosure does *not* work, see Sidebar 11-3.) Christey and Wysopal [CHR02] have proposed a vulnerability reporting process that meets constraints of timeliness, fair play, and responsibility. They call the user reporting a suspected vulnerability a "reporter" and the manufacturer the "vendor." A third party such as a computer emergency response center called a "coordinator" could also play a role when a conflict or power issue arises between reporter and vendor. Basically, the process requires reporter and vendor to do the following:

 The vendor must acknowledge a vulnerability report confidentially to the reporter.
 The vendor must agree that the vulnerability exists (or argue otherwise) confidentially
to the reporter.
 The vendor must inform users of the vulnerability and any available countermeasures
within 30 days or request additional time from the reporter as needed.
 After informing users, the vendor may request from the reporter a 30-day quiet period
to allow users time to install patches.
 At the end of the quiet period the vendor and reporter should agree upon a date at which time the vulnerability information may be released to the general public.
 The vendor should credit the reporter with having located the vulnerability.
 If the vendor does not follow these steps, the reporter should work with a coordinator
to determine a responsible way to publicize the vulnerability.

Such a proposal can only have the status of a commonly agreed-on process, since there is no authority that can enforce adherence on either users or vendors.

Quality Software

Boris Beizer, a consultant, has said, "Software should be shipped with bugs. The zero-defect notion is mythological and theoretically unachievable. That doesn't mean shipping ill-behaved or useless software; it means being open with users about the bugs we find, sending notices or including the bug list, publishing the workarounds when we have them, and being honest and open about what we have and haven't yet tested and when we do and don't plan to test in the near future." [COF02]

The whole debate over how and when to disclose vulnerabilities avoids the real issue. The world does not need faster patches, it needs better software with fewer vulnerabilities after delivery to the user. Forno [FOR01] says, "The most significant danger and vulnerability facing the Wired World is continuing to accept and standardize corporate and consumer computer environments on technology that's proven time and again to be insecure, unstable, and full of undocumented bugs ('features') that routinely place the Internet community at risk."

In January 2002, Bill Gates, CEO of Microsoft, announced that producing quality software with minimal defects was his highest priority for Microsoft, ahead of new functionality. His manager of development of the XP operating system announced he was requiring programmers involved in development of XP to attend a course in secure programming. Did the initiative work? In one five-day period in June 2002, Microsoft released six separate patches for security vulnerabilities. In November 2004, Microsoft went to once-a-month patch releases and has distributed an average of two to three new critical patches each month since then.

The issue is not how promptly a vulnerability is patched or how much detail is released with a vulnerability announcement. The issue is that, as the Anderson report [AND72] noted over three decades ago, "penetrate and patch" is a fatally flawed concept: after a flaw was patched, the penetrators always found other old flaws or new flaws introduced because of or in the patch. The issue is technical, psychological, sociological, managerial, and economic.

Until we produce consistently solid software, our entire computing infrastructure is seriously at risk.

Computer crime

The law related to contracts and employment is difficult, but at least employees, objects, contracts, and owners are fairly standard entities for which legal precedents have been developed over centuries. The definitions in copyright and patent law are strained when applied to computing because old forms must be made to fit new objects; for these situations, however, cases being decided now are establishing legal precedents. But crimes involving computers are an area of the law that is even less clear than the other areas. In this section we study computer crime and consider why new laws are needed to address some of its problems.

Why a Separate Category for Computer Crime Is Needed

Crimes can be organized into certain recognized categories, including *murder*, *robbery*, and *littering*. We do not separate crime into categories for different weapons, such as *gun crime* or *knife crime*, but we separate crime victims into categories, depending on whether they are *people* or *other objects*. Nevertheless, driving into your neighbor's picture window can be as bad as driving into his evergreen tree or pet sheep. Let us look at an example to see why these categories are not sufficient and why we need special laws relating to computers as subjects and objects of crime.

Rules of Property

Parker and Nycom [PAR84] describe the theft of a trade secret proprietary software package. The theft occurred across state boundaries by means of a telephone line; this interstate aspect is important because it means that the crime is subject to federal law as well as state law. The California Supreme Court ruled that this software acquisition was not theft because Implicit in the definition of "article" in Section 499c(a) is that it must be something *tangible*...

Based on the record here, the defendant did not carry any tangible thing... from the computer to his terminal unless the impulses which defendant allegedly caused to be transmitted over the telephone wire could be said to be tangible. *It is the opinion of the Court that such impulses are not tangible and hence do not constitute an aarticle.*"

The legal system has explicit rules about what constitutes property. Generally, property is tangible, unlike magnetic impulses. For example, unauthorized use of a neighbor's lawn mower constitutes theft, even if the lawn mower was returned in essentially the same condition as it was when taken. To a computer professional, taking a copy of a software package without permission is clear-cut theft. Fortunately, laws evolve to fit the times, and this interpretation from the 1980s has been refined so that bits are now recognized items of property.

A similar problem arises with computer services. We would generally agree that unauthorized access to a computing system is a crime. For example, if a stranger enters your garden and walks around, even if nothing is touched or damaged, the act is considered trespassing.

However, because access by computer does not involve a physical object, not all courts punish it as a serious crime.

Rules of Evidence

Computer printouts have been used as evidence in many successful prosecutions. Frequently-used are computer records generated in the ordinary course of operation, such as system audit logs.

Under the rules of evidence, courts prefer an original source document to a copy, under the assumption that the copy may be inaccurate or may have been modified in the copying process. The biggest difficulty with computer-based evidence in court is being able to demonstrate the authenticity of the evidence. Law enforcement officials operate under a chain of custody requirement: From the moment a piece of evidence is taken until it is presented in court, they track clearly and completely the order and identities of the people who had personal custody of that object. The reason for the chain of custody is to ensure that nobody has had the opportunity to alter the evidence in any way before its presentation in court. With computer-based evidence, it can be difficult to establish a chain of custody. If a crime occurred on Monday but was not discovered until Wednesday, who can verify that the log file was not altered? In fact, it probably was altered many times as different processes generated log entries. The issue is to demonstrate convincingly that the log entry for 2:37 on Monday does in fact correspond to the event that took place at that time on Monday, not some attempt on Thursday to plant a false clue long after the crime took place.

Threats to Integrity and Confidentiality

The integrity and secrecy of data are also issues in many court cases. Parker and Nycom [PAR84] describe a case in which a trespasser gained remote access to a computing system.

The computing system contained confidential records about people, and the integrity of the data was important. The prosecution of this case had to be phrased in terms of theft of computer time and valued as such, even though that was insignificant compared with loss of privacy and integrity. Why? Because the law as written recognized theft of computer time as a loss, but not loss of privacy or destruction of data.

Now, however, several federal and state laws recognize the privacy of data about individuals. For example, disclosing grades or financial information without permission is a crime, and tort law would recognize other cases of computer abuse.

Value of Data

In another computer crime, a person was found guilty of having stolen a substantial amount of data from a computer data bank. However, the court determined that the "value" of that data was the cost of the paper on which it was printed, which was only a few dollars. Because of that valuation, this crime was classified as a misdemeanor and considered to be a minor crime.

Fortunately, the courts have since determined that information and other intangibles can have significant value. The concept of what we value and how we determine its value is key to understanding the problems with computer-based law. In most economies, paper money is accepted as a valuable commodity, even if the paper on which it is printed is worth only a few cents. Cash is easy to value: A dollar bill is worth one dollar. But consider the way we determine the value of a company's assets. Usually, the valuation reflects the amount of money a person or organization is willing to pay for it. For example, the assets of a credit bureau are its files.

Banks and insurance companies willingly pay $20 or more for a credit report, even though the paper itself is worth less than a dollar. For a credit bureau, the amount a willing customer will pay for a report is a fair estimate of the report's value; this estimate is called the market value of the report. However, the credit bureau (or any company) has other assets that are not sold but are just as valuable to the company's financial viability. For instance, a confidential list of clients has no market value that can be established, but the list may be essential. Its value is apparent only when a loss is suffered, such as when the secret information is made available to a competitor. Over time, the legal system will find ways to place a value on data that is representative of its value to those who use it.

Although these methods of valuation are accepted in civil suits, they have not yet been widely accepted in criminal prosecution.

Acceptance of Computer Terminology

The law is also lagging behind technology in its acceptance of definitions of computing terms. For example, according to a federal statute, it is unlawful to commit arson within a federal enclave (18 USC 81). Part of that act relates to "machinery or building material or supplies" in the enclave, but court decisions have ruled that a motor vehicle located within a federal enclave at the time of the burning was not included under this statute. Because of that ruling, it is not clear whether computer hardware constitutes "machinery" in this context; "supplies" almost certainly does not include software. Computers and their software, media, and data must be understood and accepted by the legal system.

Why Computer Crime Is Hard to Define

From these examples, it is clear that the legal community has not accommodated advances in computers as rapidly as has the rest of society. Some people in the legal process do not understand computers and computing, so crimes involving computers are not always treated properly. Creating and changing laws are slow processes, intended to involve substantial thought about the effects of proposed changes. This deliberate process is very much out of pace with a technology that is progressing as fast as computing.

Adding to the problem of a rapidly changing technology, a computer can perform many roles in a crime. A particular computer can be the subject, object, or medium of a crime. A computer can be attacked (attempted unauthorized access), used to attack (impersonating a legitimate node on a network), and used as a means to commit crime (Trojan horse or fake login).

Computer crime statutes must address all of these evils.

Why Computer Crime Is Hard to Prosecute

Even when everyone acknowledges that a computer crime has been committed, computer crime is hard to prosecute for the following reasons.

- *Lack of understanding.* Courts, lawyers, police agents, or jurors do not necessarily understand computers. Many judges began practicing law before the invention of computers, and most began before the widespread use of the personal computer. Fortunately, computer literacy in the courts is improving as judges, lawyers, and police officers use computers in their daily activities.
- *Lack of physical evidence.* Police and courts have for years depended on tangible evidence, such as fingerprints. As readers of Sherlock Holmes know, seemingly minuscule clues can lead to solutions to the most complicated crimes (or so Doyle would have you believe). But with many computer crimes there simply are no fingerprints and no physical clues of any sort.
- *Lack of recognition of assets.* We know what cash is, or diamonds, or even negotiable securities. But are twenty invisible magnetic spots really equivalent to a million dollars?
Is computer time an asset? What is the value of stolen computer time if the system would have been idle during the time of the theft?
- *Lack of political impact.* Solving and obtaining a conviction for a murder or robbery is popular with the public, and so it gets high priority with prosecutors and police chiefs. Solving and obtaining a conviction for an obscure high-tech crime, especially one not involving obvious and significant loss, may get less attention. However, as computing becomes more pervasive, the visibility and impact of computer crime will increase.
- *Complexity of case.* Basic crimes that everyone understands, such as murder, kidnapping, or auto theft, can be easy to prosecute. A complex money-laundering or tax fraud case may be more difficult to present to a jury because jurors have a hard time following a circuitous accounting trail. But the hardest crime to present may be a high-tech crime, described, for example, as root access by a buffer overflow in which memory was overwritten by other instructions, which allowed the attacker to copy and execute code at will and then delete the code, eliminating all traces of entry (after disabling the audit logging, of course).
- *Age of defendant..* Many computer crimes are committed by juveniles. Society understands immaturity and disregards even very serious crimes by juveniles because the juveniles did not understand the impact of their actions. A more serious, related problem is

that many adults see juvenile computer crimes as childhood pranks, the modern equivalent of tipping over an outhouse.

Even when there is clear evidence of a crime, the victim may not want to prosecute because of possible negative publicity. Banks, insurance companies, investment firms, the government, and healthcare groups think their trust by the public will be diminished if a computer vulnerability is exposed. Also, they may fear repetition of the same crime by others: so-called copycat crimes. For all of these reasons, computer crimes are often not prosecuted.

Examples of Statutes

As a few examples from the 1980s have pointed out, in the early days, prosecution of computer crimes was hampered by lack of clear appreciation of the nature or seriousness of crime involving computers. Although theft, harm to persons, and damage to property have been crimes for a long time, in some cases new laws were useful to make it obvious to the courts what computer-related behavior was unacceptable. Most states now have laws covering computer crime of one sort or another. Also, computer-related crimes now appear in sentencing guidelines. In this section we highlight a few of the laws defining aspects of crime against or using computers.

U.S. Computer Fraud and Abuse Act

The primary federal statute, 18 USC 1030, was enacted in 1984 and has been amended several times since. This statute prohibits

- unauthorized access to a computer containing data protected for national defense or foreign relations concerns
- unauthorized access to a computer containing certain banking or financial information
- unauthorized access, use, modification, destruction, or disclosure of a computer or information in a computer operated on behalf of the U.S. government
- accessing without permission a "protected computer," which the courts now interpret to include any computer connected to the Internet
- computer fraud
- transmitting code that causes damage to a computer system or network
- trafficking in computer passwords

Penalties range from $5,000 to $100,000 or twice the value obtained by the offense, whichever is higher, or imprisonment from 1 year to 20 years, or both.

U.S. Economic Espionage Act

This 1996 act outlaws use of a computer for foreign espionage to benefit a foreign country or business or theft of trade secrets.

U.S. Electronic Funds Transfer Act

This law prohibits use, transport, sale, receipt, or supply of counterfeit, stolen, altered, lost, or fraudulently obtained debit instruments in interstate or foreign commerce.

U.S. Freedom of Information Act

The Freedom of Information Act provides public access to information collected by the executive branch of the federal government. The act requires disclosure of any available data, unless the data fall under one of several specific exceptions, such as national security or personal privacy. The law's original intent was to release to individuals any information the government had collected on them. However, more corporations than individuals file requests for information as a means of obtaining information about the workings of the government.

Even foreign governments can file for information. This act applies only to government agencies, although similar laws could require disclosure from private sources. The law's effect is to require increased classification and protection for sensitive information.

U.S. Privacy Act

The Privacy Act of 1974 protects the privacy of personal data collected by the government. An individual is allowed to determine what data have been collected on him or her, for what purpose, and to whom such information has been disseminated. An additional use of the law is to prevent one government agency from accessing data collected by another agency for another purpose. This act requires diligent efforts to preserve the secrecy of private data collected.

U.S. Electronic Communications Privacy Act

This law, enacted in 1986, protects against electronic wiretapping. There are some important qualifications. First, law enforcement agencies are always allowed to obtain a court order to access communications or records of them. And an amendment to the act requires Internet service providers to install equipment as needed to permit these court-ordered wiretaps.

Second, the act allows Internet service providers to read the content of communications in order to maintain service or to protect the provider itself from damage. So, for example, a provider could monitor traffic for viruses.

Gramm Leach Bliley

The U.S. Gramm Leach Bliley Act (Public Law 106-102) of 1999 covers privacy of data for customers of financial institutions. Each institution must have a privacy policy of which it informs its customers, and customers must be given the opportunity to reject any use of the data beyond the necessary business uses for which the private data were collected. The act and its implementation regulations also require financial institutions to undergo a detailed security-risk assessment. Based on the results of that assessment, the institution must adopt a comprehensive "information security program" designed to protect against unauthorized access to or use of customers' nonpublic personal information.

HIPAA

In 1996, Public Law 104-191, the Health Insurance Portability and Accountability Act (HIPAA) was passed in the United States. Although the first part of the law concerned the rights of workers to maintain health insurance coverage after their employment was terminated, the second part of the law required protection of the privacy of individuals' medical records. HIPAA and its associated implementation standards mandate protection of "individually identifiable healthcare information," that is, medical data that can be associated with an identifiable individual. To protect the privacy of individuals' healthcare data, healthcare providers must perform standard security practices, such as the following:

- Enforce need to know.
- Ensure minimum necessary disclosure.
- Designate a privacy officer.
- Document information security practices.
- Track disclosures of information.
- Develop a method for patients' inspection and copying of their information.
- Train staff at least every three years.

Perhaps most far-reaching is the requirement for healthcare organizations to develop "business associate contracts," which are coordinated agreements on how data shared among entities will be protected. This requirement could affect the sharing and transmittal of patient information among doctors, clinics, laboratories, hospitals, insurers, and any other organizations that handle such data.

USA Patriot Act

Passed in 2001 in reaction to terrorist attacks in the United States, the USA Patriot Act includes a number of provisions supporting law enforcement's access to electronic communications. Under this act, law enforcement need only convince a court that a target is probably an agent of a foreign power in order to obtain a wiretap order. The main computer security provision of the Patriot Act is an amendment to the Computer Fraud and Abuse Act:

- Knowingly causing the transmission of code resulting in damage to a protected computer is a felony.
- Recklessly causing damage to a computer system as a consequence of unauthorized access is also a felony.
- Causing damage (even unintentionally) as a consequence of unauthorized access to a protected computer is a misdemeanor.

The CAN SPAM Act

Unsolicited "junk" e-mail or spam is certainly a problem. Analysts estimate that as much as 70 percent of all e-mail traffic is spam. To address pressure from their constituents, in 2003 U.S. lawmakers passed the Controlling the Assault of Non-Solicited Pornography and Marketing (CAN SPAM) Act. (One wonders how many staff members it took to find a sequence of words to yield that acronym.) Key requirements of the law are these:

- It bans false or misleading header information.

- It prohibits deceptive subject lines.
- It requires commercial e-mail to give recipients an opt-out method.
- It bans sale or transfer of e-mail addresses of people who have opted out.
- It requires that commercial e-mail be identified as an advertisement.

Critics of the law point out that it preempts state laws, and some states had stronger laws. It also can be read as permitting commercial e-mail as long as the mail is not deceptive. Finally, and most importantly, it does little to regulate spam that comes from offshore: a spam sender simply sends spam from a foreign mailer, perhaps in a country more interested in generating business for its national ISPs than in controlling worldwide junk e-mail. The most telling result:

The volume of spam has not declined since the law.

California Breach Notification

The first state in the U.S. to enact such a law, California passed SB1386, effective in 2003. This law requires any company doing business in California or any California government agency to notify individuals of any breach that has, or is reasonably believed to have, compromised personal information on any California resident. As a state law, it is limited to California residents and California companies. At least 20 other states have since followed with some form of breach notification.

The most widely reported application of the law was in February 2005 when Choicepoint disclosed that some California residents had been affected by loss of 145,000 pieces of personal identity information. Initially only affected California residents were informed, but after news of that disclosure was made public, Choicepoint revealed how many people total were involved and began notifying them.

International Dimensions

So far we have explored laws in the United States. But many people outside the United States will read this book, perhaps wondering why they should learn about laws from a foreign country. This question has two answers.

Technically, computer security laws in the United States are similar to those in many other countries: Lawmakers in each country learn about subtle legal points and interpretation or enforcement difficulties from laws passed in other countries. Many other countries, such as Australia, Canada, Brazil, Japan, the Czech Republic, and India, have recently enacted computer crime laws. These laws cover offenses such as fraud, unauthorized computer access, data privacy, and computer misuse. Schjolberg [SCH02] has compiled a survey of different countries' laws to counter unauthorized access.

The second reason to study laws from a foreign country is that the Internet is an international entity. Citizens in one country are affected by users in other countries, and users in one country may be subject to the laws in other countries. Therefore, you need to know which laws may affect you. The international nature of computer crime makes life much more complicated. For example, a citizen of country A may sit in country B, dial into an ISP in country C, use a compromised host in country D, and attack machines in country E (not to mention traveling on communications lines through dozens of other countries). To prosecute this crime may require cooperation of all five countries. The attacker may need to be extradited from B to E to be prosecuted there, but there may be no extradition treaty for computer crimes between B and E. And the evidence obtained in D may be inadmissible in E because of the manner in which it was obtained or stored. And the crime in E may not be a crime in B, so the law enforcement authorities, even if sympathetic, may be unable to act.
Although computer crime is truly international, differing statutes in different jurisdictions inhibit prosecution of international computer crime. In the remainder of this section we briefly discuss laws around the world that differ from U.S. laws and that should be of interest to computer security students.

Council of Europe Agreement on Cybercrime

In November 2001, the United States, Canada, Japan, and 22 European countries signed the Council of Europe Agreement on Cybercrime to define cybercrime activities and support their investigation and prosecution across national boundaries. The significance of this treaty is not so much that these activities are illegal but that the countries acknowledged them as crimes across their borders, making it easier for law enforcement agencies to cooperate and for criminals to be extradited for offenses against one country committed from within another country. But to really support investigation, prosecution, and conviction of computer criminals, more than just these 25 countries will have to be involved.

The treaty requires countries that ratify it to adopt similar criminal laws on hacking, computer-related fraud and forgery, unauthorized access, infringements of copyright, network disruption, and child pornography. The treaty also contains provisions on investigative powers and procedures, such as the search of computer networks and interception of communications, and requires cross-border law enforcement cooperation in searches and seizures and extradition. The original treaty has been supplemented by an additional protocol making any publication of racist and xenophobic propaganda via computer networks a criminal offense.

E.U. Data Protection Act

The E.U. Data Protection Act, based on the European Privacy Directive, is model legislation for all the countries in the European Union. It establishes privacy rights and protection responsibilities for all citizens of member countries. The act governs the collection and storage of personal data about individuals, such as name, address, and identification numbers. The law requires a business purpose for collecting the data, and it controls against disclosure. Dating from 1994 in its initial form, this law was one of the first to establish protection requirements for the privacy of personal data. Most significantly, the act requires equivalent protection in non-E.U. countries if organizations in the European Union pass protected data outside the European Union. Chapter 10 contains more detail on this directive.

Restricted Content

Some countries have laws controlling Internet content allowed in their countries. Singapore requires service providers to filter content allowed in. China bans material that disturbs social order or undermines social stability. Tunisia has a law that applies the same controls on critical speech as for other media forms [HRW99].

Further laws have been proposed to make it illegal to transmit outlawed content *through* a country, regardless of whether the source or destination of the content is in that country.

Given the complex and unpredictable routing structure of the Internet, complying with these laws, let alone enforcing them, is effectively impossible.

Use of Cryptography

Cryptography is the fourth major area in which different countries have developed laws. We survey these laws in a subsequent section.

Why Computer Criminals Are Hard to Catch

As if computer crime laws and prosecution were not enough, it is also difficult for law enforcement agencies to catch computer criminals. There are two major reasons for this.

First, computer crime is a multinational activity that must usually be pursued on a national or local level. There are no international laws on computer crime. Even though the major industrial nations cooperate very effectively on tracking computer criminals, criminals know there are "safe havens" from which they cannot be caught. Often, the trail of a criminal stops cold at the boundary of a country. Riptech Inc. [BEL02] studies Internet attack trends by many factors. For the period JanuaryJune 2002 the United States led the world in source of Internet attacks (40 percent) followed by Germany (7 percent). But when you normalize these data for number of users, a very different pattern emerges. Per Internet user, Israel and Hong Kong lead among those nations with more than 1 million users, and Kuwait and Iran top the list among nations with fewer than 1 million users. Nations all over the globe appear on these lists, which demonstrates that attackers can and do operate from many different countries.

Complexity is an even more significant factor than country of origin. As we have stated throughout this book, networked attacks are hard to trace and investigate because they can involve so many steps. A smart attacker will "bounce" an attack through many places to obscure the trail. Each step along the way makes the investigator complete more legal steps.

If the trail leads from server A to B to C, the law enforcement investigators need a search warrant for data at A, and others for B and C. Even after obtaining the search warrants, the investigator has to find the right administrator and serve the warrants to begin obtaining data.

In the time the investigator has to get and serve warrants, not to mention follow leads and correlate findings, the attacker has carefully erased the digital evidence. In a CNET News article, Sandoval [SAN02] says law enforcement agencies are rarely able to track down hackers sophisticated enough to pull off complicated attacks. Sandoval quotes Richard Power, editorial director of the Computer Security Institute: "It's a world class business." Independent investigator Dan Clements says, "only about 10 percent of active hackers are savvy enough to work this way consistently, but they are almost always successful."

What Computer Crime Does Not Address

Even with the definitions included in the statutes, the courts must interpret what a computer is. Legislators cannot define precisely what a computer is because computer technology is used in many other devices, such as robots, calculators, watches, automobiles, microwave ovens, and medical instruments. More importantly, we cannot predict what kinds of devices may be invented ten or fifty years from now. Therefore, the language in each of these laws indicates the kinds of devices the legislature seeks to include as computers and leaves it up to the court to rule on a specific case. Unfortunately, it takes awhile for courts to build up a pattern of cases, and different courts may rule differently in similar situations. The interpretation of each of these terms will be unsettled for some time to come.

Value presents a similar problem. As noted in some of the cases presented, the courts have trouble separating the intrinsic value of an object (such as a sheet of paper with writing on it) from its cost to reproduce. The courts now recognize that a Van Gogh painting is worth more than the cost of the canvas and paint. But the courts have not agreed on the value of printed computer output. The cost of a blank diskette is miniscule, but it may have taken thousands of hours of data gathering and machine time to produce the data encoded on a diskette. The courts are still striving to determine the fair value of computer objects. Both the value of a person's privacy and the confidentiality of data about a person are even less settled. In a later section we consider how ethics and individual morality take over where the law stops.

Cryptography and the Law

The law is used to regulate people for their own good and for the greater good of society. Murder, theft, drinking, and smoking are circumscribed by laws. Generally, the balance between personal freedom and the good of society is fairly easy to judge; for example, one's right to fire a gun ends when the bullet hits someone. Cryptography is also a regulated activity, but the issues are a little less clear-cut, in part because there is little open discussion of the subject.

People want to protect their privacy, including the secrecy of communications with others. Businesses want similar confidentiality. Criminals want secrecy so that they can communicate criminal plans in private. Governments want to track illegal activity, both to prevent crime and to apprehend and convict criminals after a crime has been committed. Finally, nations want to know the military and diplomatic plans of other nations. As shown throughout this book, cryptography can be a powerful tool to protect confidentiality, but being able to break cryptography can be a potent tool for government. Phrased differently, it suits governments' interests if people cannot use cryptography that is too good (meaning, unbreakable by the government).

Controls on Use of Cryptography

Closely related to restrictions on content are restrictions on the use of cryptography imposed on users in certain countries. In China, for example, State Council Order 273 requires foreign organizations or individuals to apply for permission to use encryption in China. Pakistan requires that all encryption hardware and software be inspected and approved by the Pakistan Telecommunication Authority. And in Iraq, use of even the Internet is strictly limited, and unauthorized use of encryption carries heavy penalties.

France's encryption policy is probably the most widely discussed. Import of encryption products is subject to a registration requirement: A vendor's registration for a mass-market commercial product is valid for all imports of that product. Use of encryption for authentication is unlimited. Use of encryption with a key length up to 128 for confidentiality requires only the vendor's registration. Use of products with a key length greater than 128 bits requires that the key be escrowed with a trusted third party.

Such laws are very difficult to enforce individually. Cryptography, steganography, and secret writing have been used for centuries. The governments know they cannot prevent two cooperating people from concealing their communications. However, governments can limit widespread computer-based use by limiting cryptography in mass-market products. Although policing 50 million computer users is impossible, controlling a handful of major computer manufacturers is feasible, especially ones whose profits would be affected by not being able to sell any products in a particular country. Thus, governments have addressed cryptography use at the source: the manufacturer and vendor.

Controls on Export of Cryptography

Until 1998, the United States led other industrialized nations in controlling cryptography. It did this by controlling export of cryptographic products, using the same category as munitions, such as bombs and atomic missiles. Although the law applied to everyone, in practice it could be enforced reasonably only against mass-market software manufacturers. Software makers could export freely[1] any product using symmetric encryption with a key length of 40 bits or less. Exceptions allowed stronger encryption for financial institutions and for multinational corporations using the encryption for intracompany communication. Cryptography solely for authentication (for example, digital signatures) was also permitted. Although the law did not control the *use* of cryptography, limiting export effectively limited its use because major vendors could not sell products worldwide with strong encryption.

[1] That is, they could export to all but a handful of so-called rogue nations subject to stringent controls on munitions.

U.S. policy was especially important because most mass-market software vendors were based in the United States, and many users were in the United States. The United States could also pressure software vendors not to write programs in such a way that someone could add the cryptography at an overseas location. Although a software vendor could move to or open a subsidiary in an uncontrolled country, a new vendor has a hard time obtaining a significant share of the market against large, established competitors. If such a vendor were able to take a significant amount of business away from U.S. companies, there would be an outcry and possible political pressure from the U.S. government. Thus, U.S. policy on this issue would and did dominate the world market.

Cryptography and Free Speech

Cryptography involves not just products; it involves ideas, too. Although governments effectively control the flow of products across borders, controlling the flow of ideas, either in people's heads or on the Internet, is almost impossible.

In a decision akin to splitting hairs, the U.S. courts ruled that computer object code was subject to the export restrictions, but a printed version of the corresponding source code was an idea that could not be restricted. The case in question involved Phil Zimmermann, the inventor of PGP e-mail encryption. In 1997, Zimmermann "exported" books containing the printed source code to PGP, and volunteers in Europe spent 1000 hours scanning the pages of the book; they then posted this source code publicly on the Internet. To highlight the vacuousness of this distinction, people reduced the object code of the PGP program to a bar code and printed that code on T-shirts with the caption "Warning, this T-shirt may be a controlled munition."

Cryptographic Key Escrow

Although laws enable governments to read encrypted communications, the governments don't really want to read *all* of them. A joking e-mail message or a file with your tax data is seldom a national security concern. But suppose there was evidence of cheating on your taxes or your writings were seditious. In these cases the government could convince a court to allow it to search your home, office, or computer files. It might then have reason and justification for wanting to read your encrypted data. So the government devised a scheme in which your encryption keys would become available only with court authorization.

In 1996 the U.S. government offered to relax the export restriction for so-called escrowed encryption, in which the government would be able to obtain the encryption key for any encrypted communication. The key escrow approach was a part of an initiative known under names such as Clipper, Capstone, and Fortezza. Ultimately this approach failed; the public feared what the government could actually access. See [HOF95] and [DEN99] for more discussion on the key escrow debate.

Current Policy

The U.S. National Research Council (NRC) reported the results of an 18-month study [NRC96] to recommend a cryptographic policy for the U.S. federal government. The report carefully weighed all the factors affected by cryptographic policy, such as protecting sensitive information for U.S. companies and individuals as well as foreign ones, international commerce, enforcing laws (prevention, investigation, and prosecution), and intelligence gathering. The report's recommendations for policy include the following:

- No law should bar the manufacture, sale, or use of any form of encryption within the United States.
- Export controls on cryptography should be relaxed but not eliminated.
- Products providing confidentiality at a level that meets most general commercial requirements should be easily exportable. In 1996, that level included products that incorporate 56-bit key DES, and so these products should be easily exportable.
- Escrowed encryption should be studied further, but, as it is not yet a mature technology, its use should not be mandated.
- Congress should seriously consider legislation that would impose criminal penalties on the use of encrypted communications in interstate commerce with the intent to commit a crime.
- The U.S. government should develop a mechanism to promote information security in the private sector.

In September 1998, the U.S. government announced that it was opening up export of encryption. Export of single (56-bit) key DES would be allowed to all countries except seven that supported terrorism. Unlimited size encryption would be exportable to 45 major industrial countries for use by financial institutions, medical providers, and e-commerce companies.

Furthermore, the process for applying for permission, which had been another formidable deterrent, was simplified to a review taking no more than a week in most cases.

Summary of Legal Issues in Computer Security

This section has described four aspects of the relationship between computing and the law. First, we presented the legal mechanisms of copyright, patent, and trade secret as means to protect the secrecy of computer hardware, software, and data. These mechanisms were designed before the invention of the computer, so their applicability to computing needs is somewhat limited. However, program protection is especially desired, and software companies are pressing the courts to extend the interpretation of these means of protection to include computers.

We also explored the relationship between employers and employees, in the context of writers of software. Well-established laws and precedents control the acceptable access an employee has to software written for a company.

Third, we examined the legal side of software vulnerabilities: Who is liable for errors in software, and how is that liability enforced? Additionally, we considered alternative ways to report software errors.

Fourth, we noted some of the difficulties in prosecuting computer crime. Several examples showed how breaches of computer security are treated by the courts. In general, the courts have not yet granted computers, software, and data appropriate status, considering value of assets and seriousness of crime. The legal system is moving cautiously in its acceptance of computers. We described several important pieces of computer crime legislation that represent slow progress forward.

Privacy- Ethical issues in computer society

This final section helps clarify thinking about the ethical issues involved in computer security. We offer no answers. Rather, after listing and explaining some ethical principles, we present several case studies to which the principles can be applied. Each case is followed by a list of possible ethical issues involved, although the list is not necessarily all-inclusive or conclusive. The primary purpose of this section is to explore some of the ethical issues associated with computer security and to show how ethics functions as a control.

Differences Between the Law and Ethics

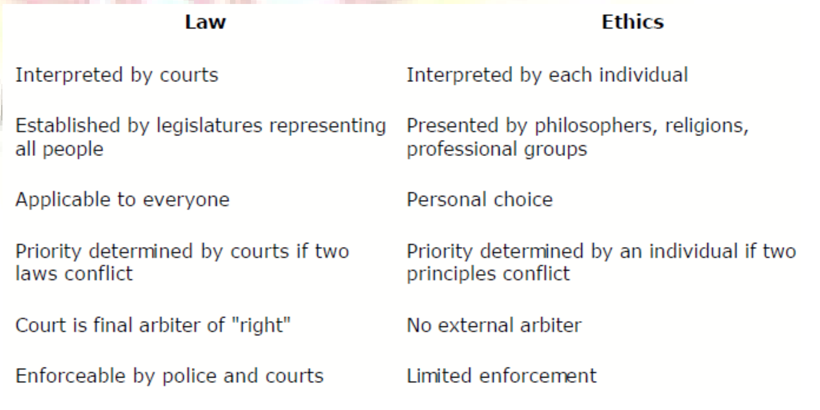
As we noted earlier, law is not always the appropriate way to deal with issues of human behavior. It is difficult to define a law to preclude only the events we want it to. For example, a law that restricts animals from public places must be refined to *permit* guide dogs for the blind. Lawmakers, who are not computer professionals, are hard pressed to think of all the exceptions when they draft a law concerning computer affairs. Even when a law is well conceived and well written, its enforcement may be difficult. The courts are overburdened, and prosecuting relatively minor infractions may be excessively time consuming relative to the benefit.

Thus, it is impossible or impractical to develop laws to describe and enforce all forms of behavior acceptable to society. Instead, society relies on ethics or morals to prescribe generally accepted standards of proper behavior. (In this section the terms ethics and morals are used interchangeably.) An ethic is an objectively defined standard of right and wrong. Ethical standards are often idealistic principles because they focus on one objective. In a given situation, however, several moral objectives may be involved, so people have to determine an action that is appropriate considering all the objectives. Even though religious groups and professional organizations promote certain standards of ethical behavior, ultimately each person is responsible for deciding what to do in a specific situation. Therefore, through our choices, each of us defines a personal set of ethical practices. A set of ethical principles is called an ethical system.

An ethic is different from a law in several important ways. First, laws apply to everyone: One may disagree with the intent or the meaning of a law, but that is not an excuse for disobeying the law. Second, the courts have a regular process for determining which law supersedes which if two laws conflict. Third, the laws and the courts identify certain actions as right and others as wrong. From a legal standpoint, anything that is not illegal is right. Finally, laws can be enforced to rectify wrongs done by unlawful behavior.
By contrast, ethics are personal: two people may have different frameworks for making moral judgments. What one person deems perfectly justifiable, another would never consider doing.

Second, ethical positions can and often do come into conflict. As an example, the value of a human life is very important in most ethical systems. Most people would not cause the sacrifice of one life, but in the right context some would approve of sacrificing one person to save another, or one to save many others. The value of one life cannot be readily measured against the value of others, and many ethical decisions must be founded on precisely this ambiguity. Yet, there is no arbiter of ethical positions: when two ethical goals collide, each person must choose which goal is dominant. Third, two people may assess ethical values differently; no universal standard of right and wrong exists in ethical judgments. Nor can one person simply look to what another has done as guidance for choosing the right thing to do. Finally, there is no enforcement for ethical choices.
These differences are summarized in Table 11-3.
Described by formal, written documents Described by unwritten principles

Table 11-3. Contrast of



Law	Ethics
Interpreted by courts	Interpreted by each individual
Established by legislatures representing all people	Presented by philosophers, religions, professional groups
Applicable to everyone	Personal choice
Priority determined by courts if two laws conflict	Priority determined by an individual if two principles conflict
Court is final arbiter of "right"	No external arbiter
Enforceable by police and courts	Limited enforcement

Studying Ethics

The study of ethics is not easy because the issues are complex. Sometimes people confuse ethics with religion because many religions supply a framework in which to make ethical choices. However, ethics can be studied apart from any religious connection. Difficult choices would be easier to make if there were a set of universal ethical principles to which everyone agreed. But the variety of social, cultural, and religious beliefs makes the identification of such a set of universal principles impossible. In this section we explore some of these problems and then consider how understanding ethics can help in dealing with issues of computer security.

Ethics and Religion

Ethics is a set of principles or norms for justifying what is right or wrong in a given situation. To understand what ethics *is* we may start by trying to understand what it *is not.* Ethical principles are different from religious beliefs. Religion is based on personal notions about the creation of the world and the existence of controlling forces or beings. Many moral principles are embodied in the major religions, and the basis of a personal morality is a matter of belief and conviction, much the same as for religions. However, two people with different religious backgrounds may develop the same ethical philosophy, while two exponents of the same religion might reach opposite ethical conclusions in a particular situation. Finally, we can analyze a situation from an ethical perspective and reach ethical conclusions without appealing to any particular religion or religious framework. Thus, it is important to distinguish ethics from religion.

Ethical Principles Are Not Universal

Ethical values vary by society, and from person to person within a society. For example, the concept of privacy is important in Western cultures. But in Eastern cultures, privacy is not desirable because people associate privacy with having something to hide. Not only is a Westerner's desire for privacy not understood but in fact it has a negative connotation.

Therefore, the attitudes of people may be affected by culture or background. Also, an individual's standards of behavior may be influenced by past events in life. A person who grew up in a large family may place greater emphasis on personal control and ownership of possessions than would an only child who seldom had to share. Major events or close contact with others can also shape one's ethical position. Despite these differences, the underlying principles of how to make moral judgment are the same.

Although these aspects of ethics are quite reasonable and understandable, they lead people to distrust ethics because it is not founded on basic principles all can accept. Also, people from a scientific or technical background expect precision and universality.

Ethics Does Not Provide Answers

Ethical pluralism is recognizing or admitting that more than one position may be ethically justifiableeven equally soin a given situation. Pluralism is another way of noting that two people may legitimately disagree on issues of ethics. We expect and accept disagreement in such areas as politics and religion.

However, in the scientific and technical fields, people expect to find unique, unambiguous, and unequivocal answers. In science, one answer must be correct or demonstrable in some sense.

Science has provided life with fundamental explanations. Ethics is rejected or misunderstood by some scientists because it is "soft," meaning that it has no underlying framework or it does not depend on fundamental truths.

One need only study the history of scientific discovery to see that science itself is founded largely on temporary truths. For many years the earth was believed to be the center of the solar system.

Ptolemy developed a complicated framework of epicycles, orbits within orbits of the planets, to explain the inconsistency of observed periods of rotation. Eventually his theory was superseded by the Copernican model of planets that orbit the sun. Similarly, Einstein's relativity theory opposed the traditional quantum basis of physics. Science is littered with theories that have fallen from favor as we learned or observed more and as new explanations were proposed. As each new theory is proposed, some people readily accept the new proposal, while others cling to the old.

But the basis of science is presumed to be "truth." A statement is expected to be provably true, provably false, or unproven, but a statement can never be both true and false. Scientists are uncomfortable with ethics because ethics does not provide these clean distinctions.

Worse, there is no higher authority of ethical truth. Two people may disagree on their opinion of the ethics of a situation, but there is no one to whom to appeal for a final determination of who is "right." Conflicting answers do not deter one from considering ethical issues in computer security.

Nor do they excuse us from making and defending ethical choices.

Ethical Reasoning

Most people make ethical judgments often, perhaps daily. (Is it better to buy from a hometown merchant or from a nationwide chain? Should I spend time with a volunteer organization or with my friends? Is it acceptable to release sensitive data to someone who might not have justification for access to that data?) Because we all engage in ethical choice, we should clarify how we do this so that we can learn to apply the principles of ethics in professional situations, as we do in private life.

Study of ethics can yield two positive results. First, in situations in which we already know what is right and what is wrong, ethics should help us justify our choice. Second, if we do not know the ethical action to take in a situation, ethics can help us identify the issues involved so that we can make reasoned judgments.

Examining a Case for Ethical Issues

How, then, can issues of ethical choice in computer security be approached? Here are several steps to making and justifying an ethical choice.

1. *Understand the situation.* Learn the facts of the situation. Ask questions of interpretation or clarification. Attempt to find out whether any relevant forces have not been considered.
2. *Know several theories of ethical reasoning.* To make an ethical choice, you have to know how those choices can be justified.
3. *List the ethical principles involved.* What different philosophies could be applied in this case?
Do any of these include others?
4. *Determine which principles outweigh others.* This is a subjective evaluation. It often involves extending a principle to a logical conclusion or determining cases in which one principle clearly supersedes another.

The most important steps are the first and third. Too often people judge a situation on incomplete information, a practice that leads to judgments based on prejudice, suspicion, or misinformation.

Considering all the different ethical issues raised forms the basis for evaluating the competing interests of step four.

Examples of Ethical Principles

There are two different schools of ethical reasoning: one based on the good that results from actions and one based on certain prima facie duties of people.

Consequence-Based Principles

The teleological theory of ethics focuses on the consequences of an action. The action to be chosen is that which results in the greatest future good and the least harm. For example, if a fellow student asks you to write a program he was assigned for a class, you might consider the good (he will owe you a favor) against the bad (you might get caught, causing embarrassment and possible disciplinary action, plus your friend will not learn the techniques to be gained from writing the program, leaving him deficient). The negative consequences clearly outweigh the positive, so you would refuse. *Teleology* is the general name applied to many theories of behavior, all of which focus on the goal, outcome, or consequence of the action.

There are two important forms of teleology. Egoism is the form that says a moral judgment is based on the positive benefits to the person taking the action. An egoist weighs the outcomes of all possible acts and chooses the one that produces the most personal good for him or her with the least negative consequence. The effects on other people are not relevant. For example, an egoist trying to justify the ethics of writing shoddy computer code when pressed for time might argue as follows. "If I complete the project quickly, I will satisfy my manager, which will bring me a raise and other good things. The customer is unlikely to know enough about the program to complain, so I am not likely to be blamed. My company's reputation may be tarnished, but that will not be tracked directly to me. Thus, I can justify writing shoddy code."

The principle of utilitarianism is also an assessment of good and bad results, but the reference group is the entire universe. The utilitarian chooses that action that will bring the greatest collective good for all people with the least possible negative for all. In this situation, the utilitarian would assess personal good and bad, good and bad for the company, good and bad for the customer, and, perhaps, good and bad for society at large. For example, a developer designing software to monitor smokestack emissions would need to assess its effects on everyone breathing. The utilitarian might perceive greater good to everyone by taking the time to write high-quality code, despite the negative personal consequence of displeasing management.

Rule-Based Principles

Another ethical theory is deontology, which is founded in a sense of duty. This ethical principle states that certain things are good in and of themselves. These things that are naturally good are good rules or acts, which require no higher justification. Something just *is* good; it does not have to be judged for its effect.

Examples (from Frankena [FRA73]) of intrinsically good things are

- truth, knowledge, and true opinion of various kinds; understanding, wisdom
- just distribution of good and evil; justice
- pleasure, satisfaction; happiness; life, consciousness
- peace, security, freedom
- good reputation, honor, esteem; mutual affection, love, friendship, cooperation; morally good dispositions or virtues
- beauty, aesthetic experience

Rule-deontology is the school of ethical reasoning that believes certain universal, self-evident, natural rules specify our proper conduct. Certain basic moral principles are adhered to because of our responsibilities to one another; these principles are often stated as rights: the right to know, the right to privacy, the right to fair compensation for work. Sir David Ross [ROS30] lists various duties incumbent on all human beings:

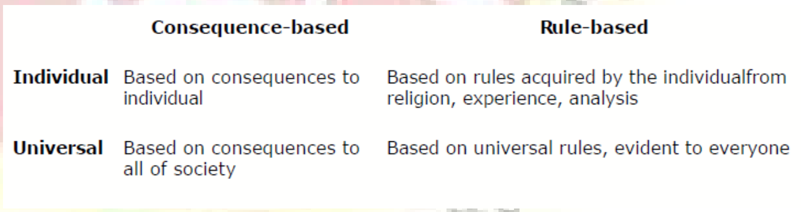
- *fidelity,* or truthfulness
- *reparation,* the duty to recompense for a previous wrongful act
- *gratitude,* thankfulness for previous services or kind acts
- *justice,* distribution of happiness in accordance with merit
- *beneficence,* the obligation to help other people or to make their lives better
- *nonmaleficence,* not harming others
- *self-improvement,* to become continually better, both in a mental sense and in a moral sense (for example, by not committing a wrong a second time)

Another school of reasoning is based on rules derived by each individual. Religion, teaching, experience, and reflection lead each person to a set of personal moral principles. The answer to an ethical question is found by weighing values in terms of what a person believes to be right behavior.

Summary of Ethical Theories

We have seen two bases of ethical theories, each applied in two ways. Simply stated, the two bases are consequence based and rule based, and the applications are either individual or universal. These theories are depicted in Table 11-4.

Table 11-4. Taxonomy of Ethical Theories.



	Consequence-based	**Rule-based**
Individual	Based on consequences to individual	Based on rules acquired by the individualfrom religion, experience, analysis
Universal	Based on consequences to all of society	Based on universal rules, evident to everyone

In the next section, we apply these theories to analyze certain situations that arise in the ethics of computer security.

Case studies of ethics.

To understand how ethics affects professional actions, ethicists often study example situations. The remainder of this section consists of several representative examples. These cases are modeled after ones developed by Parker [PAR79] as part of the AFIPS/NSF study of ethics in computing and technology. Each case study is designed to bring out certain ethical points, some of which are listed following the case. You should reflect on each case, determining for yourself what the most influential points are. These cases are suitable for use in a class discussion, during which other values will certainly be mentioned. Finally, each case reaches no conclusion because each individual must assess the ethical situation alone. In a class discussion it may be appropriate to take a vote. Remember, however, that ethics are not determined by majority rule.

Those siding with the majority are not "right," and the rest are not "wrong."

Case I: Use of Computer Services

This case concerns deciding what is appropriate use of computer time. Use of computer time is a question both of access by one person and of availability of quality of service to others. The person involved is permitted to access computing facilities for a certain purpose. Many companies rely on an unwritten standard of behavior that governs

the actions of people who have legitimate access to a computing system. The ethical issues involved in this case can lead to an understanding of that unwritten standard.
The Case
Dave works as a programmer for a large software company. He writes and tests utility programs such as compilers. His company operates two computing shifts: During the day program development and online applications are run; at night batch production jobs are completed. Dave has access to workload data and learns that the evening batch runs are complementary to daytime programming tasks; that is, adding programming work during the night shift would not adversely affect performance of the computer to other users.
Dave comes back after normal hours to develop a program to manage his own stock portfolio. His drain on the system is minimal, and he uses very few expendable supplies, such as printer paper.
Is Dave's behavior ethical?
Values Issues
Some of the ethical principles involved in this case are listed below.
 Ownership of resources. The company owns the computing resources and provides them for its own computing needs.
 Effect on others. Although unlikely, a flaw in Dave's program could adversely affect other users, perhaps even denying them service because of a system failure.
 Universalism principle. If Dave's action is acceptable, it should also be acceptable for others to do the same. However, too many employees working in the evening could reduce system effectiveness.
 Possibility of detection, punishment. Dave does not know whether his action would be wrong or right if discovered by his company. If his company decided it was improper use, Dave could be punished.
What other issues are involved? Which principles are more important than others?
Analysis
The utilitarian would consider the total excess of good over bad for all people. Dave receives benefit from use of computer time, although for this application the amount of time is not large.
Dave has a possibility of punishment, but he may rate that as unlikely. The company is neither harmed nor helped by this. Thus, the utilitarian could argue that Dave's use is justifiable.
The universalism principle seems as if it would cause a problem because clearly if everyone did this, quality of service would degrade. A utilitarian would say that each new user has to weigh good and bad separately. Dave's use might not burden the machine, and neither might Ann's; but when Bill wants to use the machine, it is heavily enough used that Bill's use *would* affect other people.
Alternative Situations
Would it affect the ethics of the situation if any of the following actions or characteristics were considered?
 Dave began a business managing stock portfolios for many people for profit.
 Dave's salary was below average for his background, implying that Dave was due the computer use as a fringe benefit.
 Dave's employer knew of other employees doing similar things and tacitly approved by not seeking to stop them.
 Dave worked for a government office instead of a private company and reasoned that the computer belonged "to the people."

Case II: Privacy Rights

In this case the central issue is the individual's right to privacy. Privacy is both a legal and an ethical issue because of the pertinent laws discussed in the previous section.
The Case
Donald works for the county records department as a computer records clerk, where he has access to files of property tax records. For a scientific study, a researcher, Ethel, has been granted access to the numerical portion but not the corresponding names of some records.
Ethel finds some information that she would like to use, but she needs the names and addresses corresponding with certain properties. Ethel asks Donald to retrieve the

names and addresses so she can contact these people for more information and for permission to do further study.
Should Donald release the names and addresses?
Some Principles Involved
Here are some of the ethical principles involved in this case. What are other ethical principles?
Which principles are subordinate to which others?
 Job responsibility. Donald's job is to manage individual records, not to make determinations of appropriate use. Policy decisions should be made by someone of higher authority.
 Use. The records are used for legitimate scientific study, not for profit or to expose sensitive data. (However, Ethel's access is authorized only for the numerical data, not for the private information relating property conditions to individuals.)
 Possible misuse. Although he believes Ethel's motives are proper, Donald cannot guarantee that Ethel will use the data only to follow up on interesting data items.
 Confidentiality. Had Ethel been intended to have names and addresses, they would have been given initially.
 Tacit permission. Ethel has been granted permission to access parts of these records for research purposes, so she should have access to complete her research.
 Propriety. Because Ethel has no authority to obtain names and addresses and because the names and addresses represent the confidential part of the data, Donald should deny Ethel's request for access.
Analysis
A rule-deontologist would argue that privacy is an inherent good and that one should not violate the privacy of another. Therefore, Donald should not release the names.
Extensions to the Basic Case
We can consider several possible extensions to the scenario. These extensions probe other ethical issues involved in this case.
• Suppose Donald were responsible for determining allowable access to the files.
What ethical issues would be involved in his deciding whether to grant access to Ethel?
• Should Ethel be allowed to contact the individuals involved? That is, should the health department release individuals' names to a researcher? What are the ethical issues for the health department to consider?
• Suppose Ethel contacts the individuals to ask their permission, and one-third of them respond giving permission, one-third respond denying permission, and one-third do not respond. Ethel claims that at least one-half of the individuals are needed to make a valid study. What options are available to Ethel? What are the ethical issues involved in deciding which of these options to pursue?
To show that ethics can be context dependent, let us consider some variations of the situation. Notice that these changes affect the domain of the problem, but not the basic question: access to personal data.
If the domain were medical records, the case would be covered by HIPAA, and so we would first consider a legal issue, not an ethical one. Notice, however, how the case changes subtly depending on the medical condition involved. You may reach one conclusion if the records deal with "ordinary" conditions (colds, broken legs, muscle injuries), but a different conclusion if the cases are for sexually transmitted diseases or AIDS. You may also reach a different conclusion if the research involves genetic conditions of which the subject may be unaware (for example, being a carrier for Huntington's disease or hemophilia).
But change the context once more, and consider web surfing habits. If Donald works for an Internet service provider and could determine all the web sites a person had visited, would that be fair to disclose?

Case III: Denial of Service

This case addresses issues related to the effect of one person's computation on other users. This situation involves people with legitimate access, so standard access controls should not exclude them. However, because of the actions of some, other people are denied legitimate access to the system. Thus, the focus of this case is on the rights of all users.
The Case
Charlie and Carol are students at a university in a computer science program. Each writes a program for a class assignment. Charlie's program happens to uncover a flaw in a

compiler that ultimately causes the entire computing system to fail; all users lose the results of their current computation. Charlie's program uses acceptable features of the language; the compiler is at fault. Charlie did not suspect his program would cause a system failure. He reports the program to the computing center and tries to find ways to achieve his intended result without exercising the system flaw.

The system continues to fail periodically, for a total of ten times (beyond the first failure). When the system fails, sometimes Charlie is running a program, but sometimes Charlie is not. The director contacts Charlie, who shows all of his program versions to the computing center staff.

The staff concludes that Charlie may have been inadvertently responsible for some, but not all, of the system failures, but that his latest approach to solving the assigned problem is unlikely to lead to additional system failures.

On further analysis, the computing center director notes that Carol has had programs running each of the first eight (of ten) times the system failed. The director uses administrative privilege to inspect Carol's files and finds a file that exploits the same vulnerability as did Charlie's program. The director immediately suspends Carol's account, denying Carol access to the computing system. Because of this, Carol is unable to complete her assignment on time, she receives a D in the course, and she drops out of school.

Analysis

In this case the choices are intentionally not obvious. The situation is presented as a completed scenario, but in studying it you are being asked to suggest alternative actions the players *could have taken*. In this way, you build a repertoire of actions that you can consider in similar situations that might arise.

• What additional information is needed?

• Who has rights in this case? What rights are those? Who has a responsibility to protect those rights? (This step in ethical study is used to clarify who should be considered as the reference group for a deontological analysis.)

• Has Charlie acted responsibly? By what evidence do you conclude so? Has Carol? How? Has the computing center director acted responsibly? How? (In this step you look for past judgments that should be confirmed or wrongs that should be redressed.)

• What are some alternative actions Charlie or Carol or the director could have taken that would have been more responsible?

Case IV: Ownership of Programs

In this case we consider who owns programs: the programmer, the employer, the manager, or all. From a legal standpoint, most rights belong to the employer, as presented earlier in this chapter. However, this case expands on that position by presenting several competing arguments that might be used to support positions in this case. As described in the previous section, legal controls for secrecy of programs can be complicated, time consuming, and expensive to apply. In this case we search for individual ethical controls that can prevent the need to appeal to the legal system.

The Case

Greg is a programmer working for a large aerospace firm, Star Computers, which works on many government contracts; Cathy is Greg's supervisor. Greg is assigned to program various kinds of simulations.

To improve his programming abilities, Greg writes some programming tools, such as a cross-reference facility and a program that automatically extracts documentation from source code. These are not assigned tasks for Greg; he writes them independently and uses them at work, but he does not tell anyone about them. Greg has written them in the evenings, at home, on his personal computer.

Greg decides to market these programming aids by himself. When Star's management hears of this, Cathy is instructed to tell Greg that he has no right to market these products since, when he was employed, he signed a form stating that all inventions become the property of the company. Cathy does not agree with this position because she knows that Greg has done this work on his own. She reluctantly tells Greg that he cannot market these products. She also asks Greg for a copy of the products.

Cathy quits working for Star and takes a supervisory position with Purple Computers, a competitor of Star. She takes with her a copy of Greg's products and distributes it to the people who work with her. These products are so successful that they substantially improve the effectiveness of her employees, and Cathy is praised by her management and receives a healthy bonus. Greg hears of this, and contacts Cathy, who contends that because the product was determined to belong to Star and because Star worked largely on government funding, the products were really in the public domain and therefore they belonged to no one in particular.

Analysis

This case certainly has major legal implications. Probably everyone could sue everyone else and, depending on the amount they are willing to spend on legal expenses, they could keep the cases in the courts for several years. Probably no judgment would satisfy all.

Let us set aside the legal aspects and look at the ethical issues. We want to determine who might have done what, and what changes might have been possible to prevent a tangle for the courts to unscramble.

First, let us explore the principles involved.

• *Rights.* What are the respective rights of Greg, Cathy, Star, and Purple?

• *Basis.* What gives Greg, Cathy, Star, and Purple those rights? What principles of fair play, business, property rights, and so forth are involved in this case?

• *Priority.* Which of these principles are inferior to which others? Which ones take precedence? (Note that it may be impossible to compare two different rights, so the outcome of this analysis may yield some rights that are important but that cannot be ranked first, second, third.)

• *Additional information.* What additional facts do you need in order to analyze this case? What assumptions are you making in performing the analysis?

Next, we want to consider what events led to the situation described and what alternative actions could have prevented the negative outcomes.

• What could Greg have done differently before starting to develop his product?

After developing the product? After Cathy explained that the product belonged to Star?

• What could Cathy have done differently when she was told to tell Greg that his products belonged to Star? What could Cathy have done differently to avert this decision by her management? What could Cathy have done differently to prevent the clash with Greg after she went to work at Purple?

• What could Purple have done differently upon learning that it had products from Star (or from Greg)?

• What could Greg and Cathy have done differently after Greg spoke to Cathy at Purple?

• What could Star have done differently to prevent Greg from feeling that he owned his products? What could Star have done differently to prevent Cathy from taking the products to Purple?

Case V: Proprietary Resources

In this case, we consider the issue of access to proprietary or restricted resources. Like the previous one, this case involves access to software. The focus of this case is the rights of a software developer in contrast with the rights of users, so this case concerns determining legitimate access rights.

The Case

Suzie owns a copy of G-Whiz, a proprietary software package she purchased legitimately. The software is copyrighted, and the documentation contains a license agreement that says that the software is for use by the purchaser only. Suzie invites Luis to look at the software to see if it will fit his needs. Luis goes to Suzie's computer and she demonstrates the software to him. He says he likes what he sees, but he would like to try it in a longer test.

Extensions to the Case

So far the actions have all been ethically sound. The next steps are where ethical responsibilities arise. Take each of the following steps as independent; that is, do not assume that any of the other steps has occurred in your analysis of one step.

 Suzie offers to copy the disk for Luis to use.

 Suzie copies the disk for Luis to use, and Luis uses it for some period of time.

 Suzie copies the disk for Luis to use; Luis uses it for some period of time and then buys a copy for himself.

 Suzie copies the disk for Luis to try out overnight, under the restriction that he must bring the disk back to her tomorrow and must not copy it for himself. Luis does so.

 Suzie copies the disk with the same restrictions, but Luis makes a copy for himself before returning it to Suzie.
 Suzie copies the disk with the same restrictions, and Luis makes a copy for himself, but he then purchases a copy.
 Suzie copies the disk with the same restrictions, but Luis does not return it.
For each of these extensions, describe who is affected, which ethical issues are involved, and which principles override which others.

Case VI: Fraud

In previous cases, we have dealt with people acting in situations that were legal or, at worst, debatable. In this case, we consider outright fraud, which is illegal. However, the case really concerns the actions of people who are asked to do fraudulent things.

The Case

Alicia works as a programmer in a corporation. Ed, her supervisor, tells her to write a program to allow people to post entries directly to the company's accounting files ("the books"). Alicia knows that ordinarily programs that affect the books involve several steps, all of which have to balance. Alicia realizes that with the new program, it will be possible for one person to make changes to crucial amounts, and there will be no way to trace who made these changes, with what justification, or when.

Alicia raises these concerns to Ed, who tells her not to be concerned, that her job is simply to write the programs as he specifies. He says that he is aware of the potential misuse of these programs, but he justifies his request by noting that periodically a figure is mistakenly entered in the books and the company needs a way to correct the inaccurate figure.

Extensions

First, let us explore the options Alicia has. If Alicia writes this program, she might be an accomplice to fraud. If she complains to Ed's superior, Ed or the superior might reprimand or fire her as a troublemaker. If she refuses to write the program, Ed can clearly fire her for failing to carry out an assigned task. We do not even know that the program is desired for fraudulent purposes; Ed suggests an explanation that is not fraudulent.

She might write the program but insert extra code that creates a secret log of when the program was run, by whom, and what changes were made. This extra file could provide evidence of fraud, or it might cause trouble for Alicia if there is no fraud but Ed discovers the secret log.

At this point, here are some of the ethical issues involved.

• Is a programmer responsible for the programs he or she writes? Is a programmer responsible for the results of those programs? (In contemplating this question, suppose the program were to adjust dosage in a computer-controlled medical application, and Ed's request were for a way to override the program controls to cause a lethal dosage. Would Alicia then be responsible for the results of the program?)
• Is a programmer merely an employee who follows orders (assigned tasks) unthinkingly?
• What degree of personal risk (such as possible firing) is an employee obliged to accept for opposing an action he or she thinks is improper?
• Would a program to manipulate the books as described here ever be justified? If so, in what circumstances would it be justified?
• What kinds of controls can be placed on such programs to make them acceptable?
What are some ways that a manager could legitimately ask an employee to write a program like this?
• Would the ethical issues in this situation be changed if Alicia designed and wrote this program herself?

Analysis

The act-deontologist would say that truth is good. Therefore, if Alicia thought the purpose of the program was to deceive, writing it would not be a good act. (If the purpose were for learning or to be able to admire beautiful code, then writing it might be justifiable.)

A more useful analysis is from the perspective of the utilitarian. To Alicia, writing the program brings possible harm for being an accomplice to fraud, with the gain of having cooperated with her manager. She has a possible item with which to blackmail Ed, but Ed might also turn on her and say the program was her idea. On balance, this option seems to have a strong negative slant.

By not writing the program her possible harm is being fired. However, she has a potential gain by being able to "blow the whistle" on Ed. This option does not seem to bring her much good, either. But fraudulent acts have negative consequences for the stockholders, the banks, and other innocent employees. Not writing the program brings only personal harm to Alicia, which is similar to the harm described earlier. Thus, it seems as if not writing the program is the more positive option.

There is another possibility. The program may *not* be for fraudulent purposes. If so, then there is no ethical conflict. Therefore, Alicia might try to determine whether Ed's motives are fraudulent.

Case VII: Accuracy of Information

For our next case, we consider responsibility for accuracy or integrity of information. Again, this is an issue addressed by database management systems and other access control mechanisms.

However, as in previous cases, the issue here is access by an *authorized* user, so the controls do not prevent access.

The Case

Emma is a researcher at an institute where Paul is a statistical programmer. Emma wrote a grant request to a cereal manufacturer to show the nutritional value of a new cereal, Raw Bits. The manufacturer funded Emma's study. Emma is not a statistician. She has brought all of her data to Paul to ask him to perform appropriate analyses and to print reports for her to send to the manufacturer. Unfortunately, the data Emma has collected seem to refute the claim that Raw Bits is nutritious, and, in fact, they may indicate that Raw Bits is harmful.

Paul presents his analyses to Emma but also indicates that some other correlations could be performed that would cast Raw Bits in a more favorable light. Paul makes a facetious remark about his being able to use statistics to support either side of any issue.

Ethical Concerns

Clearly, if Paul changed data values in this study, he would be acting unethically. But is it any more ethical for him to suggest analyzing correct data in a way that supports two or more different conclusions? Is Paul obligated to present both the positive and the negative analyses?

Is Paul responsible for the use to which others put his program results?

If Emma does not understand statistical analysis, is she acting ethically in accepting Paul's positive conclusions? His negative conclusions? Emma suspects that if she forwards negative results to the manufacturer, they will just find another researcher to do another study. She suspects that if she forwards both sets of results to the manufacturer, they will publicize only the positive ones. What ethical principles support her sending both sets of data? What principles support her sending just the positive set? What other courses of action has she?

Case VIII: Ethics of Hacking or Cracking

What behavior is acceptable in cyberspace? Who owns or controls the Internet? Does malicious or nonmalicious intent matter? Legal issues are involved in the answers to these questions, but as we have pointed out previously, laws and the courts cannot protect everything, nor should we expect them to. Some people separate investigating computer security vulnerabilities from exploiting them, calling the former "white hat" hacking and the latter "black hat." It is futile to try to stop people from learning nor should we even try, for the sake of society, as Cross [CRO06] points out. There is reasonable debate over publication or dissemination of knowledge: Is theworld safer if only a few are allowed to know how to build sophisticated weapons? Or how to break certain security systems? Is the public better served by open knowledge of system vulnerabilities? We recommend students, researchers, faculty, and technologists, and certainly users, join in thoughtful debate of this issue, one of the largest ethical matters in our field.

In this final case study we consider ethical behavior in a shared-use computing environment, such as the Internet. The questions are similar to "what behavior is acceptable in outer space?" or "who owns the oceans?"

Goli is a computer security consultant; she enjoys the challenge of finding and fixing security vulnerabilities. Independently wealthy, she does not need to work, so she has ample spare time in which to test the security of systems.

In her spare time, Goli does three things: First, she aggressively attacks commercial products for vulnerabilities. She is quite proud of the tools and approach she has developed, and she is quite successful at finding flaws. Second, she probes accessible systems on the Internet, and when she finds vulnerable sites, she contacts the owners to offer her services repairing the problems.

Finally, she is a strong believer in high-quality pastry, and she plants small programs to slow performance in the web sites of pastry shops that do not use enough butter in their pastries. Let us examine these three actions in order.

Vulnerabilities in Commercial Products

We have already described a current debate regarding the vulnerability reporting process. Now let us explore the ethical issues involved in that debate. Clearly from a rule-based ethical theory, attackers are wrong to perform malicious attacks. The appropriate theory seems to be one of consequence: who is helped or hurt by finding and publicizing flaws in products? Relevant parties are attackers, the vulnerability finder, the vendor, and the using public. Notoriety or credit for finding the flaw is a small interest. And the interests of the vendor (financial, public relations) are less important than the interests of users to have secure products. But how are the interests of users best served?

- *Full disclosure* helps users assess the seriousness of the vulnerability and apply appropriate protection. But it also gives attackers more information with which to formulate attacks. Early full disclosure before the vendor has countermeasures ready may actually harm users by leaving them vulnerable to a now widely known attack.
- *Partial disclosure* the general nature of the vulnerability but not a detailed exploitation Scenario may forestall attackers. One can argue that the vulnerability details are there to be discovered; when a vendor announces a patch for an unspecified flaw in a product, the attackers will test that product aggressively and study the patch carefully to try to determine the vulnerability. Attackers will then spread a complete description of the vulnerability to other attackers through an underground network, and attacks will start against users who may not have applied the vendor's fix.
- *No disclosure*. Perhaps users are best served by a scheme in which every so often new code is released, sometimes fixing security vulnerabilities, sometimes fixing things that are not security related, and sometimes adding new features. But without a sense of significance or urgency, users may not install this new code.

Searching for Vulnerabilities and Customers

What are the ethical issues involved in searching for vulnerabilities? Again, the party of greatest interest is the user community and the good or harm that can come from the search.

On the positive side, searching may find vulnerabilities. Clearly, it would be wrong for Goli to report vulnerabilities that were not there simply to get work, and it would also be wrong to report some but not all vulnerabilities to be able to use the additional vulnerabilities as future leverage against the client.

But suppose Goli does a diligent search for vulnerabilities and reports them to the potential client. Is that not similar to a service station owner's advising you that a headlight is not operating when you take your car in for gasoline? Not quite, you might say. The headlight flaw can be seen without any possible harm to your car; probing for vulnerabilities might cause your system to fail.

The ethical question seems to be which is greater: the potential for good or the potential for harm? And if the potential for good is stronger, how much stronger does it need to be to override the risk of harm?

This case is also related to the common practice of ostensible nonmalicious probing for vulnerabilities: Hackers see if they can access your system without your permission, perhaps by guessing a password. Spafford [SPA98] points out that many crackers simply want to look around, without damaging anything. As discussed in Sidebar 11-4, Spafford compares this seemingly innocent activity with entry into your house when the door is unlocked. Even when done without malicious intent, cracking can be a serious offense; at its worst, it has caused millions of dollars in damage. Although crackers are prosecuted severely with harsh penalties, cracking continues to be an appealing crime, especially to juveniles.

Politically Inspired Attacks

Finally, consider Goli's interfering with operation of web sites whose actions she opposes. We have purposely phrased the issue in a situation that arouses perhaps only a few gourmands and pâtissiers. We can dismiss the interest of the butter fans as an insignificant minority on an insignificant issue. But you can certainly think of many other issues that have brought on wars. (See Denning's excellent article on cybercriminals [DEN99a] for real examples of politically motivated computer activity.)

The ethical issues abound in this scenario. Some people will see the (butter) issue as one of inherent good, but is butter use one of the fundamental good principles, such as honesty or fairness or not doing harm to others? Is there universal agreement that butter use is good?

Probably there will be a division of the world into the butter advocates (x%), the unrestricted pastry advocates (y%), and those who do not take a position (z%). By how much does x have to exceed y for Goli's actions to be acceptable? What if the value of z is large? Greatest good for the greatest number requires a balance among these three percentages and some measure of benefit or harm.

Is butter use so patently good that it justifies harm to those who disagree? Who is helped and who suffers? Is the world helped if only good, but more expensive, pastries are available, so poor people can no longer afford pastry? Suppose we could determine that 99.9 percent of people in the world agreed that butter use was a good thing. Would that preponderance justify overriding the interests of the other 0.1 percent?

Codes of Ethics

Because of ethical issues such as these, various computer groups have sought to develop codes of ethics for their members. Most computer organizations, such as the Association for Computing

Machinery (ACM), the Institute of Electrical and Electronics Engineers (IEEE), and the Data Processing Management Association (DPMA), are voluntary organizations. Being a member of one of these organizations does not certify a level of competence, responsibility, or experience in computing. For these reasons, codes of ethics in these organizations are primarily advisory.

Nevertheless, these codes are fine starting points for analyzing ethical issues. IEEE
The IEEE has produced a code of ethics for its members. The IEEE is an organization of engineers, not limited to computing. Thus, their code of ethics is a little broader than might be expected for computer security, but the basic principles are applicable in computing situations.
The IEEE Code of Ethics is shown in Figure 11-1.

Figure 11-1. IEEE Code of Ethics. (Reprinted courtesy of the Institute of Electrical and Electronics Engineers © 1996.)

We, the members of the IEEE, in recognition of the importance of our technologies in affecting the quality of life throughout the world, and in accepting a personal obligation to our profession, its members, and the communities we serve, do hereby commit ourselves to conduct of the highest ethical and professional manner and agree might endanger the public or the environment;

1. to accept responsibility in making engineering decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that
2. to avoid real or perceived conflicts of interest whenever possible, and to disclose them to affected parties when they do exist;
3. to be honest and realistic in stating claims or estimates based on available data;
4. to reject bribery in all of its forms;
5. to improve understanding of technology, its appropriate application, and potential consequences;
6. to maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations;
7. to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others;
8. to treat fairly all persons regardless of such factors as race, religion, gender, disability, age, or national origin;

9. to avoid injuring others, their property, reputation, or employment by false or malicious action;
10. to assist colleagues and coworkers in their professional development and to support them in following this code of ethics.

ACM

The ACM code of ethics recognizes three kinds of responsibilities of its members: general moral imperatives, professional responsibilities, and leadership responsibilities, both inside the association and in general. The code of ethics has three sections (plus a fourth commitment section), as shown in Figure 11-2.

Figure 11-2. ACM Code of Ethics and Professional Conduct.
(Reprinted courtesy of the Association for Computing Machinery © 1993.)

As an ACM member I will ...
1.1 Contribute to society and human well-being
1.2 Avoid harm to others
1.3 Be honest and trustworthy
1.4 Be fair and take action not to discriminate
1.5 Honor property rights including copyrights and patents
1.6 Give proper credit for intellectual property
1.7 Respect the privacy of others
1.8 Honor confidentiality
As an ACM computing professional I will ...
2.1 Strive to achieve the highest quality, effectiveness, and dignity in both the process and products of professional work
2.2 Acquire and maintain professional competence
2.3 Know and respect existing laws pertaining to professional work
2.4 Accept and provide appropriate professional review
2.5 Give comprehensive and thorough evaluations of computer systems and their impacts, including analysis of possible risks
2.6 Honor contracts, agreements, and assigned responsibilities
2.7 Improve public understanding of computing and its consequences
2.8 Access computing and communication resources only when authorized to do so
As an ACM member and an organization leader, I will ...
3.1 Articulate social responsibilities of members of an organizational unit and encourage full acceptance of those responsibilities
3.2 Manage personnel and resources
3.3 Acknowledge and support proper and authorized uses of an organization's computing and communication resources
3.4 Ensure that users and those who will be affected by a system have their needs clearly articulated during the assessment and design of requirements; later the system must be validated to meet requirements
3.5 Articulate and support policies that protect the dignity of users and others affected by a computing system
3.6 Create opportunities for members of the organization to learn the principles and limitations of computer systems
As an ACM member, I will ...
4.1 Uphold and promote the principles of this code
4.2 Treat violations of this code as inconsistent with membership in the ACM

Computer Ethics Institute

The Computer Ethics Institute is a nonprofit group that aims to encourage people to consider the ethical aspects of their computing activities. The organization has been in existence since the mid-1980s, founded as a joint activity of IBM, the Brookings Institution, and the Washington Theological Consortium. The group has published its ethical guidance as ten commandments of computer ethics, listed in Figure 11-3.

Figure 11-3. The Ten Commandments of Computer Ethics.
(Reprinted with permission, Computer Ethics Institute, Washington, D.C.)

1. Thou shalt not use a computer to harm other people.

2. Thou shalt not interfere with other people's computer work.
3. Thou shalt not snoop around in other people's computer files.
4. Thou shalt not use a computer to steal.
5. Thou shalt not use a computer to bear false witness.
6. Thou shalt not copy or use proprietary software for which you have not paid.
7. Thou shalt not use other people's computer resources without authorization or proper compensation.
8. Thou shalt not appropriate other people's intellectual output.
9. Thou shalt think about the social consequences of the program you are writing or the system you are designing.
10. Thou shalt always use a computer in ways that insure consideration and respect for your fellow humans.

Many organizations take ethics seriously and produce a document guiding the behavior of its members or employees. Some corporations require new employees to read its code of ethics and sign a promise to abide by it. Others, especially at universities and research centers, have special boards that must approve proposed research and ensure that projects and team members act ethically. As an individual professional, it may be useful for you to review these codes of ethics and compose a code of your own, reflecting your ideas about appropriate behavior in likely situations. A code of ethics can help you assess situations quickly and act in a consistent, comfortable, and ethical manner.

Conclusion of Computer Ethics

In this study of ethics, we have tried not to decide right and wrong, or even to brand certain acts as ethical or unethical. The purpose of this section is to stimulate thinking about ethical issues concerned with confidentiality, integrity, and availability of data and computations.

The cases presented show complex, conflicting ethical situations. The important first step in acting ethically in a situation is to obtain the facts, ask about any uncertainties, and acquire any additional information needed. In other words, first we must understand the situation.

The second step is to identify the ethical principles involved. Honesty, fair play, proper compensation, and respect for privacy are all ethical principles. Sometimes these conflict, and then we must determine which principles are more important than others. This analysis may not lead to one principle that obviously overshadows all others. Still, a ranking to identify the major principles involved is needed.

The third step is choosing an action that meets these ethical principles. Making a decision and taking action are difficult, especially if the action has evident negative consequences. However, taking action based on a *personal* ranking of principles is necessary. The fact that other equally sensible people may choose a different action does not excuse us from taking some action.

This section is not trying to force the development of rigid, inflexible principles. Decisions may vary, based on fine differences between two situations. Or a person's views can change over time in response to experience and changing context. Learning to reason about ethical situations is not quite the same as learning "right" from "wrong." Terms such as *right* and *wrong* or *good* and *bad* imply a universal set of values. Yet we know that even widely accepted principles are overridden by some people in some situations. For example, the principle of not killing people may be violated in the case of war or capital punishment. Few, if any, values are held by everyone or in all cases. Therefore, our purpose in introducing this material has been to stimulate you to recognize and think about ethical principles involved in cases related to computer security. Only by recognizing and analyzing principles can you act consistently, thoughtfully, and responsibly.

1. Recommended Texts

1. C. P. Pfleeger, and S. L. Pfleeger, Security in Computing, Pearson Education, 4th Edition, 2003

2. Matt Bishop, Computer Security: Art and Science, Pearson Education, 2003.

2. Reference Books

1. Stallings, Cryptography And Network Security: Principles and practice, 4th Edition, 2006
2. Kaufman, Perlman, Speciner, Network Security, Prentice Hall, 2nd Edition, 2003
3. Eric Maiwald, Network Security : A Beginner's Guide, TMH, 1999
4. Macro Pistoia, Java Network Security, Pearson Education, 2nd Edition, 1999
5. Whitman, Mattord, Principles of information security, Thomson, 2nd Edition, 2005

******