Computer Security

Asst. Prof. Dr. Ali Kadhim

Chapter 1. Is There a Security Problem in Computing?

In this chapter

- The risks involved in computing
- The goals of secure computing: confidentiality, integrity, availability
- The threats to security in computing: interception, interruption, modification, fabrication
- Controls available to address these threats: encryption, programming controls, operating systems, network controls, administrative controls, law, and ethics

1.1. What Does "Secure" Mean?

How do we protect our most valuable assets? One option is to place them in a safe place, like a bank. We seldom hear of a bank robbery these days, even though it was once a fairly lucrative undertaking. In the American Wild West, banks kept large amounts of cash on hand, as well as gold and silver, which could not be traced easily. In those days, cash was much more commonly used than checks. Communications and transportation were primitive enough that it might have been hours before the legal authorities were informed of a robbery and days before they could actually arrive at the scene of the crime, by which time the robbers were long gone. To control the situation, a single guard for the night was only marginally effective. Should you have wanted to commit a robbery, you might have needed only a little common sense and perhaps several days to analyze the situation; you certainly did not require much sophisticated training. Indeed, you usually learned on the job, assisting other robbers in a form of apprenticeship. On balance, all these factors tipped very much in the favor of the criminal, so bank robbery was, for a time, considered to be a profitable business. Protecting assets was difficult and not always effective.

Today, however, asset protection is easier, with many factors working against the potential criminal. Very sophisticated alarm and camera systems silently protect secure places like banks whether people are around or not. The techniques of criminal investigation have become so effective that a person can be identified by genetic material (DNA), fingerprints, retinal patterns, voice, a composite sketch, ballistics evidence, or other hard-to-mask characteristics. The assets are stored in a safer form. For instance, many bank branches now contain less cash than some large retail stores because much of a bank's business is conducted with checks, electronic transfers, credit cards, or debit cards. Sites that must store large amounts of cash or currency are protected with many levels of security: several layers of physical systems, complex locks, multiple-party systems requiring the agreement of several people to allow access, and other schemes. Significant improvements in transportation and communication mean that police can be at the scene of a crime in minutes; dispatchers can alert other officers in seconds about the suspects to watch for. From the criminal's point of view, the risk and required sophistication are so high that there are usually easier ways than bank robbery to make money.

Protecting Valuables

This book is about protecting our computer-related assets, not about protecting our money and gold bullion. That is, we plan to discuss security for computing systems, not banks. But we can learn from our analysis of banks because they tell us some general principles about protection. In other words, when we think about protecting valuable information, we can learn a lot from the way we have protected other valuables in the past. For example, Table 1-1 presents the differences between how people protect computing systems and how banks protect money. The table reinforces the point that we have many challenges to address when protecting computers and data, but the nature of the challenges may mean that we need different and more effective approaches than we have used in the past.

Table 1-1. Protecting Money vs. Protecting Information.

Characteristic	Bank Protecting Money	People Protecting Information
Size and portability	Sites storing money are large, unwieldy, not at all portable. Buildings require guards, vaults, many levels of physical security to protect money.	Items storing valuable assets are very small and portable. The physical devices in computing can be so small that thousands of dollars worth of computing gear can fit comfortably in a briefcase.
Ability to avoid physical contact	Difficult. When banks deal with physical currency, a criminal must physically demand the money and carry it away from the bank's premises.	Simple. When information is handled electronically, no physical contact is necessary. Indeed, when banks handle money electronically, almost all transactions can be done without any physical contact. Money can be transferred through computers, mail, or telephone.
Value of assets	Very high.	Variable, from very high to very low. Some information, such as medical history, tax payments, investments, or educational background, is confidential. Other information, about troop movements, sales strategies, buying patterns, can be very sensitive. Still other information, such as address and phone number, may be of no consequence and easily accessible by other means.

Protecting our valuables, whether they are expressed as information or in some other way, ranges from quite unsophisticated to very sophisticated. We can think of the Wild West days as an example of the "unsophisticated" end of the security spectrum. And even today, when we have more sophisticated means of protection than ever before, we still see a wide range in how people and businesses actually use the protections available to them.

In fact, we can find far too many examples of computer security that seem to be back in the Wild West days. Although some organizations recognize computers and their data as valuable and vulnerable resources and have applied appropriate protection, others are dangerously deficient in their security measures. In some cases, the situation is even worse than that in the Wild West; as Sidebar 1-1 illustrates, some enterprises do not even recognize that their resources should be controlled and protected. And as software consumers, we find that the lack of protection is all the more dangerous when we are not even aware that we are susceptible to software piracy or corruption.

Sidebar 1-1: Protecting Software in Automobile Control Systems

The amount of software installed in an automobile grows larger from year to year. Most cars, especially more expensive ones, use dozens of microcontrollers to provide a variety of features to entice buyers. There is enough variation in microcontroller range and function that the Society of Automotive Engineers (Warrendale, Pennsylvania) has set standards for the U.S. automotive industry's software. Software in the microcontrollers ranges through three classes:

- low speed (class Aless than 10 kb per second) for convenience features, such as radios
- medium speed (class B10 to 125 kb per second) for the general transfer of information, such as that related to emissions, speed, or instrumentation
- high speed (class Cmore than 125 kb per second) for real-time control, such as the power train or a brake-by-wire system

These digital cars use software to control individual subsystems, and then more software to connect the systems in a network [WHI01].

However, the engineers designing and implementing this software see no reason to protect it from hackers. Whitehorn-Umphres reports that, from the engineers' point of view, the software is too complicated to be understood by a hacker. "And even if they could [understand it], they wouldn't want to."

Whitehorn-Umphres points out a major difference in thinking between hardware designers and software designers. "As hardware engineers, they assumed that, perhaps aside from bolt-on aftermarket parts, everything else is and should be a black box." But software folks have a different take: "As a software designer, I assume that all digital technologies are fair game for being played with. . . . it takes a special kind of personality to look at a software-enabled device and see the potential for manipulation and change a hacker personality."

He points out that hot-rodders and auto enthusiasts have a long history of tinkering and tailoring to make specialized changes to mass-produced cars. And the unprotected software beckons them to continue the tradition. For instance, there are reports of recalibrating the speedometer of two types of Japanese motorcycles to fool the bike about how fast it is really going (and thereby enabling faster-than-legal speeds). Whitehorn-Umphres speculates that soon you will be able to "download new ignition mappings from your PC. The next step will be to port the PC software to handheld computers so as to make on-the-road modifications that much easier."

The possibility of crime is bad enough. But worse yet, in the event of a crime, some organizations neither investigate nor prosecute for fear that the revelation will damage their public image. For example, would you feel safe depositing your money in a bank that had just suffered a several million-dollar loss through computer-related embezzlement? In fact, the breach of security makes that bank painfully aware of all its security weaknesses. Once bitten, twice shy; after the loss, the bank will probably enhance its security substantially, quickly becoming safer than a bank that had not been recently victimized.

Even when organizations want to take action against criminal activity, criminal investigation and prosecution can be hindered by statutes that do not recognize electromagnetic signals as property. The news media sometimes portrays computer intrusion by teenagers as a prank no more serious than tipping over an outhouse. But, as we see in later chapters, computer intrusion can hurt businesses and even take lives. The legal systems around the world are rapidly coming to grips with the nature of electronic property as intellectual property critical to organizational or mission success; laws are being implemented and court decisions declared that acknowledge the value of information stored or transmitted via computers. But this area is still new to many courts, and few precedents have been established.

Throughout this book, we look at examples of how computer security affects our lives directly and indirectly. And we examine techniques to prevent security breaches or at least to mitigate their effects. We address the security concerns of software practitioners as well as those professionals, managers, and users whose products, services, and well-being depend on the proper functioning of computer systems. By studying this book, you can develop an understanding of the basic problems underlying computer security and the methods available to deal with them.

In particular, we do the following:

- examine the risks of security in computing
- consider available countermeasures or controls
- stimulate thought about uncovered vulnerabilities
- identify areas where more work is needed

In this chapter, we begin by examining what kinds of vulnerabilities computing systems are prone to. We then consider why these vulnerabilities are exploited: the different kinds of attacks that are possible. This chapter's third focus is on who is involved: the kinds of people who contribute to the security problem. Finally, we introduce how to prevent possible attacks on systems.

Characteristics of Computer Intrusion

Any part of a computing system can be the target of a crime. When we refer to a **computing system**,[1] we mean a collection of hardware, software, storage media, data, and people that an organization uses to perform computing tasks. Sometimes, we assume that parts of a computing system are not valuable to an outsider, but often we are mistaken. For instance, we tend to think that the most valuable property in a bank is the cash, gold, or silver in the vault. But in fact the customer information in the bank's computer may be far more valuable. Stored on paper, recorded on a storage medium, resident in memory, or transmitted over telephone lines or satellite links, this information can be used in myriad ways to make money illicitly. A competing bank can use this information to steal clients or even to disrupt service and discredit the bank. An unscrupulous individual could move money from one account to another without the owner's permission. A group of con artists could contact large depositors and convince them to invest in fraudulent schemes. The variety of targets and attacks makes computer security very difficult.

[1] In this book, boldface identifies new terms being introduced.

Any system is most vulnerable at its weakest point. A robber intent on stealing something from your house will not attempt to penetrate a two-inch-thick metal door if a window gives easier access. Similarly, a sophisticated perimeter physical security system does not compensate for unguarded access by means of a simple telephone line and a modem. We can codify this idea as one of the principles of computer security.

Principle of Easiest Penetration: An intruder must be expected to use any available means of penetration. The penetration may not necessarily be by the most obvious means, nor is it necessarily the one against which the most solid defense has been installed. And it certainly does not have to be the way we want the attacker to behave.

This principle implies that computer security specialists must consider all possible means of penetration. Moreover, the penetration analysis must be done repeatedly, and especially whenever the system and its security change. People sometimes underestimate the determination or creativity of attackers. Remember that computer security is a game with rules only for the defending team: The attackers can (and will) use any means they can. Perhaps the hardest thing for people outside the security community to do is to think like the attacker. One group of creative security researchers investigated a wireless security system and reported a vulnerability to the system's chief designer, who replied "that would work, but no attacker would try it" [BON06]. Don't believe that for a minute: No attack is out of bounds.

Strengthening one aspect of a system may simply make another means of penetration more appealing to intruders. For this reason, let us look at the various ways by which a system can be breached.

1.2. Attacks

When you test any computer system, one of your jobs is to imagine how the system could malfunction. Then, you improve the system's design so that the system can withstand any of the problems you have identified. In the same way, we analyze a system from a security perspective, thinking about ways in which the system's security can malfunction and diminish the value of its assets.

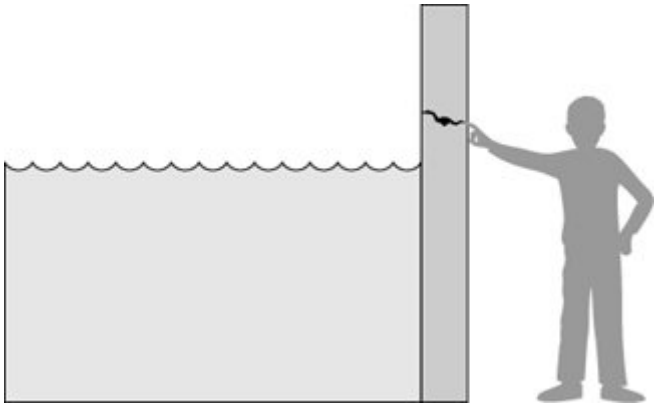
Vulnerabilities, Threats, Attacks, and Controls

A computer-based system has three separate but valuable components: **hardware**, **software,** and **data**. Each of these assets offers value to different members of the community affected by the system. To analyze security, we can brainstorm about the ways in which the system or its information can experience some kind of loss or harm. For example, we can identify data whose format or contents should be protected in some way. We want our security system to make sure that no data are disclosed to unauthorized parties. Neither do we want the data to be modified in illegitimate ways. At the same time, we must ensure that legitimate users have access to the data. In this way, we can identify weaknesses in the system.

A **vulnerability** is a weakness in the security 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 **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 1-1. 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 1-1. Threats, Controls, and Vulnerabilities.



However, we can see a small crack in the walla 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 set 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 flood servers with more messages than they can handle. (We take a closer look at denial of service in Chapter 7.)

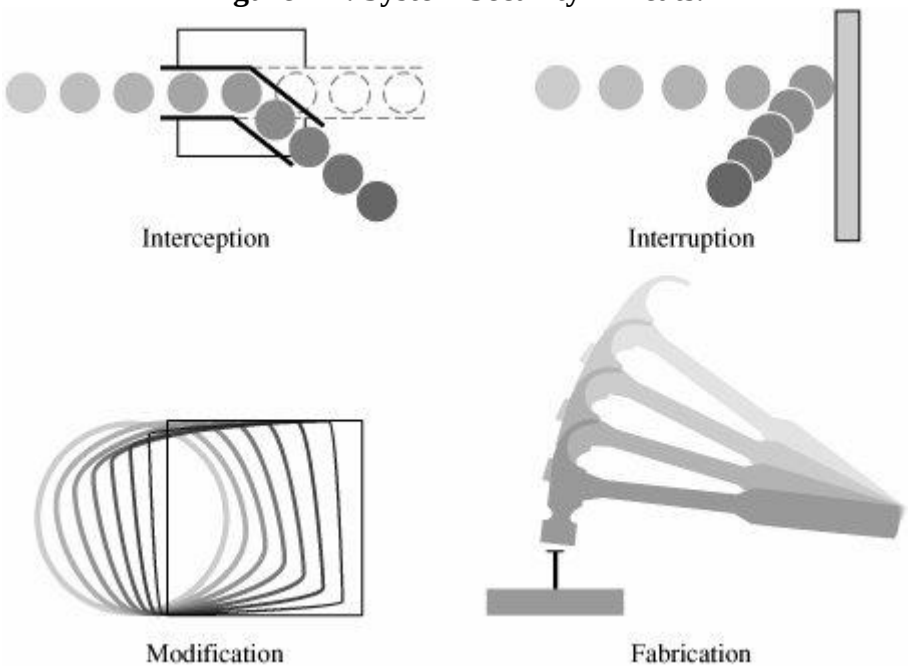
How do we address these problems? We use a **control** as a protective measure. That is, a control is an action, device, procedure, or technique that removes or reduces a vulnerability. In Figure 1-1, the man is placing his finger in the hole, controlling the threat of water leaks until he finds a more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

A threat is blocked by control of a vulnerability.

Much of the rest of this book is devoted to describing a variety of controls and understanding the degree to which they enhance a system's security.

To devise controls, we must know as much about threats as possible. We can view any threat as being one of four kinds: interception, interruption, modification, and fabrication. Each threat exploits vulnerabilities of the assets in computing systems; the threats are illustrated in Figure 1-2.

Figure 1-2. System Security Threats.



- An **interception** means that some unauthorized party has gained access to an asset. The outside party can be a person, a program, or a computing system. Examples of this type of failure are illicit copying of program or data files, or wiretapping to obtain data in a network. Although a loss may be discovered fairly quickly, a silent interceptor may leave no traces by which the interception can be readily detected.
- In an **interruption**, an asset of the system becomes lost, unavailable, or unusable. An example is malicious destruction of a hardware device, erasure of a program or data file, or malfunction of an operating system file manager so that it cannot find a particular disk file.
- If an unauthorized party not only accesses but tampers with an asset, the threat is a **modification**. For example, someone might change the values in a database, alter a program so that it performs an additional computation, or modify data being transmitted electronically. It is even possible to modify hardware. Some cases of modification can be detected with simple measures, but other, more subtle, changes may be almost impossible to detect.

- Finally, an unauthorized party might create a **fabrication** of counterfeit objects on a computing system. The intruder may insert spurious transactions to a network communication system or add records to an existing database. Sometimes these additions can be detected as forgeries, but if skillfully done, they are virtually indistinguishable from the real thing.

These four classes of threats interception, interruption, modification, and fabrication describe the kinds of problems we might encounter. In the next section, we look more closely at a system's vulnerabilities and how we can use them to set security goals.

Method, Opportunity, and Motive

A malicious attacker must have three things:

- *method:* the skills, knowledge, tools, and other things with which to be able to pull off the attack
- *opportunity:* the time and access to accomplish the attack
- *motive:* a reason to want to perform this attack against this system

(Think of the acronym "MOM.") Deny any of those three things and the attack will not occur. However, it is not easy to cut these off.

Knowledge of systems is widely available. Mass-market systems (such as the Microsoft or Apple or Unix operating systems) are readily available, as are common products, such as word processors or database management systems. Sometimes the manufacturers release detailed specifications on how the system was designed or operates, as guides for users and integrators who want to implement other complementary products. But even without documentation, attackers can purchase and experiment with many systems. Often, only time and inclination limit an attacker.

Many systems are readily available. 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.

Sidebar 1-2: Why Universities Are Prime Targets

Universities make very good targets for attack, according to an Associated Press story from June 2001 [HOP01]. Richard Power, editorial director for the Computer Security Institute, has reported that universities often run systems with vulnerabilities and little monitoring or management. Consider that the typical university research or teaching lab is managed by a faculty member who has many other responsibilities or by a student manager who may have had little training. Universities are havens for free exchange of ideas. Thus, their access controls typically are configured to promote sharing and wide access to a population that changes significantly every semester.

A worse problem is that universities are really loose federations of departments and research groups. The administrator for one group's computers may not even know other administrators, let alone share intelligence or tools. Often, computers are bought for a teaching or research project, but there is not funding for ongoing maintenance, either buying upgrades or installing patches. Steve Hare, managing director of the computer security research group at Purdue University, noted that groups are usually strapped for resources.

David Dittrich, a security engineer at the University of Washington, said he is certain that cracker(s) who attacked the eBay and CNN.com web sites in 2000 first practiced on university computers. The large and frequently changing university student body gives the attacker great opportunity to maintain anonymity while developing an attack.

Finally, it is difficult to determine motive for an attack. Some places are what are called "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 that a successful attack shows the attacker's prowess). Other systems are attacked because they are easy. (See Sidebar 1-2 on universities as targets.) And other systems are attacked simply because they are there: random, unassuming victims.
Protecting against attacks can be difficult. Anyone can be a victim of an attack perpetrated by an unhurried, knowledgeable attacker. In the remainder of this book we discuss the nature of attacks and how to protect against them.

1.3. The Meaning of Computer Security

We have seen that any computer-related system has both theoretical and real weaknesses. The purpose of computer security is to devise ways to prevent the weaknesses from being exploited. To understand what preventive measures make the most sense, we consider what we mean when we say that a system is "secure."

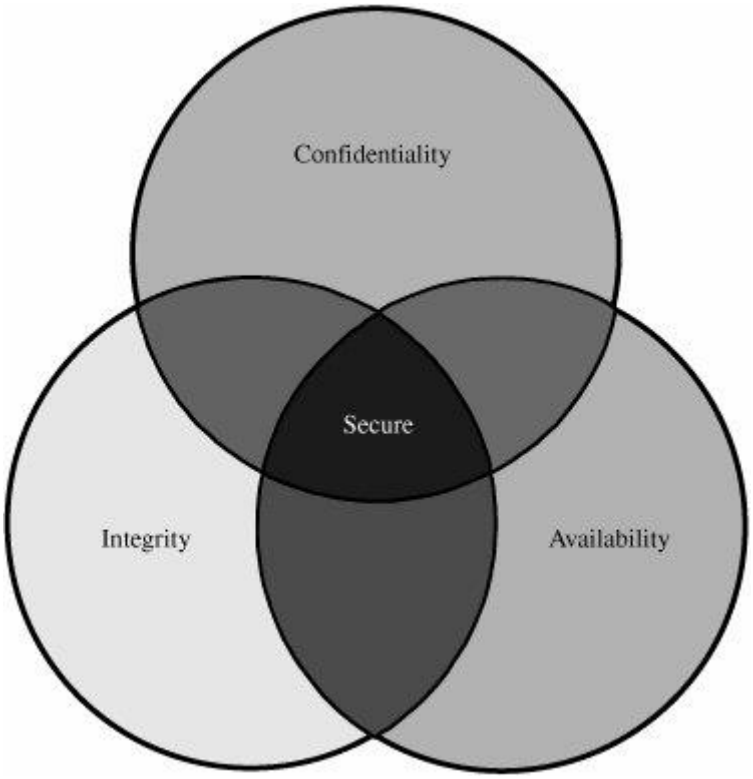
Security Goals

We use the term "security" in many ways in our daily lives. A "security system" protects our house, warning the neighbors or the police if an unauthorized intruder tries to get in. "Financial security" involves a set of investments that are adequately funded; we hope the investments will grow in value over time so that we have enough money to survive later in life. And we speak of children's "physical security," hoping they are safe from potential harm. Just as each of these terms has a very specific meaning in the context of its use, so too does the phrase "computer security."
When we talk about computer security, we mean that we are addressing three important aspects of any computer-related system: **confidentiality**, **integrity,** and **availability.**

- Confidentiality ensures that computer-related assets are accessed only by authorized parties. That is, only those who should have access to something will actually get that access. By "access," we mean not only reading but also viewing, printing, or simply knowing that a particular asset exists. Confidentiality is sometimes called **secrecy** or **privacy**.
- Integrity means that assets can be modified only by authorized parties or only in authorized ways. In this context, modification includes writing, changing, changing status, deleting, and creating.
- Availability means that assets are accessible to authorized parties at appropriate times. In other words, if some person or system has legitimate access to a particular set of objects, that access should not be prevented. For this reason, availability is sometimes known by its opposite, denial of service.

Security in computing addresses these three goals. One of the challenges in building a secure system is finding the right balance among the goals, which often conflict. For example, it is easy to preserve a particular object's confidentiality in a secure system simply by preventing everyone from reading that object. However, this system is not secure, because it does not meet the requirement of availability for proper access. That is, there must be a balance between confidentiality and availability.
But balance is not all. In fact, these three characteristics can be independent, can overlap (as shown in Figure 1-3), and can even be mutually exclusive. For example, we have seen that strong protection of confidentiality can severely restrict availability. Let us examine each of the three qualities in depth.

Figure 1-3. Relationship between Confidentiality, Integrity, and Availability.



Confidentiality

You may find the notion of confidentiality to be 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 those data to other parties?

Confidentiality is the security property we understand best because its meaning is narrower than the other two. We also understand confidentiality well because we can relate computing examples to those of preserving confidentiality in the real world.

Integrity

Integrity is much harder to pin down. As Welke and Mayfield [WEL90, MAY91, NCS91b] 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. Some forms of integrity are well represented in the real world, and those precise representations can be implemented in a computerized environment. But not all interpretations of integrity are well reflected by computer implementations.

Availability

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

- It is present in a usable form.
- It has capacity enough 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. We say a data item, service, or system is available if

- There is a timely response to our request.
- Resources are allocated fairly so that some requesters are not favored over others.
- 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.
- The service or system can be used easily and in the way it was intended to be used.
- Concurrency is controlled; that is, simultaneous access, deadlock management, and exclusive access are supported as required.

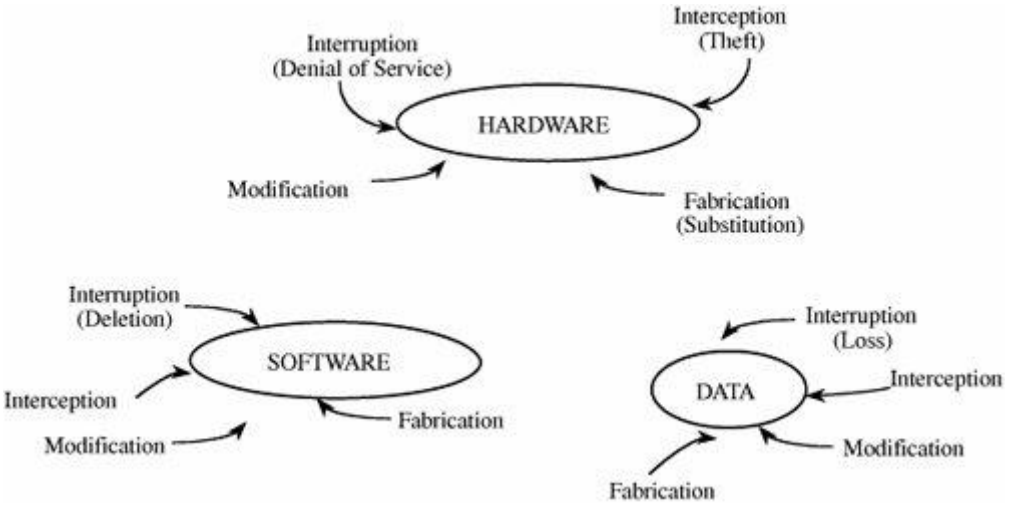
As you can see, expectations of availability are far-reaching. Indeed, the security community is just beginning to understand what availability implies and how to ensure it. A small, centralized control of access is fundamental to preserving confidentiality and integrity, but it is not clear that a single access control point can enforce availability. Much of computer security's past success has focused on confidentiality and integrity; full implementation of availability is security's next great challenge.

Vulnerabilities

When we prepare to test a system, we usually try to imagine how the system can fail; we then look for ways in which the requirements, design, or code can enable such failures. In the same way, when we prepare to specify, design, code, or test a secure system, we try to imagine the vulnerabilities that would prevent us from reaching one or more of our three security goals.

It is sometimes easier to consider vulnerabilities as they apply to all three broad categories of system resources (hardware, software, and data), rather than to start with the security goals themselves. Figure 1-4 shows the types of vulnerabilities we might find as they apply to the assets of hardware, software, and data. These three assets and the connections among them are all potential security weak points. Let us look in turn at the vulnerabilities of each asset.

Figure 1-4. Vulnerabilities of Computing Systems.



Hardware Vulnerabilities

Hardware is more visible than software, largely because it is composed of physical objects. Because we can see what devices are hooked to the system, it is rather simple to attack by adding devices, changing them, removing them, intercepting the traffic to them, or flooding them with traffic until they can no longer function. However, designers can usually put safeguards in place.

But there are other ways that computer hardware can be attacked physically. Computers have been drenched with water, burned, frozen, gassed, and electrocuted with power surges. People have spilled soft drinks, corn chips, ketchup, beer, and many other kinds of food on computing devices. Mice have chewed through cables. Particles of dust, and especially ash in cigarette smoke, have threatened precisely engineered moving parts. Computers have been kicked, slapped, bumped, jarred, and punched. Although such attacks might be intentional, most are not; this abuse might be considered "involuntary machine slaughter": accidental acts not intended to do serious damage to the hardware involved.

A more serious attack, "voluntary machine slaughter" or "machinicide," usually involves someone who actually wishes to harm the computer hardware or software. Machines have been shot with guns, stabbed with knives, and smashed with all kinds of things. Bombs, fires, and collisions have destroyed computer rooms. Ordinary keys, pens, and screwdrivers have been used to short-out circuit boards and other components. Devices and whole systems have been carried off by thieves. The list of the kinds of human attacks perpetrated on computers is almost endless.

In particular, deliberate attacks on equipment, intending to limit availability, usually involve theft or destruction. Managers of major computing centers long ago recognized these vulnerabilities and installed physical security systems to protect their machines. However, the proliferation of PCs, especially laptops, as office equipment has resulted in several thousands of dollars worth of equipment sitting unattended on desks outside the carefully protected computer room. (Curiously, the supply cabinet, containing only a few hundred dollars' worth of pens, stationery, and paper clips, is often locked.) Sometimes the security of hardware components can be enhanced greatly by simple physical measures such as locks and guards.

Laptop computers are especially vulnerable because they are designed to be easy to carry. (See Sidebar 1-3 for the story of a stolen laptop.) Safe ware Insurance reported 600,000 laptops stolen in 2003. Credent Technologies reported that 29 percent were stolen from the office, 25 percent from a car, and 14 percent in an airport. Stolen laptops are almost never recovered: The FBI reports 97 percent were not returned [SAI05].

Sidebar 1-3: Record Record Loss

The record for number of personal records lost stands at 26.5 million.

Yes, 26.5 million records were on the hard drive of a laptop belonging to the U.S. Veterans Administration (V.A.) The lost data included names, addresses, social security numbers, and birth dates of all veterans who left the service after 1975, as well as any disabled veterans who filed a claim for disability after 1975, as well as some spouses. The data were contained on the hard drive of a laptop stolen on 3 May 2006 near Washington D.C. A V.A. employee took the laptop home to work on the data, a practice that had been going on for three years.

The unasked, and therefore unanswered, question in this case is why the employee needed names, social security numbers, and birth dates of all veterans at home? One supposes the employee was not going to print 26.5 million personal letters on a home computer. Statistical trends, such as number of claims, type of claim, or time to process a claim, could be determined without birth dates and social security numbers.

Computer security professionals repeatedly find that the greatest security threat is from insiders, in part because of the quantity of data to which they need access to do their jobs. The V.A. chief testified to Congress that his agency had failed to heed years of warnings of lax security procedures. Now all employees have been ordered to attend a cybersecurity training course.

Software Vulnerabilities

Computing equipment is of little use without the software (operating system, controllers, utility programs, and application programs) that users expect. Software can be replaced, changed, or destroyed maliciously, or it can be modified, deleted, or misplaced accidentally. Whether intentional or not, these attacks exploit the software's vulnerabilities.

Sometimes, the attacks are obvious, as when the software no longer runs. More subtle are attacks in which the software has been altered but seems to run normally. Whereas physical equipment usually shows some mark of inflicted injury when its boundary has been breached, the loss of a line of source or object code may not leave an obvious mark in a program. Furthermore, it is possible to change a program so that it does all it did before, and then some. That is, a malicious intruder can "enhance" the software to enable it to perform functions you may not find desirable. In this case, it may be very hard to detect that the software has been changed, let alone to determine the extent of the change.

A classic example of exploiting software vulnerability is the case in which a bank worker realized that software truncates the fractional interest on each account. In other words, if the monthly interest on an account is calculated to be $14.5467, the software credits only $14.54 and ignores the $.0067. The worker amended the software so that the throw-away interest (the $.0067) was placed into his own account. Since the accounting practices ensured only that all accounts balanced, he built up a large amount of money from the thousands of account throw-aways without detection. It was only when he bragged to a colleague of his cleverness that the scheme was discovered.

Software Deletion

Software is surprisingly easy to delete. Each of us has, at some point in our careers, accidentally erased a file or saved a bad copy of a program, destroying a good previous copy. Because of software's high value to a commercial computing center, access to software is usually carefully controlled through a process called **configuration management** so that software cannot be deleted, destroyed, or replaced accidentally. Configuration management uses several techniques to ensure that each version or release retains its integrity. When configuration management is used, an old version or release can be replaced with a newer version only when it has been

thoroughly tested to verify that the improvements work correctly without degrading the functionality and performance of other functions and services.

Software Modification

Software is vulnerable to modifications that either cause it to fail or cause it to perform an unintended task. Indeed, because software is so susceptible to "off by one" errors, it is quite easy to modify. Changing a bit or two can convert a working program into a failing one. Depending on which bit was changed, the program may crash when it begins or it may execute for some time before it falters.

With a little more work, the change can be much more subtle: The program works well most of the time but fails in specialized circumstances. For instance, the program may be maliciously modified to fail when certain conditions are met or when a certain date or time is reached. Because of this delayed effect, such a program is known as a **logic bomb.** For example, a disgruntled employee may modify a crucial program so that it accesses the system date and halts abruptly after July 1. The employee might quit on May l and plan to be at a new job miles away by July.

Another type of change can extend the functioning of a program so that an innocuous program has a hidden side effect. For example, a program that ostensibly structures a listing of files belonging to a user may also modify the protection of all those files to permit access by another user.

Other categories of software modification include

- **Trojan horse:** a program that overtly does one thing while covertly doing another
- **virus:** a specific type of Trojan horse that can be used to spread its "infection" from one computer to another
- **trapdoor:** a program that has a secret entry point
- **information leaks** in a program: code that makes information accessible to unauthorized people or programs

More details on these and other software modifications are provided in Chapter 3.

Of course, it is possible to invent a completely new program and install it on a computing system. Inadequate control over the programs that are installed and run on a computing system permits this kind of software security breach.

Software Theft

This attack includes unauthorized copying of software. Software authors and distributors are entitled to fair compensation for use of their product, as are musicians and book authors. Unauthorized copying of software has not been stopped satisfactorily. As we see in Chapter 11, the legal system is still grappling with the difficulties of interpreting paper-based copyright laws for electronic media.

Data Vulnerabilities

Hardware security is usually the concern of a relatively small staff of computing center professionals. Software security is a larger problem, extending to all programmers and analysts who create or modify programs. Computer programs are written in a dialect intelligible primarily to computer professionals, so a "leaked" source listing of a program might very well be meaningless to the general public.

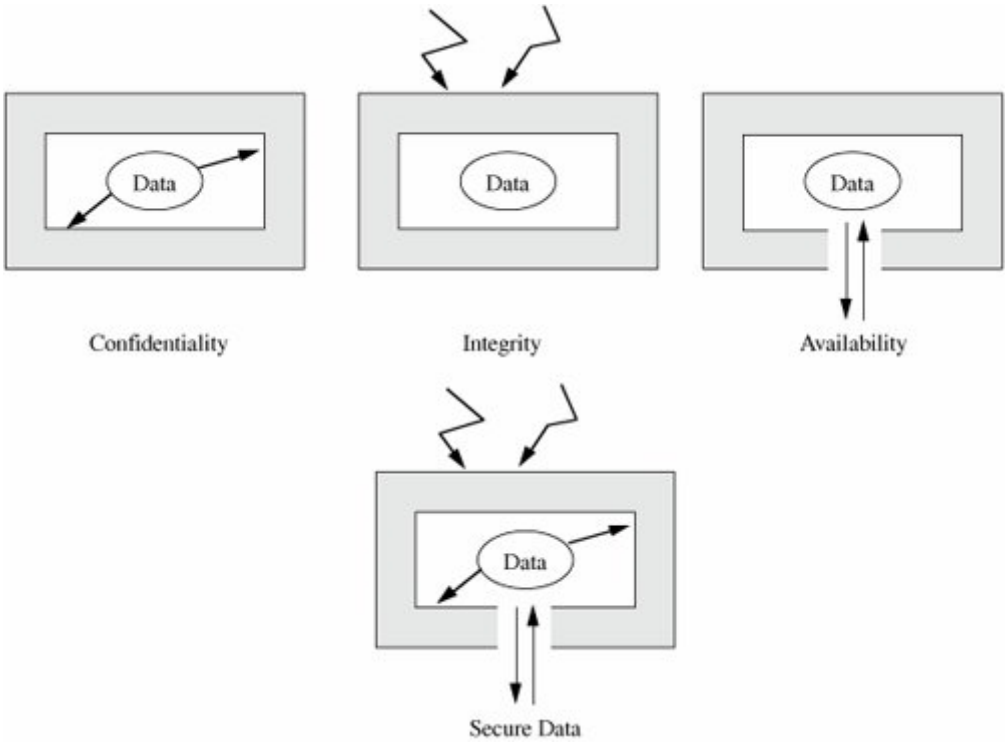
Printed data, however, can be readily interpreted by the general public. Because of its visible nature, a data attack is a more widespread and serious problem than either a hardware or software attack. Thus, data items have greater public value than hardware and software because more people know how to use or interpret data.

By themselves, out of context, pieces of data have essentially no intrinsic value. For example, if you are shown the value "42," it has no meaning for you unless you know what the number

represents. Likewise, "326 Old Norwalk Road" is of little use unless you know the city, state, and country for the address. For this reason, it is hard to measure the value of a given data item.
On the other hand, data items in context do relate to cost, perhaps measurable by the cost to reconstruct or redevelop damaged or lost data. For example, confidential data leaked to a competitor may narrow a competitive edge. Data incorrectly modified can cost human lives. To see how, consider the flight coordinate data used by an airplane that is guided partly or fully by software, as many now are. Finally, inadequate security may lead to financial liability if certain personal data are made public. Thus, data have a definite value, even though that value is often difficult to measure.
Typically, both hardware and software have a relatively long life. No matter how they are valued initially, their value usually declines gradually over time. By contrast, the value of data over time is far less predictable or consistent. Initially, data may be valued highly. However, some data items are of interest for only a short period of time, after which their value declines precipitously.
To see why, consider the following example. In many countries, government analysts periodically generate data to describe the state of the national economy. The results are scheduled to be released to the public at a predetermined time and date. Before that time, access to the data could allow someone to profit from advance knowledge of the probable effect of the data on the stock market. For instance, suppose an analyst develops the data 24 hours before their release and then wishes to communicate the results to other analysts for independent verification before release. The data vulnerability here is clear, and, to the right people, the data are worth more before the scheduled release than afterward. However, we can protect the data and control the threat in simple ways. For example, we could devise a scheme that would take an outsider more than 24 hours to break; even though the scheme may be eminently breakable (that is, an intruder could eventually reveal the data), it is adequate for those data because confidentiality is not needed beyond the 24-hour period.
Data security suggests the second principle of computer security.

Principle of Adequate Protection: Computer items must be protected only until they lose their value. They must be protected to a degree consistent with their value.
This principle says that things with a short life can be protected by security measures that are effective only for that short time. The notion of a small protection window applies primarily to data, but it can in some cases be relevant for software and hardware, too.
Sidebar 1-4 confirms that intruders take advantage of vulnerabilities to break in by whatever means they can.
Figure 1-5 illustrates how the three goals of security apply to data. In particular, confidentiality prevents unauthorized disclosure of a data item, integrity prevents unauthorized modification, and availability prevents denial of authorized access.

Figure 1-5. Security of Data.



Data Confidentiality

Data can be gathered by many means, such as tapping wires, planting bugs in output devices, sifting through trash receptacles, monitoring electromagnetic radiation, bribing key employees, inferring one data point from other values, or simply requesting the data. Because data are often available in a form people can read, the confidentiality of data is a major concern in computer security.

Sidebar 1-4: Top Methods of Attack

In 2006, the U.K. Department of Trade and Industry (DTI) released results of its latest annual survey of businesses regarding security incidents [PWC06]. Of companies surveyed, 62 percent reported one or more security breaches during the year (down from 74 percent two years earlier). The median number of incidents was 8.

In 2006, 29 percent of respondents (compared to 27 percent in 2004) reported an accidental security incident, and 57 percent (compared to 68 percent) reported a malicious incident. The percentage reporting a serious incident fell to 23 percent from 39 percent.

The top type of attack was virus or other malicious code at 35 percent (down significantly from 50 percent two years earlier). Staff misuse of data or resources was stable at 21 percent (versus 22 percent). Intrusion from outside (including hacker attacks) was constant at 17 percent in both periods, incidents involving fraud or theft were down to 8 percent form 11 percent, and failure of equipment was up slightly to 29 percent from 27 percent.

Attempts to break into a system from outside get much publicity. Of the respondents, 5 percent reported they experienced hundreds of such attacks a day, and 17 percent reported "several a day."

Data are not just numbers on paper; computer data include digital recordings such as CDs and DVDs, digital signals such as network and telephone traffic, and broadband communications such as cable and satellite TV. Other forms of data are biometric identifiers embedded in

passports, online activity preferences, and personal information such as financial records and votes. Protecting this range of data types requires many different approaches.

Data Integrity

Stealing, buying, finding, or hearing data requires no computer sophistication, whereas modifying or fabricating new data requires some understanding of the technology by which the data are transmitted or stored, as well as the format in which the data are maintained. Thus, a higher level of sophistication is needed to modify existing data or to fabricate new data than to intercept existing data. The most common sources of this kind of problem are malicious programs, errant file system utilities, and flawed communication facilities.

Data are especially vulnerable to modification. Small and skillfully done modifications may not be detected in ordinary ways. For instance, we saw in our truncated interest example that a criminal can perform what is known as a **salami attack**: The crook shaves a little from many accounts and puts these shavings together to form a valuable result, like the meat scraps joined in a salami.

A more complicated process is trying to reprocess used data items. With the proliferation of telecommunications among banks, a fabricator might intercept a message ordering one bank to credit a given amount to a certain person's account. The fabricator might try to **replay** that message, causing the receiving bank to credit the same account again. The fabricator might also try to modify the message slightly, changing the account to be credited or the amount, and then transmit this revised message.

Other Exposed Assets

We have noted that the major points of weakness in a computing system are hardware, software, and data. However, other components of the system may also be possible targets. In this section, we identify some of these other points of attack.

Networks

Networks are specialized collections of hardware, software, and data. Each network node is itself a computing system; as such, it experiences all the normal security problems. In addition, a network must confront communication problems that involve the interaction of system components and outside resources. The problems may be introduced by a very exposed storage medium or access from distant and potentially untrustworthy computing systems.

Thus, networks can easily multiply the problems of computer security. The challenges are rooted in a network's lack of physical proximity, use of insecure shared media, and the inability of a network to identify remote users positively.

Access

Access to computing equipment leads to three types of vulnerabilities. In the first, an intruder may steal computer time to do general-purpose computing that does not attack the integrity of the system itself. This theft of computer services is analogous to the stealing of electricity, gas, or water. However, the value of the stolen computing services may be substantially higher than the value of the stolen utility products or services. Moreover, the unpaid computing access spreads the true costs of maintaining the computing system to other legitimate users. In fact, the unauthorized access risks affecting legitimate computing, perhaps by changing data or programs.

A second vulnerability involves malicious access to a computing system, whereby an intruding person or system actually destroys software or data. Finally, unauthorized access may deny service to a legitimate user. For example, a user who has a time-critical task to perform may depend on the availability of the computing system. For all three of these reasons, unauthorized access to a computing system must be prevented.

Key People

People can be crucial weak points in security. If only one person knows how to use or maintain a particular program, trouble can arise if that person is ill, suffers an accident, or leaves the organization (taking her knowledge with her). In particular, a disgruntled employee can cause serious damage by using inside knowledge of the system and the data that are manipulated. For this reason, trusted individuals, such as operators and systems programmers, are usually selected carefully because of their potential ability to affect all computer users.

We have described common assets at risk. In fact, there are valuable assets in almost any computer system. (See Sidebar 1-5 for an example of exposed assets in ordinary business dealings.)

Next, we turn to the people who design, build, and interact with computer systems, to see who can breach the systems' confidentiality, integrity, and availability.

Sidebar 1-5: Hollywood at Risk

Do you think only banks, government sites, and universities are targets? Consider Hollywood. In 2001, Hollywood specifically the motion picture industry was hit with a series of attacks. Crackers entered computers and were able to obtain access to scripts for new projects, and digital versions of films in production, including Ocean's 11 at Warner Brothers and The One at Columbia Pictures. The attackers also retrieved and made public executives' e-mail messages.

But, as is true of many computer security incidents, at least one attacker was an insider. Global Network Security Services, a security consulting firm hired by several Hollywood companies to test the security of their networks, found that an employee was copying the day's (digital) film, taking it home, and allowing his roommate to post it to an Internet site.

1.4. Computer Criminals

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, most computer 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 **crackers**,[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 "hacker," someone who (non maliciously) programs, manages, or uses computing systems, and a "cracker," someone who attempts to access computing systems for malicious purposes. Crackers are the "evildoers." 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.)

1.5. Methods of Defense
In Chapter 11, 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

- *prevent* it, by blocking the attack or closing the vulnerability
- *deter* it, by making the attack harder but not impossible
- *deflect* it, by making another target more attractive (or this one less so)
- *detect* it, either as it happens or some time after the fact
- *recover* from its effects

Sidebar 1-6: 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 the hard drive contained what appeared to be files from a senior al Qaeda operative. 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 more than 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. There were also copies of news stories on terrorist activities. He also found documents indicating that al Qaeda were 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 e-mail messages of the kind one would find in a typical office: recommendations for promotions, justifications for petty cash expenditures, 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 merchant who did not know its contents.

But this computer illustrates an important point 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 as if it will happen immediately.

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

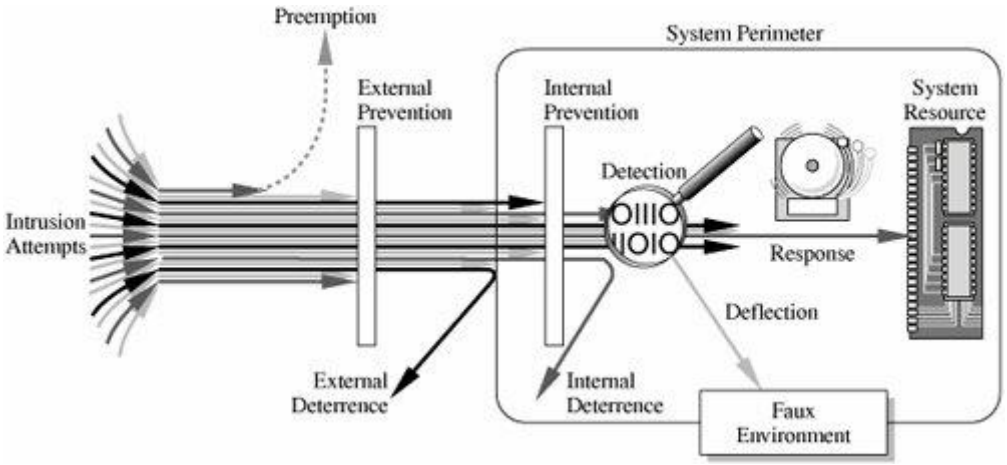
- a strong gate or door, to repel invaders
- heavy walls to withstand objects thrown or projected against them
- a surrounding moat, to control access
- arrow slits, to let archers shoot at approaching enemies
- crenellations to allow inhabitants to lean out from the roof and pour hot or vile liquids on attackers
- a drawbridge to limit access to authorized people
- 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 1-6. Multiple 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 clear text, and transform them so that they are unintelligible to the outside observer; the transformed data are called enciphered text or cipher text. 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:

- internal program controls: parts of the program that enforce security restrictions, such as access limitations in a database management program
- operating system and network system controls: limitations enforced by the operating system or network to protect each user from all other users
- independent control programs: application programs, such as password checkers, intrusion detection utilities, or virus scanners, that protect against certain types of vulnerabilities
- 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

- hardware or smart card implementations of encryption
- locks or cables limiting access or deterring theft
- devices to verify users' identities
- firewalls
- intrusion detection systems
- 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 used and used properly to 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.

Sidebar 1-7: Barn Door Wide Open
In 2001, Wilshire Associates, Inc., a Santa Monica, California-based investment company that manages about $10 billion of other people's money, found that its e-mail system had been operating for months with little security. Outsiders potentially had access to internal messages containing confidential information about clients and their investments, as well as sensitive company information.
According to a Washington Post article [OHA01], Wilshire had hired an outside security investigator in 1999 to review the security of its system. Thomas Stevens, a senior managing director of Wilshire said, "We had a report back that said our firewall is like Swiss cheese. We plugged the holes. We didn't plug all of them." Company officials were "not overly concerned" about that report because they are "not in the defense business." In 2001, security analyst George Imbruglia checked the system's security on his own, from the outside (with the same limited knowledge an attacker would have) and found it was "configured to be available to everyone; all you need to do is ask."
Wilshire's system enabled employees to access their e-mail remotely. A senior Wilshire director suggested that the e-mail messages in the system should have been encrypted.

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.

Sidebar 1-8: U.S. Government's Computer Security Report Card
The U.S. Congress requires government agencies to supply annual reports to the Office of Management and Budget (OMB) on the state of computer security in the agencies. The agencies must report efforts to protect their computer networks against crackers, terrorists, and other attackers.
In November 2001, for the third edition of this book, two-thirds of the government agencies received a grade of F (the lowest possible) on the computer security report card based on the OMB data. The good news for this edition is that in 2005 only 8 of 24 agencies received grades of F and 7 agencies received a grade of A. The bad, and certainly sad, news is that the average grade was D+. Also disturbing is that the grades of 7 agencies fell from 2004 to 2005. Among the failing agencies were Defense, State, Homeland Security, and Veterans Affairs. The Treasury Department received a D-. A grades went to Labor, Social Security Administration, and the National Science Foundation, among others. (Source: U.S. House of Representatives Government Reform Committee.)

Exercises

1 Distinguish among vulnerability, threat, and control.

2 Theft usually results in some kind of harm. For example, if someone steals your car, you may suffer financial loss, inconvenience (by losing your mode of transportation), and emotional upset (because of invasion of your personal property and space). List three kinds of harm a company might experience from theft of computer equipment.

3 List at least three kinds of harm a company could experience from electronic espionage or unauthorized viewing of confidential company materials.

4 List at least three kinds of damage a company could suffer when the integrity of a program or company data is compromised.

5 Describe two examples of vulnerabilities in automobiles for which auto manufacturers have instituted controls. Tell why you think these controls are effective, somewhat effective, or ineffective.

6 One control against accidental software deletion is to save all old versions of a program. Of course, this control is prohibitively expensive in terms of cost of storage. Suggest a less costly control against accidental software deletion. Is your control effective against all possible causes of software deletion? If not, what threats does it not cover?

7 On a typical multiuser computing system (such as a shared Unix system at a university or an industry), who can modify the code (software) of the operating system? Of a major application program such as a payroll program or a statistical analysis package? Of a program developed and run by a single user? Who should be permitted to modify each of these examples of code?

8 Suppose a program to print paychecks secretly leaks a list of names of employees earning more than a certain amount each month. What controls could be instituted to limit the vulnerability of this leakage?

9 Some terms have been introduced intentionally without definition in this chapter. You should be able to deduce their meanings. What is an electronic spy? What is an information broker?

10 Preserving confidentiality, integrity, and availability of data is a restatement of the concern over interruption, interception, modification, and fabrication. How do the first three concepts relate to the last four? That is, is any of the four equivalent to one or more of the three? Is one of the three encompassed by one or more of the four?

11 Do you think attempting to break in to (that is, obtain access to or use of) a computing system without authorization should be illegal? Why or why not?

12 Describe an example (other than the one mentioned in this chapter) of data whose confidentiality has a short timeliness, say, a day or less. Describe an example of data whose confidentiality has a timeliness of more than a year.

13 Do you currently use any computer security control measures? If so, what? Against what attacks are you trying to protect?

14 Describe an example in which absolute denial of service to a user (that is, the user gets no response from the computer) is a serious problem to that user. Describe another example where 10 percent denial of service to a user (that is, the user's computation progresses, but at a rate 10 percent slower than normal) is a serious problem to that user. Could access by unauthorized people to a computing system result in a 10 percent denial of service to the legitimate users? How?

15 When you say that software is of high quality, what do you mean? How does security fit into your definition of quality? For example, can an application be insecure and still be "good"?

16 Developers often think of software quality in terms of faults and failures. Faults are problems, such as loops that never terminate or misplaced commas in statements, that developers can see by looking at the code. Failures are problems, such as a system crash or the invocation of the wrong function, that are visible to the user. Thus, faults can exist in programs but never become failures, because the conditions under which a fault becomes a failure are never reached. How do software vulnerabilities fit into this scheme of faults and failures? Is every fault a vulnerability? Is every vulnerability a fault?

17 Consider a program to display on your web site your city's current time and temperature. Who might want to attack your program? What types of harm might they want to cause? What kinds of vulnerabilities might they exploit to cause harm?

18 Consider a program that allows consumers to order products from the web. Who might want to attack the program? What types of harm might they want to cause? What kinds of vulnerabilities might they exploit to cause harm?

19 Consider a program to accept and tabulate votes in an election. Who might want to attack the program? What types of harm might they want to cause? What kinds of vulnerabilities might they exploit to cause harm?

20 Consider a program that allows a surgeon in one city to assist in an operation on a patient in another city via an Internet connection. Who might want to attack the program? What types of harm might they want to cause? What kinds of vulnerabilities might they exploit to cause harm?

21 Reports of computer security failures appear frequently in the daily news. Cite a reported failure that exemplifies one (or more) of the principles listed in this chapter: easiest penetration, adequate protection, effectiveness, weakest link.

3.3. 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.

Sidebar 3-3: Non-malicious Flaws Cause Failures

In 1989, Crocker and Bernstein [CRO89] studied the root causes of the known catastrophic failures of what was then called the ARPANET, the predecessor of today's Internet. From its initial deployment in 1969 to 1989, the authors found 17 flaws that either did cause or could have caused catastrophic failure of the network. They use "catastrophic failure" to mean a situation that causes the entire network or a significant portion of it to fail to deliver network service.

The ARPANET was the first network of its sort, in which data are communicated as independent blocks (called "packets") that can be sent along different network routes and are reassembled at the destination. As might be expected, faults in the novel algorithms for delivery and reassembly were the source of several failures. Hardware failures were also significant. But as the network grew from its initial three nodes to dozens and hundreds, these problems were identified and fixed.

More than ten years after the network was born, three interesting non malicious flaws appeared. The initial implementation had fixed sizes and positions of the code and data. In 1986, a piece of code was loaded into memory in a way that overlapped a piece of security code. Only one critical node had that code configuration, and so only that one node would fail, which made it difficult to determine the cause of the failure.

In 1987, new code caused Sun computers connected to the network to fail to communicate. The first explanation was that the developers of the new Sun code had written the system to function as other manufacturers' code did, not necessarily as the specification dictated. It was later found that the developers had optimized the code incorrectly, leaving out some states the system could reach. But the first explanation designing to practice, not to specifications a common failing.

The last reported failure occurred in 1988. When the system was designed in 1969, developers specified that the number of connections to a subnetwork, and consequently the number of entries in a table of connections, was limited to 347, based on analysis of the expected topology. After 20 years, people had forgotten the (undocumented) limit, and a 348th connection was added, which caused the table to overflow and the system to fail. But the system derived this table gradually by communicating with neighboring nodes. So when any node's table reached 348 entries, it crashed, and when restarted, it started building its table anew. Thus, nodes throughout the system would crash seemingly randomly after running perfectly well for a while (with unfull tables).

None of these flaws were malicious nor could they have been exploited by a malicious attacker to cause a failure. But they show the importance of the analysis, design, documentation, and maintenance steps in development of a large, long-lived system.

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 non malicious 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.

Sidebar 3-4: Rapidly Approaching Zero

Y2K or the year 2000 problem, when dire consequences were forecast for computer clocks with 2-digit year fields that would turn from 99 to 00, was an ideal problem: The threat was easy to define, time of impact was easily predicted, and plenty of advance warning was given. Perhaps as a consequence, very few computer systems and people experienced significant harm early in the morning of 1 January 2000. Another countdown clock has computer security researchers much more concerned.

The time between general knowledge of a product vulnerability and appearance of code to exploit that vulnerability is shrinking. The general exploit timeline follows this sequence:

- An attacker discovers a previously unknown vulnerability.
- The manufacturer becomes aware of the vulnerability.
- Someone develops code (called proof of concept) to demonstrate the vulnerability in a controlled setting.
- The manufacturer develops and distributes a patch or wor-around that counters the vulnerability.
- Users implement the control.
- Someone extends the proof of concept, or the original vulnerability definition, to an actual attack.

As long as users receive and implement the control before the actual attack, no harm

occurs. An attack before availability of the control is called a zero day exploit. Time between proof of concept and actual attack has been shrinking. Code Red, one of the most virulent pieces of malicious code, in 2001 exploited vulnerabilities for which the patches had been distributed more than a month before the attack. But more recently, the time between vulnerability and exploit has steadily declined. On 18 August 2005, Microsoft issued a security advisory to address a vulnerability of which the proof of concept code was posted to the French SIRT (Security Incident Response Team) web site frsirt.org. A Microsoft patch was distributed a week later. On 27 December 2005 a vulnerability was discovered in Windows metafile (.WMF) files. Within hours hundreds of sites began to exploit the vulnerability to distribute malicious code, and within six days a malicious code toolkit appeared, by which anyone could easily create an exploit. Microsoft released a patch in nine days.

But what exactly is a zero day exploit? It depends on who is counting. If the vendor knows of the vulnerability but has not yet released a control, does that count as zero day, or does the exploit have to surprise the vendor? David Litchfield of Next Generation Software in the U.K. identified vulnerabilities and informed Oracle. He claims Oracle took an astonishing 800 days to fix two of them and others were not fixed for 650 days. Other customers are disturbed by the slow patch cycle Oracle released no patches between January 2005 and March 2006 [GRE06]. Distressed by the lack of response, Litchfield finally went public with the vulnerabilities to force Oracle to improve its customer support. Obviously, there is no way to determine if a flaw is known only to the security community or to the attackers as well unless an attack occurs.

Shrinking time between knowledge of vulnerability and exploit puts pressure on vendors and users both, and time pressure is not conducive to good software development or system management.

The worse problem cannot be controlled: vulnerabilities known to attackers but not to the security community.

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 war. Legend tells how the Greeks tricked the Trojans into breaking their defense wall to take a wooden horse, filled with the bravest of Greek soldiers, into their citadel. In the night, the soldiers descended and signaled their troops that the way in was now clear, and Troy was 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.

A **trapdoor** or **backdoor** is a feature in a program by which someone can access the program 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 non malicious 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.

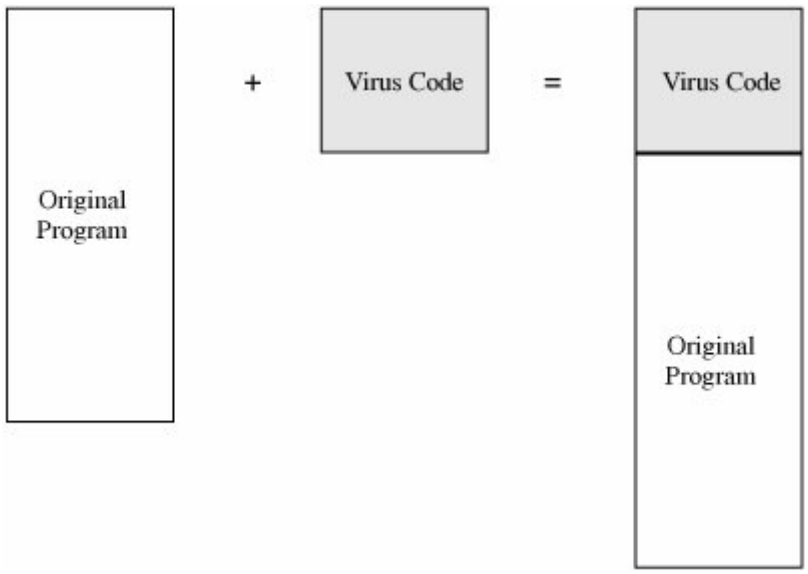
A more common means of virus activation is as an attachment to an e-mail message. In this attack, the virus writer tries to convince the victim (the recipient of the e-mail message) to open the attachment. Once the viral attachment is opened, the activated virus can do its work. Some modern e-mail handlers, 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.

Figure 3-4. Virus Appended to a Program.



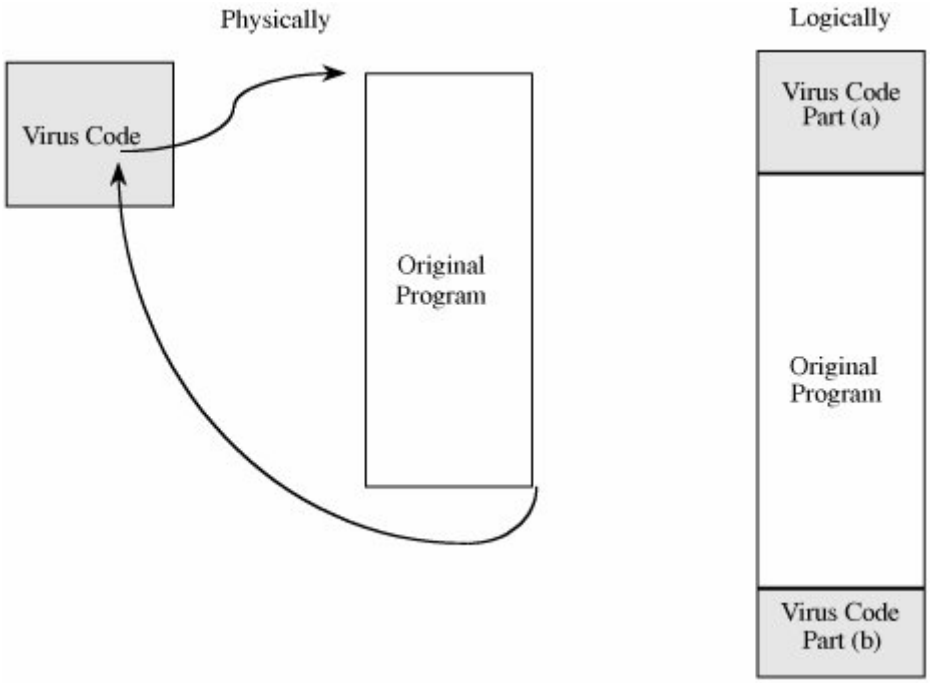
This kind of attachment is simple and usually effective. The virus writer does not need to know anything about the program to which the virus will attach, and often the attached program simply serves as a carrier for the virus. The virus performs its task and then transfers to the original program. Typically, 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.

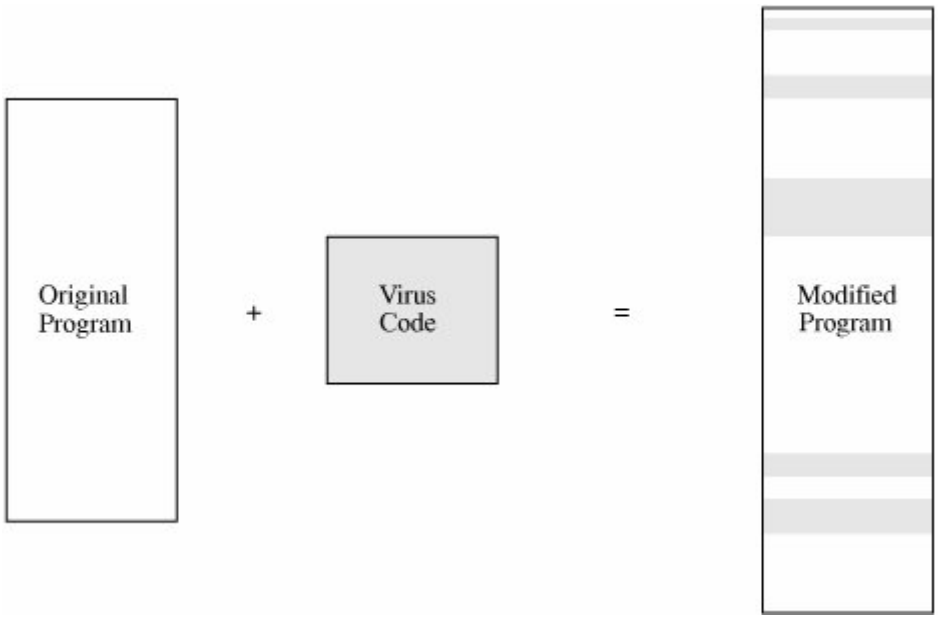
Figure 3-5. Virus Surrounding a Program.



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.

Figure 3-6. Virus Integrated into a Program.



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.

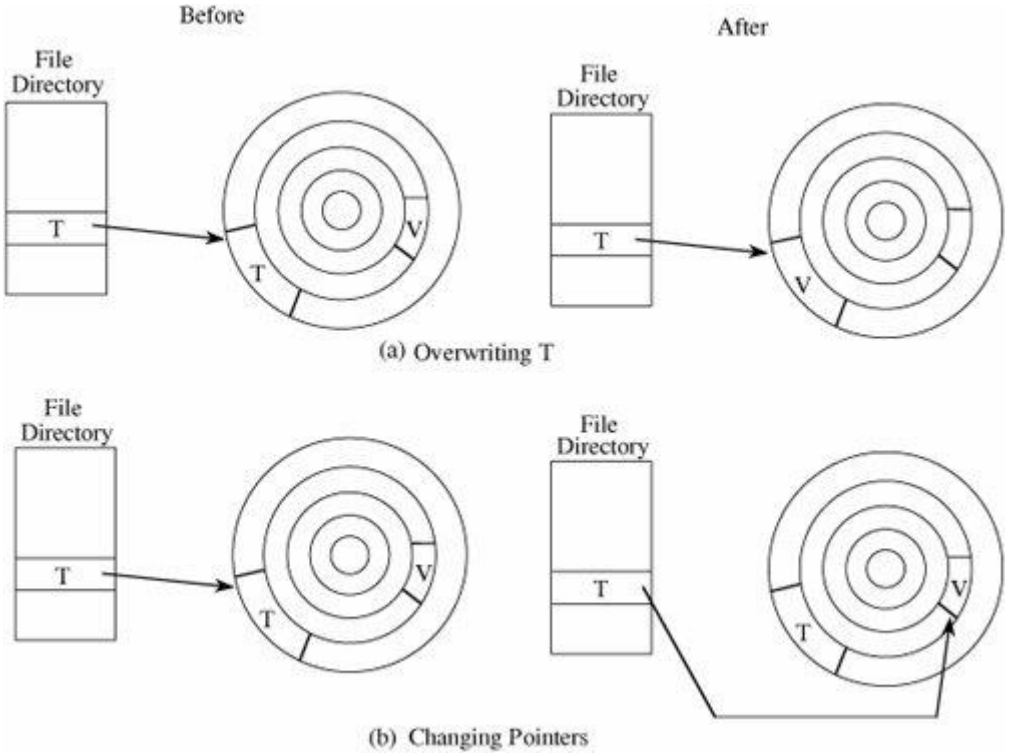
The ordinary user usually sees only the content of the document (its text or data), so the virus writer simply includes the virus in the commands part of the document, as in the integrated program virus.

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.

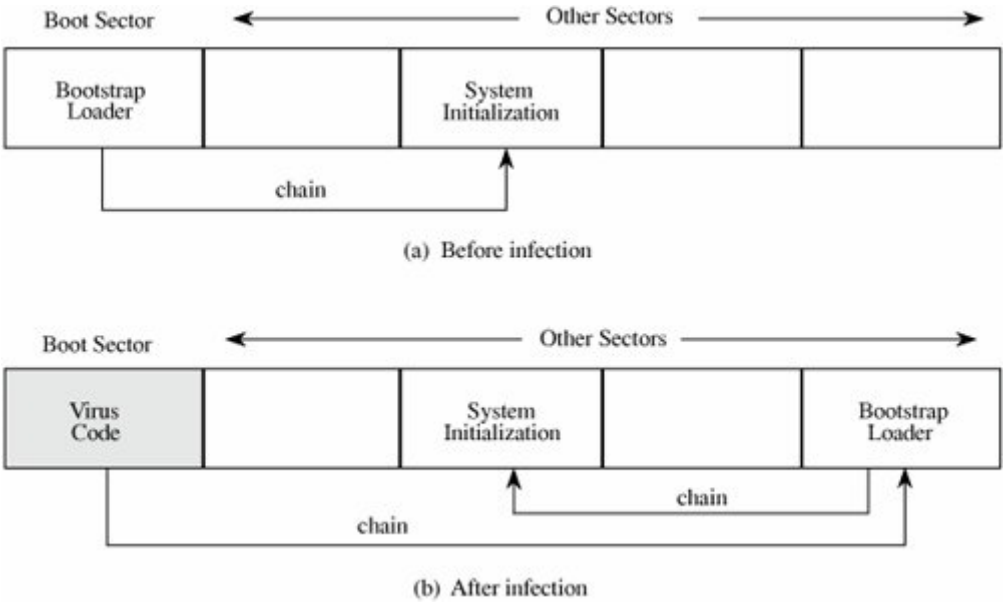
Boot Sector Viruses

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.

Figure 3-8. Boot Sector Virus Relocating Code.



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**, that can be found by a program that looks for it. The virus's signature is important for 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?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
%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.

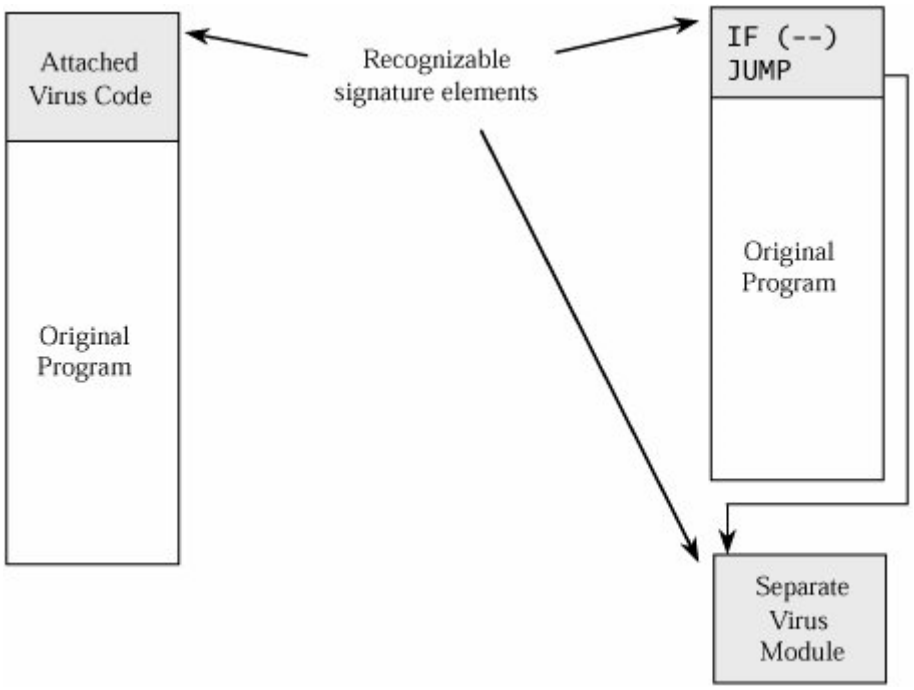
Sidebar 3-5: The Malware Threat

Security firm Symantec reports that malicious code threats rose in 2005, as in previous years. In 2005, they found 21,858 new instances of viruses and worms, compared to 11,846 for 2004 [SYM06]. Curiously, the number of distinct families of malicious code decreased from 335 for 2004 to 274 for 2005, perhaps showing that malicious code writers are becoming more adept at modifying a base attack code type or that self-modifying malicious code is on the rise. E-mail is still the preferred medium of delivery, with 92 percent of attacks using that for delivery. Other popular methods were peer-to-peer sharing protocols at 14 percent and remote exploitation of a system or software vulnerability at 13 percent. (A single malicious code strain could use more than one propagation method, accounting for the sum of methods exceeding 100 percent.)

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
Remain in memory	• Intercept interrupt by modifying interrupt handler address table • Load self in non transient 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 "HA! 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 may or may not be 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 non executable 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 ways and 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 not! 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 "hidden" or "read-only" 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. True, 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 hardware settings (for example, the size of disk installed) are retained in memory called "nonvolatile RAM," but these locations are not directly accessible by programs and are written only by programs run from read-only memory (ROM) during hardware 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 "worm," 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 accomplish hed or had code that appeared to try to accomplish hits 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 cipher text in the file is readable by anyone. (This visibility is the system error.) The worm encrypted various popular passwords and compared their cipher text to the cipher ext 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 send mail 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 security or stealth was 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 time was 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 non malicious 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 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:
HELLO!
Welcome to
http://www.worm.com !
Hacked by Chinese!

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 non blocking 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.

Sidebar 3-6: Is the Cure Worse Than the Disease?

These days, a typical application program such as a word-processor or spreadsheet package is sold to its user with no guarantee of quality. As problems are discovered by users or developers, patches are made available to be downloaded from the web and applied to the faulty system. This style of "quality control" relies on the users and system administrators to keep up with the history of releases and patches and to apply the patches in a timely manner. Moreover, each patch usually assumes that earlier patches can be applied; ignore a patch at your peril.

For example, Forno [FOR01] points out that an organization hoping to secure a web server running Windows NT 4.0's IIS had to apply over 47 patches as part of a service pack or available as a download from Microsoft. Such stories suggest that it may cost more to maintain an application or system than it cost to buy the application or system in the first place! Many organizations, especially small businesses, lack the resources for such an effort. As a consequence, they neglect to fix known system problems, which can then be exploited by hackers writing malicious code.

Blair [BLA01] describes a situation shortly after the end of the Cold War when the United States discovered that Russia was tracking its nuclear weapons materials by

using a paper-based system. That is, the materials tracking system consisted of boxes of paper filled with paper receipts. In a gesture of friendship, the Los Alamos National Lab donated to Russia the Microsoft software it uses to track its own nuclear weapons materials. However, experts at the renowned Kurchatov Institute soon discovered that over time some files become invisible and inaccessible! In early 2000, they warned the United States. To solve the problem, the United States told Russia to upgrade to the next version of the Microsoft software. But the upgrade had the same problem, plus a security flaw that would allow easy access to the database by hackers or unauthorized parties.

Sometimes patches themselves create new problems as they are fixing old ones. It is well known in the software reliability community that testing and fixing sometimes reduce reliability, rather than improve it. And with the complex interactions between software packages, many computer system managers prefer to follow the adage "if it ain't broke, don't fix it," meaning that if there is no apparent failure, they would rather not risk causing one from what seems like an unnecessary patch. So there are several ways that the continual bug-patching approach to security may actually lead to a less secure product than you started with.

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.

This information can be used to track where and when you read a document, what your 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.