

# Chapter 1

## An Overview of Computer Security

---

ANTONIO: Whereof what's past is prologue, what to come  
In yours and my discharge.  
—*The Tempest*, II, i, 257–258.

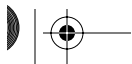
This chapter presents the basic concepts of computer security. The remainder of the book will elaborate on these concepts in order to reveal the logic underlying the principles of these concepts.

We begin with basic security-related services that protect against threats to the security of the system. The next section discusses security policies that identify the threats and define the requirements for ensuring a secure system. Security mechanisms detect and prevent attacks and recover from those that succeed. Analyzing the security of a system requires an understanding of the mechanisms that enforce the security policy. It also requires a knowledge of the related assumptions and trust, which lead to the threats and the degree to which they may be realized. Such knowledge allows one to design better mechanisms and policies to neutralize these threats. This process leads to risk analysis. Human beings are the weakest link in the security mechanisms of any system. Therefore, policies and procedures must take people into account. This chapter discusses each of these topics.

---

### 1.1 The Basic Components

Computer security rests on confidentiality, integrity, and availability. The interpretations of these three aspects vary, as do the contexts in which they arise. The interpretation of an aspect in a given environment is dictated by the needs of the individuals, customs, and laws of the particular organization.



### 1.1.1 Confidentiality

Confidentiality is the concealment of information or resources. The need for keeping information secret arises from the use of computers in sensitive fields such as government and industry. For example, military and civilian institutions in the government often restrict access to information to those who need that information. The first formal work in computer security was motivated by the military's attempt to implement controls to enforce a "need to know" principle. This principle also applies to industrial firms, which keep their proprietary designs secure lest their competitors try to steal the designs. As a further example, all types of institutions keep personnel records secret.

Access control mechanisms support confidentiality. One access control mechanism for preserving confidentiality is cryptography, which scrambles data to make it incomprehensible. A *cryptographic key* controls access to the unscrambled data, but then the cryptographic key itself becomes another datum to be protected.

EXAMPLE: Enciphering an income tax return will prevent anyone from reading it. If the owner needs to see the return, it must be deciphered. Only the possessor of the cryptographic key can enter it into a deciphering program. However, if someone else can read the key when it is entered into the program, the confidentiality of the tax return has been compromised.

Other system-dependent mechanisms can prevent processes from illicitly accessing information. Unlike enciphered data, however, data protected only by these controls can be read when the controls fail or are bypassed. Then their advantage is offset by a corresponding disadvantage. They can protect the secrecy of data more completely than cryptography, but if they fail or are evaded, the data becomes visible.

Confidentiality also applies to the existence of data, which is sometimes more revealing than the data itself. The precise number of people who distrust a politician may be less important than knowing that such a poll was taken by the politician's staff. How a particular government agency harassed citizens in its country may be less important than knowing that such harassment occurred. Access control mechanisms sometimes conceal the mere existence of data, lest the existence itself reveal information that should be protected.

Resource hiding is another important aspect of confidentiality. Sites often wish to conceal their configuration as well as what systems they are using; organizations may not wish others to know about specific equipment (because it could be used without authorization or in inappropriate ways), and a company renting time from a service provider may not want others to know what resources it is using. Access control mechanisms provide these capabilities as well.

All the mechanisms that enforce confidentiality require supporting services from the system. The assumption is that the security services can rely on the kernel, and other agents, to supply correct data. Thus, assumptions and trust underlie confidentiality mechanisms.

### 1.1.2 Integrity

Integrity refers to the trustworthiness of data or resources, and it is usually phrased in terms of preventing improper or unauthorized change. Integrity includes data integrity (the content of the information) and origin integrity (the source of the data, often called *authentication*). The source of the information may bear on its accuracy and credibility and on the trust that people place in the information. This dichotomy illustrates the principle that the aspect of integrity known as credibility is central to the proper functioning of a system. We will return to this issue when discussing malicious logic.

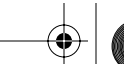
**EXAMPLE:** A newspaper may print information obtained from a leak at the White House but attribute it to the wrong source. The information is printed as received (preserving data integrity), but its source is incorrect (corrupting origin integrity).

Integrity mechanisms fall into two classes: *prevention* mechanisms and *detection* mechanisms.

Prevention mechanisms seek to maintain the integrity of the data by blocking any unauthorized attempts to change the data or any attempts to change the data in unauthorized ways. The distinction between these two types of attempts is important. The former occurs when a user tries to change data which she has no authority to change. The latter occurs when a user authorized to make certain changes in the data tries to change the data in other ways. For example, suppose an accounting system is on a computer. Someone breaks into the system and tries to modify the accounting data. Then an unauthorized user has tried to violate the integrity of the accounting database. But if an accountant hired by the firm to maintain its books tries to embezzle money by sending it overseas and hiding the transactions, a user (the accountant) has tried to change data (the accounting data) in unauthorized ways (by moving it to a Swiss bank account). Adequate authentication and access controls will generally stop the break-in from the outside, but preventing the second type of attempt requires very different controls.

Detection mechanisms do not try to prevent violations of integrity; they simply report that the data's integrity is no longer trustworthy. Detection mechanisms may analyze system events (user or system actions) to detect problems or (more commonly) may analyze the data itself to see if required or expected constraints still hold. The mechanisms may report the actual cause of the integrity violation (a specific part of a file was altered), or they may simply report that the file is now corrupt.

Working with integrity is very different from working with confidentiality. With confidentiality, the data is either compromised or it is not, but integrity includes both the correctness and the trustworthiness of the data. The origin of the data (how and from whom it was obtained), how well the data was protected before it arrived at the current machine, and how well the data is protected on the current machine all affect the integrity of the data. Thus, evaluating integrity is often very difficult,



because it relies on assumptions about the source of the data and about trust in that source—two underpinnings of security that are often overlooked.

### 1.1.3 Availability

Availability refers to the ability to use the information or resource desired. Availability is an important aspect of reliability as well as of system design because an unavailable system is at least as bad as no system at all. The aspect of availability that is relevant to security is that someone may deliberately arrange to deny access to data or to a service by making it unavailable. System designs usually assume a statistical model to analyze expected patterns of use, and mechanisms ensure availability when that statistical model holds. Someone may be able to manipulate use (or parameters that control use, such as network traffic) so that the assumptions of the statistical model are no longer valid. This means that the mechanisms for keeping the resource or data available are working in an environment for which they were not designed. As a result, they will often fail.

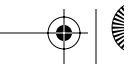
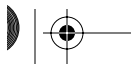
**EXAMPLE:** Suppose Anne has compromised a bank's secondary system server, which supplies bank account balances. When anyone else asks that server for information, Anne can supply any information she desires. Merchants validate checks by contacting the bank's primary balance server. If a merchant gets no response, the secondary server will be asked to supply the data. Anne's colleague prevents merchants from contacting the primary balance server, so all merchant queries go to the secondary server. Anne will never have a check turned down, regardless of her actual account balance. Notice that if the bank had only one server (the primary one), this scheme would not work. The merchant would be unable to validate the check.

Attempts to block availability, called *denial of service attacks*, can be the most difficult to detect, because the analyst must determine if the unusual access patterns are attributable to deliberate manipulation of resources or of environment. Complicating this determination is the nature of statistical models. Even if the model accurately describes the environment, atypical events simply contribute to the nature of the statistics. A deliberate attempt to make a resource unavailable may simply look like, or be, an atypical event. In some environments, it may not even appear atypical.

---

## 1.2 Threats

A *threat* is a potential violation of security. The violation need not actually occur for there to be a threat. The fact that the violation *might* occur means that those actions that could cause it to occur must be guarded against (or prepared for). Those actions



are called *attacks*. Those who execute such actions, or cause them to be executed, are called *attackers*.

The three security services—confidentiality, integrity, and availability—counter threats to the security of a system. Shirey [916] divides threats into four broad classes: *disclosure*, or unauthorized access to information; *deception*, or acceptance of false data; *disruption*, or interruption or prevention of correct operation; and *usurpation*, or unauthorized control of some part of a system. These four broad classes encompass many common threats. Because the threats are ubiquitous, an introductory discussion of each one will present issues that recur throughout the study of computer security.

*Snooping*, the unauthorized interception of information, is a form of disclosure. It is passive, suggesting simply that some entity is listening to (or reading) communications or browsing through files or system information. *Wiretapping*, or *passive wiretapping*, is a form of snooping in which a network is monitored. (It is called “wiretapping” because of the “wires” that compose the network, although the term is used even if no physical wiring is involved.) Confidentiality services counter this threat.

*Modification* or *alteration*, an unauthorized change of information, covers three classes of threats. The goal may be deception, in which some entity relies on the modified data to determine which action to take, or in which incorrect information is accepted as correct and is released. If the modified data controls the operation of the system, the threats of disruption and usurpation arise. Unlike snooping, modification is active; it results from an entity changing information. *Active wiretapping* is a form of modification in which data moving across a network is altered; the term “active” distinguishes it from snooping (“passive” wiretapping). An example is the *man-in-the-middle* attack, in which an intruder reads messages from the sender and sends (possibly modified) versions to the recipient, in hopes that the recipient and sender will not realize the presence of the intermediary. Integrity services counter this threat.

*Masquerading* or *spoofing*, an impersonation of one entity by another, is a form of both deception and usurpation. It lures a victim into believing that the entity with which it is communicating is a different entity. For example, if a user tries to log into a computer across the Internet but instead reaches another computer that claims to be the desired one, the user has been spoofed. Similarly, if a user tries to read a file, but an attacker has arranged for the user to be given a different file, another spoof has taken place. This may be a passive attack (in which the user does not attempt to authenticate the recipient, but merely accesses it), but it is usually an active attack (in which the masquerader issues responses to mislead the user about its identity). Although primarily deception, it is often used to usurp control of a system by an attacker impersonating an authorized manager or controller. Integrity services (called “authentication services” in this context) counter this threat.

Some forms of masquerading may be allowed. *Delegation* occurs when one entity authorizes a second entity to perform functions on its behalf. The distinctions between delegation and masquerading are important. If Susan delegates to Thomas the authority to act on her behalf, she is giving permission for him to perform specific actions as though she were performing them herself. All parties are aware of the delegation. Thomas will not pretend to be Susan; rather, he will say, “I am Thomas

8 **Chapter 1** An Overview of Computer Security

and I have authority to do this on Susan's behalf." If asked, Susan will verify this. On the other hand, in a masquerade, Thomas will pretend to be Susan. No other parties (including Susan) will be aware of the masquerade, and Thomas will say, "I am Susan." Should anyone discover that he or she is dealing with Thomas and ask Susan about it, she will deny that she authorized Thomas to act on her behalf. In terms of security, masquerading is a violation of security, whereas delegation is not.

*Repudiation of origin*, a false denial that an entity sent (or created) something, is a form of deception. For example, suppose a customer sends a letter to a vendor agreeing to pay a large amount of money for a product. The vendor ships the product and then demands payment. The customer denies having ordered the product and by law is therefore entitled to keep the unsolicited shipment without payment. The customer has repudiated the origin of the letter. If the vendor cannot prove that the letter came from the customer, the attack succeeds. A variant of this is denial by a user that he created specific information or entities such as files. Integrity mechanisms cope with this threat.

*Denial of receipt*, a false denial that an entity received some information or message, is a form of deception. Suppose a customer orders an expensive product, but the vendor demands payment before shipment. The customer pays, and the vendor ships the product. The customer then asks the vendor when he will receive the product. If the customer has already received the product, the question constitutes a denial of receipt attack. The vendor can defend against this attack only by proving that the customer did, despite his denials, receive the product. Integrity and availability mechanisms guard against these attacks.

*Delay*, a temporary inhibition of a service, is a form of usurpation, although it can play a supporting role in deception. Typically, delivery of a message or service requires some time  $t$ ; if an attacker can force the delivery to take more than time  $t$ , the attacker has successfully delayed delivery. This requires manipulation of system control structures, such as network components or server components, and hence is a form of usurpation. If an entity is waiting for an authorization message that is delayed, it may query a secondary server for the authorization. Even though the attacker may be unable to masquerade as the primary server, she might be able to masquerade as that secondary server and supply incorrect information. Availability mechanisms can thwart this threat.

*Denial of service*, a long-term inhibition of service, is a form of usurpation, although it is often used with other mechanisms to deceive. The attacker prevents a server from providing a service. The denial may occur at the source (by preventing the server from obtaining the resources needed to perform its function), at the destination (by blocking the communications from the server), or along the intermediate path (by discarding messages from either the client or the server, or both). Denial of service poses the same threat as an infinite delay. Availability mechanisms counter this threat.

Denial of service or delay may result from direct attacks or from nonsecurity-related problems. From our point of view, the cause and result are important; the intention underlying them is not. If delay or denial of service compromises system security, or is part of a sequence of events leading to the compromise of a system,



then we view it as an attempt to breach system security. But the attempt may not be deliberate; indeed, it may be the product of environmental characteristics rather than specific actions of an attacker.

---

## 1.3 Policy and Mechanism

Critical to our study of security is the distinction between policy and mechanism.

**Definition 1–1.** A *security policy* is a statement of what is, and what is not, allowed.

**Definition 1–2.** A *security mechanism* is a method, tool, or procedure for enforcing a security policy.

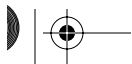
Mechanisms can be nontechnical, such as requiring proof of identity before changing a password; in fact, policies often require some procedural mechanisms that technology cannot enforce.

As an example, suppose a university's computer science laboratory has a policy that prohibits any student from copying another student's homework files. The computer system provides mechanisms for preventing others from reading a user's files. Anna fails to use these mechanisms to protect her homework files, and Bill copies them. A breach of security has occurred, because Bill has violated the security policy. Anna's failure to protect her files does not authorize Bill to copy them.

In this example, Anna could easily have protected her files. In other environments, such protection may not be easy. For example, the Internet provides only the most rudimentary security mechanisms, which are not adequate to protect information sent over that network. Nevertheless, acts such as the recording of passwords and other sensitive information violate an implicit security policy of most sites (specifically, that passwords are a user's confidential property and cannot be recorded by anyone).

Policies may be presented mathematically, as a list of allowed (secure) and disallowed (nonsecure) states. For our purposes, we will assume that any given policy provides an axiomatic description of secure states and nonsecure states. In practice, policies are rarely so precise; they normally describe in English what users and staff are allowed to do. The ambiguity inherent in such a description leads to states that are not classified as "allowed" or "disallowed." For example, consider the homework policy discussed above. If someone looks through another user's directory without copying homework files, is that a violation of security? The answer depends on site custom, rules, regulations, and laws, all of which are outside our focus and may change over time.

When two different sites communicate or cooperate, the entity they compose has a security policy based on the security policies of the two entities. If those policies are inconsistent, either or both sites must decide what the security policy for the



combined site should be. The inconsistency often manifests itself as a security breach. For example, if proprietary documents were given to a university, the policy of confidentiality in the corporation would conflict with the more open policies of most universities. The university and the company must develop a mutual security policy that meets both their needs in order to produce a consistent policy. When the two sites communicate through an independent third party, such as an Internet Service Provider, the complexity of the situation grows rapidly.

### 1.3.1 Goals of Security

Given a security policy's specification of "secure" and "nonsecure" actions, these security mechanisms can prevent the attack, detect the attack, or recover from the attack. The strategies may be used together or separately.

*Prevention* means that an attack will fail. For example, if one attempts to break into a host over the Internet and that host is not connected to the Internet, the attack has been prevented. Typically, prevention involves implementation of mechanisms that users cannot override and that are trusted to be implemented in a correct, unalterable way, so that the attacker cannot defeat the mechanism by changing it. Preventative mechanisms often are very cumbersome and interfere with system use to the point that they hinder normal use of the system. But some simple preventative mechanisms, such as passwords (which aim to prevent unauthorized users from accessing the system), have become widely accepted. Prevention mechanisms can prevent compromise of parts of the system; once in place, the resource protected by the mechanism need not be monitored for security problems, at least in theory.

*Detection* is most useful when an attack cannot be prevented, but it can also indicate the effectiveness of preventative measures. Detection mechanisms accept that an attack will occur; the goal is to determine that an attack is underway, or has occurred, and report it. The attack may be monitored, however, to provide data about its nature, severity, and results. Typical detection mechanisms monitor various aspects of the system, looking for actions or information indicating an attack. A good example of such a mechanism is one that gives a warning when a user enters an incorrect password three times. The login may continue, but an error message in a system log reports the unusually high number of mistyped passwords. Detection mechanisms do not prevent compromise of parts of the system, which is a serious drawback. The resource protected by the detection mechanism is continuously or periodically monitored for security problems.

*Recovery* has two forms. The first is to stop an attack and to assess and repair any damage caused by that attack. As an example, if the attacker deletes a file, one recovery mechanism would be to restore the file from backup tapes. In practice, recovery is far more complex, because the nature of each attack is unique. Thus, the type and extent of any damage can be difficult to characterize completely. Moreover, the attacker may return, so recovery involves identification and fixing of the vulnerabilities used by the attacker to enter the system. In some cases, retaliation (by attacking the attacker's system or taking legal steps to hold the attacker accountable) is part





of recovery. In all these cases, the system's functioning is inhibited by the attack. By definition, recovery requires resumption of correct operation.

In a second form of recovery, the system continues to function correctly while an attack is underway. This type of recovery is quite difficult to implement because of the complexity of computer systems. It draws on techniques of fault tolerance as well as techniques of security and is typically used in safety-critical systems. It differs from the first form of recovery, because at no point does the system function incorrectly. However, the system may disable nonessential functionality. Of course, this type of recovery is often implemented in a weaker form whereby the system detects incorrect functioning automatically and then corrects (or attempts to correct) the error.

---

## 1.4 Assumptions and Trust

How do we determine if the policy correctly describes the required level and type of security for the site? This question lies at the heart of all security, computer and otherwise. Security rests on assumptions specific to the type of security required and the environment in which it is to be employed.

**EXAMPLE:** Opening a door lock requires a key. The assumption is that the lock is secure against lock picking. This assumption is treated as an axiom and is made because most people would require a key to open a door lock. A good lock picker, however, can open a lock without a key. Hence, in an environment with a skilled, untrustworthy lock picker, the assumption is wrong and the consequence invalid.

If the lock picker is trustworthy, the assumption is valid. The term "trustworthy" implies that the lock picker will not pick a lock unless the owner of the lock authorizes the lock picking. This is another example of the role of trust. A well-defined exception to the rules provides a "back door" through which the security mechanism (the locks) can be bypassed. The trust resides in the belief that this back door will not be used except as specified by the policy. If it is used, the trust has been misplaced and the security mechanism (the lock) provides no security.

Like the lock example, a policy consists of a set of axioms that the policy makers believe can be enforced. Designers of policies always make two assumptions. First, the policy correctly and unambiguously partitions the set of system states into "secure" and "nonsecure" states. Second, the security mechanisms prevent the system from entering a "nonsecure" state. If either assumption is erroneous, the system will be nonsecure.

These two assumptions are fundamentally different. The first assumption asserts that the policy is a correct description of what constitutes a "secure" system. For example, a bank's policy may state that officers of the bank are authorized to shift money among accounts. If a bank officer puts \$100,000 in his account, has the bank's security



## 12 Chapter 1 An Overview of Computer Security

been violated? Given the aforementioned policy statement, no, because the officer was authorized to move the money. In the “real world,” that action would constitute embezzlement, something any bank would consider a security violation.

The second assumption says that the security policy can be enforced by security mechanisms. These mechanisms are either *secure*, *precise*, or *broad*. Let  $P$  be the set of all possible states. Let  $Q$  be the set of secure states (as specified by the security policy). Let the security mechanisms restrict the system to some set of states  $R$  (thus,  $R \subseteq P$ ). Then we have the following definition.

**Definition 1–3.** A security mechanism is *secure* if  $R \subseteq Q$ ; it is *precise* if  $R = Q$ ; and it is *broad* if there are states  $r$  such that  $r \in R$  and  $r \notin Q$ .

Ideally, the union of all security mechanisms active on a system would produce a single precise mechanism (that is,  $R = A$ ). In practice, security mechanisms are broad; they allow the system to enter nonsecure states. We will revisit this topic when we explore policy formulation in more detail.

Trusting that mechanisms work requires several assumptions.

1. Each mechanism is designed to implement one or more parts of the security policy.
2. The union of the mechanisms implements all aspects of the security policy.
3. The mechanisms are implemented correctly.
4. The mechanisms are installed and administered correctly.

Because of the importance and complexity of trust and of assumptions, we will revisit this topic repeatedly and in various guises throughout this book.

---

## 1.5 Assurance

Trust cannot be quantified precisely. System specification, design, and implementation can provide a basis for determining “how much” to trust a system. This aspect of trust is called *assurance*. It is an attempt to provide a basis for bolstering (or substantiating or specifying) how much one can trust a system.

**EXAMPLE:** In the United States, aspirin from a nationally known and reputable manufacturer, delivered to the drugstore in a safety-sealed container, and sold with the seal still in place, is considered trustworthy by most people. The bases for that trust are as follows.

- The testing and certification of the drug (aspirin) by the Food and Drug Administration. The FDA has jurisdiction over many types of

medicines and allows medicines to be marketed only if they meet certain clinical standards of usefulness.

- The manufacturing standards of the company and the precautions it takes to ensure that the drug is not contaminated. National and state regulatory commissions and groups ensure that the manufacture of the drug meets specific acceptable standards.
- The safety seal on the bottle. To insert dangerous chemicals into a safety-sealed bottle without damaging the seal is very difficult.

The three technologies (certification, manufacturing standards, and preventative sealing) provide some degree of assurance that the aspirin is not contaminated. The degree of trust the purchaser has in the purity of the aspirin is a result of these three processes.

In the 1980s, drug manufacturers met two of the criteria above, but none used safety seals.<sup>1</sup> A series of “drug scares” arose when a well-known manufacturer’s medicines were contaminated after manufacture but before purchase. The manufacturer promptly introduced safety seals to assure its customers that the medicine in the container was the same as when it was shipped from the manufacturing plants.

Assurance in the computer world is similar. It requires specific steps to ensure that the computer will function properly. The sequence of steps includes detailed specifications of the desired (or undesirable) behavior; an analysis of the design of the hardware, software, and other components to show that the system will not violate the specifications; and arguments or proofs that the implementation, operating procedures, and maintenance procedures will produce the desired behavior.

**Definition 1–4.** A system is said to *satisfy* a specification if the specification correctly states how the system will function.

This definition also applies to design and implementation satisfying a specification.

### 1.5.1 Specification

A *specification* is a (formal or informal) statement of the desired functioning of the system. It can be highly mathematical, using any of several languages defined for that purpose. It can also be informal, using, for example, English to describe what the system should do under certain conditions. The specification can be low-level, combining program code with logical and temporal relationships to specify ordering of events. The defining quality is a statement of what the system is allowed to do or what it is not allowed to do.

---

<sup>1</sup> Many used childproof caps, but they prevented only young children (and some adults) from opening the bottles. They were not designed to protect the medicine from malicious adults.



EXAMPLE: A company is purchasing a new computer for internal use. They need to trust the system to be invulnerable to attack over the Internet. One of their (English) specifications would read “The system cannot be attacked over the Internet.”

Specifications are used not merely in security but also in systems designed for safety, such as medical technology. They constrain such systems from performing acts that could cause harm. A system that regulates traffic lights must ensure that pairs of lights facing the same way turn red, green, and yellow at the same time and that at most one set of lights facing cross streets at an intersection is green.

A major part of the derivation of specifications is determination of the set of requirements relevant to the system’s planned use. Section 1.6 discusses the relationship of requirements to security.

### 1.5.2 Design

The *design* of a system translates the specifications into components that will implement them. The design is said to *satisfy* the specifications if, under all relevant circumstances, the design will not permit the system to violate those specifications.

EXAMPLE: A design of the computer system for the company mentioned above had no network interface cards, no modem cards, and no network drivers in the kernel. This design satisfied the specification because the system would not connect to the Internet. Hence it could not be attacked over the Internet.

An analyst can determine whether a design satisfies a set of specifications in several ways. If the specifications and designs are expressed in terms of mathematics, the analyst must show that the design formulations are consistent with the specifications. Although much of the work can be done mechanically, a human must still perform some analyses and modify components of the design that violate specifications (or, in some cases, components that cannot be shown to satisfy the specifications). If the specifications and design do not use mathematics, then a convincing and compelling argument should be made. Most often, the specifications are nebulous and the arguments are half-hearted and unconvincing or provide only partial coverage. The design depends on assumptions about what the specifications mean. This leads to vulnerabilities, as we will see.

### 1.5.3 Implementation

Given a design, the implementation creates a system that satisfies that design. If the design also satisfies the specifications, then by transitivity the implementation will also satisfy the specifications.

The difficulty at this step is the complexity of proving that a program correctly implements the design and, in turn, the specifications.



**Definition 1–5.** A program is *correct* if its implementation performs as specified.

Proofs of correctness require each line of source code to be checked for mathematical correctness. Each line is seen as a function, transforming the input (constrained by preconditions) into some output (constrained by postconditions derived from the function and the preconditions). Each routine is represented by the composition of the functions derived from the lines of code making up the routine. Like those functions, the function corresponding to the routine has inputs and outputs, constrained by preconditions and postconditions, respectively. From the combination of routines, programs can be built and formally verified. One can apply the same techniques to sets of programs and thus verify the correctness of a system.

There are three difficulties in this process. First, the complexity of programs makes their mathematical verification difficult. Aside from the intrinsic difficulties, the program itself has preconditions derived from the environment of the system. These preconditions are often subtle and difficult to specify, but unless the mathematical formalism captures them, the program verification may not be valid because critical assumptions may be wrong. Second, program verification assumes that the programs are compiled correctly, linked and loaded correctly, and executed correctly. Hardware failure, buggy code, and failures in other tools may invalidate the preconditions. A compiler that incorrectly compiles

```
x := x + 1
```

to

```
move x to regA
subtract 1 from contents of regA
move contents of regA to x
```

would invalidate the proof statement that the value of  $x$  after the line of code is 1 more than the value of  $x$  before the line of code. This would invalidate the proof of correctness. Third, if the verification relies on conditions on the input, the program must reject any inputs that do not meet those conditions. Otherwise, the program is only partially verified.

Because formal proofs of correctness are so time-consuming, *a posteriori* verification techniques known as *testing* have become widespread. During testing, the tester executes the program (or portions of it) on data to determine if the output is what it should be and to understand how likely the program is to contain an error. Testing techniques range from supplying input to ensure that all execution paths are exercised to introducing errors into the program and determining how they affect the output to stating specifications and testing the program to see if it satisfies the specifications. Although these techniques are considerably simpler than the more formal methods, they do not provide the same degree of assurance that formal methods do.



Furthermore, testing relies on test procedures and documentation, errors in either of which could invalidate the testing results.

Although assurance techniques do not guarantee correctness or security, they provide a firm basis for assessing what one must trust in order to believe that a system is secure. Their value is in eliminating possible, and common, sources of error and forcing designers to define precisely what the system is to do.

---

## 1.6 Operational Issues

Any useful policy and mechanism must balance the benefits of the protection against the cost of designing, implementing, and using the mechanism. This balance can be determined by analyzing the risks of a security breach and the likelihood of it occurring. Such an analysis is, to a degree, subjective, because in very few situations can risks be rigorously quantified. Complicating the analysis are the constraints that laws, customs, and society in general place on the acceptability of security procedures and mechanisms; indeed, as these factors change, so do security mechanisms and, possibly, security policies.

### 1.6.1 Cost-Benefit Analysis

Like any factor in a complex system, the benefits of computer security are weighed against their total cost (including the additional costs incurred if the system is compromised). If the data or resources cost less, or are of less value, than their protection, adding security mechanisms and procedures is not cost-effective because the data or resources can be reconstructed more cheaply than the protections themselves. Unfortunately, this is rarely the case.

**EXAMPLE:** A database provides salary information to a second system that prints checks. If the data in the database is altered, the company could suffer grievous financial loss; hence, even a cursory cost-benefit analysis would show that the strongest possible integrity mechanisms should protect the data in the database.

Now suppose the company has several branch offices, and every day the database downloads a copy of the data to each branch office. The branch offices use the data to recommend salaries for new employees. However, the main office makes the final decision using the original database (not one of the copies). In this case, guarding the integrity of the copies is not particularly important, because branch offices cannot make any financial decisions based on the data in their copies. Hence, the company cannot suffer any financial loss.

Both of these situations are extreme situations in which the analysis is clear-cut. As an example of a situation in which the analysis is less clear, consider the need



for confidentiality of the salaries in the database. The officers of the company must decide the financial cost to the company should the salaries be disclosed, including potential loss from lawsuits (if any); changes in policies, procedures, and personnel; and the effect on future business. These are all business-related judgments, and determining their value is part of what company officers are paid to do.

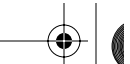
Overlapping benefits are also a consideration. Suppose the integrity protection mechanism can be augmented very quickly and cheaply to provide confidentiality. Then the cost of providing confidentiality is much lower. This shows that evaluating the cost of a particular security service depends on the mechanism chosen to implement it and on the mechanisms chosen to implement other security services. The cost-benefit analysis should take into account as many mechanisms as possible. Adding security mechanisms to an existing system is often more expensive (and, incidentally, less effective) than designing them into the system in the first place.

### 1.6.2 Risk Analysis

To determine whether an asset should be protected, and to what level, requires analysis of the potential threats against that asset and the likelihood that they will materialize. The level of protection is a function of the probability of an attack occurring and the effects of the attack should it succeed. If an attack is unlikely, protecting against it has a lower priority than protecting against a likely one. If the unlikely attack would cause long delays in the company's production of widgets but the likely attack would be only a nuisance, then more effort should be put into preventing the unlikely attack. The situations between these extreme cases are far more subjective.

Let's revisit our company with the salary database that transmits salary information over a network to a second computer that prints employees' checks. The data is stored on the database system and then moved over the network to the second system. Hence, the risk of unauthorized changes in the data occurs in three places: on the database system, on the network, and on the printing system. If the network is a local (company-wide) one and no wide area networks are accessible, the threat of attackers entering the systems is confined to untrustworthy internal personnel. If, however, the network is connected to the Internet, the risk of geographically distant attackers attempting to intrude is substantial enough to warrant consideration.

This example illustrates some finer points of risk analysis. First, risk is a function of environment. Attackers from a foreign country are not a threat to the company when the computer is not connected to the Internet. If foreign attackers wanted to break into the system, they would need physically to enter the company (and would cease to be "foreign" because they would then be "local"). But if the computer is connected to the Internet, foreign attackers become a threat because they can attack over the Internet. An additional, less tangible issue is the faith in the company. If the company is not able to meet its payroll because it does not know *whom* it is to pay, the company will lose the faith of its employees. It may be unable to hire anyone, because the people hired would not be sure they would get paid. Investors would not



fund the company because of the likelihood of lawsuits by unpaid employees. The risk arises from the environments in which the company functions.

Second, the risks change with time. If a company's network is not connected to the Internet, there seems to be no risk of attacks from other hosts on the Internet. However, despite any policies to the contrary, someone could connect a modem to one of the company computers and connect to the Internet through the modem. Should this happen, any risk analysis predicated on isolation from the Internet would no longer be accurate. Although policies can forbid the connection of such a modem and procedures can be put in place to make such connection difficult, unless the responsible parties can guarantee that no such modem will ever be installed, the risks can change.

Third, many risks are quite remote but still exist. In the modem example, the company has sought to minimize the risk of an Internet connection. Hence, this risk is "acceptable" but not nonexistent. As a practical matter, one does not worry about acceptable risks; instead, one worries that the risk will become unacceptable.

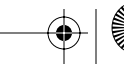
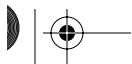
Finally, the problem of "analysis paralysis" refers to making risk analyses with no effort to act on those analyses. To change the example slightly, suppose the company performs a risk analysis. The executives decide that they are not sure if all risks have been found, so they order a second study to verify the first. They reconcile the studies then wait for some time to act on these analyses. At that point, the security officers raise the objection that the conditions in the workplace are no longer those that held when the original risk analyses were done. The analysis is repeated. But the company cannot decide how to ameliorate the risks, so it waits until a plan of action can be developed, and the process continues. The point is that the company is paralyzed and cannot act on the risks it faces.

### 1.6.3 Laws and Customs

Laws restrict the availability and use of technology and affect procedural controls. Hence, any policy and any selection of mechanisms must take into account legal considerations.

**EXAMPLE:** Until the year 2000, the United States controlled the export of cryptographic hardware and software (considered munitions under United States law). If a U.S. software company worked with a computer manufacturer in London, the U.S. company could not send cryptographic software to the manufacturer. The U.S. company first would have to obtain a license to export the software from the United States. Any security policy that depended on the London manufacturer using that cryptographic software would need to take this into account.

**EXAMPLE:** Suppose the law makes it illegal to read a user's file without the user's permission. An attacker breaks into the system and begins to download users' files. If the system administrators notice this and observe what the attacker is reading, they will be reading the victim's files without his permission and therefore will be violat-





ing the law themselves. For this reason, most sites require users to give (implicit or explicit) permission for system administrators to read their files. In some jurisdictions, an explicit exception allows system administrators to access information on their systems without permission in order to protect the quality of service provided or to prevent damage to their systems.

Complicating this issue are situations involving the laws of multiple jurisdictions—especially foreign ones.

**EXAMPLE:** In the 1990s, the laws involving the use of cryptography in France were very different from those in the United States. The laws of France required companies sending enciphered data out of the country to register their cryptographic keys with the government. Security procedures involving the transmission of enciphered data from a company in the United States to a branch office in France had to take these differences into account.

**EXAMPLE:** If a policy called for prosecution of attackers and intruders came from Russia to a system in the United States, prosecution would involve asking the United States authorities to extradite the alleged attackers from Russia. This undoubtedly would involve court testimony from company personnel involved in handling the intrusion, possibly trips to Russia, and more court time once the extradition was completed. The cost of prosecuting the attackers might be considerably higher than the company would be willing (or able) to pay.

Laws are not the only constraints on policies and selection of mechanisms. Society distinguishes between *legal* and *acceptable* practices. It may be legal for a company to require all its employees to provide DNA samples for authentication purposes, but it is not socially acceptable. Requiring the use of social security numbers as passwords is legal (unless the computer is one owned by the U.S. government) but also unacceptable. These practices provide security but at an unacceptable cost, and they encourage users to evade or otherwise overcome the security mechanisms.

The issue that laws and customs raise is the issue of psychological acceptability. A security mechanism that would put users and administrators at legal risk would place a burden on these people that few would be willing to bear; thus, such a mechanism would not be used. An unused mechanism is worse than a nonexistent one, because it gives a false impression that a security service is available. Hence, users may rely on that service to protect their data, when in reality their data is unprotected.

---

## 1.7 Human Issues

Implementing computer security controls is complex, and in a large organization procedural controls often become vague or cumbersome. Regardless of the strength



of the technical controls, if nontechnical considerations affect their implementation and use, the effect on security can be severe. Moreover, if configured or used incorrectly, even the best security control is useless at best and dangerous at worst. Thus, the designers, implementers, and maintainers of security controls are essential to the correct operation of those controls.

### 1.7.1 Organizational Problems

Security provides no direct financial rewards to the user. It limits losses, but it also requires the expenditure of resources that could be used elsewhere. Unless losses occur, organizations often believe they are wasting effort related to security. After a loss, the value of these controls suddenly becomes appreciated. Furthermore, security controls often add complexity to otherwise simple operations. For example, if concluding a stock trade takes two minutes without security controls and three minutes with security controls, adding those controls results in a 50% loss of productivity.

Losses occur when security protections are in place, but such losses are expected to be less than they would have been without the security mechanisms. The key question is whether such a loss, combined with the resulting loss in productivity, would be greater than a financial loss or loss of confidence should one of the nonsecured transactions suffer a breach of security.

Compounding this problem is the question of who is responsible for the security of the company's computers. The power to implement appropriate controls must reside with those who are responsible; the consequence of not doing so is that the people who can most clearly see the need for security measures, and who are responsible for implementing them, will be unable to do so. This is simply sound business practice; responsibility without power causes problems in any organization, just as does power without responsibility.

Once clear chains of responsibility and power have been established, the need for security can compete on an equal footing with other needs of the organization. The most common problem a security manager faces is the lack of people trained in the area of computer security. Another common problem is that knowledgeable people are overloaded with work. At many organizations, the "security administrator" is also involved in system administration, development, or some other secondary function. In fact, the security aspect of the job is often secondary. The problem is that indications of security problems often are not obvious and require time and skill to spot. Preparation for an attack makes dealing with it less chaotic, but such preparation takes enough time and requires enough attention so that treating it as a secondary aspect of a job means that it will not be performed well, with the expected consequences.

Lack of resources is another common problem. Securing a system requires resources as well as people. It requires time to design a configuration that will provide an adequate level of security, to implement the configuration, and to administer the system. It requires money to purchase products that are needed to build an adequate security system or to pay someone else to design and implement security mea-



tures. It requires computer resources to implement and execute the security mechanisms and procedures. It requires training to ensure that employees understand how to use the security tools, how to interpret the results, and how to implement the nontechnical aspects of the security policy.

### 1.7.2 People Problems

The heart of any security system is people. This is particularly true in computer security, which deals mainly with technological controls that can usually be bypassed by human intervention. For example, a computer system authenticates a user by asking that user for a secret code; if the correct secret code is supplied, the computer assumes that the user is authorized to use the system. If an authorized user tells another person his secret code, the unauthorized user can masquerade as the authorized user with significantly less likelihood of detection.

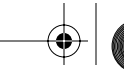
People who have some motive to attack an organization and are not authorized to use that organization's systems are called *outsiders* and can pose a serious threat. Experts agree, however, that a far more dangerous threat comes from disgruntled employees and other *insiders* who are authorized to use the computers. Insiders typically know the organization of the company's systems and what procedures the operators and users follow and often know enough passwords to bypass many security controls that would detect an attack launched by an outsider. Insider *misuse* of authorized privileges is a very difficult problem to solve.

Untrained personnel also pose a threat to system security. As an example, one operator did not realize that the contents of backup tapes needed to be verified before the tapes were stored. When attackers deleted several critical system files, she discovered that none of the backup tapes could be read.

System administrators who misread the output of security mechanisms, or do not analyze that output, contribute to the probability of successful attacks against their systems. Similarly, administrators who misconfigure security-related features of a system can weaken the site security. Users can also weaken site security by misusing security mechanisms (such as selecting passwords that are easy to guess).

Lack of training need not be in the technical arena. Many successful break-ins have arisen from the art of *social engineering*. If operators will change passwords based on telephone requests, all an attacker needs to do is to determine the name of someone who uses the computer. A common tactic is to pick someone fairly far above the operator (such as a vice president of the company) and to feign an emergency (such as calling at night and saying that a report to the president of the company is due the next morning) so that the operator will be reluctant to refuse the request. Once the password has been changed to one that the attacker knows, he can simply log in as a normal user. Social engineering attacks are remarkably successful and often devastating.

The problem of misconfiguration is aggravated by the complexity of many security-related configuration files. For instance, a typographical error can disable key protection features. Even worse, software does not always work as advertised.



One widely used system had a vulnerability that arose when an administrator made too long a list that named systems with access to certain files. Because the list was too long, the system simply assumed that the administrator meant to allow those files to be accessed without restriction on who could access them—exactly the opposite of what was intended.

---

## 1.8 Tying It All Together

The considerations discussed above appear to flow linearly from one to the next (see Figure 1–1). Human issues pervade each stage of the cycle. In addition, each stage of the cycle feeds back to the preceding stage, and through that stage to all earlier stages. The operation and maintenance stage is critical to the life cycle. Figure 1–1 breaks it out so as to emphasize the impact it has on all stages. The following example shows the importance of feedback.

**EXAMPLE:** A major corporation decided to improve its security. It hired consultants, determined the threats, and created a policy. From the policy, the consultants derived several specifications that the security mechanisms had to meet. They then developed a design that would meet the specifications.

During the implementation phase, the company discovered that employees could connect modems to the telephones without being detected. The design required

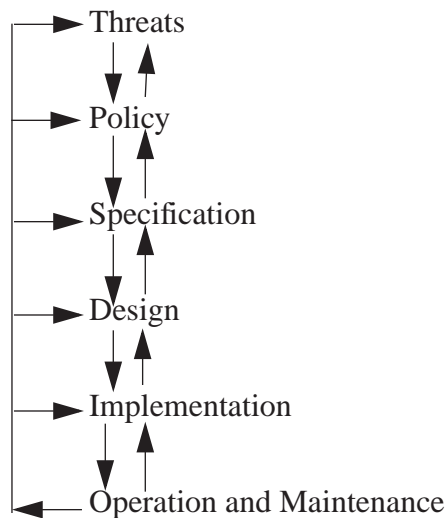
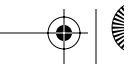
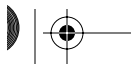


Figure 1–1 The security life cycle.



all incoming connections to go through a firewall. The design had to be modified to divide systems into two classes: systems connected to “the outside,” which were put outside the firewall; and all other systems, which were put behind the firewall. The design needed other modifications as well.

When the system was deployed, the operation and maintenance phase revealed several unexpected threats. The most serious was that systems were repeatedly misconfigured to allow sensitive data to be sent across the Internet in the clear. The implementation made use of cryptographic software very difficult. Once this problem had been remedied, the company discovered that several “trusted” hosts (those allowed to log in without authentication) were physically outside the control of the company. This violated policy, but for commercial reasons the company needed to continue to use these hosts. The policy element that designated these systems as “trusted” was modified. Finally, the company detected proprietary material being sent to a competitor over electronic mail. This added a threat that the company had earlier discounted. The company did not realize that it needed to worry about insider attacks.

Feedback from operation is critical. Whether or not a program is tested or proved to be secure, operational environments always introduce unexpected problems or difficulties. If the assurance (specification, design, implementation, and testing/proof) phase is done properly, the extra problems and difficulties are minimal. The analysts can handle them, usually easily and quickly. If the assurance phase has been omitted or done poorly, the problems may require a complete reevaluation of the system. The tools used for the feedback include auditing, in which the operation of the system is recorded and analyzed so that the analyst can determine what the problems are.

---

## 1.9 Summary

Computer security depends on many aspects of a computer system. The threats that a site faces, and the level and quality of the countermeasures, depend on the quality of the security services and supporting procedures. The specific mix of these attributes is governed by the site security policy, which is created after careful analysis of the value of the resources on the system or controlled by the system and of the risks involved.

Underlying all this are key assumptions describing what the site and the system accept as true or trustworthy; understanding these assumptions is the key to analyzing the strength of the system’s security. This notion of “trust” is the central notion for computer security. If trust is well placed, any system can be made acceptably secure. If it is misplaced, the system cannot be secure in any sense of the word.

Once this is understood, the reason that people consider security to be a relative attribute is plain. Given enough resources, an attacker can often evade the security procedures and mechanisms that are in place. Such a desire is tempered by the cost of the attack, which in some cases can be very expensive. If it is less expensive to regenerate the data than to launch the attack, most attackers will simply regenerate the data.

This chapter has laid the foundation for what follows. All aspects of computer security begin with the nature of threats and countering security services. In future chapters, we will build on these basic concepts.

---

## 1.10 Research Issues

Future chapters will explore research issues in the technical realm. However, other, nontechnical issues affect the needs and requirements for technical solutions, and research into these issues helps guide research into technical areas.

A key question is how to quantify risk. The research issue is how to determine the effects of a system's vulnerabilities on its security. For example, if a system can be compromised in any of 50 ways, how can a company compare the costs of the procedures (technical and otherwise) needed to prevent the compromises with the costs of detecting the compromises, countering them, and recovering from them? Many methods assign weights to the various factors, but these methods are *ad hoc*. A rigorous technique for determining appropriate weights has yet to be found.

The relationships of computer security to the political, social, and economic aspects of the world are not well understood. How does the ubiquity of the Internet change a country's borders? If someone starts at a computer in France, transits networks that cross Switzerland, Germany, Poland, Norway, Sweden, and Finland, and launches an attack on a computer in Russia, who has jurisdiction? How can a country limit the economic damage caused by an attack on its computer networks? How can attacks be traced to their human origins?

This chapter has also raised many technical questions. Research issues arising from them will be explored in future chapters.

---

## 1.11 Further Reading

Risk analysis arises in a variety of contexts. Molak [725] presents essays on risk management and analysis in a variety of fields. Laudan [610] provides an enjoyable introduction to the subject. Neumann [772] discusses the risks of technology and recent problems. Software safety (Leveson [622]) requires an understanding of the risks posed in the environment. Peterson [804] discusses many programming errors in a readable way. All provide insights into the problems that arise in a variety of environments.

Many authors recount stories of security incidents. The earliest, Parker's wonderful book [799], discusses motives and personalities as well as technical details. Stoll recounts the technical details of uncovering an espionage ring that began as the result of a 75¢ accounting error [973, 975]. Hafner and Markoff describe the same episode in a study of "cyberpunks" [432]. The Internet worm [322, 432, 845, 953]

brought the problem of computer security into popular view. Numerous other incidents [374, 432, 642, 914, 931, 968] have heightened public awareness of the problem.

Several books [59, 61, 824, 891] discuss computer security for the layperson. These works tend to focus on attacks that are visible or affect the end user (such as pornography, theft of credit card information, and deception). They are worth reading for those who wish to understand the results of failures in computer security.

---

## 1.12 Exercises

1. Classify each of the following as a violation of confidentiality, of integrity, of availability, or of some combination thereof.
  - a. John copies Mary's homework.
  - b. Paul crashes Linda's system.
  - c. Carol changes the amount of Angelo's check from \$100 to \$1,000.
  - d. Gina forges Roger's signature on a deed.
  - e. Rhonda registers the domain name "AddisonWesley.com" and refuses to let the publishing house buy or use that domain name.
  - f. Jonah obtains Peter's credit card number and has the credit card company cancel the card and replace it with another card bearing a different account number.
  - g. Henry spoofs Julie's IP address to gain access to her computer.
2. Identify mechanisms for implementing the following. State what policy or policies they might be enforcing.
  - a. A password changing program will reject passwords that are less than five characters long or that are found in the dictionary.
  - b. Only students in a computer science class will be given accounts on the department's computer system.
  - c. The login program will disallow logins of any students who enter their passwords incorrectly three times.
  - d. The permissions of the file containing Carol's homework will prevent Robert from cheating and copying it.
  - e. When World Wide Web traffic climbs to more than 80% of the network's capacity, systems will disallow any further communications to or from Web servers.
  - f. Annie, a systems analyst, will be able to detect a student using a program to scan her system for vulnerabilities.





12. Computer viruses are programs that, among other actions, can delete files without a user's permission. A U.S. legislator wrote a law banning the deletion of any files from computer disks. What was the problem with this law from a computer security point of view? Specifically, state which security service would have been affected if the law had been passed.
13. Users often bring in programs or download programs from the Internet. Give an example of a site for which the benefits of allowing users to do this outweigh the dangers. Then give an example of a site for which the dangers of allowing users to do this outweigh the benefits.
14. A respected computer scientist has said that no computer can ever be made perfectly secure. Why might she have said this?
15. An organization makes each lead system administrator responsible for the security of the system he or she runs. However, the management determines what programs are to be on the system and how they are to be configured.
  - a. Describe the security problem(s) that this division of power would create.
  - b. How would you fix them?
16. The president of a large software development company has become concerned about competitors learning proprietary information. He is determined to stop them. Part of his security mechanism is to require all employees to report any contact with employees of the company's competitors, even if it is purely social. Do you believe this will have the desired effect? Why or why not?
17. The police and the public defender share a computer. What security problems does this present? Do you feel it is a reasonable cost-saving measure to have all public agencies share the same (set of) computers?
18. Companies usually restrict the use of electronic mail to company business but do allow minimal use for personal reasons.
  - a. How might a company detect excessive personal use of electronic mail, other than by reading it? (*Hint*: Think about the personal use of a company telephone.)
  - b. Intuitively, it seems reasonable to ban *all* personal use of electronic mail on company computers. Explain why most companies do not do this.
19. Argue for or against the following proposition. Ciphers that the government cannot cryptanalyze should be outlawed. How would your argument change if such ciphers could be used provided that the users registered the keys with the government?
20. For many years, industries and financial institutions hired people who broke into their systems once those people were released from prison. Now, such a conviction tends to prevent such people from being hired.



28 **Chapter 1** An Overview of Computer Security

Why you think attitudes on this issue changed? Do you think they changed for the better or for the worse?

21. A graduate student accidentally releases a program that spreads from computer system to computer system. It deletes no files but requires much time to implement the necessary defenses. The graduate student is convicted. Despite demands that he be sent to prison for the maximum time possible (to make an example of him), the judge sentences him to pay a fine and perform community service. What factors do you believe caused the judge to hand down the sentence he did? What would you have done were you the judge, and what extra information would you have needed to make your decision?

