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

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

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.[1] 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.

[1] 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-

sures. 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

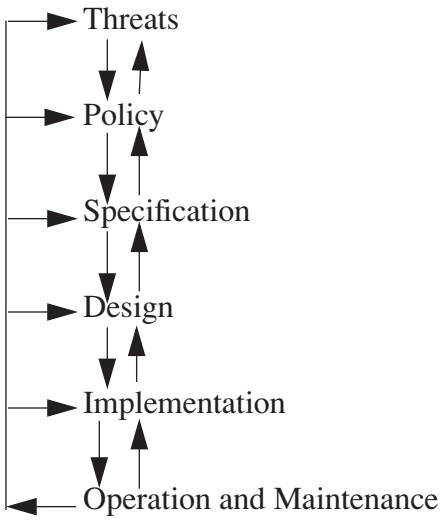

Figure 1–1 The security life cycle.

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.

g. A program used to submit homework will turn itself off just after the due date.

3. The aphorism "security through obscurity" suggests that hiding information provides some level of security. Give an example of a situation in which hiding information does not add appreciably to the security of a system. Then give an example of a situation in which it does.
4. Give an example of a situation in which a compromise of confidentiality leads to a compromise in integrity.
5. Show that the three security services—confidentiality, integrity, and availability—are sufficient to deal with the threats of disclosure, disruption, deception, and usurpation.
6. In addition to mathematical and informal statements of policy, policies can be implicit (not stated). Why might this be done? Might it occur with informally stated policies? What problems can this cause?
7. For each of the following statements, give an example of a situation in which the statement is true.
   a. Prevention is more important than detection and recovery.
   b. Detection is more important than prevention and recovery.
   c. Recovery is more important than prevention and detection.
8. Is it possible to design and implement a system in which *no* assumptions about trust are made? Why or why not?
9. Policy restricts the use of electronic mail on a particular system to faculty and staff. Students cannot send or receive electronic mail on that host. Classify the following mechanisms as secure, precise, or broad.
   a. The electronic mail sending and receiving programs are disabled.
   b. As each letter is sent or received, the system looks up the sender (or recipient) in a database. If that party is listed as faculty or staff, the mail is processed. Otherwise, it is rejected. (Assume that the database entries are correct.)
   c. The electronic mail sending programs ask the user if he or she is a student. If so, the mail is refused. The electronic mail receiving programs are disabled.
10. Consider a very high-assurance system developed for the military. The system has a set of specifications, and both the design and implementation have been proven to satisfy the specifications. What questions should school administrators ask when deciding whether to purchase such a system for their school's use?
11. How do laws protecting privacy impact the ability of system administrators to monitor user activity?

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.

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?

Chapter 13
Design Principles

FALSTAFF: If I had a thousand sons, the
first human principle I would teach them should
be, to forswear thin potations and to addict
themselves to sack.
—*The Second Part of King Henry the Fourth*, IV, iii, 133–136.

Specific design principles underlie the design and implementation of mechanisms for supporting security policies. These principles build on the ideas of simplicity and restriction. This chapter discusses those basic ideas and eight design principles.

13.1 Overview

Saltzer and Schroeder [865] describe eight principles for the design and implementation of security mechanisms. The principles draw on the ideas of simplicity and restriction.

Simplicity makes designs and mechanisms easy to understand. More importantly, less can go wrong with simple designs. Minimizing the interaction of system components minimizes the number of sanity checks on data being transmitted from one component to another.

EXAMPLE: The program *sendmail* reads configuration data from a binary file. System administrators generated the binary file by "freezing," or compiling, a text version of the configuration file. This created three interfaces: the mechanism used to edit the text file, the mechanism used to freeze the file, and the mechanism *sendmail* used to read the frozen file. The second interface required manual intervention and was often overlooked. To minimize this problem, *sendmail* checked that the frozen file was newer than the text file. If not, it warned the user to update the frozen configuration file.

The security problem lies in the assumptions that *sendmail* made. For example, the compiler would check that a particular option had an integer value. However, *sendmail* would not recheck; it assumed that the compiler had done the checking.

Errors in the compiler checks, or *sendmail*'s assumptions being inconsistent with those of the compiler, could produce security problems. If the compiler allowed the default UID to be a user name (say, *daemon* with a UID of 1), but *sendmail* assumed that it was an integer UID, then *sendmail* would scan the string "daemon" as though it were an integer. Most input routines would recognize that this string is not an integer and would default the return value to 0. Thus, *sendmail* would deliver mail with the *root* UID rather than with the desired *daemon* UID.

Simplicity also reduces the potential for inconsistencies within a policy or set of policies.

EXAMPLE: A college rule requires any teaching assistant who becomes aware of cheating to report it. A different rule ensures the privacy of student files. A TA contacts a student, pointing out that some files for a program were not submitted. The student tells the TA that the files are in the student's directory, and asks the TA to get the files. The TA does so, and while looking for the files notices two sets, one with names beginning with "x" and the other set not. Unsure of which set to use, the TA takes the first set. The comments show that they were written by a second student. The TA gets the second set, and the comments show that they were written by the first student. On comparing the two sets, the TA notes that they are identical except for the names in the comments. Although concerned about a possible countercharge for violation of privacy, the TA reports the student for cheating. As expected, the student charges the TA with violating his privacy by reading the first set of files. The rules conflict. Which charge or charges should be sustained?

Restriction minimizes the power of an entity. The entity can access only information it needs.

EXAMPLE: Government officials are denied access to information for which they have no need (the "need to know" policy). They cannot communicate that which they do not know.

Entities can communicate with other entities only when necessary, and in as few (and narrow) ways as possible.

EXAMPLE: All communications with prisoners are monitored. Prisoners can communicate with people on a list (given to the prison warden) through personal visits or mail, both of which are monitored to prevent the prisoners from receiving contraband such as files for cutting through prison bars or weapons to help them break out. The only exception to the monitoring policy is when prisoners meet with their attorneys. Such communications are privileged and so cannot be monitored.

"Communication" is used in its widest possible sense, including that of imparting information by not communicating.

EXAMPLE: Bernstein and Woodward, the reporters who broke the Watergate scandal, describe an attempt to receive information from a source without the source directly answering the question. They suggested a scheme in which the source would hang up if the information was inaccurate and remain on the line if the information was accurate. The source remained on the line, confirming the information [85].

13.2 Design Principles

The principles of secure design discussed in this section express common-sense applications of simplicity and restriction in terms of computing. We will discuss detailed applications of these principles throughout the remainder of Part 5, and in Part 8, "Practicum." However, we will mention examples here.

13.2.1 Principle of Least Privilege

This principle restricts how privileges are granted.

> **Definition 13–1.** The *principle of least privilege* states that a subject should be given only those privileges that it needs in order to complete its task.

If a subject does not need an access right, the subject should not have that right. Furthermore, the *function* of the subject (as opposed to its identity) should control the assignment of rights. If a specific action requires that a subject's access rights be augmented, those extra rights should be relinquished *immediately* on completion of the action. This is the analogue of the "need to know" rule: if the subject does not need access to an object to perform its task, it should not have the right to access that object. More precisely, if a subject needs to append to an object, but not to alter the information already contained in the object, it should be given append rights and not write rights.

In practice, most systems do not have the granularity of privileges and permissions required to apply this principle precisely. The designers of security mechanisms then apply this principle as best they can. In such systems, the consequences of security problems are often more severe than the consequences for systems that adhere to this principle.

EXAMPLE: The UNIX operating system does not apply access controls to the user *root*. That user can terminate any process and read, write, or delete any file. Thus, users who create backups can also delete files. The *administrator* account on Windows has the same powers.

This principle requires that processes should be confined to as small a protection domain as possible.

EXAMPLE: A mail server accepts mail from the Internet and copies the messages into a spool directory; a local server will complete delivery. The mail server needs the rights to access the appropriate network port, to create files in the spool directory, and to alter those files (so it can copy the message into the file, rewrite the delivery address if needed, and add the appropriate "Received" lines). It should surrender the right to access the file as soon as it has finished writing the file into the spool directory, because it does not need to access that file again. The server should not be able to access any user's files, or any files other than its own configuration files.

13.2.2 Principle of Fail-Safe Defaults

This principle restricts how privileges are initialized when a subject or object is created.

> **Definition 13–2.** The *principle of fail-safe defaults* states that, unless a subject is given explicit access to an object, it should be denied access to that object.

This principle requires that the default access to an object is *none*. Whenever access, privileges, or some security-related attribute is not *explicitly* granted, it should be denied. Moreover, if the subject is unable to complete its action or task, it should undo those changes it made in the security state of the system before it terminates. This way, even if the program fails, the system is still safe.

EXAMPLE: If the mail server is unable to create a file in the spool directory, it should close the network connection, issue an error message, and stop. It should *not* try to store the message elsewhere or to expand its privileges to save the message in another location, because an attacker could use that ability to overwrite other files or fill up other disks (a denial of service attack). The protections on the mail spool directory itself should allow create and write access only to the mail server and read and delete access only to the local server. No other user should have access to the directory.

In practice, most systems will allow an administrator access to the mail spool directory. By the principle of least privilege, that administrator should be able to access *only* the subjects and objects involved in mail queueing and delivery. As we have seen, this constraint minimizes the threats if that administrator's account is compromised. The mail system can be damaged or destroyed, but nothing else can be.

13.2.3 Principle of Economy of Mechanism

This principle simplifies the design and implementation of security mechanisms.

> **Definition 13–3.** The *principle of economy of mechanism* states that security mechanisms should be as simple as possible.

If a design and implementation are simple, fewer possibilities exist for errors. The checking and testing process is less complex, because fewer components and cases need to be tested. Complex mechanisms often make assumptions about the system and environment in which they run. If these assumptions are incorrect, security problems may result.

EXAMPLE: The *ident* protocol [861] sends the user name associated with a process that has a TCP connection to a remote host. A mechanism on host *A* that allows access based on the results of an ident protocol result makes the assumption that the originating host is trustworthy. If host *B* decides to attack host *A*, it can connect and then send any identity it chooses in response to the ident request. This is an example of a mechanism making an incorrect assumption about the environment (specifically, that host *B* can be trusted).

Interfaces to other modules are particularly suspect, because modules often make implicit assumptions about input or output parameters or the current system state; should any of these assumptions be wrong, the module's actions may produce unexpected, and erroneous, results. Interaction with external entities, such as other programs, systems, or humans, amplifies this problem.

EXAMPLE: The *finger* protocol transmits information about a user or system [1072]. Many client implementations assume that the server's response is well-formed. However, if an attacker were to create a server that generated an infinite stream of characters, and a finger client were to connect to it, the client would print all the characters. As a result, log files and disks could be filled up, resulting in a denial of service attack on the querying host. This is an example of incorrect assumptions about the input to the client.

13.2.4 Principle of Complete Mediation

This principle restricts the caching of information, which often leads to simpler implementations of mechanisms.

> **Definition 13–4.** The *principle of complete mediation* requires that all accesses to objects be checked to ensure that they are allowed.

Whenever a subject attempts to read an object, the operating system should mediate the action. First, it determines if the subject is allowed to read the object. If so, it provides the resources for the read to occur. If the subject tries to read the object again, the system should check that the subject is still allowed to read the object. Most systems would not make the second check. They would cache the results of the first check and base the second access on the cached results.

EXAMPLE: When a UNIX process tries to read a file, the operating system determines if the process is allowed to read the file. If so, the process receives a file descriptor encoding the allowed access. Whenever the process wants to read the file, it presents the file descriptor to the kernel. The kernel then allows the access.

If the owner of the file disallows the process permission to read the file after the file descriptor is issued, the kernel still allows access. This scheme violates the principle of complete mediation, because the second access is not checked. The cached value is used, resulting in the denial of access being ineffective.

EXAMPLE: The Domain Name Service (DNS) caches information mapping host names into IP addresses. If an attacker is able to "poison" the cache by implanting records associating a bogus IP address with a name, one host will route connections to another host incorrectly. Section 14.6.1.2 discusses this in more detail.

13.2.5 Principle of Open Design

This principle suggests that complexity does not add security.

> **Definition 13–5.** The *principle of open design* states that the security of a mechanism should not depend on the secrecy of its design or implementation.

Designers and implementers of a program must not depend on secrecy of the details of their design and implementation to ensure security. Others can ferret out such details either through technical means, such as disassembly and analysis, or through nontechnical means, such as searching through garbage receptacles for source code listings (called "dumpster-diving"). If the strength of the program's security depends on the ignorance of the user, a knowledgeable user can defeat that security mechanism. The term "security through obscurity" captures this concept exactly.

This is especially true of cryptographic software and systems. Because cryptography is a highly mathematical subject, companies that market cryptographic software or use cryptography to protect user data frequently keep their algorithms secret. Experience has shown that such secrecy adds little if anything to the security of the system. Worse, it gives an aura of strength that is all too often lacking in the actual implementation of the system.

Keeping cryptographic keys and passwords secret does *not* violate this principle, because a key is not an algorithm. However, keeping the enciphering and deciphering algorithms secret would violate it.

Issues of proprietary software and trade secrets complicate the application of this principle. In some cases, companies may not want their designs made public, lest their competitors use them. The principle then requires that the design and implementation be available to people barred from disclosing it outside the company.

EXAMPLE: The Content Scrambling System (CSS) is a cryptographic algorithm that protects DVD movie disks from unauthorized copying. The DVD disk has an authentication key, a disk key, and a title key. The title key is enciphered with the disk key. A block on the DVD contains several copies of the disk key, each enciphered by a different player key, and a checksum of the disk key. When a DVD is inserted into a DVD player, the algorithm reads the authentication key. It then deciphers the disk keys using the DVD player's unique key. When it finds a deciphered key with the correct hash, it uses that key to decipher the title key, and it uses the title key to decipher the movie [971]. (Figure 13–1 shows the layout of the keys.) The authentication and disk keys are not located in the file containing the movie, so if one copies the file, one still needs the DVD disk in the DVD player to be able to play the movie.

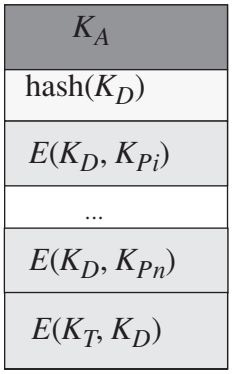

Figure 13–1 DVD key layout. K_A is the authentication key, K_T the title key, K_D the disk key, and K_{Pi} the key for DVD player i. The disk key is enciphered once for each player key.

In 1999, a group in Norway acquired a (software) DVD playing program that had an unenciphered key. They also derived an algorithm completely compatible with the CSS algorithm from the software. This enabled them to decipher any DVD movie file. Software that could perform these functions rapidly became available throughout the Internet, much to the discomfort of the DVD Copyright Control Association, which promptly sued to prevent the code from being made public [783, 798]. As if to emphasize the problems of providing security by concealing algorithms, the plaintiff's lawyers filed a declaration containing the source code of an implementation of the CSS algorithm. When they realized this, they requested that the declaration be sealed from public view. By then, the declaration had been posted on several Internet sites, including one that had more than 21,000 downloads of the declaration before the court sealed it [671].

13.2.6 Principle of Separation of Privilege

This principle is restrictive because it limits access to system entities.

> **Definition 13–6.** The *principle of separation of privilege* states that a system should not grant permission based on a single condition.

This principle is equivalent to the separation of duty principle discussed in Section 6.1. Company checks for more than $75,000 must be signed by two officers of the company. If either does not sign, the check is not valid. The two conditions are the signatures of both officers.

Similarly, systems and programs granting access to resources should do so only when more than one condition is met. This provides a fine-grained control over the resource as well as additional assurance that the access is authorized.

EXAMPLE: On Berkeley-based versions of the UNIX operating system, users are not allowed to change from their accounts to the *root* account unless two conditions are met. The first condition is that the user knows the *root* password. The second condition is that the user is in the *wheel* group (the group with GID 0). Meeting either condition is not sufficient to acquire *root* access; meeting both conditions is required.

13.2.7 Principle of Least Common Mechanism

This principle is restrictive because it limits sharing.

> **Definition 13–7.** The *principle of least common mechanism* states that mechanisms used to access resources should not be shared.

Sharing resources provides a channel along which information can be transmitted, and so such sharing should be minimized. In practice, if the operating system provides support for virtual machines, the operating system will enforce this privilege automatically to some degree (see Chapter 17, "Confinement Problem"). Otherwise, it will provide some support (such as a virtual memory space) but not complete support (because the file system will appear as shared among several processes).

EXAMPLE: A Web site provides electronic commerce services for a major company. Attackers want to deprive the company of the revenue it obtains from that Web site. They flood the site with messages and tie up the electronic commerce services. Legitimate customers are unable to access the Web site and, as a result, take their business elsewhere.

Here, the sharing of the Internet with the attackers' sites caused the attack to succeed. The appropriate countermeasure would be to restrict the attackers' access to the segment of the Internet connected to the Web site. Techniques for doing this include proxy servers such as the Purdue SYN intermediary [893] or traffic throttling (see Section 26.4, "Availability and Network Flooding"). The former targets suspect connections; the latter reduces the load on the relevant segment of the network indiscriminately.

13.2.8 Principle of Psychological Acceptability

This principle recognizes the human element in computer security.

> **Definition 13–8.** The *principle of psychological acceptability* states that security mechanisms should not make the resource more difficult to access than if the security mechanisms were not present.

Configuring and executing a program should be as easy and as intuitive as possible, and any output should be clear, direct, and useful. If security-related software is too complicated to configure, system administrators may unintentionally set up the software in a nonsecure manner. Similarly, security-related user programs must be easy to use and must output understandable messages. If a password is rejected, the password changing program should state why it was rejected rather than giving a cryptic error message. If a configuration file has an incorrect parameter, the error message should describe the proper parameter.

EXAMPLE: The *ssh* program [1065] allows a user to set up a public key mechanism for enciphering communications between systems. The installation and configuration mechanisms for the UNIX version allow one to arrange that the public key be stored locally without any password protection. In this case, one need not supply a password to connect to the remote system, but will still obtain the enciphered connection. This mechanism satisfies the principle of psychological acceptability.

On the other hand, security requires that the messages impart no unnecessary information.

EXAMPLE: When a user supplies the wrong password during login, the system should reject the attempt with a message stating that the login failed. If it were to say that the password was incorrect, the user would know that the account name was legitimate. If the "user" were really an unauthorized attacker, she would then know the name of an account for which she could try to guess a password.

In practice, the principle of psychological acceptability is interpreted to mean that the security mechanism may add some extra burden, but that burden must be both minimal and reasonable.

EXAMPLE: A mainframe system allows users to place passwords on files. Accessing the files requires that the program supply the password. Although this mechanism violates the principle as stated, it is considered sufficiently minimal to be acceptable. On an interactive system, where the pattern of file accesses is more frequent and more transient, this requirement would be too great a burden to be acceptable.

13.3 Summary

The design principles discussed in this chapter are fundamental to the design and implementation of security mechanisms. They encompass not only technical details but also human interaction. Several principles come from nontechnical environments, such as the principle of least privilege. Each principle involves the restriction

of privilege according to some criterion, or the minimization of complexity to make the mechanisms less likely to fail.

13.4 Research Issues

These principles pervade all research touching on the design and implementation of secure systems. The principle of least privilege raises the issue of granularity of privilege. Is a "write" privilege sufficient, or should it be fragmented—for example, into "write" and "write at the end" or "append," or into the ability to write to specific blocks? How does the multiplicity of rights affect system administration and security management? How does it affect architecture and performance? How does it affect the user interface and the user's model of the system?

Least common mechanism problems arise when dealing with denial of service attacks, because such attacks exploit shared media. The principle of least common mechanism plays a role in handling covert channels, which are discussed further in Chapter 17.

Separation of privilege arises in the creation of user and system roles. How much power should administrative accounts have? How should they work together? These issues arise in role-based access control, which is discussed in Section 7.4.

The principle of complete mediation runs counter to the philosophy of caching. One caches data to keep from having to retrieve the information when it is next needed, but complete mediation requires the retrieval of access permissions. How are these conflicting forces balanced in practice?

Research in software and systems design and implementation studies the application of the principle of economy of mechanism. How can interfaces be made simple and consistent? How can the various design paradigms lead to better-crafted, simpler software and systems?

Whether "open source" software (software the source of which is publicly available) is more secure than other software is a complex question. Analysts can check open source software for security problems more easily than they can software for which no source is available. Knowing that one's coding will be available for public scrutiny should encourage programmers to write better, tighter code. On the other hand, attackers can also look at the source code for security flaws, and various pressures (such as time to market) weigh against careful coding. Furthermore, the debate ignores security problems introduced by misconfigured software, or software used incorrectly.

Experimental data for the debate about the efficacy of open source software is lacking. An interesting research project would be to design an experiment that would provide evidence either for or against the proposition that if source code for software is available, then that software has (or causes) fewer security problems than software for which source code is not available. Part of the research would be to determine how to make this question precise, what metrics and statistical techniques should be used to analyze the data, and how the data should be collected.

13.5 Further Reading

Many papers discuss the application of these principles to security mechanisms. Succeeding chapters will present references for this aspect of the principles. Other papers present different sets of principles. These papers are generally specializations or alternative views of Saltzer and Schroeder's principles, tailored for particular environments. Abadi and Needham [2] and Anderson and Needham [32] discuss principles for the design of cryptographic protocols; Syverson discusses their limits [986]. Moore [729] and Abadi [1] describe problems in cryptographic protocols. Wood [1057, 1058] discusses principles for secure systems design with an emphasis on groupware. Bonyun [133] focuses on architectural principles. Landwehr and Goldschlag [615] present principles for Internet security.

13.6 Exercises

1. The PostScript language [11] describes page layout for printers. Among its features is the ability to request that the interpreter execute commands on the host system.
    a. Describe a danger that this feature presents when the language interpreter is running with administrative or *root* privileges.
    b. Explain how the principle of least privilege could be used to ameliorate this danger.
2. A common technique for inhibiting password guessing is to disable an account after three consecutive failed login attempts.
    a. Discuss how this technique might prevent legitimate users from accessing the system. Why is this action a violation of the principle of least common mechanism?
    b. One can argue that this is an example of fail-safe defaults, because by blocking access to an account under attack, the system is defaulting to a known, safe state. Do you agree or disagree with this argument? Justify your answer.
3. Kernighan and Plauger [565] argue a minimalist philosophy of tool building. Their thesis is that each program should perform exactly one task, and more complex programs should be formed by combining simpler programs. Discuss how this philosophy fits in with the principle of economy of mechanism. In particular, how does the advantage of the simplicity of each component of a software system offset the disadvantage of a multiplicity of interfaces among the various components?

4. Design an experiment to determine the performance impact of checking access permissions for each file *access* (as opposed to once at the file's opening). If you have access to a system on which you can modify the file access mechanism, run your experiment and determine the impact.
5. A company publishes the design of its security software product in a manual that accompanies the executable software.
    a. In what ways does this satisfy the principle of open design? In what ways does it not?
    b. Given that the design is known, what advantages does keeping the source code unavailable give the company and those who purchase the software? What disadvantages does it cause?
6. Assume that processes on a system share no resources. Is it possible for one process to block another process' access to a resource? Why or why not? From your answer, argue that denial of service attacks are possible or impossible.
7. Given that the Internet is a shared network, discuss whether preventing denial of service attacks is inherently possible or not possible. Do systems connected to the Internet violate the principle of least common mechanism?
8. A program called *lsu* [111] gives access to role accounts. The user's access rights are checked, and the user is required to enter her password. If access rules allow the change and the user's password is correct, *lsu* allows the change. Given that Mary uses *lsu* from her account, why does *lsu* require her to enter her password? Name the principles involved, and why they require this.
9. Recall the S/Key one-time password algorithm discussed in Section 12.3.2. When a user prints a list of S/Key passwords for future use, the system encodes each hash value as a set of six short words and prints them. Why does it not merely print out the hash values?
10. The program *su* enables a UNIX user to access another user's account. Unless the first user is the superuser, *su* requires that the password of the second user be given. A (possibly apocryphal) version of *su* would ask for the user's password and, if it could not determine if the password was correct because the password file could not be opened, *immediately* grant superuser access so that the user could fix the problem. Discuss which of the design principles this approach meets, and which ones it violates.