CHAPTER 1

Security Goals

The two main objectives in the first three chapters of this book are to establish the key goals of computer security and to provide an overview of the core principles of secure systems design.

Chapter 1 focuses on the role that technological security plays in the design of a large secure system, and the seven key concepts in the field of security:

- Authentication
- Authorization
- Confidentiality
- Data/message integrity
- Accountability
- Availability
- Non-repudiation

After discussing these concepts, we will then illustrate the role they play in the scope of a larger system by looking at an example of a web client interacting with a web server, and examining the contribution these key concepts make in that interaction.

1.1. Security Is Holistic

Technological security and all the other computer security mechanisms we discuss in this book make up only one component of ensuring overall, holistic security to your system. By technological security, we mean application security, operating system (OS) security, and network security. In addition to discussing what it means to have application, OS, and network security, we will touch upon physical security, and policies and procedures. Achieving holistic security requires physical security, technological security, and good policies and procedures. Having just one or two of these types of security is usually not sufficient to achieve security: all three are typically required. An organization that has advanced technological security mechanisms in place but does not train its employees to safeguard their passwords with care will not be secure overall. The bulk of this book focuses on technological security, and we briefly comment on physical security and policies and procedures in this chapter, as security is holistic. However, our coverage of physical security and policies and procedures do not do the topics justice—for more information, we would encourage you to do the following:

- Read standards such as ISO 17799 (see `www.iso.org/iso/en/prods-services/popstds/informationsecurity.html` and `www.computersecuritynow.com`).
- Visit sites such as the SANS Security Policy Project (`www.sans.org/resources/policies`) for more information on security policies and procedures.
- Read *Practical UNIX and Network Security*, by Simson Garfinkel, Gene Spafford, and Alan Schwartz, for more on operating system and network security.

1.1.1. Physical Security

Physically securing your system and laying down good policies for employees and users is often just as important as using the technological security mechanisms that we cover in this book. All of your servers should be behind locked doors, and only a privileged set of employees (typically system and security administrators) should have access to them. In addition, data centers used to house farms of servers can employ cameras, card reader and biometric locks, and even "vaults" of various kinds, depending upon the sensitivity of data stored on the servers.

In addition to mechanisms that limit access to a physical space to prevent asset theft and unauthorized entry, there are also mechanisms that protect against information leakage and document theft. Documents containing sensitive information can be shredded before they're disposed of so that determined hackers can be prevented from gathering sensitive information by sifting through the company's garbage. Such an attack is often referred to as *dumpster diving*.

1.1.2. Technological Security

In addition to physical security, there are many technical levels of security that are important. Technological security can be divided into three components: application security, OS security, and network security.

Note that our use of the word *technological* to group together application, OS, and network security may not be the best of terms! Clearly, various types of technology can also be used to achieve physical security. For example, employees can be given electronic badges, and badge readers can be put on the door of the server room. The badges and readers clearly employ technology—but here, we use the term *technological security* to refer to software-related application, OS, and network security technology.

Application Security

A web server is an example of an application that can suffer from security problems. In this chapter, and throughout the rest of the book, we use web servers to illustrate many application-layer security vulnerabilities. The deployment scenario that we have in mind for a web server is shown in Figure 1-1.

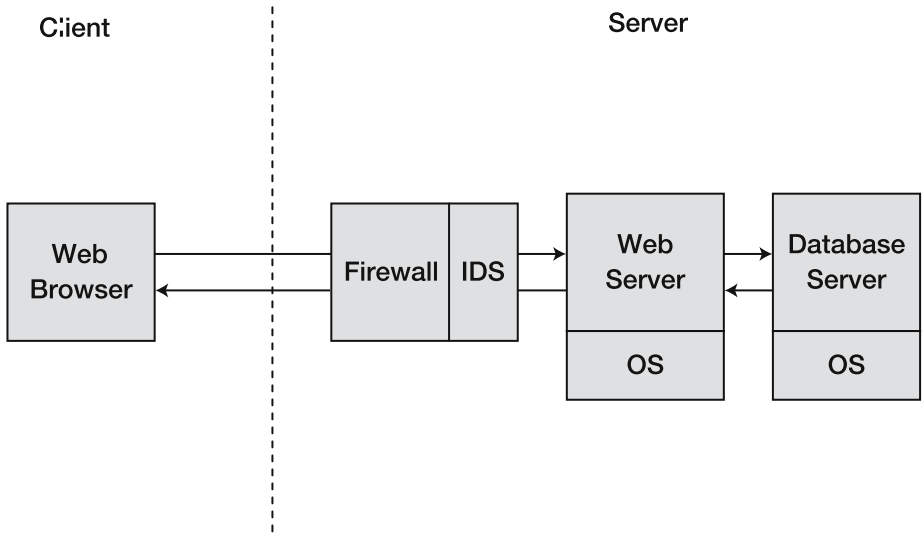

Figure 1-1. *A typical web server deployment scenario*

Consider a scenario in which a web server is configured to allow only certain users to download valuable documents. In this scenario, a vulnerability can arise if there is a bug in how it ascertains the identity of the user. If the user's identity is not ascertained properly, it may be possible for an attacker to get access to valuable documents to which she would not otherwise be granted access.

In addition to ensuring that there are no flaws in the identity verification process used by a web server, it is also important that you configure your web server correctly. Web servers are complicated pieces of software that have many options that can be turned on or off. For instance, a web server may have an option that, when turned on, allows it to serve content from a database; or when turned off, only allows it to serve files from its local file system. Administrators must ensure that their web servers are configured correctly, so as to minimize the possible methods of attack.

By restricting a web server to only serve files from its local file system, you prevent an attacker from taking advantage of vulnerabilities in how a web server uses a back-end database. It is possible for a malicious user to trick a web server into sending user-submitted data to the database as a command, and thereby take control of the database. (One example of such an attack is a SQL injection attack—we cover such attacks in Chapter 8.)

However, even if a web server is not configured to connect to a database, other configuration options might, for instance, make available files on the local file system that a web server administrator did not intend to make accessible. For instance, if a web server is configured to make all types of files stored on its file system available for download, then any sensitive spreadsheets stored on the local file system, for example, could be downloaded just as easily as web documents and images. An attacker may not even need to probe web servers individually to find such documents. A search engine can inadvertently crawl and index sensitive documents, and the attacker can simply enter the right keywords into the search engine to discover such sensitive documents (Long 2004).

Another example of an application that could have a security vulnerability is a web browser. Web browsers download and interpret data from web sites on the Internet. Sometimes web browsers do not interpret data in a robust fashion, and can be directed to download data from malicious web sites. A malicious web site can make available a file that exploits a vulnerability in web browser code that can give the attacker control of the machine that the web browser is running on. As a result of poor coding, web browser code needs to be regularly "patched" to eliminate such vulnerabilities, such as buffer overflows (as discussed in Chapter 6). The creators of the web browser can issue patches that can be installed to eliminate the vulnerabilities in the web browser. A *patch* is an updated version of the software. The patch does not have to consist of an entirely updated version, but may contain only components that have been fixed to eliminate security-related bugs.

OS Security

In addition to application security, OS security is also important. Your operating system—whether it is Linux, Windows, or something else—also must be secured. Operating systems themselves are not inherently secure or insecure. Operating systems are made up of tens or hundreds of millions of lines of source code, which most likely contain vulnerabilities. Just as is the case for applications, OS vendors typically issue patches regularly to eliminate such vulnerabilities. If you use Windows, chances are that you have patched your operating system at least once using the Windows Update feature. The Windows Update feature periodically contacts Microsoft's web site to see if any critical system patches (including security patches) need to be installed on your machine. If so, Windows pops up a small dialog box asking you if it is OK to download the patch and reboot your machine to install it.

It is possible that an attacker might try to exploit some vulnerability in the operating system, even if you have a secure web server running. If there is a security vulnerability in the operating system, it is possible for an attacker to work around a secure web server, since web servers rely on the operating system for many functions.

Network Security

Network layer security is important as well—you need to ensure that only valid data packets are delivered to your web server from the network, and that no malicious traffic can get routed to your applications or operating system. Malicious traffic typically consists of data packets that contain byte sequences that, when interpreted by software, will produce a result unexpected to the user, and may cause the user's machine to fail, malfunction, or provide access to privileged information to an attacker. Firewalls and intrusion detection systems (IDSs) are two types of tools that you can use to help deal with potentially malicious network traffic.

1.1.3. Policies and Procedures

Finally, it is important to recognize that even if your system is physically and technologically secure, you still need to establish a certain set of policies and procedures for all of your employees to ensure overall security. For example, each employee may need to be educated to never give out his or her password for any corporate system, even if asked by a security administrator. Most good password systems are designed so that security and system administrators have the capability to reset passwords, and should never need to ask a user for her existing password to reset it to a new one.

Attackers can potentially exploit a gullible employee by impersonating another employee within the company and convincing him (say, over the phone) to tell him his username or password. Such an attack is called a *social engineering* attack, and is geared at taking advantage of unsuspecting employees. Even if your applications, operating systems, and networks are secure, and your servers are behind locked doors, an attacker can still conduct social engineering attacks to work around the security measures you have in place.

As evidenced by the threat of social engineering, it is important to have policies and procedures in place to help guard sensitive corporate information. Writing down such policies and procedures on paper or posting them on the company intranet is not enough. Your employees need to be aware of them, and they need to be educated to be somewhat paranoid and vigilant to create a secure environment. A combination of physical security, technological security mechanisms, and employees who follow policies and procedures can result in improved overall security for your environment.

It is often said that "security is a process, not a product" (Schneier 2000). There is much more to security than just technology, and it is important to weigh and consider risks from all relevant threat sources.

ARCHETYPAL CHARACTERS

We are going to spend the rest of this chapter illustrating seven key technological security goals (authentication, authorization, confidentiality, message/data integrity, accountability, availability, and non-repudiation). We will do so with the help of a few fictitious characters that are often used in the field of computer security. The first two fictitious characters are Alice and Bob, who are both "good guys" trying to get some useful work done. Their work may often involve the exchange of secret information. Alice and Bob unfortunately have some adversaries that are working against them—namely Eve and Mallory.

Another person that we will occasionally use in our examples is a gentleman by the name of Trent. Trent is a trusted third party. In particular, Trent is trusted by Alice and Bob. Alice and Bob can rely on Trent to help them get some of their work accomplished. We will provide more details about Alice, Bob, Eve, Mallory, and Trent as necessary, and we encourage you to learn more about them by reading "The Story of Alice and Bob" (Gordon 1984).

1.2. Authentication

Authentication is the act of verifying someone's identity. When exploring authentication with our fictitious characters Alice and Bob, the question we want to ask is: if Bob wants to communicate with Alice, how can he be sure that he is communicating with Alice and not someone trying to impersonate her? Bob may be able to authenticate and verify Alice's identity based on one or more of three types of methods: something you know, something you have, and something you are.

1.2.1. Something You Know

The first general method Bob can use to authenticate Alice is to ask her for some secret only she should know, such as a password. If Alice produces the right password, then Bob can

assume he is communicating with Alice. Passwords are so prevalently used that we dedicate Chapter 9 to studying how to properly build a password management system.

There are advantages and disadvantages to using passwords. One advantage is that password schemes are simple to implement compared to other authentication mechanisms, such as biometrics, which we will discuss later in this chapter. Another advantage of password security systems is that they are simple for users to understand.

There are, however, disadvantages to using password security systems. First, most users do not choose strong passwords, which are hard for attackers to guess. Users usually choose passwords that are simple concatenations of common names, common dictionary words, common street names, or other easy-to-guess terms or phrases. Attackers interested in hacking into somebody's account can use password-cracking programs to try many common login names and concatenations of common words as passwords. Such password cracking programs can easily determine 10 to 20 percent of the usernames and passwords in a system. Of course, to gain access to a system, an attacker typically needs only one valid username and password. Passwords are relatively easy to crack, unless users are somehow forced to choose passwords that are hard for such password-cracking programs to guess. A second disadvantage of password security systems is that a user needs to reuse a password each time she logs into a system—that gives an attacker numerous opportunities to "listen in" (see Section 1.4) on that password. If the attacker can successfully listen in on a password just once, the attacker can then log in as the user.

A one-time password (OTP) system, which forces the user to enter a new password each time she logs in, eliminates the risks of using a password multiple times. With this system, the user is given a list of passwords—the first time she logs in, she is asked for the first password; the second time she logs in, she is asked the second password; and so on. The major problem with this system is that no user will be able to remember all these passwords. However, a device could be used that keeps track of all the different passwords the user would need to use each time she logs in. This basic idea of such a device naturally leads us from the topic of "something you know" to the topic of "something you have."

1.2.2. Something You Have

A second general method of authenticating a user is based on something that the user has.

OTP Cards

OTP products generate a new password each time a user needs to log in. One such product, offered by RSA Security, is the SecurID card (other companies have different names for such cards). The SecurID card is a device that flashes a new password to the user periodically (every 60 seconds or so). When the user wants to log into a computer system, he enters the number displayed on the card when prompted by the server. The server knows the algorithm that the SecurID card uses to generate passwords, and can verify the password that the user enters. There are many other variations of OTP systems as well. For instance, some OTP systems generate passwords for their users only when a personal identification number (PIN) is entered. Also, while OTP systems traditionally required users to carry additional devices, they are sometimes now integrated into personal digital assistants (PDAs) and cell phones.

Smart Cards

Another mechanism that can authenticate users based on something that they have is a *smart card*. A smart card is tamper-resistant, which means that if a bad guy tries to open the card or gain access to the information stored on it, the card will self-destruct. The card will not self-destruct in a manner similar to what comes to mind when you think of *Mission Impossible*. Rather, the microprocessor, memory, and other components that make up the "smart" part of the smart card are epoxied (or glued) together such that there is no easy way to take the card apart. The only feasible way to communicate with the microprocessor is through its electronic interface. Smart cards were designed with the idea that the information stored in the card's memory would only be accessible through the microprocessor. A smart card's microprocessor runs software that can authenticate a user while guarding any secret information stored on the card. In a typical scenario, a user enters a smart card into a smart card reader, which contains a numeric keypad. The smart card issues a "challenge" to the reader. The user is required to enter a PIN into the reader, and the reader computes a response to the challenge. If the smart card receives a correct response, the user is considered authenticated, and access to use the secret information stored on the smart card is granted.

One problem with using smart cards for authentication is that the smart card reader (into which the PIN is entered) must be trusted. A rogue smart card reader that is installed by a bad guy can record a user's PIN, and if the bad guy can then gain possession of the smart card itself, he can authenticate himself to the smart card as if he were the user. While such an attack sounds as if it requires quite a bit of control on the part of the attacker, it is very feasible. For example, an attacker could set up a kiosk that contains a rogue smart card reader in a public location, such as a shopping mall. The kiosk could encourage users to enter their smart cards and PINs by displaying an attractive message such as "Enter your smart card to receive a 50 percent discount on all products in this shopping mall!" Such types of attacks have occurred in practice. Attacks against smart cards have also been engineered by experts such as Paul Kocher, who runs a security company called Cryptography Research (`www.cryptography.com`). By studying a smart card's power consumption as it conducted various operations, Kocher was able to determine the contents stored on the card. While such attacks are possible, they require a reasonable amount of expertise on the part of the attacker. However, over time, such attacks may become easier to carry out by an average attacker.

ATM Cards

The ATM (automatic teller machine) card is another example of a security mechanism based on some secret the user has. On the back of an ATM card is a magnetic stripe that stores data—namely the user's account number. This data is used as part of the authentication process when a user wants to use the ATM. However, ATM cards, unlike smart cards, are not tamper-resistant—anyone who has a magnetic stripe reader can access the information stored on the card, without any additional information, such as a PIN. In addition, it is not very difficult to make a copy of an ATM card onto a blank magnetic stripe card. Since the magnetic stripe on an ATM card is so easy to copy, credit card companies also sometimes incorporate holograms or other hard-to-copy elements on the cards themselves. However, it's unlikely that a cashier or point-of-sale device will actually check the authenticity of the hologram or other elements of the card.

In general, the harder it is for an attacker to copy the artifact that the user has, the stronger this type of authentication is. Magnetic stripe cards are fairly easy to copy. Smart cards, however, are harder to copy because of their tamper-resistance features.

1.2.3. Something You Are

The third general method of authenticating a user is based on something that the user *is*. Most of the authentication techniques that fall into this category are biometric techniques, in which something about the user's biology is measured. When considering a biometric authentication technique as part of your system, it is important to consider its effectiveness and social acceptability.

The first biometric authentication technique that we consider is a palm scan in which a reader measures the size of a person's hand and fingers, and the curves that exist on their palm and fingers. It also incorporates fingerprint scans on each of the fingers. In this way, the palm scan technique is much more effective than simply taking a single fingerprint of the user.

A second technique used to biometrically authenticate someone is to scan their iris. In this technique, a camera takes a picture of a person's iris and stores certain features about it in the system. Studies have been conducted to measure how comfortable people are with such scans, and the iris scan appears to be more socially acceptable than the palm scan. In the palm scan technique, the user is required to actually put her hand on the reader for a few seconds, while in the iris scan, a camera just takes a quick picture of the user's iris. The iris scan is less intrusive since the user does not have to do anything except look in a particular direction.

Another biometric technique is a retinal scan, in which infrared light is shot into a user's eyes, and the pattern of retinal blood vessels is read to create a signature that is stored by a computer system. In a retinal scan, the user puts his head in front of a device, and then the device blows a puff of air and shoots a laser into the user's eye. As you can imagine, a retinal scan is more intrusive than an iris scan or a palm scan.

Another biometric authentication technique is fingerprinting. In fingerprinting, the user places her finger onto a reader that scans the set of curves that makes up her fingerprint. Fingerprinting is not as socially accepted as other biometric identification techniques since people generally associate taking fingerprints with criminal activity. In addition, fingerprinting provides less information than a palm scan.

Voice identification is a mechanism in which a computer asks a user to say a particular phrase. The computer system then takes the electrically coded signals of the user's voice, compares them to a databank of previous signals, and determines whether there is close enough of a match.

Facial recognition involves a camera taking a picture of a person's face and a computer system trying to recognize its features.

Another technique, signature dynamics, records not only a user's signature, but also the pressure and timing at which the user makes various curves and motions while writing. The advantage of signature dynamics over simple signature matching is that it is far more difficult to replicate.

The key disadvantages to these biometric authentication techniques are the number of false positives and negatives generated, their varying social acceptance, and key management issues.

A *false positive* occurs when a user is indeed an authentic user of the system, but the biometric authentication device rejects the user. A *false negative*, on the other hand, occurs when an impersonator successfully impersonates a user.

Social acceptance is another issue to take into account when considering biometric authentication techniques. All the biometric authentication techniques discussed here are less socially accepted than entering a password.

The final disadvantage for biometric authentication techniques is the key management issue. In each of these biometric authentication techniques, measurements of the user's biology are used to construct a key, a supposedly unique sequence of zeros and ones that corresponds only to a particular user. If an attacker is able to obtain a user's biological measurements, however, the attacker will be able to impersonate the user. For example, a criminal may able to "copy" a user's fingerprint by re-creating it with a wax imprint that the criminal puts on top of his finger. If you think of the user's fingerprint as a "key," then the key management issue in this case is that we cannot revoke the user's key because the user cannot get a new fingerprint—even though her original fingerprint has been stolen. By contrast, the keys in password systems are generated from passwords, and users can easily have their passwords changed if they are ever stolen or compromised. Biometric authentication becomes ineffective once attackers are able to impersonate biometric measurements.

1.2.4. Final Notes on Authentication

Combining various authentication techniques can be more effective than using a single authentication technique. For example, in the previous section, we discussed some of the disadvantages of using biometric authentication alone. However, if you combine biometric authentication with another technique, such as a password or a token, then the authentication process becomes more effective.

The term *two-factor authentication* is used to describe the case in which a user is to be authenticated based upon two methods. ATM cards are an example of two-factor authentication at work. ATM cards have magnetic stripes that have the user's name and account number. When the card is used, the user is required to enter not only the card into the teller machine, but also a PIN, which can basically be thought of as a password. In such an example of *two-factor authentication*, the bank requires the user to be authenticated based upon two methods—in this case, something that the user has and something that the user knows.

There are other factors that can be taken into account when conducting authentication. For instance, Alice's location can be considered a factor. Alice may carry around a cell phone that has a GPS (Global Positioning System) chip inside of it. When Alice is standing in front of an ATM requesting to withdraw money, Alice's bank could ask her cell phone company's computer system where she currently is. If the cell phone company's computer responds with a latitude and longitude that corresponds to the expected location of the ATM, the bank can approve the withdrawal request. However, if Alice's ATM card and PIN were stolen by a bad guy who is trying to withdraw money, then taking Alice's location (or specifically, the location of her cell phone) into account could help thwart such a fraudulent withdrawal request. If Alice's cell phone is still in her possession, when an attacker attempts to use her card at an ATM, the location of the ATM will not correspond to the location of Alice's cell phone, and the bank will deny the withdrawal request (unless, of course, Alice and her cell phone are being held captive in front of the ATM). In this example, it is advantageous for Alice to keep her cell phone and her ATM card in different places; she should not, say, keep both of them in her purse.

In all the examples discussed so far, we have talked about people authenticating people or people authenticating themselves to computers. In a large distributed system, however, computers are also interacting with other computers. The computers may have to authenticate themselves to each other because all computers cannot be trusted equally. There are many protocols that can be used to allow computer-to-computer authentication, and these protocols will, in general, support three types of authentication: client authentication, server authentication, and mutual authentication.

Client authentication involves the server verifying the client's identity, *server authentication* involves the client verifying the server's identity, and *mutual authentication* involves the client and server verifying each other's identity. When we discuss protocols, such as Secure Sockets Layer (SSL) in Chapter 15, we will discuss the different modes they use to support client, server, and mutual authentication.

Whether client, server, or mutual authentication is done often depends upon the nature of the application and the expected threats. Many e-commerce web sites provide server authentication once a user is ready to make a purchase because they do not want the client to submit a credit card number to a spoofed or impostor web site. Spoofed web sites are a significant security threat because they do not cost much to set up.

On the other hand, in older cell phone networks, only client authentication was required. Cell phone towers (servers) would only check that a phone (client) that attempted to communicate with it was owned by an authentic customer. The phones did not authenticate the cell phone towers because cell phone towers were costly to set up, and an attacker would require significant capital to spoof a cell phone tower. On the other hand, the cell phones themselves were much cheaper, and hence wireless carriers only required phones to be authenticated. Today, the cost of cell phone base stations is significantly cheaper, and modern-day cell phone networks use mutual authentication.

Now that we have completed our discussion of authentication, we are going to explore our next security concept: authorization.

1.3. Authorization

Authorization is the act of checking whether a user has permission to conduct some action. Whereas authentication is about verifying identity, authorization is about verifying a user's authority. To give a concrete example, let us examine the case in which Alice authenticates herself at an ATM by putting in her ATM card and entering her PIN. Alice may want to deduct $500, but may only be authorized to deduct a maximum of $300 per day. If Alice enters $500 as the amount that she is requesting to deduct, the system will not authorize her transaction even if she successfully authenticates herself.

In the previous example, an authorization check questions whether Alice has the authority to deduct a certain amount of money. Operating systems such as Windows and Linux do authorization checks all the time. For example, when Alice attempts to delete a file, the operating system checks whether Alice is allowed to do so. A general mechanism called an *access control list* (ACL) is used by many operating systems to determine whether users are authorized to conduct different actions.

1.3.1. Access Control Lists (ACLs)

Minimally, an ACL is a set of users and a corresponding set of resources they are allowed to access. For example, Alice may have access to all the files in her home directory,[1] but may not have access to Bob's files. Suppose Alice's home directory is `/home/Alice`, and Bob's home directory is `/home/Bob`. An ACL that models this would list Alice as the principal,[2] and it would also list the set of files in her home directory that she is allowed to access, as shown in Table 1-1. In the table, an asterisk (*) is used as a wildcard to indicate all files and subdirectories within a particular home directory. An ACL may optionally include privileges that are associated with resources. The Privilege column indicates that Alice and Bob are allowed to read, write, and execute files in their respective home directories.

Table 1-1. *A Simple ACL*

User	Resource	Privilege
Alice	`/home/Alice/*`	Read, write, execute
Bob	`/home/Bob/*`	Read, write, execute

In some more sophisticated ACL schemes, another piece of information called a *role* is added, which enables a user or principal to access particular resources. Table 1-2 shows an example mapping of users to roles, and Table 1-3 shows a role-based ACL. In Table 1-2, Alice is both a programmer and an administrator, and Bob is both a programmer and a backup operator.[3]

Table 1-2. *A User-Role Mapping*

User	Role
Alice	Administrator, Programmer
Bob	Backup Operator, Programmer

Table 1-3. *A Role-Based ACL*

Role	Resource	Privilege
Backup Operator	`/home/*`	Read
Administrator	`/*`	Read, write, execute

1. A user's *home directory* is the location on a file system where her files are stored.
2. An entity (or a process) that is capable of being authenticated is often referred to as a principal.
3. A backup operator is responsible for backing up all user files on a periodic basis.

1.3.2. Access Control Models

ACLs can be used to implement one of three access control models—the mandatory access control (MAC) model, the discretionary access control (DAC) model, and the role-based access control (RBAC) model—sometimes called the non-discretionary access model.

Mandatory Access Control (MAC)

In the MAC model, the computer system decides exactly who has access to which resources in the system. In the MAC model, if Alice creates a new document, the system can decide that no one but Alice is allowed to access that document. Alice herself does not have the right to decide who else is allowed to access the file that she authored. Even if she wants to share the document she authored with her friend Bob, she is not authorized to make that decision. For instance, if Alice creates a file `/home/Alice/product_specs.txt` in a system with a MAC model, there would be no way for Alice to decide on her own to allow Bob to see that file. In a MAC model, only the computer system determines who is authorized to access documents that Alice creates.

Discretionary Access Control (DAC)

The DAC model is different from the MAC model in that users are authorized to determine which other users can access files or other resources that they create, use, or own. In a discretionary access system, Alice could let Bob access a file at her discretion by issuing a command to the system, and then Bob would be given access to that file. For instance, in UNIX, which uses a DAC model, Alice could issue the command `chmod a+r /home/Alice/product_specs.txt` to allow all users on the system to read the file. The ACL that results from such a command is shown in Table 1-4, in which the third row specifies that every user (denoted by *) has read privileges for the file `/home/Alice/product_specs.txt`.

Table 1-4. *The Resulting ACL*

User	Resource	Privilege
Alice	`/home/Alice/*`	Read, write, execute
Bob	`/home/Bob/*`	Read, write, execute
*	`/home/Alice/product_specs.txt`	Read

Role-Based Access Control (RBAC)

The third access control model is the RBAC model, which is similar to the MAC model in the sense that the system decides exactly which users are allowed to access which resources—but the system does this in a special way. A RBAC system will incorporate the user's role into its access decision. For instance, the system may know about the user's position (or role) within a company (e.g., administrative assistant, manager, or CEO) and give the user different privileges based on that role. For instance, the CEO may be allowed to access salary information about any employee in the company, whereas a manager may only be able to access salary information about his or her subordinates.

As per the role-based ACL shown in Table 1-3, a backup operator is allowed to read data from all user home directories (`/home/*`) so that the data can be archived. However, a principal with an administrator role, such as Alice, may be able to read, write, and execute files anywhere on the file system. Users that have multiple roles would declare their role just prior to conducting an action, such as doing a backup or modifying a file. While a user such as Bob may have both read and write privileges to some files (such as those in his home directory), the purpose of the role would be to ensure that he could not inadvertently modify a file while doing a backup.

Another example might use the concept of a group in the UNIX operating system to implement RBAC. All users with a particular role would be placed in a group with the same name as their role (e.g., Alice and Bob would be members of the group `programmer`). To make the file `/home/Alice/product_specs.txt` available to all programmers, one could use the command `chgrp programmer /home/Alice/product_specs.txt`. As long as the file has group read privileges, all users within the `programmer` group will have read privileges for the file. The results of such a command are shown in Table 1-5, which contains a third row that specifies that any user with the programmer role can read the file `/home/Alice/product_specs.txt`.

Table 1-5. *The ACL Based on the RBAC Model*

Role	Resource	Privilege
Backup Operator	`/home/*`	Read
Administrator	`/*`	Read, write, execute
Programmer	`/home/Alice/product_specs.txt`	Read

■ **Note** Our illustrations of various types of access control models using UNIX have been shown for conceptual clarity only. Various implementations of UNIX may implement ACLs using different data structures than in the tables we have used.

Now that we have summarized the three different types of access control models, we will examine an access control model called the Bell-LaPadula model. The Bell-LaPadula model can be used to implement either a mandatory or discretionary access model, depending upon the particular details of the implementation.

1.3.3. The Bell-LaPadula Model

The *Bell-LaPadula* model is a popular access control model used by many government and military organizations. In this model, all resources within the system are classified with a certain level of access. The classifications are, in order of increasing privilege: unclassified, confidential, secret, and top secret, as shown in Figure 1-2. In addition to associating a classification with resources, all users are also given a classification (unclassified, confidential, secret, or top secret).

The key innovation in the Bell-LaPadula model is not the idea of adding classifications to users and resources, it is the use of various rules used to guide the decisions about who is allowed to access the resources. There are three rules that guide the decisions about which users are allowed to access which files: the simple property, the star property, and the tranquility property.

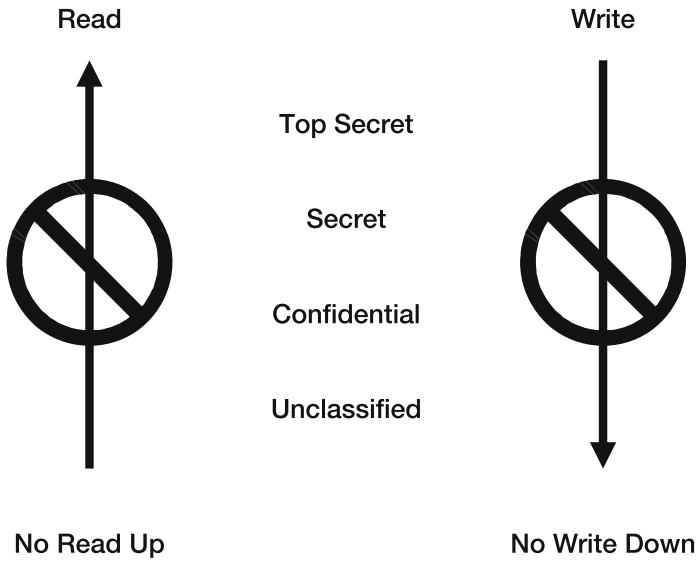

Figure 1-2. *The Bell-LaPadula model*

The first rule, the *simple property*, states that if a user has a particular level of access, then that user is not allowed to access any information resources that have a higher classification than the user does. In essence, a user that has only unclassified access will only be able to access unclassified files. A user with confidential access will be able to access confidential and unclassified files, but not secret or top secret files. The simple property is an intuitive rule that is very often called *no read up*.

The *star property*, also called the confinement property, is the second rule. If a user has secret level access, then the user is not allowed to write any files or create any resources that have a lower level of access. For example, if a user logs into a system and has secret level access, that user is not allowed to write any files that would be accessible by someone with only confidential or unclassified access. The idea behind this *no write down* strategy is that we would not want any information to leak from a higher level to a lower level. With this strategy, it would be impossible for someone with secret level access to write out any file in a system that could be read by a user that has only unclassified or confidential access. The goal of the star property is to restrict secret-level information only to the appropriate level of classification or above.

The third property of the Bell-LaPadula model is the *tranquility property*. The tranquility property states that the classification of a file cannot be changed while that file is in use by any user of the system. (The file is not considered to be tranquil while it is being edited or written.) For example, if the status of a confidential file is to be changed to unclassified, one has to wait

until all users currently using (and potentially writing to) that file to stop using it. The reason to wait for tranquility is that it may be possible that some document could get declassified while some user with confidential access is still writing confidential information into the document. The tranquility property is a synchronization constraint placed upon all the resources in the system that uses the Bell-LaPadula model.

1.4. Confidentiality

The goal of *confidentiality* is to keep the contents of a transient communication or data on temporary or persistent storage secret.

If Alice and Bob want to exchange some information that they do not want Eve to see, the challenge is to make sure that Eve is not able to understand that information, even if Eve can see the bits that are being transferred over the network.

Suppose Eve is an eavesdropper who may be able to listen in on the contents of Alice and Bob's secret conversations. If Alice and Bob are communicating over a network, then Eve is able to see the bits—the zeros and ones—that make up Alice and Bob's conversation go back and forth over the wires (or over the air, in the case Alice and Bob are using a wireless network).

A real-world Eve might employ various existing software tools to eavesdrop. On an Ethernet network that uses a hub (as opposed to a switch), for instance, each computer is capable of actually seeing all the network traffic that is generated and received by any other computer. A computer's operating system is typically responsible for only allowing applications running on that computer to access traffic that is directed to or from that computer, and filtering out traffic that originates or is destined for other computers on the same network. However, if a user has root or administrator privileges on a computer, that user can use a software package such as Ethereal, tcpdump, or dsniff to access network traffic. These software packages are run in a "promiscuous mode," in which the operating system provides the software access to *all* traffic on the network instead of providing filtered traffic that is just directed to or from the computer on which it is running. While such packages exist to help network administrators and engineers debug problems, they can be used for eavesdropping. Attackers may not have administrator privileges, but can obtain them by first getting access to *some* account, and then exploiting software vulnerabilities in the operating system to gain such privileges.

Usually, some kind of encryption technology is used to achieve confidentiality. Most encryption technologies use a key to encrypt the communication between Alice and Bob. A *key* is a secret sequence of bits that Alice and Bob know (or share) that is not known to potential attackers.[4] A key may be derived from a password that is known to both Alice and Bob. An encryption algorithm will take the key as input, in addition to the message that Alice wants to transfer to Bob, and will scramble the message in a way that is mathematically dependent on the key. The message is scrambled such that when Eve sees the scrambled communication, she will not be able to understand its contents. Bob can use the key to unscramble the message by computing the mathematical inverse of the encryption algorithm. If Alice and Bob use good encryption technology and keep the key secret, then Eve will not be able to understand their communication.

---

4. In this chapter, we use the term *key* to refer to a secret key. In some encryption schemes (covered in Chapter 13), some keys can be made public.

1.5. Message/Data Integrity

When Alice and Bob exchange messages, they do not want a third party such as Mallory to be able to modify the contents of their messages.

Mallory has capabilities similar to Eve, but Eve is a passive eavesdropper while Mallory is an active eavesdropper. Though Eve is able to see the zeros and ones go by, she is unable to modify them. Eve therefore cannot modify any part of the conversation. On the other hand, Mallory has the ability to modify, inject, or delete the zeros and ones, and thus change the contents of the conversation—a potentially more significant kind of attack. Mallory is sometimes referred to as a *man in the middle*.

Alice and Bob can use an *integrity check* to detect if an active eavesdropper like Mallory has modified the messages in an attempt to corrupt or disrupt their conversation. That is, Alice and Bob want to protect the *message integrity* of their conversation. One approach that they can take to ensure message integrity is to add redundancy to their messages.

Consider a hypothetical scenario in which Alice wants to send an "I owe you" (IOU) message such as "`I, Alice, owe you, Bob, $1.00`," and Mallory has the ability to change only one character in the message. If Mallory wants Alice to be in more debt to Bob, she could change the message to "`I, Alice, owe you, Bob, $1000`" by changing the dot to a zero. On the other hand, if Mallory wants to cheat Bob out of his dollar, she could change the message to "`I, Alice, owe you, Bob, $0.00`." Assuming Mallory can only change a single character in a message, Alice could add redundancy to her message by repeating the dollar amount twice so that Bob could detect tampering. For example, if Alice sends the message "`I, Alice, owe you, Bob, $1.00. Confirm, $1.00`," then Mallory would not be able to change both of the dollar values in the message, and Bob would be able to detect tampering by Mallory. If Mallory changes one of the amounts in the message, Bob will see a mismatch between the two dollar amounts and discard the message. In this manner, redundancy can be used to provide message integrity.

While Mallory may not be able to tamper with Alice's IOU if she uses redundancy, she may still be able to conduct a denial-of-service attack. If Mallory changes one of the dollar amounts in the IOU each time Alice tries to send it to Bob, and Bob is forced to discard the message each time because of the mismatched dollar amounts, Bob will never receive the IOU he rightly deserves! (Denial-of-service attacks are discussed further in Section 1.7.)

Unfortunately, a real-world active eavesdropper will typically have the power to change much more than a single character in a message, and the simple approach of repeating the dollar amount will not work. In addition, repeating information more than once requires extra communications bandwidth and is not terribly efficient.

In networking communications protocols, approaches such as CRCs (cyclic redundancy checks) can be used to achieve integrity and detect when bits in a message have been lost or altered due to inadvertent communications failures. These techniques compute short codes that are functions of the message being sent. Alice can attach a short code to the message such that if the message or code are modified, Bob can determine whether they were tampered with.

However, while CRCs are sufficient to detect inadvertent communications failures, they are typically not good enough to deal with adversaries such as Mallory. If Mallory knows that a CRC is being used, and she has no restrictions on how many bytes she can modify, she can also change the short code to match her modified message.

Instead, message authentication codes (MACs) are typically used to achieve message integrity in real-world security protocols. A MAC is not only a function of the message itself, but is also a function of a key known only to Alice and Bob, such that even if Mallory is able to modify the bytes of a message, she will not be able to appropriately modify the corresponding MAC. (MACs are covered in more detail in Chapter 15.)

While the goal in confidentiality is to make sure that the contents of Alice and Bob's communication cannot be understood by a third party like Eve or Mallory, there is no such requirement for message integrity. For message integrity to be achieved, it does not matter whether the eavesdropper can see the data in the message so long as she is unable to change it undetected. The goal of message integrity is to make sure that even if Mallory can "look," she cannot "touch" the contents of the message.

1.6. Accountability

While authentication and authorization are important, *accountability* is another key security goal (especially for a company's internal systems). The goal of accountability is to ensure that you are able to determine who the attacker or principal is in the case that something goes wrong or an erroneous transaction is identified. In the case of a malicious incident, you want to be able to prosecute and prove that the attacker conducted illegitimate actions. In the case of an erroneous transaction, you want to identify which principal made the mistake. Most computer systems achieve accountability through authentication and the use of logging and audit trails. To obtain accountability, you can have a system write log entries every time a user authenticates, and use the log to keep a list of all the actions that the user conducted.

The chief financial officer (CFO) of a company may have the authority to transfer money from the company's bank account to any another, but you want to hold the CFO accountable for any actions that could be carried out under her authority. The CFO should have the ability to transfer money from the company account to other accounts because the company may have certain financial commitments to creditors, vendors, or investors, and part of the CFO's job may involve satisfying those commitments. Yet, the CFO could abuse that capability. Suppose the CFO, after logging into the system, decides to transfer some money from the company's bank account to her own personal account, and then leave the country. When the missing funds are discovered, the system log can help you ascertain whether or not it was the CFO who abused her privileges. Such a system log could even potentially be used as evidence in a court of law.

It is also crucial to make sure that when the logging is done and audit trails are kept, the logs cannot be deleted or modified after the fact. For example, you would not want the CFO to be able to transfer money into her own personal account and then delete or change the audit trail so that transaction no longer appears, or is covered up in any way to appear as if the transaction had a different recipient. To prevent logs from being deleted or altered, they could immediately be transferred to another system that hopefully an attacker would not be able to access as easily. Also, Chapter 15 discusses how MACs (message authentication codes) can be used to construct integrity check tokens that can either be added to each entry of a log or associated with an entire log file to allow you to detect any potential modifications to the system log. You can also use *write once, read many* (WORM) media to store system logs, since once written, these logs may be hard (or even physically impossible) to modify—short of destroying the media completely.

A good logging or audit trail facility also provides for accurate timestamping. When actions are written to an entry in a log, the part of the entry that contains the time and date at which the action occurred is called a *timestamp*. You need to ensure that no user can modify timestamps recorded in the log. The operating system, together with all the other computers on the network, must be in agreement on the current time. Otherwise, an attacker can log into a computer whose clock is ahead or behind the real time to cause confusion about when certain actions actually occurred. A protocol such as Network Time Protocol (NTP) can be used to keep the clocks of multiple computers synchronized.

One problem with many of today's systems is that logging facilities do not have secure timestamping and integrity checking facilities. As a result, after attackers hack into a system, they can change the logs such that no one can detect that they hacked in. Therefore, it is especially important to think carefully about a secure audit trail facility when you design secure systems. If existing or third-party software tools are used when constructing systems, they may have to be instrumented or modified to satisfy accountability goals.

1.7. Availability

An *available* system is one that can respond to its users' requests in a reasonable timeframe. While availability is typically thought of as a performance goal, it can also be thought of as a security goal. If an attacker is able to make a system unavailable, a company may lose its ability to earn revenue. For example, if an online bookstore's web site is attacked, and legitimate customers are unable to make purchases, the company will lose revenue. An attacker that is interested in reducing the availability of a system typically launches a denial-of-service (DoS) attack. If the online bookstore web site were run on a single web server, and an attacker transmitted data to the web server to cause it to crash, it would result in a DoS attack in which legitimate customers would be unable to make purchases until the web server was started again. Most web sites are not run using just a single web server, but even multiple web servers running a web site can be vulnerable to an attack against availability.

In a *distributed denial-of-service* (DDoS) attack, perpetrators commandeer weakly protected personal computers and install malicious software (malware) on them that sends excessive amounts of network traffic to the victim web sites.[5] The servers running the victim web sites are then overwhelmed with the large number of packets arriving from the commandeered computers, and are unable to respond to legitimate users.

In February 2000, the eBay, E*TRADE, Amazon, CNN, and Yahoo web sites were victims of DDoS attacks, and some were disabled for almost an entire business day. This meant lost revenues and interruption of service for legitimate users. One study by the Yankee Group estimated the damage due to lost capitalization, lost revenues, and cost of security upgrades to be $1.2 billion (Kovar 2000); this cost figure was also cited in a FBI congressional statement on cybercrime (Gonzalez 2000).

We include availability as a security goal because it is sometimes difficult to provide a system that is both highly secure and available all the time. There is sometimes an interesting trade-off between availability and security. For example, if a computer is disconnected from

5. Such attacks are called network-layer denial-of-service attacks. Application-layer denial-of-service attacks are also possible, in which vulnerabilities in applications are exploited to make systems unavailable.

the Internet and stored in a physically secure location where no one is allowed to access it, the computer will be very secure. The problem is that such a computer is not readily available to anyone for use.

You want to design systems whose functionality is available to the largest possible intended audience while being as secure as possible. A service like PayPal (`www.paypal.com`), which supports person-to-person payments, is an example of a system that generates more revenue the more users take advantage of it, and as such, its availability is critical—users may get very upset if they cannot access their funds at a moment's notice.

How does one achieve availability in a system? One method is to add redundancy to eliminate any single point of failure. For example, consider a telephone network. In such a network, phones connect to a switch (central office) that directs calls. If someone wants to attack your ability to place phone calls, he might cut the telephone line that connects to that particular central office, and as a result you would not be able to make calls. Attackers sometimes cut off a victim's ability to communicate prior to launching an attack.

One potential way to avoid single points of failure is to add redundancy. (Note that we are referring to a different type of redundancy than the redundancy we referred to in our discussion of message integrity.) A second switch can be added to the network so that if an attacker disables the first switch, the system will automatically connect you to the second.

Another potential DoS attack can be conducted by filling up a system's disk. Suppose users are sharing a disk on a server that is used to store their photos. That server may be running critical processes that need some disk space themselves. If an attacker can sign up as a user (or compromise an existing account) and fill up the shared disk with his own photos (or garbage data), then the critical processes may not be able to properly function, and system failure may ensue.

If you impose limits on the amount of disk space that each user can use, then even if the attacker is able to compromise one user's account, he will only be able to use up a certain amount of disk space. The attacker would need to compromise additional accounts to use up more disk space. In such a system, even if a user is a legitimate, paying customer, that user should not be trusted with more than her fair share of disk space because her account could be compromised.

Now that we have covered availability, let us move on to the last key security goal we consider in this chapter: non-repudiation.

1.8. Non-repudiation

The goal of *non-repudiation* is to ensure undeniability of a transaction by any of the parties involved. A trusted third party, such as Trent, can be used to accomplish this.

For example, let us say Alice interacted with Bob at some point, and she does not want Bob to deny that she interacted with him. Alice wants to prove to some trusted third party (i.e., Trent) that she did communicate with Bob. If, for instance, Alice sent a payment for a bill to Bob over the Web, she may want her payment to be non-repudiable. That is, she does not want Bob to be able to deny that he received the payment at some later point for any reason.

Alice, for example, may feel comfortable sending money to Trent, but not directly to Bob. Bob also trusts Trent. Trent may say to Bob, "Yes, Alice gave me the $500, so you can ship her the goods, and then I will pay you." In such an example, Trent is playing the role of an escrow agent, but trusted third parties may be able to serve in many other types of trusted roles

beyond being escrow agents. Because Alice and Bob trust Trent, they may be able to conduct certain types of transactions that they could not have accomplished otherwise.

To illustrate another example in which Alice and Bob use the help of Trent, consider that Alice might want to sign a contract to be employed by Bob. Alice might want Trent to serve as a judge so that if Bob ever tries to pay her less than the salary specified by the contract, she can call on Trent to help enforce the contract. At the same time, Bob might not want Alice to show the employment contract to another potential employer to try to get a higher offer.

Alice and Bob can accomplish both of their goals by using Trent's help. Bob can give Trent the employment contract. Trent tells Alice the amount of the offer, and agrees not to show the employment contract to other employers. Then, Alice can decide whether to accept the contract, but will not be able to use it to negotiate higher offers with other employers. Also, if Bob ever tries to cheat Alice by not issuing payment, Trent can intervene. Note that we assume that Trent is trusted to be impartial and will not collude with either Alice or Bob. To summarize, trusted third parties can help conduct non-repudiable transactions.

In general, non-repudiation protocols in the world of security are used to ensure that two parties cannot deny that they interacted with each other. In most non-repudiation protocols, as Alice and Bob interact, various sets of evidence, such as receipts, are generated. The receipts can be digitally signed statements that can be shown to Trent to prove that a transaction took place.

Unfortunately, while non-repudiation protocols sound desirable in theory, they end up being very expensive to implement, and are not used often in practice.

1.9. Concepts at Work

Now that we have covered a number of key security concepts, let us examine how those concepts work together in a typical web client/web server interaction. Suppose Alice is an employee of a company called PCs-R-Us, and her job responsibility is to order DVD drives for the company's PCs from a company called DVD-Factory. DVD-Factory has a web site that Alice uses to procure DVDs for her company. The following points examine why DVD-Factory might want to care about the security goals discussed in this chapter when implementing its web site.

- *Authentication*: If a malicious competitor is trying to steal business from DVD-Factory, the competitor could create a web site that looks exactly like the DVD-Factory web site, but at a different web address. To combat that tactic, DVD-Factory needs to make sure that the web server can be authenticated so that when Alice goes to the DVD-Factory web site, she knows she is dealing with DVD-Factory and not DVD-Factory's look-alike competitor.

  The SSL protocol is used between web clients and web servers to do secure transactions. When Alice enters the web address `https://www.dvd-factory.biz`, Alice's browser will invoke the SSL protocol to make sure that the web site authenticates itself to her browser. (We will talk more about how the SSL protocol accomplishes this later in the book, but at this point, it is important to note that the web browser authenticates the web site to make sure that it is dealing with DVD-Factory's web site and not another web site that is trying to spoof or imitate it.)

Alice then has to log into DVD-Factory and give that web site her username and password so that DVD-Factory knows that it is Alice, an authenticated principal at PCs-R-Us, who is attempting to buy DVDs from them.

- *Authorization*: While Alice is the trusted PCs-R-Us employee to order DVDs, Bob, another employee at PCs-R-Us, might be responsible for accounting and auditing. He might also have a login and password to the DVD-Factory web site because he needs to see prices and orders placed, but he may not be allowed to place orders for DVDs himself. Before accepting an order, the DVD-Factory web site conducts an authorization check to make sure that the logged-in user is allowed to place an order. If Alice tries to order DVDs from PCs-R-Us, the web site will allow it, but if Bob attempts to order DVDs, the order will be rejected.

- *Confidentiality*: DVD-Factory doesn't want competitors to be able to see exactly how many or which DVDs Alice happens to be ordering from DVD-Factory, because that may give them competitive information. The SSL protocol encrypts all of the communication between Alice and the DVD-Factory web site with an algorithm such as Triple DES. (We cover SSL in more detail in Chapter 15, and we cover Triple DES and other encryption algorithms in Chapters 12 and 13.)

- *Message Integrity*: Suppose that Alice wants to order ten DVDs from DVD-Factory, but an attacker wants to alter her order to zero DVDs. If the attacker succeeds and DVD-Factory gets a message saying that Alice has ordered zero DVDs, her job may be affected, since no DVDs are actually going to be shipped. Alice may eventually get frustrated with DVD-Factory and might decide to go to a competitor (who may be behind this mischief). Message and data integrity are very important to prevent such mischief. The SSL protocol uses message authentication codes in the messages that are sent between Alice and the web site to make sure that no competitor or other malicious party can tamper with the data.

- *Availability*: DVD-Factory may have a competitor that launches a DoS attack against the site in order that Alice will stop buying from DVD-Factory and instead come to their competing site. As part of DVD-Factory's security strategy, its web site needs to be kept running and available 24 hours a day, 7 days a week. One simple (but potentially expensive) approach that DVD-Factory might use to mitigate a DoS attack against it would be to overprovision their bandwidth to handle the increased traffic load caused by illegitimate clients. You can read more about overprovisioning and other approaches to mitigating DoS attacks in *Internet Denial of Service Attack and Defense Mechanisms*, by Jelena Mirkovic et al.

- *Accountability*: To ensure accountability, every time Alice places an order from the DVD-Factory web site, it produces a log entry so that Alice cannot later claim to have not ordered the DVDs. This may sound a bit like non-repudiation, but it is actually accountability, since the goal is to simply keep a log of what Alice has and has not done.

- *Non-repudiation*: It is possible for DVD-Factory to cheat and report that Alice ordered more DVDs than she actually did. If the web browser and web site run a non-repudiation protocol, it is then possible for Alice to prove to a third party that she only ordered, say, 10 DVDs, and not the 12 that DVD-Factory may claim she ordered.

Unfortunately, true non-repudiation is not provided by SSL, and is not implemented on most web sites, partially due to the absence of a practical protocol for non-repudiation, and partially because there are no organizations to serve as the trusted third parties.

In practice, when customers pay for services with credit cards on web sites, Visa and Mastercard take on the role of trusted third parties, but they usually end up trusting their users much more than the merchant web sites from which their users buy products. If a user claims that he or she did not place an order with a merchant, then the credit card company favors the user and issues a *chargeback*. In the physical world, merchants can fight the chargeback if they can produce a receipt that the user signed. Of course, in the context of a web transaction, there is no good proxy or replacement for a receipt signed by the user!

While we unfortunately do not see true non-repudiation on the Web today, it is possible that the Web of the future will provide better non-repudiation capabilities.