Chapter 11

Security and Protection

11.1 Introduction

Computer security is based on the answers to two questions:

- Who do you trust?
- How much do you trust them?

The specific answers to these questions are the basis for a *security policy*. A typical policy will define *roles* and *access privileges* for company employees, contractors, vendors, customers, and others. All of the security mechanisms that we will discuss in this chapter are intended to enforce that security policy.

We often think of security as protecting a system from outsiders, but it must also protect it from insiders who are attempting (either accidentally or deliberately) to do things not allowed by the security policy.

Like the objects of an executing object-oriented program, resources are accessed by processes through operations defined by a capabilities list controlled by the appropriate operating system manager. *Protection* refers to the set of policies and mechanisms that define how resources and the operations defined on them are protected from processes. In order to avoid haphazard and indiscriminant use of the operations of a resource object, the memory manager maintains an access matrix that specifies which operations on each object are accessible to the various domains that can be assigned to processes.

External security refers to the way user programs access networks, servers, other hosts, and their resources and services. In contrast to protection, where a process hopes to gain straightforward access to an operation defined on a specific resource, security is about understanding how processes, by accident or by design, attempt to carry out some action to an operating system resource that could damage it or alter it in an unacceptable way.

Another important issue in security is *access*—the ability to access the resources of another computer across the Internet by using a protocol (SMTP, HTTP, FTP, TCP, UDP, etc.). Any Internet user has access to public domain resources (usually files and lightweight applications, most of which are on the Web) while more extended access is available across the Internet, on local area networks, and directly for those who have user accounts. Requests for user account access to networks and servers are controlled by login names and passwords and (for remote access) by IP addresses and port numbers. Users and processes are controlled in this way because untrusted users (which the system attempts to filter out by a firewall) are likely to cause deliberate damage by inserting *viruses*, *worms*, or *spyware* into the system. Such users may deliberately or accidentally exploit an application or operating system vulnerability (such as the potential of a text reader to overflow its buffer if too much data is sent or the capability of circumventing the login procedure with a trap door) that has not been patched.

11.2 Problems of Security

Security must defend a system and its computers against mistakes and malicious attacks by users and by automated or intelligent programs. A system vulnerability, such as the susceptibility of a buffer or a stack to data overflow into a protected memory space, may be exposed by a mistake (usually a programming error) or exploited on purpose. A malicious attack usually involves the deliberate installation of software on a computer, which can compromise either privacy or security. For instance, such an attack could install *spyware*, which could extract enough information from files to lead to *identity theft*, *credit card fraud*, or other financial loss. Such an attack could install a virus that alters the system (attributes or functions) in some way (usually the operating system or a popular application). Such an attack could install a worm that uses the system as a springboard to propagate itself to other machines and to launch attacks on other machines that it may not be able to access directly. The worm may reduce the performance of the system through a *denial of service attack*, a disruption that floods it with useless Internet packets.

There are thus several points of distinction between the problem of protection and the problem of security. Protection is hardware and software oriented. Security is primarily software oriented. Protection assumes that the user and the processes are legitimate for the system and have been assigned to the appropriate domain, while security does not. Protection focuses on protecting resources from unauthorized use, while security focuses on reducing, mitigating, or recovering from the damage caused by programming mistakes and the harmful side effects of malicious programming.

In the case of protection, the owner of a system creates processes that want to access system resources (both hardware and software) in valid ways. It is a question of assigning the process to the right domain. Thus, protection is a bookkeeping device administered by the operating system to make sure that the applications executed by authorized users do not attempt to access resources beyond the scope of the domains on which they are defined. In the case of security, users and automated programs that may

or may not have authorization gain access to or somehow debilitate system resources and data (deliberately or by accident). Having gained access, or even by attempting to gain access, such programs are able to damage or change the system or reduce its capability of responding to legitimate requests.

11.3 Security and Protection Components

The security of a system depends on the following components:

- Physical security
- User authentication
- Protection
- Secure communications
- People

As we shall discuss throughout the rest of this chapter, it is often desirable to use several of these security components so that if one component is breached, the others will still protect the system.

11.3.1 Physical Security

A computer must be protected against being lost or stolen; otherwise, it is highly vulnerable. Additionally, when a computer or disk drive is discarded, the disk drive should be erased to ensure that others do not have access to important information. As discussed in Chapter 8, the contents of deleted files remain on the disk. Consequently, this erasure procedure should physically write to the disk, not just delete files.

If a portable computer contains highly sensitive information (e.g., credit card numbers), then it is advisable that those files and/or the entire disk be encrypted. This will reduce the possibility of this information being maliciously accessed if the computer is lost or stolen.

It is common for modern computers to have a wireless communications capability such as WiFi or Bluetooth. These wireless capabilities add useful features to the computer, but they also open up holes in the physical security of the system. Thus, it is important that other security mechanisms such as user authentication and encryption be used with all wireless communications.

11.3.2 User Authentication

User authentication is the action of the system verifying (to the best of its knowledge) that users are who they say they are and that those users are permitted access to the

system. User authentication can be in the form of *Something You Know*. This usually consists of a login name, a password, and a (network) domain name. The security of such basic login schemes can be enhanced by requiring periodic password updates, requiring a minimum password length and/or requiring that a minimum number of different types of characters (e.g., alpha, numeric) be used in a password so as to render it harder to guess. Usually, after the nth failed login attempt, further attempts by that user are disabled.

User authentication can also be achieved via *Something You Have*. Examples of this are a special ID card that is swiped or otherwise read by the computer. This technique is often used on manufacturing lines or by servers at a restaurant. This authentication technique is vulnerable to the card being lost or stolen. If it is a simple magnetic stripe card, then it can also be forged.

Newer systems will combine Something You Have with Something You Know. An example of this is using a *SmartCard*—a card that the user plugs into a port on the computer—for login purposes. The user will then be asked for a pin code (Something You Know), which will then be encrypted by the SmartCard. The encrypted value is sent to the network host, which will then verify it.

There are several forms of biometrics that are now starting to be used for user authentication:

- *Fingerprint*: Laptop computers are available that verify the user's fingerprint for logon.
- *Voice print*: User's voice is checked and verified.
- *Retina scan*: While very accurate, this is cumbersome to do and is currently restricted to very limited applications.
- *Facial imaging*: Currently, this technique is not accurate enough to uniquely identify an individual.

11.3.3 Protection

A resource is an object controlled by the operating system. It may be a software object such as a file or an application, or it may be a hardware object such as main memory, the CPU, or a peripheral device. An operation is a function defined on that resource. In the case of a file, for instance, operations that a process may execute on a file are create, open, copy, read, write, execute, close, and so forth.

A *domain* specifies the resources that a particular process may access. Protection on the system consists of giving resource access to domains that should have it and denying it to domains that should not have it. Consider the following example:

- A company has a Human Resources database and an Order database (the Resources).
- The HR department should have read and write access to the HR database.
- The Payroll department should have read access to the HR database.

- The Order Entry and Manufacturing departments should have read and write access to the Order database.

- All others typically would have no access to these databases.

A domain can be modeled as a set of ordered pairs (R, O), which specify access rights. The first component refers to the ith resource denoted by the subscript i and the second component refers to a set of k operations defined on resource i. The domain is denoted as $(R_i, \{O_1, O_2, \ldots\})$.

For example, let F be the generic symbol for a file. Then the access right-ordered pair for a file and its operations is $(F, \{create, open, read, write, execute, close\})$ operations defined on that resource.

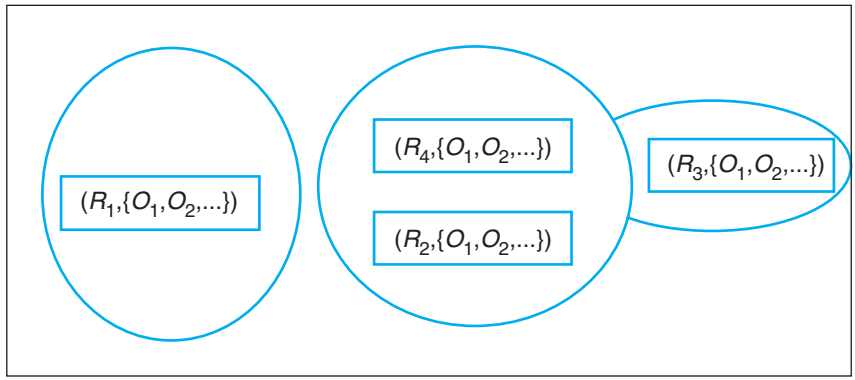

Figure 11.1 Three disjoint sets of access rights.

In Figure 11.1, the domains are denoted by ovals. The left and right domains consist of a single ordered pair, while the center domain consists of two ordered pairs. If one (or more) ordered pairs appear in the intersection of two domains, then the access right pair(s) are shared by the domains. The problem with the representation in Figure 11.1 is that it is not amenable to easy implementation.

11.3.3.1 Domains on Unix/Linux

The owner of a Unix file specifies three domains when creating a file—owner, group, and world—and three operations—read, write, and execute. The access rights to the file for each domain are specified by a group of three consecutive binary digits and separated from other domains by a hyphen (−) and are specified by the following code: $b_1b_2b_3 - b_4b_5b_6 - b_7b_8b_9$.

The first binary digit in each group of three specifies the read access; the second specifies the write access; the third specifies the execute access. If the binary digit is 1, access is permitted; otherwise it is denied. Only the owner of the file or a system administrator with access to the directory containing the file can change the access rights to the file.

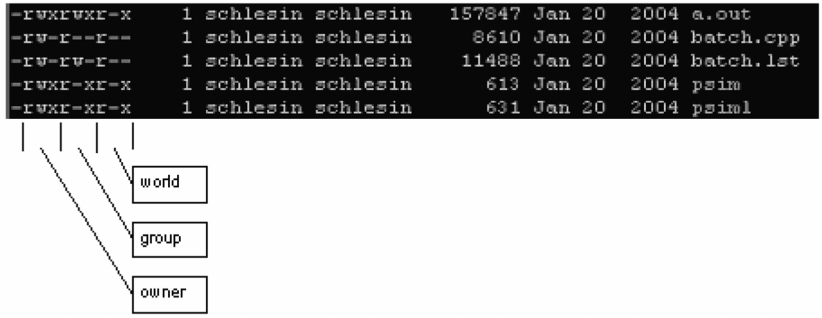

Figure 11.2 Example of permission settings on Linux.

In Figure 11.2, the owner has read and write permissions, but not execute permission, for the file `batch.cpp`. Since this file is a text file, it makes no sense to have execute permission. The group has read and execute permissions for psiml. The group does not have write permission, so no members of the group (except the owner) can change the file.

11.3.3.2 Domains on Microsoft Windows

MS Windows uses a more general domain model than Unix. On Windows, you can specify a set of individual users and a set of groups. Each of those users and groups can have different access rights for a resource. In Figure 11.3, the group named Users can

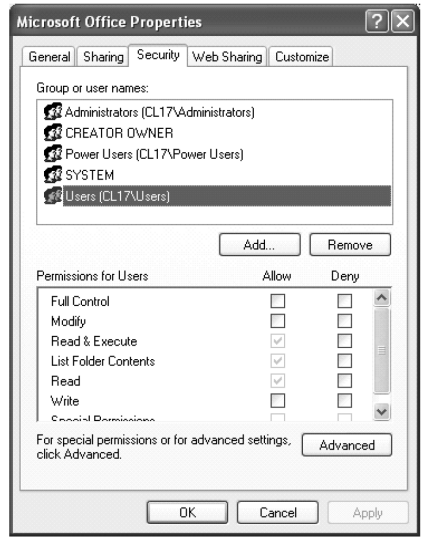

Figure 11.3 Example of permission settings on Windows.

list the contents of the folder Microsoft Office and read and execute it. Other groups may have a different set of permissions.

11.3.3.3 The Access Matrix

The domains in a system are typically modeled via an *access matrix* denoted by A. The rows of A consist of domains, the columns of A consist of objects, and the allowed operations are inserted into the cells of A. Table 11.1 shows an example of resource protection. Table 11.2 shows the access matrix for the company example.

Sometimes a domain itself can be listed as a resource (see Table 11.3). As a complex program evolves, it may want to switch domains to complete further tasks. A good example of this is when an installer program asks to run as the system administrator.

More flexibility/change can be introduced into the access matrix by introducing the operations of copy, owner, and control to the cells.

The copy right allows a process or the operating system to copy an operation for a resource defined on one domain to another domain for that same resource (same column). The owner right allows a process defined on one domain to add (or remove) an operation to (from) another domain for a particular resource by changing any cell

Table 11.1 An Access Matrix for Resource Protection

Domains/Resources	R_1	R_2	R_3	R_4
D_1	Stack alloc			
D_2		o, r, w, c	p	
D_3				ss, st, sth

Table 11.2 Access Matrix for a Company Example

Domains/Resources	HR Database	Order Database
Human resources	read, write	
Payroll	read	
Order entry		read, write
Manufacturing		read, write
Others		

Table 11.3 An Extended Access Matrix for Processes That Are Able to Switch Domains

Domains/Resources	R_1	R_2	R_3	R_4	D_1	D_2	D_3
D_1	ss, st					sw	
D_2		o, r, w, c	p		sw		sw
D_3				o, sth, cn, c	sw		

Table 11.4 An Access Matrix Where Domain Has Owner Rights and Copy(*) Rights

Domains/Resources	R_1	R_2	R_3	R_4
D_1	Stack alloc			
D_2		o, r*, w*, c	p*	
D_3		r, w		ss, st, sth

in the same (resource) column. Table 11.4 shows an access matrix in which certain domains have owner and/or copy rights.

Now the process defined on D_2 will remove read and write access for the file R_2 for D_3 and exercise the copy right to add print access to the file R_3 to D_3. Table 11.5 shows the access matrix after this change.

Table 11.5 Modified Access Matrix with Domain Access Rights

Domains/Resources	R_1	R_2	R_3	R_4
D_1	Stack alloc			
D_2		o, r*, w*, c	p*	
D_3			p	ss, st, sth

11.3.3.4 Memory Protection

Memory protection is a special case of protection in which the operating system will set up the appropriate access tables, but the enforcement is handled by the hardware. Memory protection is used to help ensure that a program bug does not have a dangerous effect. Typically, permission bits for read, write, and execute will be associated with each entry in the page table. The operating system will set these bits based on information that the compiler places in the program file for the program being executed. Table 11.6 shows the meaning and usage of the various combinations of permission bits.

Table 11.6 Memory Protection

Read	Write	Execute	Meaning	Usage
0	0	1	Execute-only	Instructions
0	1	0	Write-only	Not used
0	1	1	Write-execute	Not used
1	0	0	Read-only	Constants
1	0	1	Read-execute	Instructions, constants
1	1	0	Read-write	Data
1	1	1	Read-write-exec	No protection

Although the concept of having separate read, write, and execute permissions was developed about 40 years ago, not all CPUs support this concept. Some have no memory protection support and others only support read and write permissions. In fact, the CPUs used in PCs did not support a separate execute permission until 2006. The lack of execute permission provides an opening for hackers to penetrate a system.

Without a separate execute permission, any data area of a program is effectively executable. Thus, if a hacker can trick a program into executing something in its data, the program can be led to execute instructions provided by the hacker (in what the program believes is data). This is the basis for the Buffer Overflow and Stack Overflow security attacks.

Note that since the OS sets the permission bits based on the information provided by the compiler, it is necessary for the application developer to take advantage of this security feature in order for it to be fully enabled.

11.3.3.5 Capabilities and Higher Level Protection

Analyzing the rows across the access matrix provides a list of objects and the operations that a process defined on the domain can access on these objects. A list of all objects accessible to a domain (together with the operations allowed on these objects) is called

a *capability list*. The address of an object is referred to as its *capability*. Possession of the capability implies access to all operations defined on the object.

The capability list of a domain is not accessible to the process executing on that domain. The capability list is itself a protected object maintained by the operating system. Capabilities are distinguished from other kinds of objects by a tag or by an extended address, part of which is accessible to the program process and the other part, containing the capability list, is accessible to the operating system only.

Normally protection is implemented at the hardware/software interface by the operating system kernel. Some protection can be implemented in hardware to reduce the overhead of having the kernel inspect and validate every attempt to access a resource. However, protection can also be implemented at the compiler level or even at the (programming) language (source code) level. Such a scheme can increase the flexibility of a protection policy. One way to extend flexibility is to allow applications programmers to define functions that perform operations on resources suitable to that particular application.

Control of the access to a shared resource can be made by extending the data type syntax of a programming language. For instance, the access to a file created by a program can also be controlled by that program with this type of facility (e.g., by opening a file for reading or writing for processes defined on specified domains). Each application can define its own access control matrix for the resources that it requires, rather than allowing the kernel to control the overall access matrix for all applications.

The advantage of such a system is that protection needs are declared at the beginning of a program instead of programmed as a sequence of calls on the operating system. Such an arrangement appears to be independent of the operating system. But in fact, the protection enforcement mechanism is usually left up to the kernel of the operating system.

A disadvantage of program- (or compiler-) based protection is that a priori assumptions are made by the applications programmer (compiler) about the objects of the system and the operations that can be performed on them. Compiler-based protection ignores the problem of enforcement. By leaving enforcement up to some other entity, it is not completely solving the protection problem. Ultimately, platform-independent access control is more of a goal than an accomplished fact. The application program must collaborate with hardware or the operating system kernel to correctly interpret, and then enforce and implement, the protection schemes it defines.

If the compiler or interpreter happens to provide software for enforcing protection, then it is easier to change it than to change hardware or kernel code. The tradeoff for the enhanced flexibility of higher-level software implementations of protection is the reduced efficiency relative to lower-level software and hardware implementations.

Recent versions of the Java programming language implement language-based protection. The Java Virtual Machine (JVM) loads a class in response to a program's request to create instances (objects) of that class. Java implements protection at the class level rather than at the process level. Pairs of classes can be labeled as mutually distrusting. Individual classes are labeled as trusted or untrusted. The JVM assigns a protection domain to untrusted classes and mediates protection between mutually distrusting classes.

Depending on the protection domain assigned to the class by the JVM, the method requiring a resource (object) may assert the privilege of being able to access a particular operation (method) of that resource. Every thread in the JVM maintains a stack of calls to methods in the process of execution. For each class, the stack frame is annotated with the resource access privileges allowed for each class. When an object of that class requests access to a protected resource, the stack frame annotation is checked before permission is granted; the technical term for this procedure is *stack inspection*. If access to a resource is denied to an object based on the protection domain of the method's class, then Java throws an exception.

The reason this scheme works is that, unlike programs in other programming languages, Java programs cannot directly access memory. In particular, untrusted programs are not able to manipulate their own stacks and thus change the resource access privileges assigned by the JVM.

11.3.4 Secure Communications

A system must ensure that its communications with other systems are secure. In today's world, any communications over the Internet are vulnerable to eavesdropping (see Figure 11.4). Similarly, any communications over a Wi-Fi wireless link can be intercepted. Thus, encryption should be used for any data transmitted where the physical security of the communication lines cannot be guaranteed.

Encryption is the process of scrambling data so that only an authorized receiver can unscramble it. There are two basic operations:

- `Ciphertext = Encrypt(plaintext, key1)`
- `Plaintext = Decrypt(ciphertext, key2)`

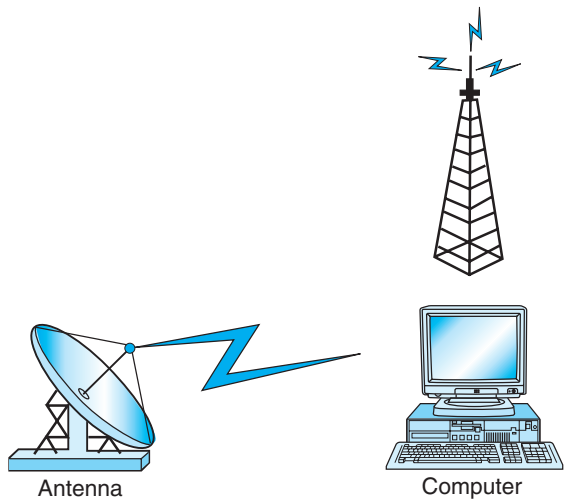

Figure 11.4 Eavesdropping.

A *symmetric encryption system* is one in which $key1 = key2$. The sender and receiver use the same key to encrypt and decrypt the data. This type of system requires that the key be kept secret. Consequently, it also requires a separate secure communications channel for the sender and receiver to communicate the key. Examples of this type of system are Data Encryption Standard (DES) and the Advanced Encryption Standard (AES).

DES was developed in the 1970s and became the standard for most encryption, especially in the financial industry. It uses a 56 bit key, which was effectively unbreakable in the 1970s; however, advances in computer technology have made it vulnerable. Thus it is now being replaced by AES, which gives the user the option of the size of key to use (128, 192, or 256 bits). This flexibility should make this system safe for the foreseeable future.

In an *asymmetric encryption system*, $key1 \neq key2$. Thus, one key is used to encrypt the data, and another is used to decrypt it. With this type of system, one of the keys (called the Public Key) is published (to the world). The other (the Private Key) is kept secret. For example, anybody can encrypt a message with Bill's public key and be confident that only Bill can decrypt it (since only he knows his private key). On the other hand, if Bill encrypts a message with his private key, anybody can decrypt it using Bill's public key. Thus, Bill can *sign* a message and anyone can verify that signature. This is called a *digital signature*, an important feature of the modern online world. When a new software component or update to an existing component is downloaded, digital signatures allow you to verify that a software component that is to be downloaded over the Internet was indeed created by whomever claims to have created it.

In modern systems, both symmetric and asymmetric systems are commonly used. Asymmetric systems provide flexibility. Symmetric systems are much faster than asymmetric systems, so a common arrangement is to use an asymmetric system to set up an encrypted communication link and a symmetric system to exchange large amounts of data.

11.3.5 Digital Certificates

A digital certificate is a mechanism that allows users to verify the authenticity of a document. When the concept was first invented, it was thought that this would be used to vouch for the authenticity of emails, legal documents, etc. While digital certificates are used for these purposes, their most common function has become verifying the authenticity of executable files downloaded from the Web.

A digital certificate system should have the following characteristics:

- It is *verifiable*. Anybody that receives the document should be able to perform a calculation to verify the accuracy of the certificate.
- It is *not forgeable*. Only the person, company, or computer system that purports to have created the certificate can actually create it.

As discussed earlier, you can create a digital certificate by encrypting a file with your private key. In practice, this leads to a very slow authentication process. To improve

performance, the file to be authenticated is *hashed* to produce a relatively short *message digest*. You can then use your private key to encrypt just the hash value.

Depending on the particular hash function that is used, this message digest will be anywhere from 128 to 256 bits. A hash function $h = H(M)$ is designed to meet the following criteria:

- It can be applied to any size of message M.
- It produces a fixed-length output h.
- It is easy to compute $h = H(M)$ for any message M.
- It is a one-way function. For a given hash value, it is not feasible to find the original message M.
- It has collision resistance. For a given message M_1, it is not feasible to find another message M_2 that has the same hash value.

A number of hash functions are in use, including MD5, SHA-1, and SHA-256. Weaknesses have been discovered in MD5 and SHA-1, thus it is currently recommended that SHA-256 be used.

Let us suppose that you receive a certificate that purports to come from Company XYZ. How do you know that this is not a forgery? There are two approaches to creating certificates:

- *Self-certification*: In this approach, a company announces its public key ahead of time. To verify a document, you obtain that company's public key and then use it to verify the certificate. This is relatively straightforward but leads to the question of whether you can really trust that company's public key. Perhaps it was published by a forger.
- *Trusted party*: In this approach, a trusted third party signs the certificate. Since you trust the third party, you can trust the authenticity of the document. This approach leads to a question of who are the trusted third parties and how do you know if you can trust them?

In practice, the term *digital certificate* usually refers to a file that conforms to the X.509 international standard. An X.509 file will include demographic information (name, address of the creator of the certificate). The X.509 standard utilizes a trusted third-party mechanism—a set of companies/organizations that are globally trusted. A digital certificate (called a *root certificate*) identifying each of these companies is included with the operating system. On MS Windows, you can view these certificates by selecting Control Panel → Internet Options → Content → Certificates (see Figure 11.5).

If a company wishes to publish a digital certificate, it must provide that certificate to one of these *trusted root authorities*, who will then sign it with their private key. Thus a digital certificate will include an identification of the trusted signer of that certificate. You can then look at that signer's digital certificate to obtain the public key to verify the certificate that you are looking at.

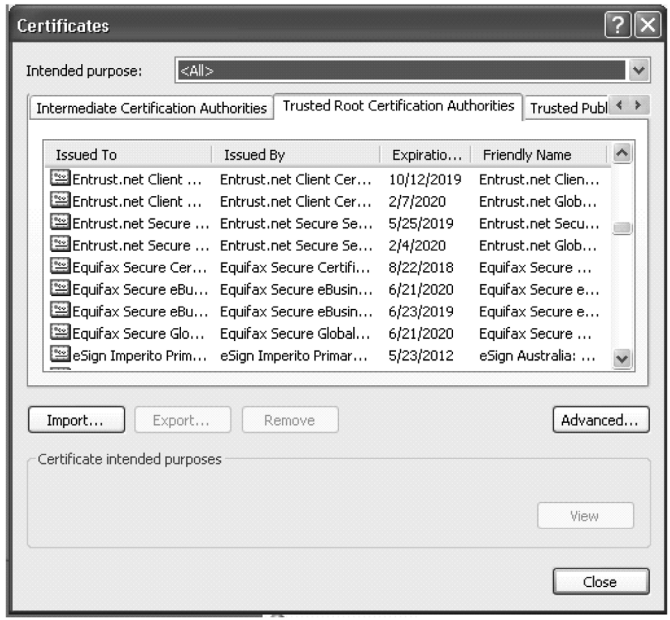

Figure 11.5 Root certificates.

Let's consider an example. Managers at Company XYZ have created a digital certificate that they wish to publish. To guarantee the authenticity of the certificate, they submit it to Company ABC, which is a designated root certificate authority. ABC verifies that XYZ is a valid company and then places their name in the certificate and signs the certificate with ABC's private key.

If you were to receive this signed certificate, you could verify its authenticity with the following steps:

1. Obtain the signer's name from the certificate.
2. Look up the signer's root certificate and obtain his or her public key.
3. Use that public key to decrypt the hash in the original certificate.
4. Verify the accuracy of the hash.

After completing these steps, you would know that this certificate was created by Company XYZ.

A digital certificate can be attached to a file that is intended for download over the Internet. In that case, what is being hashed is the entire file. Once this certificate has been properly signed by a root authority, any user can download it and be assured that it was created by the company that purports to have created it.

11.3.6 People

The people who use a system must be properly trained in basic security concepts and the protection of information. No matter how secure the system is technically, if the users do not practice good security procedures, the system will be vulnerable.

11.4 System Vulnerabilities

Software vulnerabilities are inherent in the operating system and most application programs. In some cases, they were deliberately designed by the programmer in order to facilitate his or her use of the program. In other cases, they are fundamental software defects that were not anticipated at design time.

11.4.1 Social Engineering

Social engineering is an attempt made by an attacker who pretends to be a trusted individual or institution in order to convince someone to divulge confidential information such as passwords or account numbers. There are two common examples:

- *Phishing* is the act of sending out an email that appears to be from a trusted institution, such as the receiver's bank. The user is asked to click on a web link to enter the desired confidential information. The link, of course, takes the user to an attacker's website rather than the trusted institution.
- *Pretexting* involves an attacker who makes a phone call to someone in an organization and poses as someone else and asks for confidential information.

11.4.2 Trojan Horse Programs

A *Trojan horse* is a program that masquerades as something beneficial but actually causes damage or invades privacy. It can also be a beneficial program that is accidentally or deliberately misused by a programmer. A prime example of a Trojan horse program is a terminal login emulator that hijacks/overrides an ATM or other password access controlled system.

In the first attempt to access a secure system, the user unwittingly communicates with the Trojan horse terminal emulator that intervenes between the user and the legitimate login software. The Trojan horse program logs the keystrokes that the user makes and then returns a message to the user that the password has been entered incorrectly and to try to login again. Having acquired the user's authentication data, the Trojan horse then permits the user to access the true terminal login software the next time. The user is none the wiser. The transaction is successfully completed and the user imagines that he or she must have keyed in the wrong PIN number on the first attempt. The Trojan horse then communicates the vital login information across the Internet to its control program.

11.4.3 Spyware

Spyware includes relatively innocuous *software cookies* that monitor the user's browser habits and report them back to an application when the computer is online. Browsers can be set so that a pop-up dialog box requests the browser's permission to deposit a cookie on the disk drive in exchange for access to the services provided by the website. A common form of spyware will hijack the user's web browser so that web searches are diverted to an unexpected website. Other spyware can search the hard drive for personal information, including social security numbers, credit card numbers, driver's license numbers, and other data that could be used to profile individuals, steal identities, or perpetrate credit card or other financial fraud (such as providing account numbers for bogus e-commerce and e-banking transactions).

11.4.4 Trap Doors

A *trap door* is a program fragment that issues a sequence of commands (sometimes just a special password) that the program writers have inserted to circumvent the security system that they designed to control access to their programs. Since these trap doors are not registered as normal logins, programmers who design software with trap doors have used them to carry out illegal activities undetected. Such activities include changing data in a database or tampering with accounting procedures and diverting funds from a group of accounts to a private account.

A trap door written into the login procedure is relatively easy to detect. Trap doors may be written into compilers by applications programmers. The compiler automatically adds a trap door to the executable file of any commercial software it compiles. Thus, the trap door code could not be discovered by examining the source code of the programs. The source code of such a customized compiler (which might be hard to acquire and hard to find once acquired) would have to be examined to locate the trap door.

11.4.5 Database Access Vulnerabilities

The dialog query entry text box for SQL and other database program user interfaces is supposed to be used for valid queries. However, if the program does insufficient checking of user input and simply inserts the user's text into an SQL query string, it is possible for the user to get the program to perform SQL functions that it was not intended to do. This is called an SQL injection attack.

11.4.6 Buffer and Stack Overflow

In many applications where the user must enter data into an input field, the user sends more data than the program anticipates. The data can eventually *overflow* the memory space allocated to the program and enter into a secure area in memory, such as the stack or kernel memory space (see Figure 11.6). The malicious hacker will tamper with pointers to execute code included as part of the data so that, instead of returning to the

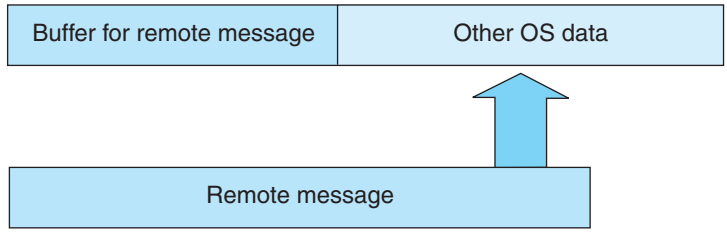

Figure 11.6 Buffer overflow illustration.

application, control is transferred to malicious code that is then executed in a secure area. Processes executing in secure areas of memory are associated with domains having broad access to a variety of resources.

Password files (stored in the root of the file system) can be located by programs running in kernel memory, and core dump files of crashed programs (intended to be used for debugging) can be sifted for useful information about the system and its programs. When vulnerability appears in a network access program, it is particularly pernicious because the attacker can use it to obtain information about the system and to execute malicious programs on the system.

Notice that both database vulnerabilities and buffer/stack overflows occur because the programmer made assumptions about the length or format of input data. One must always completely validate user input before accepting it.

11.5 Invasive and Malicious Software

Invasive and malicious software includes *viruses* and *worms* and any other programs that have the property of being self-replicating and/or self-propagating (they can spread themselves from one computer to another).

In addition to the ability to replicate themselves (often in hard-to-find places such as the boot sector of another file), viruses typically have harmful side effects. Some viruses can do something relatively innocuous such as changing wallpaper, posting flamboyant messages on the desktop, or otherwise changing the appearance of the graphical interface to the operating system. Other viruses can destabilize the operating system by deleting files, disabling applications, and causing other problems.

Viruses can be inadvertently transferred when files from a disk inserted into an external drive are copied to the internal hard disk. Viruses can also be transferred to the (internal) hard disk when email attachments are opened or downloaded. Viruses are fragments of operating system script code found in parts of files, sometimes hidden in the boot sector. Opening such a file can cause the virus script to execute. The first thing the virus code usually does is make a copy of itself to one or more files in the local file system. The adverse side effects caused when the virus changes the attributes

of an object (e.g., by changing the registry in Windows) often appear after rebooting the machine.

Worms are complete programs or sets of collaborating programs that have the ability to transfer themselves from one machine to another and to communicate with each other or to receive commands from a master program on a remote machine. They can transfer themselves to any computer logged on to the Internet that they are able to access.

In addition to the ability to migrate from one machine to another, worms often have additional capabilities. An important one is that most worms have to report back to a master process the exact machine and the conduit they used (IP address, port number, and domain) to install themselves successfully. Using this information, the master process can send command signals across the Internet to launch a distributed denial of service attack on one or more servers.

11.6 Defending the System and the User

The system can be defended against viruses in two ways. The best method of protection against viruses is to practice safe computing. This entails common sense practices such as setting the web browser to detect/reject cookies and not opening or downloading email attachments from untrusted or unknown correspondents.

But no matter how careful you might be, using the Internet and/or inserting disks into external drives and copying files from them to the hard disk drive or executing applications on them, viruses will end up infesting your computer. The best way to eliminate these viruses is by installing antivirus software that will scan all files for viruses and remove or quarantine those that it discovers. Antivirus software should be updated often since it is only good against viruses that the programmers deemed were prevalent at the time it was written.

It is more difficult to defend the system against worms. A worm often gains access to a system by exploiting known vulnerabilities of system applications, such as the buffer overflow vulnerability of some network applications. Any system that a human user can access can be accessed automatically by a worm. Worms are also harder to delete. Since most worms on a system will be executing, the worm file cannot be simply deleted, as is the case with a virus. However, worms in the process of execution can be deleted with the aid of the process manager and the file manager.

For servers and other computers with broadband access that are always on and have a fixed IP address, the best way to defend against worms and unauthorized users is to control Internet access on a machine by installing a *firewall*—a program that filters out harmful data by using high-level application gateways and low-level router gateways. Low-level filters specify protocols that allow specific IP addresses to use specific port numbers to access a specific service and deny connection to that address, port, and service to all others. Filters can be circumvented by standard techniques such as IP address spoofing. When hackers discover IP addresses, port numbers, services, and domains to which a firewall grants connections and system access and the association

between the port numbers and the services, they can spoof one of these IP addresses. The acceptable IP addresses and port numbers can be inserted into the headers of the packets that the program uses to initiate or terminate communication with the machine (e.g., SYN, ACK, RST, and FIN packets). With the aid of this information, the intruder can tunnel into the network using the protocols acceptable to the firewall.

Apart from preventive measures, the best overall strategy is to conduct periodic checkups of the system/network server/network host when it is running and connected to the Internet for any length of time. The periodic checkup may include any of the following: conducting virus scans with antivirus software, file checksum verifications, process and thread inspections (especially network daemons to locate any unidentified processes that have been running for a long time), using specialized software to search for spyware and viruses, checking access protection data for system files and other resources such as device drivers, requiring users to change passwords that are easy to guess or are out of date, and deleting unexpected pointers in the directory search path.

To defend against identity theft and financial fraud, it is good practice to avoid entering information that could be used to identify an individual into (insecure) web page dialog boxes.

11.7 Intrusion Detection Management

There are two basic patterns for intrusion detection: *signature-based detection* and *anomaly detection*. Like antivirus software, *signature detection* looks for known patterns of behavior established by previous attacks. For instance, multiple logons to an account indicate that an intruder may be trying to guess the password to an account. An application that scans port numbers by sending premature FIN packets indicates that an attacker is looking for open ports, while a connection followed by the transmission of an inordinate amount of data can indicate that the attacker is looking to exploit a buffer overflow to obtain a variety of sensitive system information that may include passwords and IP addresses that could provide access to system accounts.

Anomaly detection is a process that looks for unusual patterns in computer behavior. For instance, a worm program exploiting a port or network application daemon might be detected by the transfer of unusually large amounts of data that would not normally be transferred when the daemon is in that particular state.

In addition to scanning for attack patterns or anomalous behavior on the current state of the machine, intruders who may no longer be active or are temporarily quiescent can be detected by processing the log and audit files. With the aid of these files, past attack patterns can be detected and anomalous events can be analyzed. Information from log and audit files can be cross referenced with the information obtained from the scan of the most recent state of the machine to provide further evidence of the presence of an intruder.

In either case, if an intrusion into the system is suspected, appropriate action can be taken. A system administrator can be warned. Anomalous processes can be deleted from the system. A virtual resource, called a honey pot, can be set up to lure the

intruder into the system and extract information about the intruder program while it attempts to exploit the resource.

11.8 Security and Privacy

As we have said, *security* involves preventing the unauthorized access to information. This is different from *privacy*, which involves preventing the unauthorized disclosure of information. What is the difference? Employees who may be authorized to access certain information violate a company's privacy policy if they disclose that information to someone who is not authorized to see it.

Current commercial operating systems do nothing to enforce privacy restrictions. Once authorized users obtain information, the system allows them to do whatever they like with that information. That is because the operating system considers information to be just a set of bits, with no understanding of what those bits represent. Thus, in today's world, enforcing a privacy policy is something that is a training and administrative issue with no technical support.

11.9 Secure Systems Versus Systems Security

Most operating systems and applications in common use at the time this book was written were designed with the goal of maximum connectivity and ease of use. Software plugins are developed by industry or open source programmers to cover up additional vulnerabilities as they are discovered or exposed. When vulnerabilities for an application or an operating system are discovered, the information about the availability of patches and the patches themselves are posted on websites or emailed to users and systems managers.

As new viruses are discovered, antivirus software is updated to recognize their signatures so that the new releases can be removed or at least quarantined. As specific websites with unacceptable content and addresses that send junk email proliferate, application-level filters for browsers and email applications can be updated and adjusted as needed. Security requires overhead, but overhead becomes faster and cheaper as system hardware and software evolves.

What if worrying about systems security was not an afterthought and operating systems designers created an operating system with the forethought to build security in from the foundation up? How can this be done without compromising connectivity and ease of use? An attempt to design a secure system that is easy to use and can handle insecurity from the foundation up has already been attempted with the Java programming language and the Java Virtual Machine (JVM). The basic strategy is to protect the management of main memory from the programmer. Main memory is

viewed as the essential source of program and operating system vulnerability.

The Java programming language does not allow its objects to directly access main memory. Moreover, various processes are classified and labeled as trustworthy or not and then allowed appropriate access by the JVM by indelibly marking the stack frame where the object is loaded and which the Java programs executing the object are unable to access.

The three main vulnerabilities related to viruses and worms of an operating system (for a single host machine, a server, or a network of machines) are memory for process storage, disk drives for file storage, and the TCP/IP network protocol and the login procedure for Internet communication. Any attempt to design a future operating system with security built into the architecture must try to extend what the designers of the Java programming language did to the broader scope of the entire operating system, the network, and the resources that they manage.

11.10 Summary

This chapter defines security and protection, and explores the organization of resource protection. Domains are defined for processes. Each process is eligible to execute zero or more operations on each hardware/software resource. Protection can be expressed in terms of the access matrix, which can be extended to allow processes defined on one domain to switch to another, or to copy, add, or delete operations from one matrix cell to another. The access matrix can be implemented globally by storing and then searching ordered triples in a table. It can also be implemented by looking at the list of capabilities available to each domain on each resource.

The problem of security was discussed and contrasted to that of protection. The discussion of security begins with the observation that most systems and applications have vulnerabilities that can be accidentally or deliberately triggered to impair the system. A host of security threats, such as worms, viruses, and spyware, are discussed, as are software countermeasures for each of these threats, such as firewalls, antivirus software, and spyware. Despite the availability of these software countermeasures, the problem is that intruders still manage to access system resources and log on to system accounts. When they do, subtler means must be used to identify their presence. The two main techniques for detecting intruders are to look for pattern signatures and anomalies that can be found by looking at the present state of the system or at information about past system activity available in log and audit files.

Key Terms

access matrix	domain	root certificate
access privilege	external security	security policy
address spoof	firewall	signature detection
anomaly detection	gateway	smartcard
antivirus software	identity theft	software cookie
asymmetric encryption	intrusion detection	spyware
authentication	message digest	stack inspection
capability	packet filter	symmetric encryption
credit card fraud	phishing	system vulnerability
cryptography	pretexting	trap door
denial of service attack	protection	tunnel
	rights	virus
digital signature	role	worm

Exercises and Questions

1. List the access rights and sketch the domain implementations for the Unix file access specification given in Section 11.3.3. (*Hint*: There are a multitude of possible domains. Choose two reasonable file policies that you, as the owner of the file, might want to implement as domains for processes wanting to access the files in your directory. In each case, list the ordered pairs and sketch the relationship between the three domains in a Venn diagram, such as the one shown in Figure 11.1).

2. Sketch a domain implementation in which $i < j \Rightarrow D_i \subseteq D_j; i, j = 1, ..., 5$. Can you explain why this is called a ring implementation? If your program had a process that you wanted to have at least as many access rights as any other process could have, which domain would you select for it?

3. Write a program in pseudocode to implement an access matrix so an operating systems manager can control access to processes associated with domains that request resources from objects. Explain how you would attempt to write a similar program to implement domains of access right pairs. Why is the first task easier?

4. Write a program to implement an access matrix with a global table. When an operation M on D is executed on a resource R in domain D, the program searches the table for the ordered triple (M,D,O). Each time a process, defined on domain D, tries to execute an operation on resource O, the global table is checked for a match. If the ordered triple is located, the operation is permitted; otherwise an exception is thrown.

5. For Table 11.2, explain what kind of process could have access to domains 1, 2, and 3. What kind of resources could R_1, R_2, R_3, and R_4 be?

6. With regard to Table 11.3, define a process that would need to carry out the various operations and domain switches. Describe a scenario in which the switch operation could be used to complete a useful workload and execution cycle for the application.

7. Define a third access matrix modifier called *control* that is applicable to domain objects only. When the word *control* appears in the cell of an access matrix, the process defined on that domain has the ability to change other cells in the same row. Add the word *control* to cell a_{27} in Table 11.3. Show the matrix that results when the process with domain D_2 uses its control to remove the operations that a process defined on D_3 can access on R_4 and to add the operations that a process defined on D_3 can access on R_2.

8. Download a Java compiler/interpreter and use it to load and run an untrusted applet from a web page. Modify the applet to invoke a method to access a resource that should be protected from an untrusted class, hence not be accepted by the JVM. Check if the applet is able to access the resource or whether Java throws the exception as promised and discontinues the execution of the applet.

9. A computer has a one or ten gigahertz processor on a computer connected to a server on an Internet connection that transfers information at a rate of 10 or 100 megabits per second. Suppose that a program running on this computer is dedicated to cracking passwords. Compute how long it will take a program to guess the following passwords by trial and error:

    a. A 6-character password in which each character is a letter in the English alphabet.

    b. A 6-character password in which each character is a letter or a digit.

    c. A 10-character password in which each character is selected from among 36 different Unicode characters.

10. In the Windows operating system, object attributes (properties determined by data) of software objects are stored in a registry. Write an innocuous virus code fragment and insert it into a text file. Attach the text file to an email message and send it to yourself. When you open/download the attachment, the virus fragment should copy itself to a file in a specified directory location and exhibit a harmless side effect, such as changing the wallpaper of the desktop to your school colors by changing the registry. When you reboot the machine, the appearance of the desktop graphical interface should change.

11. Use the file search engine on your operating system file manager to locate the virus in Exercise 10 by matching it to key script commands in the code fragment that uniquely identify it. Now write your own file search program to first locate and then either quarantine or delete the innocuous virus that you created.

12. Write a white paper on how to redesign the file system manager and the directory that would make the file system secure from processes and threads in a manner analogous to the way Sun Microsystems designed the Java language (as mediated by the JVM compiler/interpreter) to make main memory secure from processes and threads.