Unit-2
Chapter -1: Protection

Protection refers to a mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system. This mechanism must provide a means for specifying the controls to be imposed, together with a means of enforcement. We distinguish between protection and security, which is a measure of confidence that the integrity of a system and its data will be preserved.

Goals of Protection

Protection was originally conceived as an adjunct to multiprogramming operating systems, so that untrustworthy users might safely share a common logical name space, such as a directory of files, or share a common physical name space, such as memory.

We need to provide protection for several reasons. The most obvious is the need to prevent the mischievous, intentional violation of an access restriction by a user.

Protection can improve reliability by detecting latent errors at the interfaces between component subsystems. A protection-oriented system provides means to distinguish between authorized and unauthorized usage.

The role of protection in a computer system is to provide a mechanism for the enforcement of the policies governing resource use. These policies can be established in a variety of ways. Some are fixed in the design of the system, while others are formulated by the management of a system.

Policies for resource use may vary by application, and they may change over time. For these reasons, protection is no longer the concern solely of the designer of an operating system. The application programmer needs to use protection mechanisms as well, to guard resources created and supported by an application subsystem against misuse.

Mechanisms are distinct from policies. Mechanisms determine how something will be done; policies decide what will be done. The separation of policy and mechanism is important for flexibility. Policies are likely to change from place to place or time to time.

Principles of Protection

Consider the analogy of a security guard with a passkey. If this key allows the guard into just the public areas that she guards, then misuse of the key will result in minimal damage. If, however, the passkey allows access to all areas, then damage from its being lost, stolen, misused, copied, or otherwise compromised will be much greater.

An operating system following the **principle of least privilege** implements its features, programs, system calls, and data structures so that failure or compromise of a component does the minimum damage and allows the minimum damage to be done.

Operating system also provides system calls and services that allow applications to be written with fine-grained access controls. It provides mechanisms to enable privileges when they are needed and to disable them when they are not needed.

Managing users with the principle of least privilege entails creating a separate account for each user, with just the privileges that the user needs. An operator who needs to mount tapes and back up files on the system has access to just those commands and files needed to accomplish the job. Some systems implement role-based access control (RBAC) to provide this functionality.

Domain of Protection

A computer system is a collection of processes and objects**. By objects,** we mean both hardware objects (such as the CPU, memory segments, printers, disks, and tape drives) and software objects (such as files, programs, and semaphores).

Domain: It is a collection of access rights, each of which is an ordered pair <object name, rights set>. Each user or each process may be a domain

The operations that are possible may depend on the object. For example, on a CPU, we can only execute. Memory segments can be read and written, whereas a CD-ROM or DVD-ROM can only be read.

A process should be allowed to access only those resources for which it has authorization. Furthermore, at any time, a process should be able to access only those resources that it currently requires to complete its task. This second requirement, commonly referred to as the **need-to-know principle**, is useful in limiting the amount of damage a faulty process can cause in the system.

For example, when process p invokes procedure A() , the procedure should be allowed to access only its own variables and the formal parameters passed to it; it should not be able to access all the variables of process p.

Domain Structure

A process operates within a protection domain, which specifies the resources that the process may access. Each domain defines a set of objects and the types of operations that may be invoked on each object. The ability to execute an operation on an object is an access right. A domain is a collection of access rights, each of which is an ordered pair <object-name, rights-set>.

For example, if domain D has the access right <file F , {read, write}>, then a process executing in domain D can both read and write file F. It cannot, however, perform any other operation on that object.

Domains may share access rights. For example, in Figure 14.1, we have three domains: D1, D2 and D3 . The access right <O4, {print}> is shared by D2 and D3, implying that a process executing in either of these two domains can print object O4 .

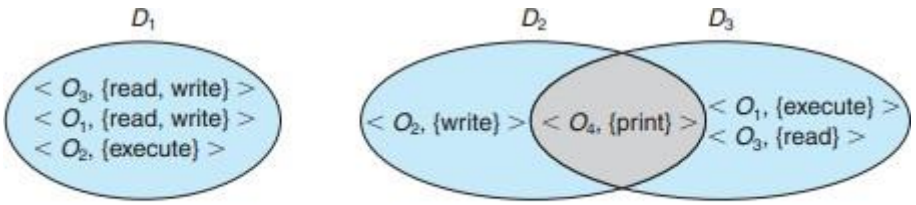


Figure 14.1 System with three protection domains.

If the association between processes and **domains is fixed,** and we want to adhere to the need-to-know principle, then a mechanism must be available to change the content of a domain. The reason stems from the fact that a process may execute in two different phases and may, for example, need read access in one phase and write access in another. If a domain is static, we must define the domain to include both read and write access.

Thus, the need-to-know principle is violated. We must allow the contents of a domain to be modified so that the domain always reflects the minimum necessary access rights.

If the **association is dynamic**, a mechanism is available to allow **domain switching**, enabling the process to switch from one domain to another. We may also want to allow the content of a domain to be changed. If we cannot change the content of a domain, we can provide the same effect by creating a new domain with the changed content and switching to that new domain when we want to change the domain content.

A domain can be realized in a variety of ways:

• Each user may be a domain. In this case, the set of objects that can be accessed depends on the identity of the user. Domain switching occurs when the user is changed — generally when one user logs out and another user logs in.

• Each process may be a domain. In this case, the set of objects that can be accessed depends on the identity of the process. Domain switching occurs when one process sends a message to another process and then waits for a response.

• Each procedure may be a domain. In this case, the set of objects that can be accessed corresponds to the local variables defined within the procedure. Domain switching occurs when a procedure call is made.

An Example: UNIX

In the UNIX operating system, a domain is associated with the user. Switching the domain corresponds to changing the user identification temporarily. This change is accomplished through the file system as follows. An owner identification and a domain bit (known as the setuid bit) are associated with each file. When the setuid bit is on, and a user executes that file, the userID is set to that of the owner of the file. When the bit is off , however, the userID does not change. For example, when a user A (that is, a user with userID = A) starts executing a file owned by B, whose associated domain bit is off ,the userID of the process is set to A. When the setuid bit is on, the userID is set to that of the owner of the file: B. When the process exits, this temporary userID change ends.

An Example: MULTICS

In the MULTICS system, the protection domains are organized hierarchically into a ring structure. Each ring corresponds to a single domain (Figure 14.2). The rings are numbered from 0 to 7. Let Di and Dj be any two domain rings. If j < i, then Di is a subset of Dj.

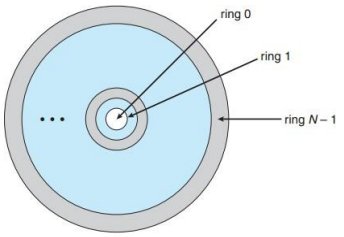


Figure 14.2 MULTICS ring structure.

That is, a process executing in domain Dj has more privileges than does a process executing in domain Di . A process executing in domain D0 has the most privileges.

MULTICS has a segmented address space; each segment is a file, and each segment is associated with one of the rings. A segment description includes an entry that identifies the ring number. In addition, it includes three access bits to control reading, writing, and execution. The association between segments and rings is a policy decision with which we are not concerned here.

Domain switching in MULTICS occurs when a process crosses from one ring to another by calling a procedure in a different ring. Obviously, this switch must be done in a controlled manner; otherwise, a process could start executing in ring 0, and no protection would be provided. To allow controlled domain switching, we modify the ring field of the segment descriptor to include the following:

- **Access bracket.:** A pair of integers, b1 and b2 ,such that $b1 \leq b2$.
- **Limit**. An integer b3 such that$b3 > b2$.
- **List of gates.** Identifies the entry points (or gates) at which the segments may be called.

If a process executing in ring i calls a procedure (or segment) with access bracket (b1 , b2), then the call is allowed if $b1 \leq i \leq b2$, and the current ring number of the process remains i. Otherwise, a trap to the operating system occurs, and the situation is handled as follows:

If **i < b1**, then the call is allowed to occur, because we have a transfer to a ring (or domain) with fewer privileges. However, if parameters are passed that refer to segments in a lower ring (that is, segments not accessible to the called procedure), then these segments must be copied into an area that can be accessed by the called procedure.

If **i > b2**, then the call is allowed to occur only if b3 is greater than or equal to i and the call has been directed to one of the designated entry points in the list of gates. This scheme allows processes with limited access rights to call procedures in lower rings that have more access rights, but only in a carefully controlled manner.

Access Matrix

Our general model of protection can be viewed abstractly as a matrix, called an access matrix. The rows of the access matrix represent domains, and the columns represent objects. Each entry in the matrix consists of a set of access rights. Because the column defines objects explicitly, we can omit the object name from the access right. The entry access(i,j) defines the set of operations that a process executing in domain Di can invoke on object Oj.

To illustrate these concepts, we consider the access matrix shown in Figure 14.3. There are four domains and four objects — three files (F1, F2, F3) and one laser printer. A process executing in domain D1 can

read files F1 and F3 .A process executing in domain D4 has the same privileges as one executing in domain D1; but in addition, it can also write onto files F1 and F3 .The laser printer can be accessed only by a process executing in domain D2.

object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Figure 14.3 Access matrix.

The access matrix can implement policy decisions concerning protection. The policy decisions involve which rights should be included in the (i,j)th entry. We must also decide the domain in which each process executes. This last policy is usually decided by the operating system.

The access matrix provides an appropriate mechanism for defining and implementing strict control for both static and dynamic association between processes and domains. When we switch a process from one domain to another, we are executing an operation (switch) on an object (the domain). We can control domain switching by including domains among the objects of the access matrix. Similarly, when we change the content of the access matrix, we are performing an operation on an object: the access matrix.

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Figure 14.4 Access matrix of Figure 14.3 with domains as objects.

Processes should be able to switch from one domain to another. Switching from domain Di to domain Dj is allowed if and only if the access right switch ∈ access(i,j). Thus, in Figure 14.4, a process executing in

domain D2 can switch to domain D3 or to domain D4. A process in domain D4 can switch to D1 and one in domain D1 can switch to D2.0

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Figure 14.5 Access matrix with ***copy*** rights.

The ability to copy an access right from one domain (or row) of the access matrix to another is denoted by an asterisk (*) appended to the access right. The copyright allows the access right to be copied only within the column (that is, for the object) for which the right is defined. For example, in Figure 14.5(a), a process executing in domain D2 can copy the read operation into any entry associated with file F2. Hence, the access matrix of Figure 14.5(a) can be modified to the access matrix shown in Figure 14.5(b).

We also need a mechanism to allow addition of new rights and removal of some rights. The owner right controls these operations. If access(i,j) includes the owner right, then a process executing in domain Di can add and remove any right in any entry in column j.

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Figure 14.6 Access matrix with `owner` rights.

For example, in Figure 14.6(a), domain D1 is the owner of F1 and thus can add and delete any valid right in column F1. Similarly, domain D2 is the owner of F2 and F3 and thus can add and remove any valid right within these two columns. Thus, the access matrix of Figure 14.6(a) can be modified to the access matrix shown in Figure 14.6(b).

The copy and owner rights allow a process to change the entries in a column. A mechanism is also needed to change the entries in a row. The control right is applicable only to domain objects. If access (i, j) includes the control right, then a process executing in domain Di can remove any access right from row j. For example, suppose that, in Figure 14.4, we include the control right in access (D2, D4). Then, a process executing in domain D2 could modify domain D4, as shown in Figure 14.7.

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Figure 14.4 Access matrix of Figure 14.3 with domains as objects.

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Figure 14.7 Modified access matrix of Figure 14.4.

Implementation of the Access Matrix

In general, the matrix will be sparse; that is, most of the entries will be empty. Although data-structure techniques are available for representing sparse matrices, they are not particularly useful for this application, because of the way in which the protection facility is used. Here, we first describe several methods of implementing the access matrix and then compare the methods.

Global Table

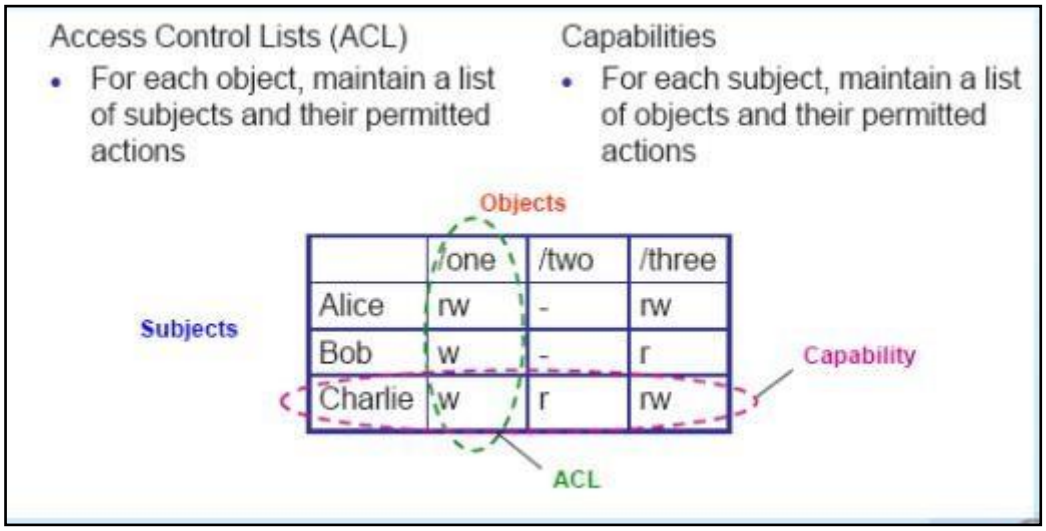
The simplest implementation of the access matrix is a global table consisting of a set of ordered triples <domain, object, rights-set >. Whenever an operation M is executed on an object Oj within domain Di, the global table is searched for a triple < Di, O j, Rk >,with M ∈ Rk. If this triple is found, the operation is allowed to continue; otherwise, an exception (or error) condition is raised.

This implementation suffers from several drawbacks. The table is usually large and thus cannot be kept in main memory, so additional I/O is needed.

Access Lists for Objects

Each column in the access matrix can be implemented as an access list for one object. The empty entries can be discarded. The resulting list for each object consists of ordered pairs <domain, rights-set>, which define all domains with a nonempty set of access rights for that object.

This approach can be extended easily to define a list plus a default set of access rights. When an operation M on an object Oj is attempted in domain Di , we search the access list for object Oj, looking for an entry < Di, Rk > with M ∈ Rk. If the entry is found, we allow the operation; if it is not, we check the default set. If M is in the default set, we allow the access. Otherwise, access is denied, and an exception condition occurs. For efficiency, we may check the default set first and then search the access list.



Capability Lists for Domains

Rather than associating the columns of the access matrix with the objects as access lists, we can associate each row with its domain. A capability list for a domain is a list of objects together with the operations

allowed on those objects. An object is often represented by its physical name or address called a **capability** .To execute operation M on object Oj, the process executes the operation M, specifying the capability (or pointer) for object Oj as a parameter. Simple possession of the capability means that access is allowed.

The capability list is associated with a domain, but it is never directly accessible to a process executing in that domain. Rather, the capability list is itself a protected object, maintained by the operating system and accessed by the user only indirectly. Capability-based protection relies on the fact that the capabilities are never allowed to migrate into any address space directly accessible by a user process (where they could be modified).

Capabilities are usually distinguished from other data in one of two ways:

Each object has a tag to denote whether it is a capability or accessible data. The tags themselves must not be directly accessible by an application program. Hardware or firmware support may be used to enforce this restriction. Although only one bit is necessary to distinguish between capabilities and other objects, more bits are often used. This extension allows all objects to be tagged with their types by the hardware. Thus, the hardware can distinguish integers, floating-point numbers, pointers, Booleans, characters, instructions, capabilities, and uninitialized values by their tags.

• Alternatively, the address space associated with a program can be split into two parts. One part is accessible to the program and contains the program's normal data and instructions. The other part, containing the capability list, is accessible only by the operating system.

A Lock–Key Mechanism

The lock – key scheme is a compromise between access lists and capability lists. Each object has a list of unique bit patterns, called locks. Similarly, each domain has a list of unique bit patterns, called keys. A process executing in a domain can access an object only if that domain has a key that matches one of the locks of the object.

Comparison

Using a global table is simple; however, the table can be quite large and often cannot take advantage of special groupings of objects or domains. Access lists correspond directly to the needs of users. When a user creates an object, he can specify which domains can access the object, as well as what operations are allowed. However, because access-right information for a particular domain is not localized, determining the set of access rights for each domain is difficult. In addition, every access to the object must be checked, requiring a search of the access list. In a large system with long access lists, this search can be time consuming.

Capability lists do not correspond directly to the needs of users, but they are useful for localizing information for a given process. The process attempting access must present a capability for that access. Then, the protection system needs only to verify that the capability is valid. Revocation of capabilities, however, may be inefficient.

The lock – key mechanism, as mention ed, is a compromise between access lists and capability lists. The mechanism can be both effective and flexible, depending on the length of the keys. The keys can be passed freely from domain to domain. In addition, access privileges can be effectively revoked by the simple technique of changing some of the locks associated with the object.

Most systems use a combination of access lists and capabilities. When a process first tries to access an object, the access list is searched. If access is denied, an exception condition occurs. Otherwise, a capability is created and attached to the process. Additional references use the capability to demonstrate swiftly that access is allowed. After the last access, the capability is destroyed. This strategy is used in the MULTICS system and in the CAL system.

Access Control

Each file and directory is assigned an owner, a group, or possibly a list of users, and for each of those entities, access-control information is assigned. A similar function can be added to other aspects of a computer system. A good example of this is found in Solaris 10.

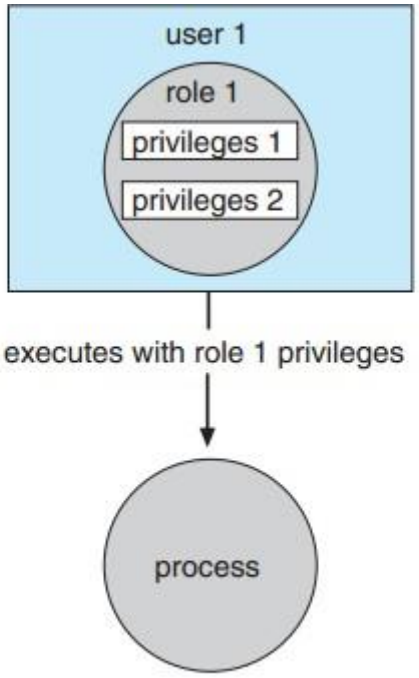


Figure 14.8 Role-based access control in Solaris 10.

Solaris 10 advances the protection available in the operating system by explicitly adding the principle of least privilege via role-based access control (RBAC). This facility revolves around privileges. A privilege is the right to execute a system call or to use an option within that system call (such as opening a file with write access). Privileges can be assigned to processes, limiting them to exactly the access they need to perform their work. Privileges and programs can also be assigned to roles. Users are assigned roles or can take roles based on passwords to the roles. In this way, a user can take a role that enables a privilege, allowing the user to run a program to accomplish a specific task, as depicted in Figure 14.8.

Revocation of Access Rights

In a dynamic protection system, we may sometimes need to revoke access rights to objects shared by different users. Various questions about revocation may arise:

- **Immediate versus delayed.** Does revocation occur immediately, or is it delayed? If revocation is delayed, can we find out when it will take place?
- **Selective versus general.** When an access right to an object is revoked, does it affect all the users who have an access right to that object, or can we specify a select group of users whose access rights should be revoked?
- **Partial versus total.** Can a subset of the rights associated with an object be revoked, or must we revoke all access rights for this object?
- **Temporary versus permanent.** Can access be revoked permanently (that is, the revoked access right will never again be available), or can access be revoked and later be obtained again?

With an access-list scheme, revocation is easy. The access list is searched for any access rights to be revoked, and they are deleted from the list. Revocation is immediate and can be general or selective, total or partial, and permanent or temporary.

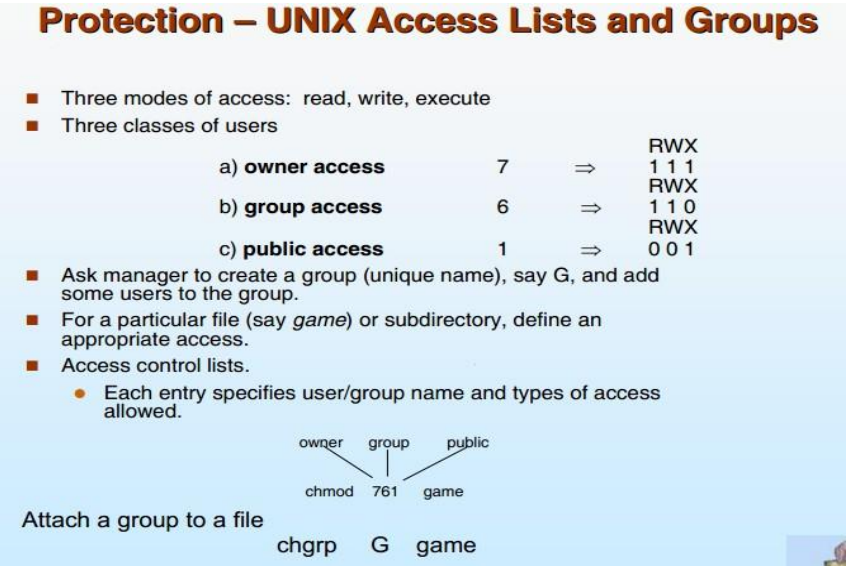
Capabilities, however, present a much more difficult revocation problem, as mentioned earlier. Since the capabilities are distributed throughout the system, we must find them before we can revoke them. Schemes that implement revocation for capabilities include the following:

- **Reacquisition.** Periodically, capabilities are deleted from each domain. If a process wants to use a capability, it may find that that capability has been deleted. The process may then try to reacquire the capability. If access has been revoked, the process will not be able to reacquire the capability.

- **Back-pointers.** A list of pointers is maintained with each object, pointing to all capabilities associated with that object. When revocation is required, we can follow these pointers, changing the capabilities as necessary. This scheme was adopted in the MULTICS system. It is quite general, but its implementation is costly.
- **Indirection.** The capabilities point indirectly, not directly, to the objects. Each capability points to a unique entry in a global table, which in turn points to the object. We implement revocation by searching the global table for the desired entry and deleting it. Then, when an access is attempted, the capability is found to point to an illegal table entry. Table entries can be reused for other capabilities without difficulty, since both the capability and the table entry contain the unique name of the object. The object for a capability and its table entry must match.
- **Keys.** A key is a unique bit pattern that can be associated with a capability. This key is defined when the capability is created, and it can be neither modified nor inspected by the process that owns the capability. A master key is associated with each object; it can be defined or replaced with the set-key operation. When a capability is created, the current value of the master key is associated with the capability. When the capability is exercised, its key is compared with the master key. If the keys match, the operation is allowed to continue; otherwise, an exception condition is raised. Revocation replaces the master key with a new value via the set-key operation,invalidating all previous capabilities for this object.

  This scheme does not allow selective revocation, since only one master key is associated with each object. If we associate a list of keys with each object, then selective revocation ca n be implemented. Finally, we can group all keys into one global table of keys. A capability is valid only if its key matches some key in the global table. We implement revocation by removing the matching key from the table.

Example:



Unit-3
Chapter -2: Security

Security requires not only an adequate protection system but also consideration of the external environment within which the system operates. A protection system is ineffective if user authentication is compromised or a program is run by an unauthorized user.

Computer resources must be guarded against unauthorized access, malicious destruction or alteration, and accidental introduction of inconsistency. These resources include information stored in the system (both data and code), as well as the CPU, memory, disks, tapes, and networking that are the computer.

The Security Problem

In many applications, ensuring the security of the computer system is worth considerable effort. Large commercial systems containing payroll or other financial data are inviting targets to thieves. Systems that contain data pertaining to corporate operations may be of interest to unscrupulous competitors. Furthermore, loss of such data, whether by accident or fraud, can seriously impair the ability of the corporation to function.

Security violations (or misuse) of the system can be categorized as intentional (malicious) or accidental. It is easier to protect against accidental misuse than against malicious misuse. For the most part, protection mechanisms are the core of protection from accidents. The following list includes several forms of accidental and malicious security violations.

We use the terms **intruder** and **cracker** for those attempting to breach security. In addition, a threat is the potential for a security violation, such as the discovery of a vulnerability, whereas an **attack** is the attempt to break security.

Various types of Security Violations

- **Breach of confidentiality**: This type of violation involves unauthorized reading of data (or theft of information). Typically, a breach of confidentiality is the goal of an intruder. Capturing secret data from a system or a data stream, such as credit-card information or identity information for identity theft, can result directly in money for the intruder.
- **Breach of Integrity:** This violation involves unauthorized modification of data. Such attacks can, for example, result in passing of liability to an innocent party or modification of the source code of an important commercial application.

- **Breach of availability:** This violation involves unauthorized destruction of data. Some crackers would rather wreak havoc and gain status or bragging rights than gain financially. Website defacement is a common example of this type of security breach.
- **Theft of service:** This violation involves unauthorized use of resources. For example, an intruder (or intrusion program) may install a daemon on a system that acts as a file server.
- **Denial of service:** This violation involves preventing legitimate use of the system. Denial-of-service (DOS) attacks are sometimes accidental. The original Internet worm turned into a DOS attack when a bug failed to delay its rapid spread.

Attackers use several standard methods in their attempts to breach security. The most common is **masquerading,** in which one participant in a communication pretends to be someone else (another host or another person). By masquerading, attackers breach authentication, the correctness of identification; they can then gain access that they would not normally be allowed or escalate their privileges—obtain privileges to which they would not normally be entitled.

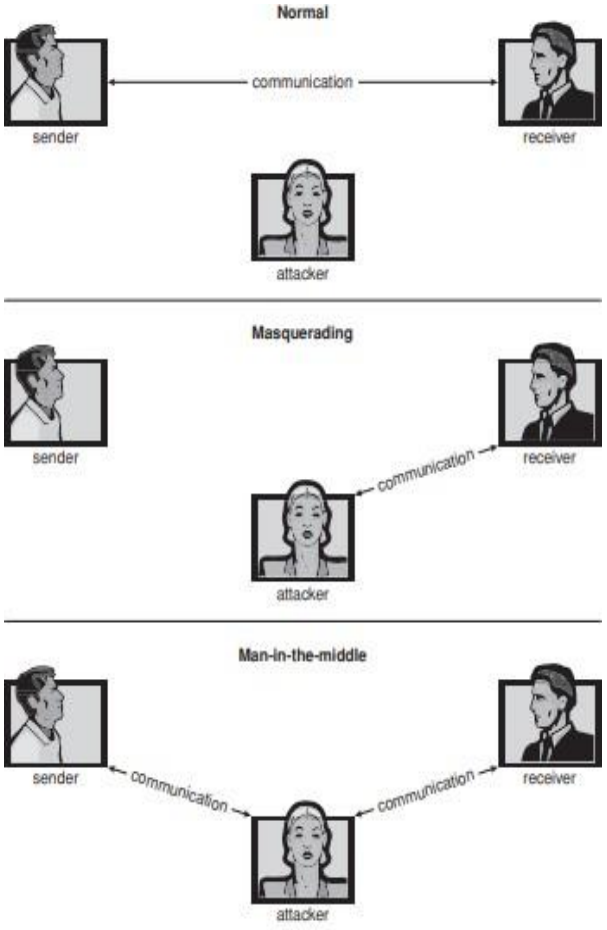


Figure 15.1 Standard security attacks.

Another common attack is to replay a captured exchange of data. A **replay attack** consists of the malicious or fraudulent repeat of a valid data transmission. Sometimes the replay comprises the entire attack— for example, in a repeat of a request to transfer money. But frequently it is done along with message modification, again to escalate privileges.

Consider the damage that could be done if a request for authentication had a legitimate user's information replaced with an unauthorized user's.

In a network communication, a man-in-the-middle attack may be preceded by a **session hijacking**, in which an active communication session is intercepted. Several attack methods are depicted in Figure 15.1.

To protect a system, we must take security measures at four levels:

- **Physical:** The site or sites containing the computer systems must be physically secured against armed or surreptitious entry by intruders. Both the machine rooms and the terminals or workstations that have access to the machines must be secured.
- **Human:** Authorization must be done carefully to assure that only appropriate users have access to the system. Even authorized users, however, may be ‒encouraged‖ to let others use their access (in exchange for a bribe, for example). They may also be tricked into allowing access via social engineering. One type of social-engineering attack is phishing. Here, a legitimate-looking e-mail or web page misleads a user into entering confidential information. Another technique is dumpster diving, a general term for attempting to gather information in order to gain unauthorized access to the computer (by looking through trash, finding phone books, or finding notes containing passwords, for example). These security problems are management and personnel issues, not problems pertaining to operating systems.
- **Operating system:** The system must protect itself from accidental or purposeful security breaches. A runaway process could constitute an accidental denial-of-service attack. A query to a service could reveal passwords. A stack overflow could allow the launching of an unauthorized process. The list of possible breaches is almost endless.
- **Network:** Much computer data in modern systems travels over private leased lines, shared lines like the Internet, wireless connections, or dial-up lines. Intercepting these data could be just as harmful as breaking into a computer, and interruption of communications could constitute a remote denial-of-service attack, diminishing users' use of and trust in the system.

Table 14.1 Threat Consequences, and the Types of Threat Actions that Cause Each Consequence (Based on RFC 2828)

Threat Consequence	Threat Action (Attack)
Unauthorized Disclosure A circumstance or event whereby an entity gains access to data for which the entity is not authorized.	**Exposure:** Sensitive data are directly released to an unauthorized entity. **Interception:** An unauthorized entity directly accesses sensitive data, traveling between authorized sources and destinations. **Inference:** A threat action whereby an unauthorized entity indirectly accesses sensitive data (but not necessarily the data contained in the communication) by reasoning from characteristics or by-products of communications. **Intrusion:** An unauthorized entity gains access to sensitive data by circumventing a system's security protections.
Deception A circumstance or event that may result in an authorized entity receiving false data and believing it to be true.	**Masquerade:** An unauthorized entity gains access to a system or performs a malicious act by posing as an authorized entity. **Falsification:** False data deceive an authorized entity. **Repudiation:** An entity deceives another by falsely denying responsibility for an act.
Disruption A circumstance or event that interrupts or prevents the correct operation of system services and functions.	**Incapacitation:** Prevents or interrupts system operation by disabling a system component. **Corruption:** Undesirably alters system operation by adversely modifying system functions or data. **Obstruction:** A threat action that interrupts delivery of system services by hindering system operation.
Usurpation A circumstance or event that results in control of system services or functions by an unauthorized entity.	**Misappropriation:** An entity assumes unauthorized logical or physical control of a system resource. **Misuse:** Causes a system component to perform a function or service that is detrimental to system security.

Table 14.2 Computer and Network Assets, with Examples of Threats

	Availability	Confidentiality	Integrity
Hardware	Equipment is stolen or disabled, thus denying service.		
Software	Programs are deleted, denying access to users.	An unauthorized copy of software is made.	A working program is modified, either to cause it to fail during execution or to cause it to do some unintended task.
Data	Files are deleted, denying access to users.	An unauthorized read of data is performed. An analysis of statistical data reveals underlying data.	Existing files are modified or new files are fabricated.
Communication Lines	Messages are destroyed or deleted. Communication lines or networks are rendered unavailable.	Messages are read. The traffic pattern of messages is observed.	Messages are modified, delayed, reordered, or duplicated. False messages are fabricated.

Table 14.4 Terminology of Malicious Programs

Name	Description
Virus	Malware that, when executed, tries to replicate itself into other executable code; when it succeeds the code is said to be infected. When the infected code is executed, the virus also executes.
Worm	A computer program that can run independently and can propagate a complete working version of itself onto other hosts on a network.
Logic bomb	A program inserted into software by an intruder. A logic bomb lies dormant until a predefined condition is met; the program then triggers an unauthorized act.
Trojan horse	A computer program that appears to have a useful function but also has a hidden and potentially malicious function that evades security mechanisms, sometimes by exploiting legitimate authorizations of a system entity that invokes the Trojan horse program.
Backdoor (trapdoor)	Any mechanism that bypasses a normal security check; it may allow unauthorized access to functionality.
Mobile code	Software (e.g., script, macro, or other portable instruction) that can be shipped unchanged to a heterogeneous collection of platforms and execute with identical semantics.
Exploits	Code specific to a single vulnerability or set of vulnerabilities.
Downloaders	Program that installs other items on a machine that is under attack. Usually, a downloader is sent in an e-mail.
Auto-rooter	Malicious hacker tools used to break into new machines remotely.
Kit (virus generator)	Set of tools for generating new viruses automatically.
Spammer programs	Used to send large volumes of unwanted e-mail.
Flooders	Used to attack networked computer systems with a large volume of traffic to carry out a denial-of-service (DoS) attack.
Keyloggers	Captures keystrokes on a compromised system.
Rootkit	Set of hacker tools used after attacker has broken into a computer system and gained root-level access.
Zombie, bot	Program activated on an infected machine that is activated to launch attacks on other machines.
Spyware	Software that collects information from a computer and transmits it to another system.
Adware	Advertising that is integrated into software. It can result in pop-up ads or redirection of a browser to a commercial site.

Hacker v/s Cracker

	Hacker	Cracker
Meaning	They are constantly looking for the flaws in the computer and internet security and their sole aim is to rectify these flaws and improve the security of the content.	The purpose of a cracker is to break the security of computers and networks.
Known for	There is a common view that Hackers build things	The crackers are believed to break the things
Sill Level	Hackers have an advanced knowledge of the computer related security.	Crackers are usually not very skilful as hackers. Very few of them are skilled enough to create their own new software and tools.
Internet Security	Hackers potentially restore the security setups across the corrupted networks and they help in catching the specific crackers.	Crackers always know that their activities are illegal and they are breaking the law so they tend to cover up their tracks.

Program Threats

Processes, along with the kernel, are the only means of accomplishing work on a computer. Therefore, writing a program that creates a breach of security, or causing a normal process to change its behaviour and create a breach, is a common goal of **crackers**. In fact, even most nonprogram security events have as their goal causing a program threat. For example, while it is useful to log in to a system without authorization, it is quite a lot more useful to leave behind a **back-door** daemon that provides information or allows easy access even if the original exploit is blocked. In this section, we describe common methods by which programs cause security breaches. Note that there is considerable variation in the naming conventions for security holes and that we use the most common or descriptive terms.

1. Trojan Horse

Many systems have mechanisms for allowing programs written by users to be executed by other users. If these programs are executed in a domain that provides the access rights of the executing

user, the other users may misuse these rights. A text-editor program, for example, may include code to search the file to be edited for certain keywords. If any are found, the entire file may be copied to a special area accessible to the creator of the text editor. A code segment that misuses its environment is called a Trojan horse .

A variation of the Trojan horse is a program that emulates a **login program**. An unsuspecting user starts to log in at a terminal and notices that he has apparently mistyped his password. He tries again and is successful. What has happened is that his authentication key and password have been stolen by the login emulator, which was left running on the terminal by the thief. The emulator stored away the password, printed out a login error message, and exited; the user was then provided with a genuine login prompt. This type of attack can be defeated by having the operating system print a usage message at the end of an interactive session or by a non-trappable key sequence such as the control-alt-delete combination used by all modern Windows operating systems.

Another variation on the Trojan horse is **spyware.** Spyware sometimes accompanies a program that the user has chosen to install. Most frequently, it comes along with freeware or shareware programs, but sometimes it is included with commercial software. The goal of spyware is to download ads to display on the user's system, create pop-up browser windows when certain sites are visited, or capture information from the user's system and return it to a central site.

Spyware is a micro example of a macro problem: violation of the principle of least privilege. Under most circumstances, a user of an operating system does not need to install network daemons. Such daemons are installed via two mistakes. First, a user may run with more privileges than necessary (for example, as the administrator), allowing programs that she runs to have more access to the system than is necessary. This is a case of human error—a common security weakness. Second, an operating system may allow by default more privileges than a normal user needs. This is a case of poor operating-system design decisions. An operating system (and, indeed, software in general) should allow fine-grained control of access and security, but it must also be easy to manage and understand. Inconvenient or inadequate security measures are bound to be circumvented, causing an overall weakening of the security they were designed to implement.

2. Trap Door(Back Door)

The designer of a program or system might leave a hole in the software that only she is capable of using. This type of security breach (or trap door) was shown in the movie War Games. For instance, the code might check for a specific user ID or password, and it might circumvent normal security procedures. Programmers have been arrested for embezzling from banks by including rounding errors in their code and having the occasional half-cent credited to their accounts. This account crediting can add up to a large amount of money, considering the number of transactions that a large bank executes.

A clever trap door could be included in a compiler. The compiler could generate standard object code as well as a trap door, regardless of the source code being compiled. This activity is particularly nefarious, since a search of the source code of the program will not reveal any problems. Only the source code of the compiler would contain the information.

Trap doors pose a difficult problem because, to detect them, we have to analyse all the source code for all components of a system. Given that software systems may consist of millions of lines of code, this analysis is not done frequently, and frequently it is not done at all!

3. Logic Bomb

Consider a program that initiates a security incident only under certain circumstances. It would be hard to detect because under normal operations, there would be no security hole. However, when a predefined set of parameters was met, the security hole would be created. This scenario is known as a logic bomb. A programmer, for example, might write code to detect whether he was still employed; if that check failed, a daemon could be spawned to allow remote access, or code could be launched to cause damage to the site.

4. Stack and Buffer Overflow

The stack- or buffer-overflow attack is the most common way for an attacker outside the system, on a network or dial-up connection, to gain unauthorized access to the target system. An authorized user of the system may also use this exploit for privilege escalation.

The attack exploits a bug in a program. The bug can be a simple case of poor programming, in which the programmer neglected to code bounds checking on an input field. In this case, the attacker sends more data than the program was expecting. By using trial and error, or by examining the source code of the attacked program if it is available, the attacker determines the vulnerability and writes a program to do the following:

- Overflow an input field, command-line argument, or input buffer—for example, on a network daemon—until it writes into the stack.

- Overwrite the current return address on the stack with the address of the exploit code loaded in step 3
- Write a simple set of code for the next space in the stack that includes the commands that the attacker wishes to execute—for instance, spawn a shell.

The result of this attack program's execution will be a root shell or other privileged command execution.

For instance, if a web-page form expects a user name to be entered into a field, the attacker could send the user name, plus extra characters to overflow the buffer and reach the stack, plus a new return address to load onto the stack, plus the code the attacker wants to run. When the buffer-reading subroutine returns from execution, the return address is the exploit code, and the code is run.

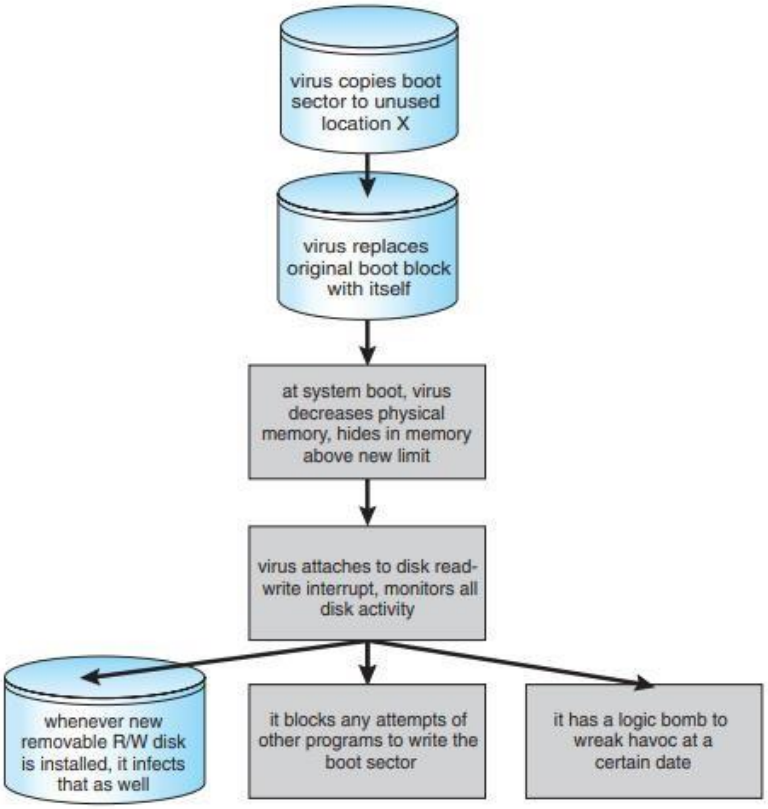
5. Viruses

Another form of program threat is a virus. A virus is a fragment of code embedded in a legitimate program. Viruses are self-replicating and are designed to ―infect‖ other programs. They can wreak havoc in a system by modifying or destroying files and causing system crashes and program malfunctions. As with most penetration attacks, viruses are very specific to architectures, operating systems, and applications. Viruses are a particular problem for users of PCs. UNIX and other multiuser operating systems generally are not susceptible to viruses because the executable programs are protected from writing by the operating system. Even if a virus does infect such a program, its powers usually are limited because other aspects of the system are protected. Viruses are usually borne via e-mail, with spam the most common vector. They can also spread when users download viral programs from Internet file-sharing services or exchange infected disks.

Once a virus reaches a target machine, a program known as a virus dropper inserts the virus into the system. The virus dropper is usually a Trojan horse, executed for other reasons but installing the virus as its core activity. Once installed, the virus may do any one of a number of things. There are literally thousands of viruses, but they fall into several main categories. Note that many viruses belong to more than one category.

- **File:** A standard file virus infects a system by appending itself to a file. It changes the start of the program so that execution jumps to its code. After it executes, it returns control to the program so that its execution is not noticed. File viruses are sometimes known as parasitic viruses, as they leave no full files behind and leave the host program still functional.

- **Boot:** A boot virus infects the boot sector of the system, executing every time the system is booted and before the operating system is loaded. It watches for other bootable media and infects them. These viruses are also known as memory viruses, because they do not appear in the file system.
- **Macro:** Most viruses are written in a low-level language, such as assembly or C. Macro viruses are written in a high-level language, such as Visual Basic. These viruses are triggered when a program capable of executing the macro is run. For example, a macro virus could be contained in a spreadsheet file.
- **Source code:** A source code virus looks for source code and modifies it to include the virus and to help spread the virus.



- **Polymorphic:** A polymorphic virus changes each time it is installed to avoid detection by antivirus software. The changes do not affect the virus's functionality but rather change the virus's signature. A virus signature is a pattern that can be used to identify a virus, typically a series of bytes that make up the virus code.
- **Encrypted:** An encrypted virus includes decryption code along with the encrypted virus, again to avoid detection. The virus first decrypts and then executes.
- **Stealth:** This tricky virus attempts to avoid detection by modifying parts of the system that could be used to detect it. For example, it could modify the read system call so that if the file it has modified is read, the original form of the code is returned rather than the infected code

- **Tunneling:** This virus attempts to bypass detection by an antivirus scanner by installing itself in the interrupt-handler chain. Similar viruses install themselves in device drivers.
- **Multipartite:** A virus of this type is able to infect multiple parts of a system, including boot sectors, memory, and files. This makes it difficult to detect and contain.
- **Armored:** An armored virus is coded to make it hard for antivirus researchers to unravel and understand. It can also be compressed to avoid detection and disinfection. In addition, virus droppers and other full files that are part of a virus infestation are frequently hidden via file attributes or unviewable file names.

System and Network Threats

Program threats typically use a breakdown in the protection mechanisms of a system to attack programs. In contrast, system and network threats involve the abuse of services and network connections. System and network threats create a situation in which operating-system resources and user files are misused. Sometimes, a system and network attack is used to launch a program attack, and vice versa.

The more open an operating system is—the more services it has enabled and the more functions it allows—the more likely it is that a bug is available to exploit. Increasingly, operating systems strive to be secure by default. For example, Solaris 10 moved from a model in which many services (FTP, telnet, and others) were enabled by default when the system was installed to a model in which almost all services are disabled at installation time and must specifically be enabled by system administrators. Such changes reduce the system's attack surface—the set of ways in which an attacker can try to break into the system.

It is important to note that **masquerading** and **replay attacks** are also commonly launched over networks between systems. In fact, these attacks are more effective and harder to counter when multiple systems are involved.

1. Worms

A worm is a process that uses the spawn mechanism to duplicate itself. The worm spawns copies of itself, using up system resources and perhaps locking out all other processes. On computer networks, worms are particularly potent, since they may reproduce themselves among systems and thus shut down an entire network. Such an event occurred in 1988 to UNIX systems on the Internet, causing the loss of system and system-administrator time worth millions of dollars.

At the close of the workday on November 2, 1988, Robert Tappan Morris, Jr., a first-year Cornell graduate student, unleashed a worm program on one or more hosts connected to the Internet. Targeting Sun Microsystems' Sun 3 workstations and VAX computers running variants of

Version 4 BSD UNIX, the worm quickly spread over great distances. Within a few hours of its release, it had consumed system resources to the point of bringing down the infected machines.

Although Morris designed the self-replicating program for rapid reproduction and distribution, some of the features of the UNIX networking environment provided the means to propagate the worm throughout the system. It is likely that Morris chose for initial infection an Internet host left open for and accessible to outside users. From there, the worm program exploited flaws in the UNIX operating system's security routines and took advantage of UNIX utilities that simplify resource sharing in local-area networks to gain unauthorized access to thousands of other connected sites. Morris's methods of attack are outlined next.

The worm was made up of two programs, a grappling hook (also called a bootstrap or vector) program and the main program. Named l1.c, the grappling hook consisted of 99 lines of C code compiled and run on each machine it accessed. Once established on the computer system under attack, the grappling hook connected to the machine where it originated and uploaded a copy of the main worm onto the hooked system (Figure 15.6). The main program proceeded to search for other machines to which the newly infected system could connect easily. In these actions, Morris exploited the UNIX networking utility rsh for easy remote task execution. By setting up special files that list host–login name pairs, users can omit entering a password each time

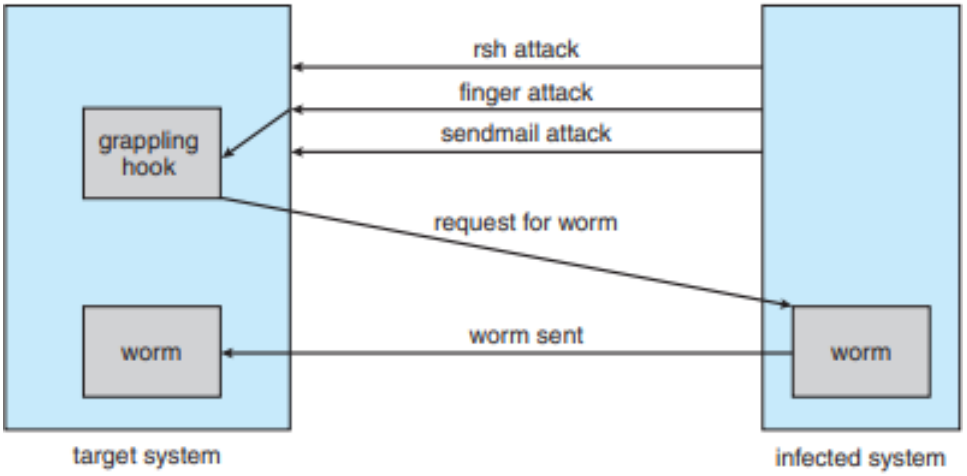


Figure 15.6 The Morris Internet worm.

they access a remote account on the paired list. The worm searched these special files for site names that would allow remote execution without a password. Where remote shells were established, the worm program was uploaded and began executing anew. The attack via remote access was one of three infection methods built into the worm. The other two methods involved operating-system bugs in the UNIX finger and sendmail programs.

A worm is a program that can replicate itself and send copies from computer to computer across network connections. Upon arrival, the worm may be activated to replicate and propagate again. In addition to propagation, the worm usually performs some unwanted function. An e-mail virus has some of the characteristics of a worm because it propagates itself from system to system. However, we can still classify it as a virus because it uses a document modified to contain viral macro content and requires human action. A worm actively seeks out more machines to infect, and each machine that is infected serves as an automated launching pad for attacks on other machines.

Network worm programs use network connections to spread from system to system. Once active within a system, a network worm can behave as a computer virus or bacteria, or it could implant Trojan horse programs or perform any number of disruptive or destructive actions.

To replicate itself, a network worm uses some sort of network vehicle. Examples include the following:

- **Electronic mail facility:** A worm mails a copy of itself to other systems, so that its code is run when the e-mail or an attachment is received or viewed.
- **Remote execution capability:** A worm executes a copy of itself on another system, either using an explicit remote execution facility or by exploiting a program flaw in a network service to subvert its operations (such as buffer overflow, described in Chapter 7).
- **Remote login capability:** A worm logs on to a remote system as a user and then uses commands to copy itself from one system to the other, where it then executes.

The new copy of the worm program is then run on the remote system where, in addition to any functions that it performs at that system, it continues to spread in the same fashion.

A network worm exhibits the same characteristics as a computer virus: a dormant phase, a propagation phase, a triggering phase, and an execution phase. The propagation phase generally performs the following functions:

1. Search for other systems to infect by examining host tables or similar repositories of remote system addresses.
2. Establish a connection with a remote system.
3. Copy itself to the remote system and cause the copy to be run.

The network worm may also attempt to determine whether a system has previously been infected before copying itself to the system. In a multiprogramming system, it may also disguise its presence by naming itself as a system process or using some other name that may not be noticed by a system operator.

As with viruses, network worms are difficult to counter.

2. Port Scanning

Port scanning is not an attack but rather a means for a cracker to detect a system's vulnerabilities to attack. Port scanning typically is automated, involving a tool that attempts to create a TCP/IP connection to a specific port or a range of ports. For example, suppose there is a known vulnerability (or bug) in sendmail. A cracker could launch a port scanner to try to connect, say, to port 25 of a particular system or to a range of systems. If the connection was successful, the cracker (or tool) could attempt to communicate with the answering service to determine if the service was indeed sendmail and, if so, if it was the version with the bug.

Now imagine a tool in which each bug of every service of every operating system was encoded. The tool could attempt to connect to every port of one or more systems. For every service that answered, it could try to use each known bug.

Frequently, the bugs are buffer overflows, allowing the creation of a privileged command shell on the system. From there, of course, the cracker could install Trojan horses, back-door programs, and so on.

There is no such tool, but there are tools that perform subsets of that functionality. For example, nmap (from http://www.insecure.org/nmap/) is a very versatile open-source utility for network exploration and security auditing. When pointed at a target, it will determine what services are running, including application names and versions. It can identify the host operating system. It can also provide information about defenses, such as what firewalls are defending the target. It does not exploit any known bugs.

Because port scans are detectable they frequently are launched from **zombie systems**. Such systems are previously compromised, independent systems that are serving their owners while being used for nefarious purposes, including denial-of-service attacks and spam relay. Zombies make crackers particularly difficult to prosecute because determining the source of the attack and the person that launched it is challenging. This is one of many reasons for securing –inconsequential‖ systems, not just systems containing –valuable‖ information or services.

3. Denial of Service

As mentioned earlier, denial-of-service attacks are aimed not at gaining information or stealing resources but rather at disrupting legitimate use of a system or facility. Most such attacks involve systems that the attacker has not penetrated. Launching an attack that prevents legitimate use is frequently easier than breaking into a machine or facility.

Denial-of-service attacks are generally network based. They fall into two categories. Attacks in the first category use so many facility resources that, in essence, no useful work can be done. For

example, a website click could download a Java applet that proceeds to use all available CPU time or to pop up windows infinitely. The second category involves disrupting the network of the facility. There have been several successful denial-of-service attacks of this kind against major websites. These attacks result from abuse of some of the fundamental functionality of TCP/IP.

For instance, if the attacker sends the part of the protocol that says –I want to start a TCP connection,‖ but never follows with the standard –The connection is now complete,‖ the result can be partially started TCP sessions. If enough of these sessions are launched, they can eat up all the network resources of the system, disabling any further legitimate TCP connections. Such attacks, which can last hours or days, have caused partial or full failure of attempts to use the target facility.

It is impossible to prevent denial-of-service attacks. The attacks use the same mechanisms as normal operation. Even more difficult to prevent and resolve are distributed denial-of-service (DDOS) attacks. These attacks are launched from multiple sites at once, toward a common target, typically by zombies. DDOS attacks have become more common and are sometimes associated with blackmail attempts. A site comes under attack, and the attackers offer to halt the attack in exchange for money. Sometimes a site does not even know it is under attack. It can be difficult to determine whether a system slowdown is an attack or just a surge in system use. Consider that a successful advertising campaign that greatly increases traffic to a site could be considered a DDOS.

There are other interesting aspects of DOS attacks. For example, if an authentication algorithm locks an account for a period of time after several incorrect attempts to access the account, then an attacker could cause all authentication to be blocked by purposely making incorrect attempts to access all accounts. Similarly, a firewall that automatically blocks certain kinds of traffic could be induced to block that traffic when it should not. These examples suggest that programmers and systems managers need to fully understand the algorithms and technologies they are deploying. Finally, computer science classes are notorious sources of accidental system DOS attacks. Consider the first programming exercises in which students learn to create subprocesses or threads. A common bug involves spawning subprocesses infinitely. The system’s free memory and CPU resources don’t stand a chance.

Cryptography as a Security Tool

It is generally considered infeasible to build a network of any scale in which the source and destination addresses of packets can be trusted in this sense. Therefore, the only alternative is somehow to eliminate the need to trust the network. This is the job of cryptography. Abstractly, cryptography is used to constrain the potential senders and/or receivers of a message.

Modern cryptography is based on secrets called **keys** that are selectively distributed to computers in a network and used to process messages. Cryptography enables a recipient of a message to verify that the message was created by some computer possessing a certain key. Similarly, a sender can encode its message so that only a computer with a certain key can decode the message.

1. Encryption

Encryption is used frequently in many aspects of modern computing. It is used to send messages securely across a network, as well as to protect database data, files, and even entire disks from having their contents read by unauthorized entities. An encryption algorithm enables the sender of a message to ensure that only a computer possessing a certain key can read the message, or ensure that the writer of data is the only reader of that data.

An encryption algorithm consists of the following components:

- A set K of keys.
- A set M of messages
- A set C of ciphertexts
- An encrypting function $E : K \rightarrow (M \rightarrow C)$. That is, for each $k \in K$, E_k is a function for generating ciphertexts from messages. Both E and E_k for any k should be efficiently computable functions. Generally, E_k is a randomized mapping from messages to ciphertexts.
- A decrypting function $D : K \rightarrow (C \rightarrow M)$. That is, for each $k \in K$, Dk is a function for generating messages from ciphertexts. Both D and Dk for any k should be efficiently computable functions.

An encryption algorithm must provide this essential property: given a ciphertext $c \in C$, a computer can compute m such that Ek (m) = c only if it possesses k. Thus, a computer holding k can decrypt ciphertexts to the plaintexts used to produce them, but a computer not holding k cannot decrypt ciphertexts. Since ciphertexts are generally exposed (for example, sent on a network), it is important that it be infeasible to derive k from the ciphertexts. There are two main types of encryption algorithms: symmetric and asymmetric.

Symmetric Encryption

In a symmetric encryption algorithm, the same key is used to encrypt and to decrypt. Therefore, the secrecy of k must be protected. Figure 15.7 shows an example of two users communicating securely via symmetric encryption over an insecure channel. Note that the key exchange can take place directly between the two parties or via a trusted third party (that is, a certificate authority)

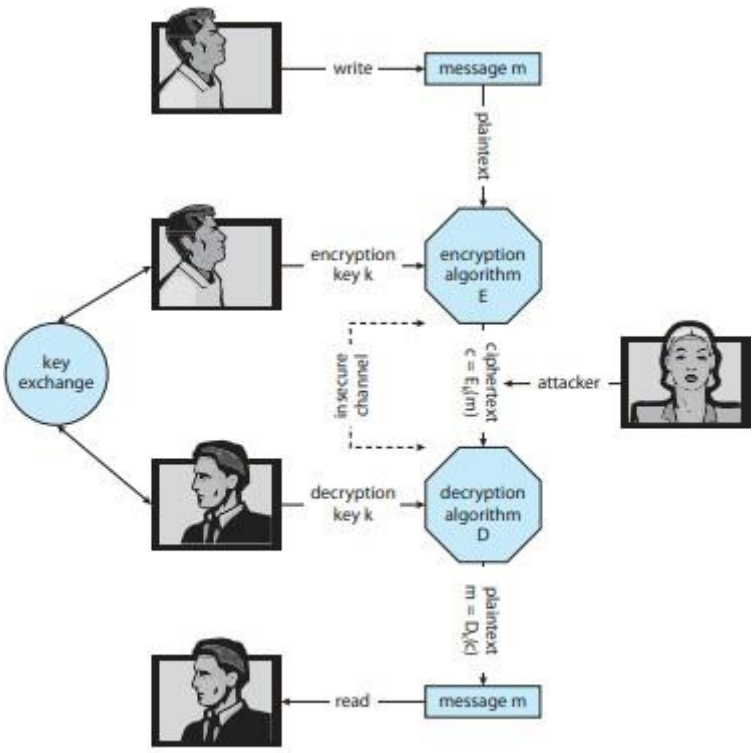


Figure 15.7 A secure communication over an insecure medium.

Asymmetric Encryption

In an asymmetric encryption algorithm, there are different encryption and decryption keys. An entity preparing to receive encrypted communication creates two keys and makes one of them (called the public key) available to anyone who wants it. Any sender can use that key to encrypt a communication, but only the key creator can decrypt the communication. This scheme, known as public-key encryption, was a breakthrough in cryptography. No longer must a key be kept secret and delivered securely. Instead, anyone can encrypt a message to the receiving entity, and no matter who else is listening, only that entity can decrypt the message.

User Authentication

User authentication process is used just to identify who the owner is or who the identified person is.

In personal computer, generally, user authentication can be perform using password.

When a computer user wants to log into a computer system, then the installed operating system (OS) on that computer system generally wants to determine or check who the user is. This process is called as user authentication.

Sometime it is too important to authenticate the user because the computer system may have some important documents of the owner.

Most methods of authenticating the computer users when they attempt or try to log into the system are based on one of the following three principles:

- Something, the user knowns
- Something, the user has
- Something, the user is

That computer users who want to cause some trouble on any specific computer system, have to first log into that computer systems, means getting past whichever the authentication method or procedure is used. Those computer users are called as hackers.

Basically, hacker is a term of honour that is reserved for or given to a great computer programmer as normal computer user or programmer can't get access into anyone's system without permission.

User can be authenticated through one of the following way:

- User authentication using password
- User authentication using physical object
- User authentication using biometric
- User authentication using countermeasures

Now let's describe briefly about all the above authentication process one by one.

User Authentication using Password

- User authentication using password is the most widely used form of authenticating the user.
- In this method of authenticating the user with password, it is to require that the user who is going to authenticate has to type their login name or id and login password.
- Authenticating the user using their password is an easy method and also easy to implement.
- Keeping a central list of pairs is the simplest implementation of user authentication using password method.
- Here, in this method, the login name typed in is looked up in the list and typed password is then compared to stored password.
- Now, if both login and password match, then the login is allowed or the user is successfully authenticated and approved to log into that system. And in case if now match occurred, then the login error is detected.

How to Improve Password Security?

Here's the list of four basic and common way to secure the password:

- Password should be minimum of eight characters
- Password should contain both uppercase and lowercase letters
- Password should contain at least one digit and one special characters
- Don't use dictionary words and known name such as stick, mouth, sun, albert etc.

What is One Time Password (OTP) ?

- One Time Password (OTP) is the most extreme form of changing the password all the time.
- One time password is a very safe way to implement.
- When OTPs are used, the user get a book containing a list of many passwords. Each login uses the next password in the list.
- Therefore, if an intruder ever discover the password, then it will not do any good for him as the next time, a different password must be used.

User Authentication using Biometric

User authentication using biometric is the third authentication method here.

This method measures the physical characteristics of the user that are very hard to forge. These are called as biometrics.

User authentication using biometric's example is a fingerprint, voiceprint, or retina scan reader in the terminal could verify the identity of the user.

Basically, the typical biometric system has the following two parts:

- Enrolment
- Identification

Now, let's describe briefly about the above two parts of the biometric system.

Enrolment

In biometric system, during enrolment, characteristics of the user are measured and the results digitized.

Then, significant features are extracted and stored in the record associated with the user.

The record can be kept or stored in a central or main database or stored on a smart card that the user carrier around and inserts into a remote reader, for example, at an ATM machine.

Identification

In identification, the user shows up and provides a login name or id. Now, again, the system makes the measurement.

Now, if the new values match the ones sampled at enrolment time, then the login is accepted, otherwise the login attempt is rejected.

User Authentication vs User Authorization

Authentication is about validating your credentials like User Name/User ID and password to verify your identity. The system determines whether you are what you say you are using your credentials. In public and private networks, the system authenticates the user identity via login passwords. Authentication is usually done by a username and password, and sometimes in conjunction with factors of authentication, which refers to the various ways to be authenticated.

Authentication factors determine the various elements the system use to verify one’s identity prior to granting him access to anything from accessing a file to requesting a bank transaction. A user’s identity

can be determined by what he knows, what he has, or what he is. When it comes to security, at least two or all the three authentication factors must be verified in order to grant someone access to the system.

Based on the security level, authentication factor can vary from one of the following:

- **Single-Factor Authentication** – It's the simplest authentication method which commonly relies on a simple password to grant user access to a particular system such as a website or a network. The person can request access to the system using only one of the credentials to verify his identity. The most common example of a single-factor authentication would be login credentials which only require a password against a username.
- **Two-Factor Authentication** – As the name suggests, it's a two-step verification process which not only requires a username and password, but also something only the user knows, to ensure an additional level of security, such as an ATM pin, which only the user knows. Using a username and password along with an additional piece of confidential information makes it virtually impossible for fraudsters to steal valuable data.
- **Multi-Factor Authentication** – It's the most advanced method of authentication which uses two or more levels of security from independent categories of authentication to grant user access to the system. All the factors should be independent of each other to eliminate any vulnerability in the system. Financial organizations, banks, and law enforcement agencies use multiple-factor authentication to safeguard their data and applications from potential threats.

For example, when you enter your ATM card into the ATM machine, the machine asks you to enter your pin. After you enter the pin correctly, the bank then confirms your identity that the card really belongs to you and you're the rightful owner of the card. By validating your ATM card pin, the bank actually verifies your identity, which is called authentication. It merely identifies who you are, nothing else.

Authorization

Authorization, on the other hand, occurs after your identity is successfully authenticated by the system, which ultimately gives you full permission to access the resources such as information, files, databases, funds, locations, almost anything. In simple terms, authorization determines your ability to access the system and up to what extent. Once your identity is verified by the system after successful authentication, you are then authorized to access the resources of the system.

Authorization is the process to determine whether the authenticated user has access to the particular resources. It verifies your rights to grant you access to resources such as information, databases, files, etc.

Authorization usually comes after authentication which confirms your privileges to perform. In simple terms, it's like giving someone official permission to do something or anything.

For example, the process of verifying and confirming employees ID and passwords in an organization is called authentication, but determining which employee has access to which floor is called authorization. Let's say you are traveling and you're about to board a flight. When you show your ticket and some identification before checking in, you receive a boarding pass which confirms that the airport authority has authenticated your identity. But that's not it. A flight attendant must authorize you to board the flight you're supposed to be flying on, allowing you access to the inside of the plane and its resources.

Access to a system is protected by both authentication and authorization. Any attempt to access thesystem might be authenticated by entering valid credentials, but it can only be accepted after successful authorization. If the attempt is authenticated but not authorized, the system will deny access to the system.

Sno	User Authentication	User Authorization
1	Authentication confirms your identity to grant access to the system.	Authorization determines whether you are authorized to access the resources.
2	It is the process of validating user credentials to gain user access.	It is the process of verifying whether access is allowed or not.
3	It determines whether user is what he claims to be.	It determines what user can and cannot access.
4	Authentication usually requires a username and a password.	Authentication factors required for authorization may vary, depending on the security level.
5	Authentication is the first step of authorization so always comes first.	Authorization is done after successful authentication.
6	For example, students of a particular university are required to authenticate themselves before accessing the student link of the university's official website. This is called authentication.	For example, authorization determines exactly what information the students are authorized to access on the university website after successful authentication.