UNIT 5 PROTECTION AND SECURITY

PROTECTION

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. Modern protection concepts have evolved to increase the reliability of any complex system that makes use of shared resources.

Protection can improve reliability by detecting latent errors at the interfaces between component subsystems. Early detection of interface errors can often prevent contamination of a healthy subsystem by a malfunctioning subsystem. An unprotected resource cannot defend against use (or misuse) by an unauthorized or incompetent user. 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. Still others are defined by the individual users to protect their own files and programs. A protection system must have the flexibility to enforce a variety of policies.

PRINCIPLES OF PROTECTION:

A key, time-tested guiding principle for protection is the principle of least privilege. It indicates that programs, users, and even systems be given just enough privileges to perform their tasks.

If this key allows the guard into just the public areas that she guards, then misuse of the key will result in minimal damage. 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. Such anoperating 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 backup 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). Each object has a unique name that differentiates it from all other objects in the system, and each can be accessed only through well- defined and meaningful operations. Objects are essentially abstract data types.

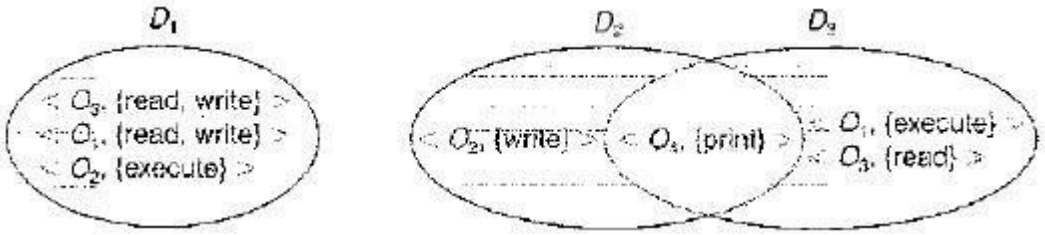
The operations that are possible may depend on the object.

Memory segments can be read and written, whereas a CD-ROM or DVD-ROM can only be read.Tape drives can be read, written, and rewound. Data files can be created, opened, read, written, closed, and deleted; program files can be read, written, executed, and deleted. A process should be allowed to access only those resources for which it has authorization. This second requirement, commonly referred to as the *need-to-knozv* principle, is useful in limiting theamount of damage a faulty process can cause in the system.
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 variablesof process *p*. Similarly, consider the case where process *p* invokes a compiler to compile a particular file. The compiler should not be able to access files arbitrarily but should have access only to a well-defined subset of files related to the file to be compiled.

Domain Structure

To facilitate this scheme, 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 objectis an **access right.** A domain is a collection of access rights, each of which is an ordered pair
<*object-iiame, rights-set*>.Domains do not need to be disjoint; they may share access rights. We have three domains: *Dir D2,* and D3. The access right < *Oi,* (print}> is shared by D? and *D3,*implying that a process executing in either of these two domains can print object O4. Note that a process must be executing in domain *D*\ to read and write object *O*\, while only processes in domain D3 may execute object *O.*
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 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:

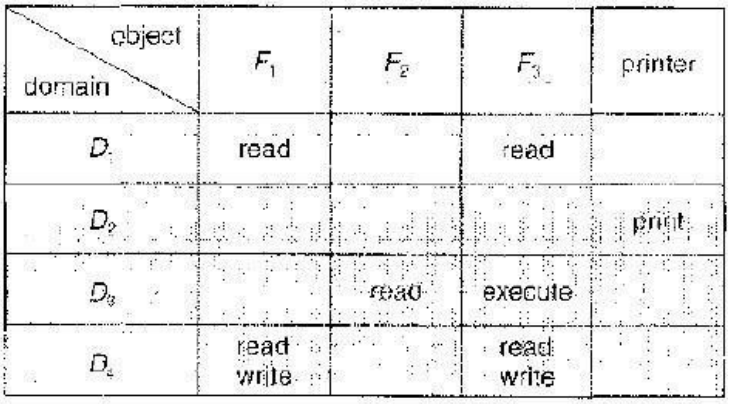
» Each *user* may be a domain. In this case, the set of objects that can be accessed depends on the identity of the user. Domain switching occurs when the user is changed—generally when one user logs out and another user logs in.

• 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 aprocedure call is made.

ACCESS MATRIX

Our 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(/,/) defines the set of operations thata process executing in domain *Dj* can invoke on object *Or* There



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	

are four domains and four objects—three files (F|, F2, F:1) and one laser printer. A process executing in domain *D* can read files Fj and F3. A process executing in domain D4 has the same privileges as one executing in domain *D\;* but in addition, it can also write onto files F| and F3

The access-matrix scheme provides us with the mechanism for specifying a variety of policies. The mechanism consists of implementing the access matrix and ensuring that the semantic properties we have outlined indeed, hold. More specifically, we must ensur e that a process executing in domain *D,* can access only those objects specified in row \, and then only as allowedby the access-matrix entries.

The access matrix can implement policy decisions concerning protection. The policy decisions involve which rights should be included in the (z',;') the entry. We must also decide the domainin 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 the static and dynamic association between processes and domains. Then we switch a process from one domain to another, we are executing an operation (switch) on anobject (the domain). We can control domain switching by including domains among the objects of the access matrix.

domain \ object	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			

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

(a)

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

(b)

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 *copy* right allows the copying of the access right only within the column (that is, for the object) for which the right is defined.

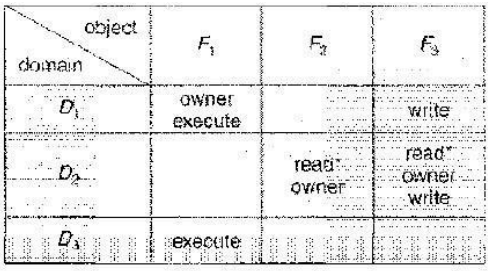
This scheme has two variants:

1. A right is copied from access(/, /) to access(/c,/); it is then removed from access(/,/). This action is a *transfer* of a right, rather than a copy.

2. Propagation of the *copy* right may be limited. That is, when the right *R** is copied from

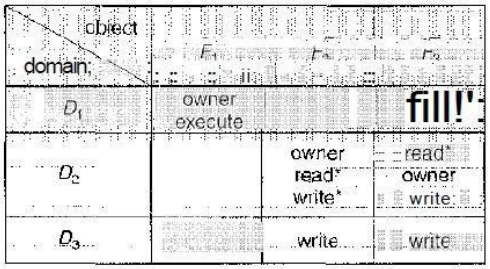
access(/,y) to access(/t,/), only the right *R* (not *R")* is created. A process executing in domain *Dk* cannot further copy the right *R.*

A system may select only one of these three *copy* rights, or it may provide all three by identifying them as separate rights: *copy*, *transfer*, and *limited copy.* We also need a mechanism to allow addition of new rights and removal of some rights. The *owner* right controls these operations. If access(/,/) includes the *oivncr* right, then a process executing in domain D, can add and remove any right in any entry in column /'.



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

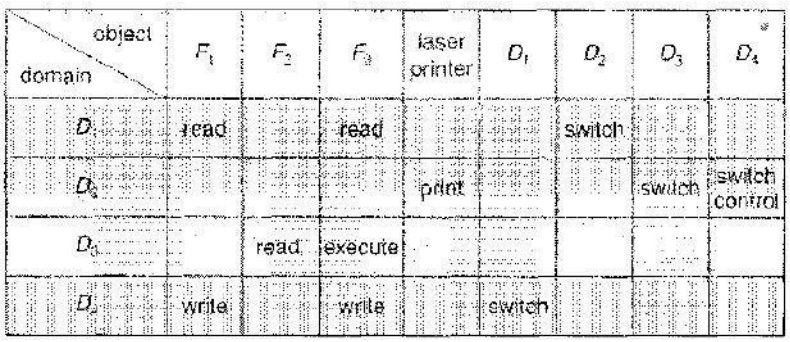
(a)



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

(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(/,/) includes the *control* right, then a process executing in domain *D.* can remove any access right from row /'.



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			

IMPLEMENTATION OF ACCESS MATRIX

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 O,within domain *D-,,* the global table is searched for a triple <*D,,* O;, *Rk*>, with *M* e R/:. 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. Virtual memory techniques are often used for managing this table.

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 <*rfomnin, 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 0/ is attempted in domain *Dj,* we search the access list for object O., looking for an entry <*D,,* R; > with M e Kj. 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.

Capability Lists for Domains

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 *0,,* the process executes the operation *M,* specifying the capability (or pointer) for object O/ as a parameter.

Simple possession of the capability means that access is allowed. The capability list is associatedwith 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 bythe 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 theycould be modified). If all capabilities are secure, the object they protect is also secure against unauthorized access.

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

Each object has a tag to denote its type either as a capability or as 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 1 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 segmented memory space (Section 8.6) is useful to support this approach

ACCESS CONTROL:

Each file and directory are assigned an owner, a group, or possibly a list of users, and for each ofthose entities, access-control information is assigned. A similar function can be added to other aspects of a computer system.

Solaris 10 advances the protection available in the Sun Microsystems 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

Revocation of Access Rights

Rights to objects shared by different users. Various questions about revocationmay arise:

• Immediate versus delayed. Does revocation occur immediately/ or is it delayed? If revocation isdelayed, 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? Capabilities, however, present a much more difficult revocation problem. 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.

Capability-Based Systems

An Example: Hydra

Hydra is a capability-based protection system that provides considerable flexibility. A fixed setof possible access rights is known to and interpreted by the system. These rights include such basic forms of access as the right to read, write, or execute a memory segment. In addition, a user(of the protection system) can declare other rights.

Operations on objects are defined procedurally. The procedures that implement such operations are themselves a form of object, and they are accessed indirectly by capabilities. The names of user-defined procedures must be identified to the protection system if it is to deal with objects of the user defined type. When the definition of an object is made known to Hydra, the names of operations on the type become auxiliary rights.

Hydra also provides rights amplification. This scheme allows a procedure to be certified as *trustworthy* to act on a formal parameter of a specified type on behalf of any process that holds a right to execute the procedure.

When a process invokes the operation *P* on an object *A,* however, the capability for access to *A* may be amplified as control passes to the code body of *P.* This amplification may be necessary to allow P the right to access the storage segment representing *A* so as to implement the operation that P defines on the abstract data type. The code body of P may be allowed to read or to write to the segment of *A* directly, even though the calling process cannot. On return from P, thecapability for *A* is restored to its original, unamplified state. This case is a typical one in which the rights held by a process for access to a protected segment must change dynamically, depending on the task to be performed. The dynamic adjustment of rights is performed to guarantee consistency of a programmer-defined abstraction. Amplification of rights can be statedexplicitly in the declaration of an abstract type to the Hydra operating system.

LANGUAGE-BASED PROTECTION

To the degree that protection is provided in existing computer systems, it is usually achieved through an operating-system kernel, which acts as a security agent to inspect and validate each attempt to access a protected resource.

we must give it hardware support to reduce the cost of each validation or we must accept that thesystem designer may compromise the goals of protection. Satisfying all these goals is difficult if the flexibility to implement protection policies is restricted by the support mechanisms provided or if protection environments are made larger than necessary to

secure greater operationalefficiency.

As operating systems have become more complex, and particularly as they have attempted to provide higher-level user interfaces, the goals of protection have become much more refined.The designers of protection systems have drawn heavily on ideas that originated in programminglanguages and especially on the concepts of abstract data types and objects. Protection systems are now concerned not only with the identity of a resource to which access is attempted but also with the functional nature of that access, in the newest protection systems, concern for the function to be invoked extends beyond a set of system-defined functions, such as standard file- access methods, to include functions that may be user-defined as well.

COMPILER-BASED ENFORCEMENT

Specifying the desired control of access to a shared resource in a system is making a declarative statement about the resource. This kind of statement can be integrated into a language by an extension of its typing facility. When protection is declared along with data typing, the designer of each subsystem can specify its requirements for protection, as well as its need for use of other resources in a system. Such a specification should be given directly as a program is composed, and in the language in which the program itself is stated. This approach has several significant advantages:

- Protection needs are simply declared, rather than programmed as a sequence of calls on procedures of an, operating system.
- Protection requirements can be stated independently of the facilities provided by a particular operating system.
- The means for enforcement need not be provided by the designer of a subsystem.
- A declarative notation is natural because access privileges are closely related to the linguistic concept of data type. A variety of techniques can be provided by a programming-language implementation to enforce protection, but any of these must depend on some degree of support from an underlying machine and its operating system.

SECURITY

THE SECURITY PROBLEM

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 forms of accidental and malicious security violations. We should note that in our discussion of security, we vise the terms *intruder* and *cracker* for those attempting to breach security. In addition, a **threat** is the potential for a security violation, stich as the discovery of a vulnerability, whereas an **attack** is the attempt to break secvirity.

• **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.

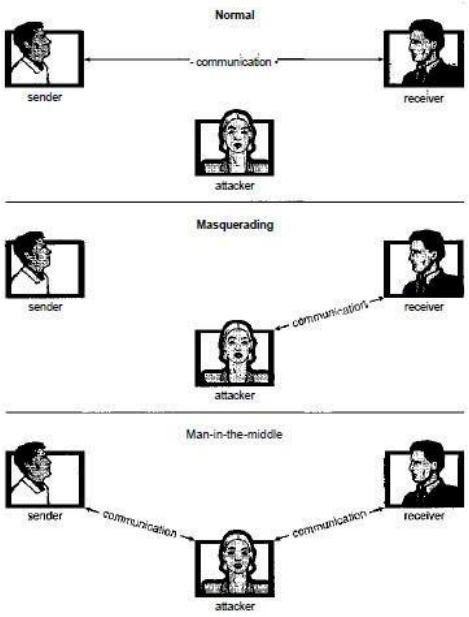
• **Breach of integrity.** This violation involves unauthorized modification of data. Such attacks can, for example, result in passing of liability to an innocent party or modification of the source code of an important commercial application.

• **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. Web-site 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,** or **DOS,** attacks are sometimes accidental. The original Internet worm turned into aDOS attack when a bug failed to delay its rapid spread.

As we have already suggested, absolute protection of the system from malicious abuse is not possible, but the cost to the perpetrator can be made sufficiently high to deter most intruders. In some cases, such as a denial-of service attack, it is preferable to prevent the attack but sufficient to detect the attack so that countermeasures can be taken.



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

1. Physical. The site or sites containing the computer systems must be physically secured againstarmed or surreptitious entry by intruders. Both the machine rooms and the terminals or workstations that have access to the machines must be secured.

2. Human. Authorizing users 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.

3. 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 aservice could reveal passwords. A stack overflow could allow the launching of an unauthorized process. The list of possible breaches is almost endless.

4. 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 justas 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.

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 behavior and create a breach, is a common goal of crackers.

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.** For instance, consider the use of the "." character in a search path. The "." tells the shell to include the current directory in the search. Thus, if a userhas "." in her search path, has set her current directory to a friend's directory, and enters the name of a normal system command, the command may be executed from the friend's directory instead.The program would run within the user's domain, allowing the program to do anything that the user is allowed to do, including deleting the user's files, for instance.

A variation of the Trojan horse is a program that emulates a login program. An unsuspecting userstarts to log in at a terminal and notices that he has apparently mistyped his password. He tries again and is successful. 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 attackcan be defeated by having the operating system print a usage message at the end of an interactivesession or by a non-trappable key sequence, such as the control - alt - delete combination used byall modern Windows operating systems.

Trap Door

The designer of a program or system might leave a hole in the software that only she is capableof 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 codebeing compiled.

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 were met, the security hole would be created. This scenario is known as a logic bomb. A programmer, for example, might write code to detect if she is still employed; if that check failed, a daemon could be spawned to allow remote access, or code couldbe launched to cause damage to the site.

Viruses

Another form of program threat is a **virus.** 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. A virus is a fragment of code embedded in a legitimate program. 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 andother 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.

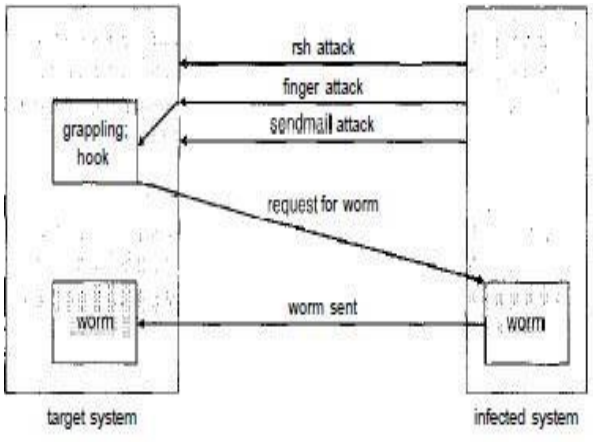
System and Network Threats

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. Sometimes a system and network attack is used to launch a program attack, and vice versa.

System and network threats create a situation in which operating-system resources and user files are misused. Here, we discuss some examples of these threats, including worms, port scanning, and denial-of-service attacks.

Worms

A **worm** is a process that uses the **spawn** mechanism to ravage system performance. 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 millions of dollars of lost system and system administrator time.



`

The worm was made up of two programs, a grappling hook (also called a bootstrap or

vector) program and the main program. Named *11.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

Port Scanning

Port scanning is not an attack but rather is 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.

Denial of Service

DOS attacks are aimed not at gaining information or stealing resources but rather at disrupting legitimate use of a system or facility. Most denial-of-service attacks involve systems that the attacker has not penetrated. Indeed, 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. The first case is an attack that uses so many facility resources that, in essence, no useful work can be done. For example, a web-site click could download a Java applet that proceeds to vise all available CPU time or to infinitely pop up windows. The second case involves disrupting the network of the facility. There have been several successful denial-of-service attacks of this kind against major web sites. They result fromabuse of some of the fundamental functionality of TCP/IP.

CRYPTOGRAPHY AS A SECURITY TOOL

There are many defenses against computer attacks, running the gamut from methodology totechnology. The broadest tool available to system designers and users is cryptography

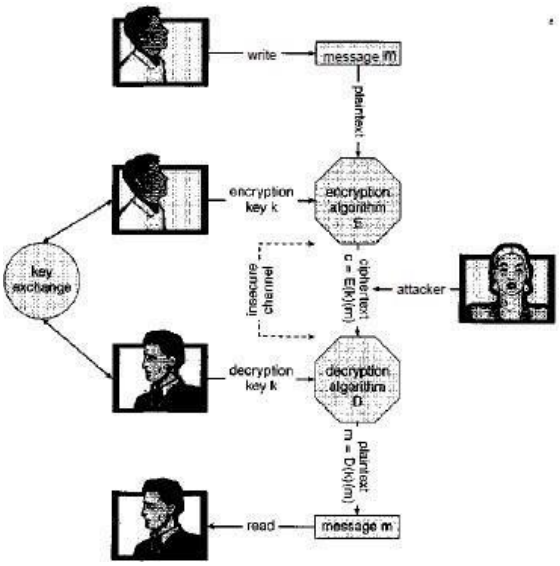
networked computer receives bits *fivm the wire* with no immediate and reliable way of determining what machine or application sent those bits. Similarly, the computer sends bits onto the network with no way of knowing who might eventually receive them.

Commonly, network addresses are used to infer the potential senders and receivers of network messages. Network packets arrive with a source address, such as an IP address. And when a computer sends a message, it names the intended receiver by specifying a destination address. However, for applications where security matters, we are asking for trouble if we assume that the source or destination address of a packet reliably determines who sent or received that packet.

Encryption

Encryption is a means for constraining the possible receivers of a message. An encryption algorithm enables the sender of a message to ensure that only a computer possessing a certainkey can read the message. Encryption of messages is an ancient practice, of course, and there have been many encryption algorithms,

dating back to before Caesar.



An encryption algorithm consists of the following components:

• A set *K* of keys.

• A set M of messages.

• A set C of ciphertexts.

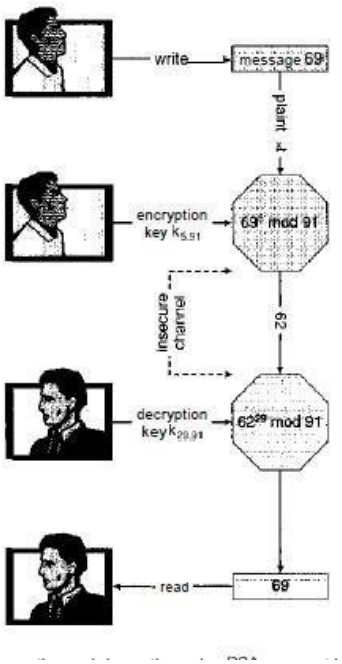
• A function *E : K* -> • *(M* - • C). That is, for each *k e K, E (k)* is a function for generating ciphertexts from messages. Both *E* and *E(k)* for any *k* should be efficiently computable functions.

• A function D : *K* -> (C -> *M).* That is, for each *k e K, D{k)* is a function for generating messages from cipher texts. Both D and *D(k)* for any *k* should be efficiently computable functions.

Authentication

We have seen that encryption offers a way of constraining the set of possible receivers of a message. Constraining the set of potential senders of a message is called **authentication.** Authentication is thus complementary to encryption. In fact, sometimes their functions overlap. Consider that an encrypted message can also prove the identity of the sender. For example, if

D{kd, N)(E(ke, N){m)) produces a valid message, then we know that the creator of the message must hold *kc*. Authentication is also useful for proving that a message has not been modified.



An authentication algorithm consists of the following components:

• A set *K* of keys.

• A set *M* of messages.

• A set A of authenticators.

• A function *S : K* —*• (M —>• *A).* That is, for each *k* e *K, S(k)* is a function for generating authenticators from messages. Both *S* and *S(k)* for any *k* should be efficiently computable functions.

A function *V* : X - > (M x y l - > {true, false}). That is, for each *k e K, V(k)* is a function for verifying authenticators on messages. Both *V* and *V(k)* for any *k* should be efficiently computable functions.

USER AUTHENTICATION

A major security problem for operating systems is **user**

authentication. The protection system depends on the ability to identify the programs and processes currently executing, which in turn depends on the ability to identify each user of the system.

Passwords

The most common approach to authenticating a user identity is the use of **passwords.** When the user identifies herself by user ID or account name, she is asked for a password. If the user-supplied password matches the password stored in the system, the system assumes that the account is being accessed by the owner of that account.

Passwords are often used to protect objects in the computer system, in the absence of more complete protection schemes. They can be considered a special case of either keys or capabilities.

Password Vulnerabilities

Passwords are extremely common because they are easy to understand and use. Unfortunately, passwords can often be guessed, accidentally exposed, sniffed, or illegally transferred from an authorized user to an unauthorized one, as we show next.

There are two common ways to guess a password. One way is for the intruder (either human or program) to know the user or to have information about the user.

The other way is to use brute force, trying enumeration—or all possible combinations of valid password characters (letters, numbers, and punctuation on some systems)—until the password is found.

Encrypted Passwords

One problem with all these approaches is the difficulty of keeping the password secret within the computer. How can the system

store a password securely yet allow its use for authentication when the user presents her password? The UNIX system uses encryption to avoid the necessity of keeping its password list secret. Each user has a password. The system contains a function that is extremely difficult—the designers hope impossible—to invert but is simple to compute. That is, given a value *x*, it is easy to compute the function value *f(x)*. Given a function value *f(x)*, however, it is impossible to compute *x** This function is used to encode all passwords. Only encoded passwords are stored. When a user presents a password, it is encoded and compared against the stored encoded password. Even if the stored encoded password is seen, it cannot be decoded, so the password cannot be determined. Thus, the password file does not need to be kept secret. The function/^) is typically an encryption algorithm that has been designed and tested rigorously.

One-Time Passwords

To avoid the problems of password sniffing and shoulder surfing, a system could use a set of **paired passwords.** When a session begins, the system randomly selects and presents one part of a password pair; the user must supply the other part. In this system, the user is **challenged** and must **respond** with the correct answer to that challenge.

This approach can be generalized to the use of an algorithm as a password. The algorithm might be an integer function, for example. The system selects a random integer and presents it to the user. The user applies the function and replies with the correct result. The system also applies the function. If the two results match, access is allowed.

In this **one-time password** system, the password is different in each instance. Anyone capturing the password from one session

and trying to reuse it in another session will fail. One-time passwords are among the only ways to prevent improper authentication due to password exposure.

One-time password systems are implemented in various ways. Commercial implementations, such as SecurlD, use hardware calculators. Most of these calculators are shaped like a credit card, a key-chain dangle, or a USB device; they include a display and may or may not also havea keypad. Some use the current time as the random seed. Others require that the user enters the shared secret, also known as a **personal identification number** or **PIN,** on the keypad. The display then shows the one-time password. The use of both a one-time password generator and a PIN is one form of **two-factor authentication.**

Two different types of components are needed in this case. Two-factor authentication offers far better authentication protection than single-factor authentication.

IMPLEMENTING SECURITY DEFENSES

The U.S. Department of Defense Trusted Computer System Evaluation Criteria specify four security classifications in systems: A, B, C, and D. This specification is widely used to determinethe security of a facility and to model security solutions, so we explore it here. The lowest-level classification is division D, or minimal protection. Division D includes only one class and is usedfor systems that have failed to meet the requirements of any of the other security classes.

For instance, MS-DOS and Windows 3.1 are in division D. Division C, the next level of security, provides discretionary protection and accountability of users and their actions through the use of audit capabilities. Division C has two levels: Cl and C2. A Cl-class system incorporates some form of controls that allow

users to protect private information and to keep other users from accidentally reading or destroying their data. A Cl environment is one in which cooperating users access data at the same levels of sensitivity. Most versions of UNIX are Cl class.

The sum total of all protection systems within a computer system (hardware, software, firmware) that correctly enforce a security policy is known as a **trusted computer base (TCB).** The TCB of a Cl system controls access between users and files by allowing the user to specify and control sharing of objects by named individuals or defined groups. In addition, the TCB requires that the users identify themselves before they start any activities that the TCB is expected to mediate. This identification is accomplished via a protected mechanism or password; the TCB protects the authentication data so that they are inaccessible to unauthorized users.

A C2-class system adds an individual-level access control to the requirements of a Cl system. For example, access rights of a file can be specified to the level of a single individual. In addition, the system administrator can selectively audit the actions of any one or more users based on individual identity. The TCB also protects itself from modification of its code or data structures.

FIREWALLING TO PROTECT SYSTEMS AND NETWORKS

We turn next to the question of how a trusted computer can be connected safely to an untrustworthy network. One solution is the use of a firewall to separate trusted and untrusted systems. A firewall is a computer, appliance, or router that sits between the trusted and the untrusted. A network firewall limits network access between the two security domains and monitors and logs all connections. It can also limit connections based on source or destination address, source or destination port, or direction of the

connection. For instance, web servers use HTTP to communicate with web browsers. A firewall therefore may allow only HTTP to pass from all hosts outside the firewall to the web server within the firewall. The Morris Internet worm used the f inger protocol to break into computers, so finger would not be allowed to pass, for example.

In fact, a network firewall can separate a network into multiple domains. A common implementation has the Internet as the untrusted domain; a semitrusted and semi-secure network, called the demilitarized zone (DMZ), as another domain; and a company's computers as a third domain (Figure 15.10). Connections are allowed from the Internet to the DMZ computers and from the company computers to the Internet but are not allowed from the Internet or DMZ computers to the company computers. Optionally, controlled communications may be allowed between the DMZ and one company computer or more. For instance, a web server on the DMZ may need to query a database server on the corporate network.

With a firewall, however, access is contained, and any DMZ systems that are broken into still are unable to access the company computers. Of course, a firewall itself must be secure and attack-proof; otherwise, its ability to secure connections can be compromised. Furthermore, firewalls do not prevent attacks that tunnel, or travel within protocols or connections that the firewall allows.

A buffer-overflow attack to a web server will not be stopped by the firewall, for example, because the HTTP connection is allowed; it is the contents of the HTTP connection that house the attack. Likewise, denial-ofservice attacks can affect firewalls as much as any other machines. Another vulnerability of firewalls is spoofing, in which an unauthorized host pretends to be an authorized host by meeting some authorization criterion. For example, if a firewall rule allows a connection from a host and identifies that host by its IP address, then another host could send packets using that same address and be allowed through the firewall.

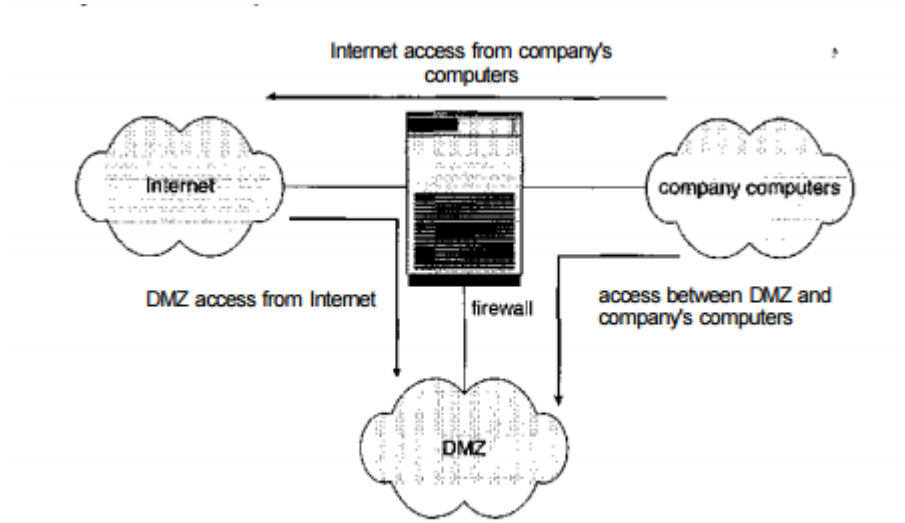


Figure 15.10 Domain separation via firewall.

COMPUTER-SECURITY CLASSIFICATIONS

The U.S. Department of Defense Trusted Computer System Evaluation Criteria specify four security classifications in systems: A, B, C, and D. This specification is widely used to determine the security of a facility and to model security solutions, so we explore it here.

The lowest-level classification is division D, or minimal protection. Division D includes only one class and is used for systems that have failed to meet the requirements of any of the other security classes. For instance, MS-DOS and Windows 3.1 are in division D. Division C, the next level of security, provides discretionary protection and accountability of users and their actions through the use of audit capabilities.

Division C has two levels: Cl and C2. A Cl-class system incorporates some form of controls that allow users to protect private information and to keep other users from accidentally reading or destroying their data. A Cl environment is one in which cooperating users access data at the same levels of sensitivity. Most versions of UNIX are Cl class.

The sum total of all protection systems within a computer system (hardware, software,

firmware) that correctly enforce a security policy is known as a trusted computer base (TCB). The TCB of a Cl system controls access between users and files by allowing the user to specify and control sharing of objects by named individuals or defined groups.

In addition, the TCB requires that the users identify themselves before they start any activities that the TCB is expected to mediate. This identification is accomplished via a protected mechanism or password; the TCB protects the authentication data so th at they are inaccessible to unauthorized users. A C2-class system adds an individual-level access control to the requirements of a Cl system.

For example, access rights of a file can be specified to the level of a single individual. In addition, the system administrator can selectively audit the actions of any one or more users based on individual identity. The TCB also protects itself from modification of its code or data structures. In addition, no information produced by a prior user is available to another user who accesses a storage object that has been released back to the system. Some special, secure versions of UNIX have been certified at the C2 level. Division-B mandatory-protection systems have all the properties of a classC2 system; in addition, they attach a sensitivity label to each object. The Bl-class TCB maintains the security label of each object in the system; the label is used for decisions pertaining to mandatory access control.

For example, a user at the confidential level could not access a file at the more sensitive secret level. The TCB also denotes the sensitivity level at the top and bottom of each page of any human-readable output. In addition to the normal user-namepassword authentication information, the TCB also maintains the clearance and authorizations of individual users and will support at least two levels of security. These levels are hierarchical, so that a user may access any objects that carry sensitivity labels equal to or lower than his security clearance.

For example, a secret-level user could access a file at the confidential level in the absence of other access controls. Processes are also isolated through the use of distinct

address spaces. A B2-class system extends the sensitivity labels to each system resource, such as storage objects.

Physical devices are assigned minimum and maximum security levels that the system uses to enforce constraints imposed by the physical environments in which the devices are located. In addition, a B2 system supports covert channels and the auditing of events that could lead to the exploitation of a covert channel. A B3-class system allows the creation of access-control lists that denote users or groups not granted access to a given named object. The TCB also contains a mechanism to monitor events that may indicate a violation .

The mechanism notifies the security administrator aid, if necessary, terminates the event in the least disruptive manner. The highest-level classification is division A. Architecturally, a class-Al system is functionally equivalent to a B3 system, but it uses formal design specifications and verification techniques, granting a high degree of assurance that the TCB has been implemented correctly. A system beyond class Al might be designed and developed in a trusted facility by trusted personnel.

The use of a TCB merely ensures that the system can enforce aspects of a security policy; the TCB does not specify what the policy should be. Typically, a given computing environment develops a security policy for certification and has the plan accredited by a security agency, such as the National Computer Security Center. Certain computing environments may require other certification, such as that supplied by TEMPEST, which guards against electronic eavesdropping. For example, a TEMPEST-certified system has terminals that are shielded to prevent electromagnetic fields from escaping. This shielding ensures that equipment outside the room or building where the terminal is housed cannot detect what information is being displayed by the terminal.