PROTECTION AND SECURITY

Protection mechanisms control access to a system by limiting the types of file access permitted to users. In operating system, protection ensures that only processes that have gained proper authorization from the operating system can operate on memory segments, the CPU and other resources. Protection mechanisms provide means for specifying the controls to be imposed together with means of enforcing them.

Security on the other hand, ensures the authentication of system users to protect the integrity of information stored in the system (data and code) and physical resources of the computer system. Security system prevents unauthorized access, malicious destruction or alteration of data, and accidental introduction of inconsistency.

In general, **protection** refers to a mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system. **Security** is a measure of confidence that the integrity of a system and its data will be preserved.

Protection

Goals of Protection

Protection in a computer system should be provided for several reasons:

- Prevention of mischievous, intentional violation of an access restriction by user
- Ensure that programs, processes and users active in a system uses system resources only in a way consistent with stated policies.
- Improving reliability through detection of errors at the interfaces between component subsystems. Early detection of errors can often prevent contamination of a healthy subsystem by a malfunctioning subsystem.
- Defend computer resources against use or misuse by unauthorized or incompetent user.

Protection provides mechanism for enforcing policies governing resource use. These policies are established in a number of ways. Some are fixed in the design of a system, others are formulated by management of system while others are defined by individual users to protect their own files and programs.

NB: mechanisms determine **how** something will be done while **policies** decide **what** will be done.

Principles of protection

A guiding principle can be used to simplify design decisions and keep the system consistent as well as easy to understand. A frequently used, and a key time tested principle of protection is the

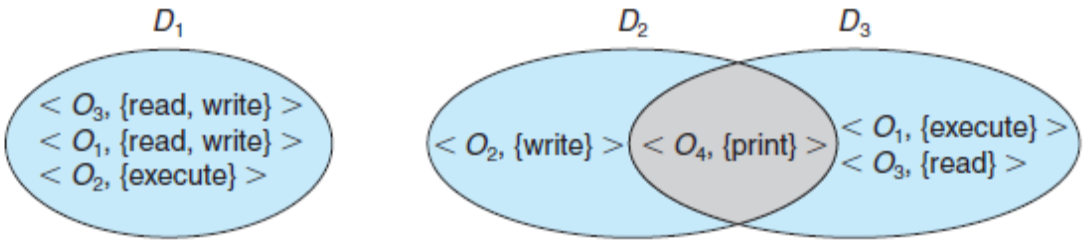
principle of least privilege. It dictates that programs, users, and even systems be given just enough privileges to perform their tasks. The operating system follows this principle to implements its features, programs, system calls, and data structures so that failure or compromise of a component does minimum damage and allows minimum damage to be done.

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.

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 below, we have three domains: *D*1, *D*2, and *D*3. The access right <*O*4, *{*print*}*> is shared by *D*2 and *D*3, implying that a process executing in either of these two domains can print object *O*4. Note that a process must be executing in domain *D*1 to read and write object *O*1, while only processes in domain *D*3 may execute object *O*1.



The association between a process and a domain may be either **static**, if the set of resources available to the process is fixed throughout the process's lifetime, or **dynamic**. If the association is dynamic, a mechanism is available to allow **domain switching**, enabling the process to switch from one domain to another.

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.

Access matrix

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 .

For the purpose of illustration, consider the table below

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	

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 $D2$.

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 policies

outlined hold. More specifically, we must ensure that a process executing in domain *Di* can access only those objects specified in row *i*, and then only as allowed by the access-matrix entries.

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 users normally decide the contents of the access-matrix entries. When a user creates a new object *Oj*, the column *Oj* is added to the access matrix with the appropriate initialization entries, as dictated by the creator. The user may decide to enter some rights in some entries in column *j* and other rights in other entries, as needed.

The access matrix provides an appropriate mechanism for defining and implementing strict control for both static and dynamic association between processes and domains.

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 defined in access(*i*, *j*).

For example, in the table below, a process executing in domain *D*2 can switch to domain *D*3 or to domain *D*4. A process in domain *D*4 can switch to *D*1, and one in domain *D*1 can switch to *D*2.

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			

Security

We say that a system is **secure** if its resources are used and accessed as intended under all circumstances. Unfortunately, total security cannot be achieved. Nonetheless, we must have mechanisms to make security breaches a rare occurrence, rather than the norm.

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. The following list includes several forms of accidental and malicious security violations.

- **Breach of confidentiality**. This type of violation involves unauthorized reading of data (or theft of information) e.g. capturing secret data from a system or a data stream such as credit-card information or identity information.
- **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. 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. These attacks are sometimes accidental.

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 sneaky 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**.
- **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.
- **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.