MODULE - 5
PROTECTION.

- The process in an operating system must be protected from one another's activities.
- In provide such protection, we can use various mechanisms to ensure that only processes that have gained proper authorization from the operating system can operate on the files, memory segments, CPU, and other resources of a system.
- 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.

Goals of Protection

- To prevent malicious misuse of the system by users or programs. See chapter 15 for a more through coverage of this goal.
- To ensure that each shared resource is used only in accordance with system policies, which may be set either by system designers or by system administrators.
- To ensure that errant programs cause the minimal amount of damage possible.
- Note that protection systems only provide the mechanisms for enforcing policies and ensuring reliable systems. It is upto administrators and users to implement those mechanisms effectively.

Principles of Protection:

- The principle of least privilege dictates the programs, users, and systems be given just enough privileges to perform their tasks.

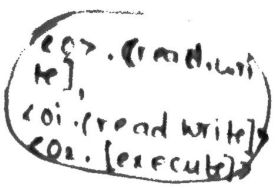
- This ensures that the failures do the least amount of harm and allow the least of harm to be done.
- For example, if a program need special previleges to perform a task, it is better to make it a SGID program with group ownership of "network" or "backup" or some other pseudo group, rather than SUID with root ownership. This limits the amount of damage that can occur if something goes wrong.
- Typically each user is given their own account, and has only enough privilege to modify their own files.
- The root account should not be used for normal day to day activities - The system administrator should also have an ordinary account, and reserve use of the root account for only those tasks which need the root privileges.

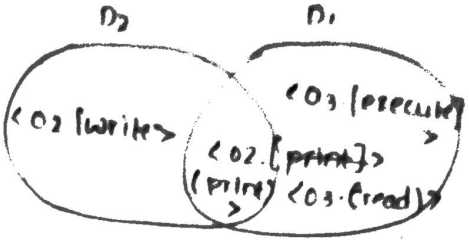
Domain of Protection:

- A computer can be viewed as a collection of processes and objects (both HW & SW).
- The need to know principle states that a process should only have access to those objects it needs to accomplish its tasks, and furthermore only in the modes for which it needs access. and only during the time frame when it needs access.
- The modes available for a particular object may depend upon its type.

Domain Structure:

• A computer





• The association between a process and a domain may be static or dynamic.

* If the association is stable, then the need-to-know principle requires a way of changing the contents of the domain dynamically.

* If the association is dynamic, then there needs to be a mechanism for domain switching.

• A domain can be realised 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 domain. In this, 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:

- The model of protection that we have been discussing can be viewed as an access matrix, in which columns represent different system resources and rows represent different protection domains. Entries within the matrix indicate what access that domain has to that resource.

Object / Domain	P_1	P_2	P_3	Printer
D_1	read		read	
D_4				print
D_3		read	execute	
D_4	read write		read write	

Access Matrix

- Domain switching can be easily supported under this model, simply by providing "switch" access to other domain.

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			

- The ability to copy right is denoted by an asterisk, indicating that processes in that domain have the right to copy that access within the same column, i.e., for the same object. There are two important variations:

* If the asterisk is removed from the original access right, then the right is transferred, rather than being copied. This may be termed a transfer right as opposed to a copy right.

* If only the right and not the asterisk is copied, then the access right is added to the new domain, but it may not be propogated further that is the new domain doesnot also receive the right to copy the access. This may be termed a limited copy right as shown in figure below.

(a)

object / Domain	F1	F2	F2
D1	execute		~~execute~~ write *
D2	~~execute~~	read *	execute
D3	~~execute~~		

(b)

object / Domain	F1	F2	F3
D1	execute		write *
D2	execute	read *	execute
D3	execute	read	

* Copy and owner rights only allow the modification of rights within a column. The addition of control rights, which only apply to domain objects, allow a process operating in one domain to affect the rights available in other domains.

* For example in the table below, a process operating in domain D2 has the right to control any of the rights in domain D4.

Object / Domain	F1	F2	F3	Faster Printer	D1	D2	D3	D4
D1	read					switch		
D2				print			switch	switch control
D3		read	execute write					
D4	write		write		switch			

Implementation of Access matrix:

1. Global Table:

- The simplest approach is one big global table with <domain, object, rights> entries.
- Unfortunately this table is very large (even if sparse) and so cannot be kept in memory (without invoking virtual memory techniques).
- There is also no good way to specify groupings. If everyone has access to some resource, then it still needs a seperate entry for every domain

2. Access lists for objects:

- Each column of the table can be kept as a list of the access rights for that particular object, discarding blank entries.
- For efficiently a seperate list of default access rights can also be kept, and checked first.

3. Capability lists for domains:

- In a similar fashion, each row of the table can be kept as a list of the capabilities of that domain.
- Capability lists are associated with each domain, but not directly accessible by the domain or any user process.
- Capability lists are themselves protected resources,

distinguished from other data in one of two ways:

* A tag, possibly hardware implemented, distinguishing this special type of data (other types may be floats, pointers, booleans, etc).

* The address space for a program may be split into multiple segments, atleast one of which is inaccessible by the program itself, and by the operating system for maintaining the process's access right capability list

4. A lock-key mechanism:

- Each resource has a list of unique bit patterns, termed locks.
- Each domain has its own list of unique bit patterns, termed keys.
- Access is granted if one of the domain's keys fits one of the resource's locks.
- Again, a process is not allowed to modify its own keys.

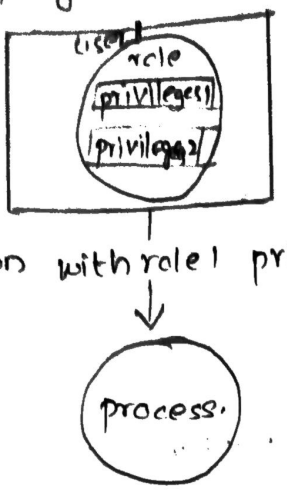
5. Comparision:

- Each of the methods here has certain advantages or disadvantages, depending on the particular situation and task at hand.
- Many systems employ some combination of the listed methods.

Access Control:

- Role-Based Access control, RBAC, assigns privileges to users, programs, or roles as appropriate, where "privilages" refer to the right to call certain system calls, or to use certain parameters with those calls.
- RBAC supports the principle of least privilege, and

reduces the susceptibility to abuse as opposed to suid or sgid programs.



Revocation of Access Rights.

- The need to revoke access rights dynamically raises several questions:
  - * Immediate versus delayed - If delayed, can we determine when the revocation will take place?
  - * Selective versus general - Does revocation of an access right to an object affect all users who have the right, or only some users?
  - * Partial versus total - Can a subset of rights for an object be revoked, or are all rights revoked at once?
  - * Temperary versus permanent - If rights are revoked is there a mechanism for processes to re-acquire some or all of the revoked rights?
- With an access list scheme revocation is easy, immediate, and can be selective, general, partial, total, temporary or permanent as desired.
- With capabilities lists the problem is more complicated because access rights are distributed throughout the system. A few schemes that have been developed include:

• Reacquisition : capabilities are periodically revoked from each domain, which must then reacquire them.
• Back-pointers: A list of pointers is maintained from each object to each capability which is held for that object.
• Indirection:- Capabilities point to an entry in a global table rather than to the object. Access rights can be revoked by changing or invalidating the table entry, which may affect multiple processes, which must there re-acquire access rights to continue.
• Keys- A unique bit pattern is associated with each capability when created, which can be neither inspected nor modified by the process.
* A master key associated with each object.
* When a capability is created, its key is set to the object's master key.
* As long as the capabilities key matches the object's key, then the capabilities remain valid.
* The object master key can be changed with the set-key command

Program Threats

• There are many common threats to modern systems. Some of them are:

1. Trojan Horse:
• A trojan horse is a program that secretly performs some maliciousness in addition to its visible actions.
• Some trojan horses are deliberately written as such, and others are the result of legitimate programs that have become infected with viruses, (See below)
• One dangerous opening for trojan horses is long search paths, and in particular paths which include the current directory (".") as part of the path.

- If a dangerous program having the same name as a legitimate program (or a common mis-spelling, such as "sl" instead of "ls") is placed anywhere on the path, then an unsuspecting user may be fooled into running the wrong program by mistake.
- Another classic trojan horse is a login emulator, which record a users account name and password, issues a "password incorrect" message, and then logs off the system. The user then tries again (with a proper login prompt), logs in successfully and doesn't realise that their information has been stolen.
- Two solutions to trojan horses are to have the system print usage statistics on logouts, and to require the typing of non-tappable key sequences such as control-alt-delete in order to login. (This is why modern Windows systems require the control-alt-delete sequence to commence logging in, which cannot be emulated or caught by ordinary programs. I.E that key sequence always transfers control over to the operating system).
- Spyware is a version of a trojan horse that is often included in "free" software downloaded off the internet. Spyware programs generate pop-up browser windows, and may also accumulate information about the user and deliver it to some central site. (This is an example of convert channels, in which surreptitious communications occur) Another common task of spyware is to send out spam e-mail messages, which then purportedly come from the infected user

2. Trap door:

- A trap door is when a designer or a programmer (or hacker) deliberately inserts a security hole that they can use later to access the system.
- Because of the possibility of trap doors, once a system has been in an untrustworthy state, that system can never be trusted again. Even the backup tapes may contain a copy of some cleverly hidden back door.
- A clever trap door could be inserted into a compiler, so that any programs compiled with that compiler would contain a security hole. This is especially dangerous, because inspection of the code being compiled would not reveal any problems.

3. Logic Bomb:

- A logic bomb is code that is not designed to cause havoc all the time, but only when a certain set of circumstances occurs, such as when a particular date or time is reached or some other noticeable event.
- A classic example is the dead-man switch, which is designed to check whether a certain person (e.G. The author) is logging in every day, and if they don't log in for a long time (presumably because they have been fired), then the logic bomb goes off and either opens up security holes or causes other problem.

4. Stack and Buffer Overflow:

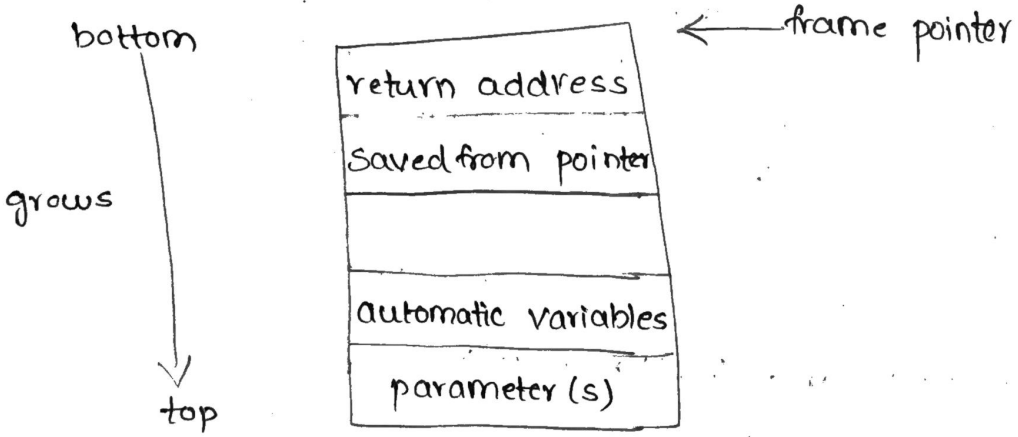
- This is a classic method of attack, which exploits bugs in system code that allows buffers to overflow.

```
#include
#define BUFFER_SIZE 256
int main (int argc, char* argv [])
{
char buffer [BUFFER_SIZE];
if(argc<2)return-1;
```

```
else
{
strcpy(buffer, argv[1]);
return 0;
  }
}
```

C program with buffer-overflow condition.

- The "bottom" of the stack is actually at a high memory address, and the stack grows towards lower addresses.
- However the address of an array is the lowest address of the array, and higher array elements extend to higher addresses. (I.E. An array "grows" towards the bottom of the stack.
- In particular, writing past the top of an array, as occurs when a buffer overflows with too much input data, can eventually overwrite the return address, effectively changing where the program jumps to when it returns.



The layout for a typical stack frame.

5. Viruses:

- A virus is a fragment of code embedded in an otherwise legitimate program, designed to replicate itself (by infecting other programs), and (eventually) wreaking havoc.