Protection

- Discuss the goals and principles of protection in a modern computer system
- Explain how protection domains combined with an access matrix are used to specify the resources a process may access
- Examine capability and language-based protection systems

Goals of Protection

Operating system consists of a collection of objects, hardware or software. Each object has a unique name and can be accessed through a well-defined set of operations. Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so.

- Obviously to prevent malicious misuse of the system by users or programs.
- 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 up to administrators and users to implement those mechanisms effectively.

Principles of Protection

Programs, users and systems should be given just enough privileges to perform their tasks

- The ***principle of least privilege*** dictates that programs, users, and systems be given just enough privileges to perform their tasks.
- This ensures that failures do the least amount of harm and allow the least of harm to be done.
- For example, if a program needs special privileges 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

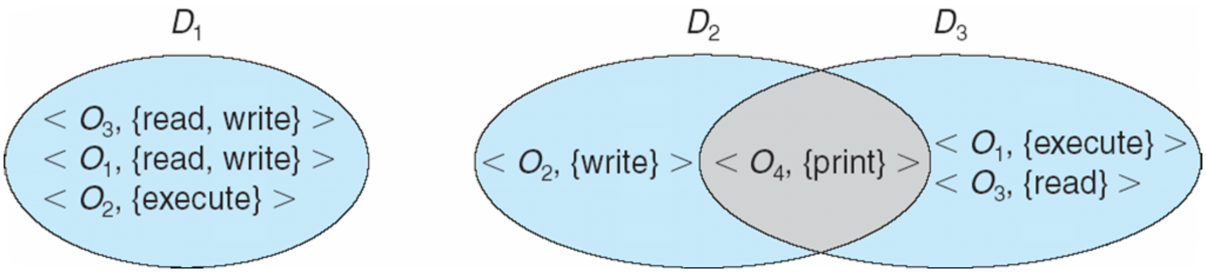
Domain Structure

- 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 task, 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.

Access-right = <*object-name*, *rights-set*>

where *rights-set* is a subset of all valid operations that can be performed on the object.

Domain = set of access-rights



System consists of 2 domains:

User

Supervisor UNIX

Domain = user-id

Domain switch accomplished via file system

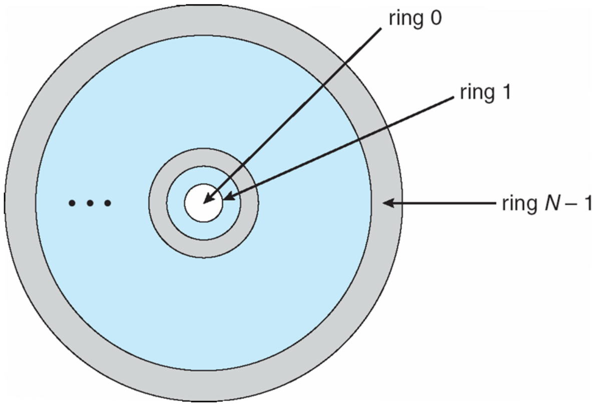
Each file has associated with it a domain bit (setuid bit)

When file is executed and setuid = on, then user-id is set to owner of the file being executed. When execution completes user-id is reset

Domain Implementation (MULTICS)

Let *Di* and *Dj* be any two domain rings

If *j* < *I* Þ *Di* Í *Dj*



- Rings are numbered from 0 to 7, with outer rings having a subset of the privileges of the inner rings.
- Each file is a memory segment, and each segment description includes an entry that indicates the ring number associated with that segment, as well as read, write, and execute privileges.
- Each process runs in a ring, according to the *current-ring-number,* a counter associated with each process.

- A process operating in one ring can only access segments associated with higher (farther out) rings, and then only according to the access bits. Processes cannot access segments associated with lower rings.
- Domain switching is achieved by a process in one ring calling upon a process operating in a lower ring, which is controlled by several factors stored with each segment descriptor:
    - An ***access bracket***, defined by integers b1 <= b2.
    - A ***limit*** b3 > b2
    - A ***list of gates,*** identifying the entry points at which the segments may be called.
- If a process operating in ring i calls a segment whose bracket is such that b1 <= i <= b2, then the call succeeds and the process remains in ring i.
- Otherwise a trap to the OS occurs, and is handled as follows:
    - If i < b1, then the call is allowed, because we are transferring to a procedure with fewer privileges. However if any of the parameters being passed are of segments below b1, then they must be copied to an area accessible by the called procedure.
    - If i > b2, then the call is allowed only if i <= b3 and the call is directed to one of the entries on the list of gates.
- Overall this approach is more complex and less efficient than other protection schemes.

Access Matrix

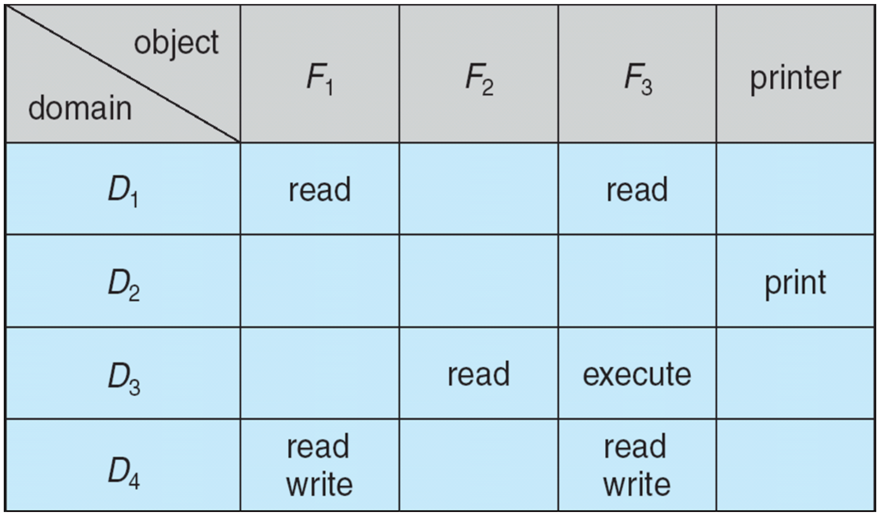
The model of protection that we have been discussing can be viewed as an **access matrix,** in which columns represent different system resources and rows represent different protection domains. Entries within the matrix indicate what access that domain has to that resource.

View protection as a matrix (*access matrix*)

Rows represent domains

Columns represent objects

Access (i, j) is the set of operations that a process executing in Domain$_i$ can invoke on Object$_j$



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	

Use of Access Matrix

If a process in Domain D_i tries to do “op” on object O_j, then “op” must be in the access matrix Can be expanded to dynamic protection Operations to add, delete access rights
Special access rights:
Owner of O_i *copy op from* O_i *to* O_j
Control – D_i *can modify* D_j *access rights*
Transfer – switch from domain D_i *to* D_j

ACCESS MATRIX DESIGN SEPARATES MECHANISM FROM POLICY

Mechanism

- Operating system provides access-matrix + rules
- If ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced

Policy

- User dictates policy
- Who can access what object and in what mode

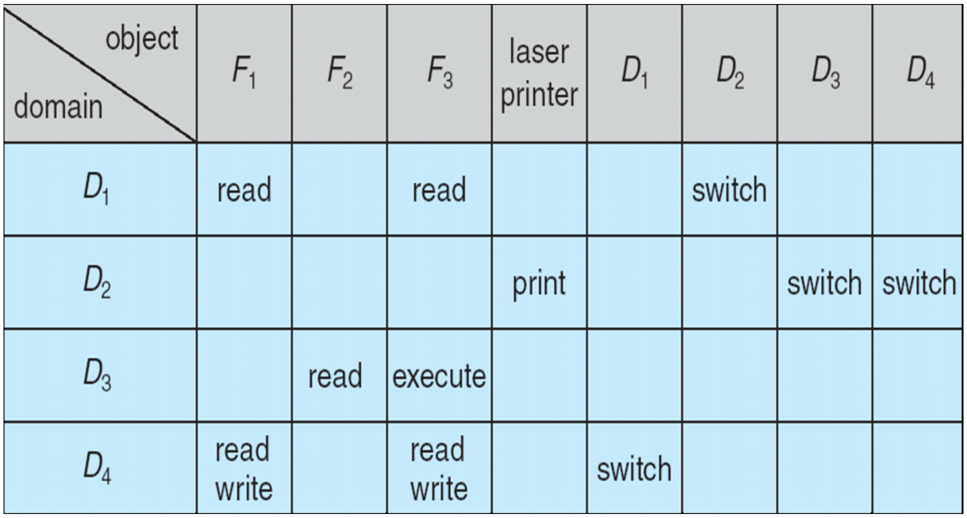
Implementation of Access Matrix

Each column = Access-control list for one object
Defines who can perform what operation.

Domain 1 = Read, Write
Domain 2 = Read
Domain 3 = Read

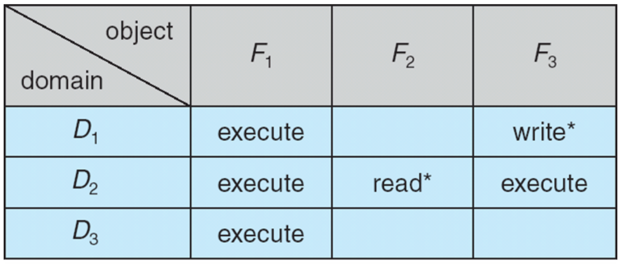
M Each Row = Capability List (like a key)
Fore each domain, what operations allowed on what objects.
Object 1 – Read
Object 4 – Read, Write, Execute
Object 5 – Read, Write, Delete, Copy

Access Matrix of Figure A With Domains as Objects



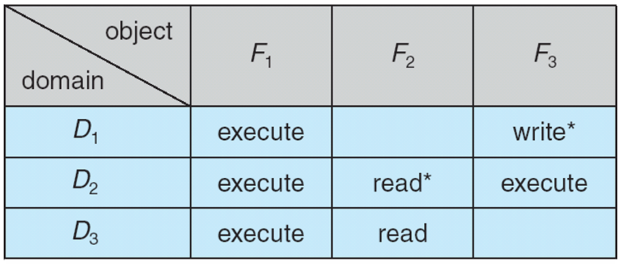
object domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Access Matrix with *Copy* Rights



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

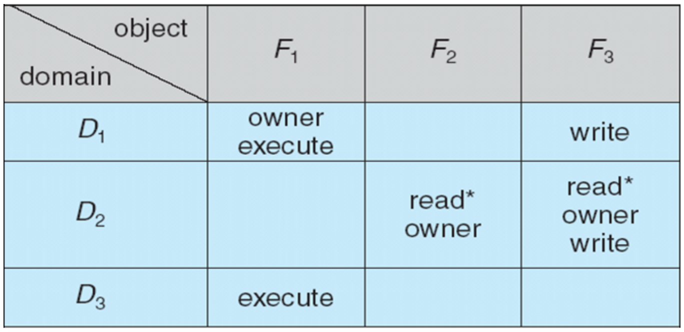
(a)



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

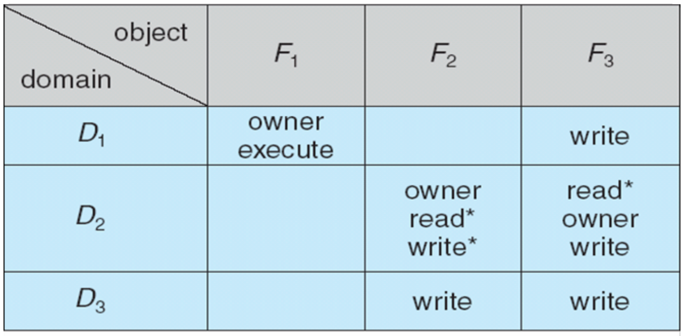
(b)

Access Matrix With *Owner* Rights



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

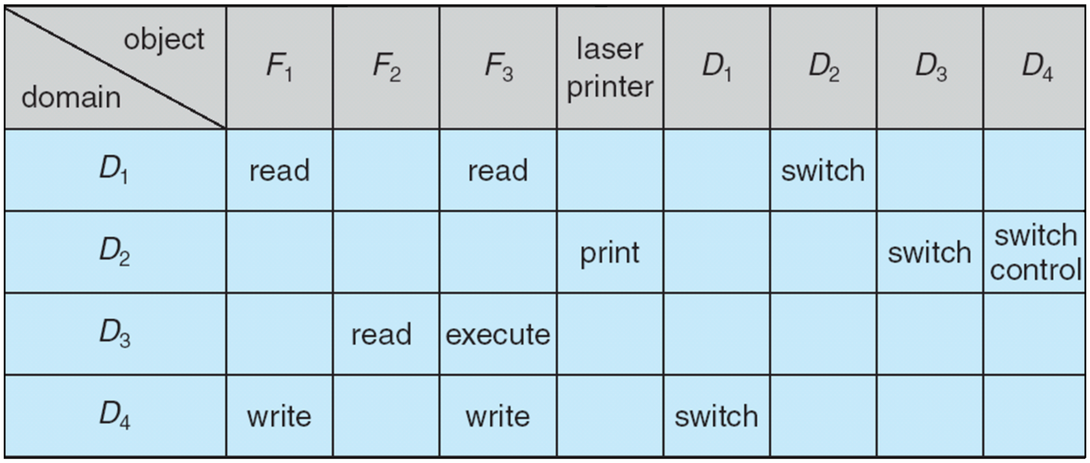
(a)



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)

Modified Access Matrix of Figure B



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 control
D_3		read	execute					
D_4	write		write		switch			

Access Control

Protection can be applied to non-file resources

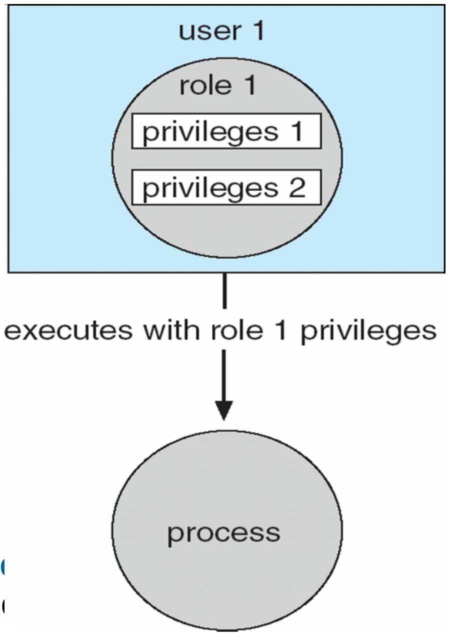
Solaris 10 provides role-based access control (RBAC) to implement least privilege

Privilege is right to execute system call or use an option within a system call

Can be assigned to processes

Users assigned roles granting access to privileges and programs

Role-based Access Control in Solaris 10



Revocation of Acc

Access List – Delet

Simple

Immediate Capability List – Scheme required to locate capability in the system before capability can be revoked

Reacquisition
Back-pointers
Indirection
Keys

Capability-Based Systems

Hydra
Fixed set of access rights known to and interpreted by the system
Interpretation of user-defined rights performed solely by user's program; system provides access protection for use of these rights Ambridge CAP System
Data capability - provides standard read, write, execute of individual storage segments associated with object
Software capability -interpretation left to the subsystem, through its protected procedures

Language-Based Protection

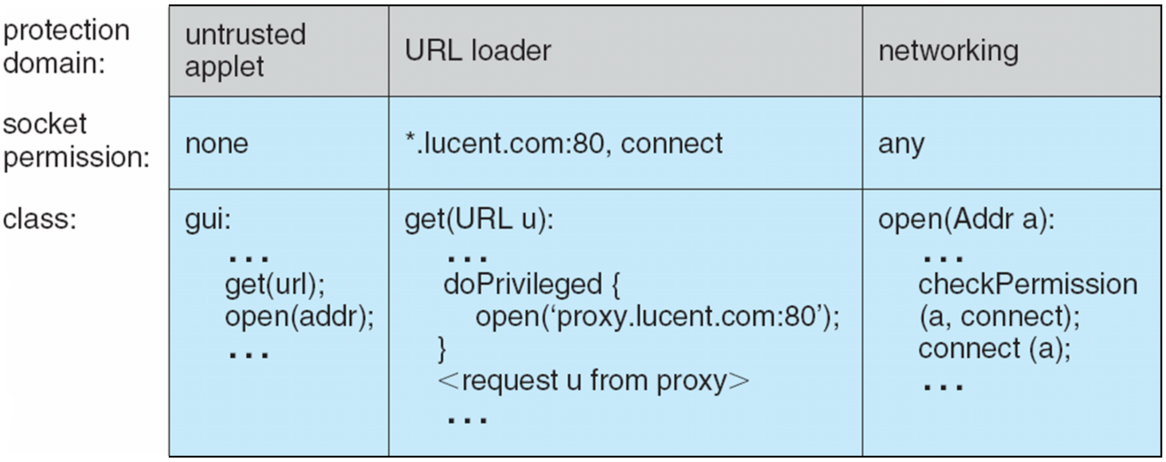
Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources Language implementation can provide software for protection enforcement when automatic hardware-supported checking is unavailable Interpret protection specifications to generate calls on whatever protection system is provided by the hardware and the operating system

Protection in Java 2

- Java was designed from the very beginning to operate in a distributed environment, where code would be executed from a variety of trusted and untrusted sources. As a result the Java Virtual Machine, JVM incorporates many protection mechanisms
- When a Java program runs, it load up classes dynamically, in response to requests to instantiates objects of particular types. These classes may come from a variety of different sources, some trusted and some not, which requires that the protection mechanism be implemented at the resolution of individual classes, something not supported by the basic operating system.
- As each class is loaded, it is placed into a separate protection domain. The capabilities of each domain depend upon whether the source URL is trusted or not, the presence or absence of any digital signatures on the class (Chapter 15), and a configurable policy file indicating which servers a particular user trusts, etc.
- When a request is made to access a restricted resource in Java, (e.g. open a local file), some process on the current ***call stack*** must specifically assert a privilege to perform the operation. In essence this method ***assumes responsibility*** for the restricted access. Naturally the method must be part of a class which resides in a protection domain that includes the capability for the requested operation. This approach is termed ***stack inspection,*** and works like this:

  - When a caller may not be trusted, a method executes an access request within a doPrivileged( ) block, which is noted on the calling stack.
  - When access to a protected resource is requested, checkPermissions( ) inspects the call stack to see if a method has asserted the privilege to access the protected resource.
    - If a suitable doPriveleged block is encountered on the stack before a domain in which the privilege is disallowed, then the request is granted.
    - If a domain in which the request is disallowed is encountered first, then the access is denied and a AccessControlException is thrown.
    - If neither is encountered, then the response is implementation dependent.
- In the example below the untrusted applet's call to get( ) succeeds, because the trusted URL loader asserts the privilege of opening the specific URL lucent.com. However when the applet tries to make a direct call to open( ) it fails, because it does not have privilege to access any sockets.

Stack Inspection



protection domain:	untrusted applet	URL loader	networking
socket permission:	none	*.lucent.com:80, connect	any

class:	gui:
	. . .
	get(url);
	open(addr);
	. . .
		get(URL u):
		. . .
		doPrivileged {
		open(‘proxy.lucent.com:80’);
		}
		<request u from proxy>
		. . .
			open(Addr a):
			. . .
			checkPermission
			(a, connect);
			connect (a);
			. . .