Protection and security

Last modified: 01.06.2021

Contents

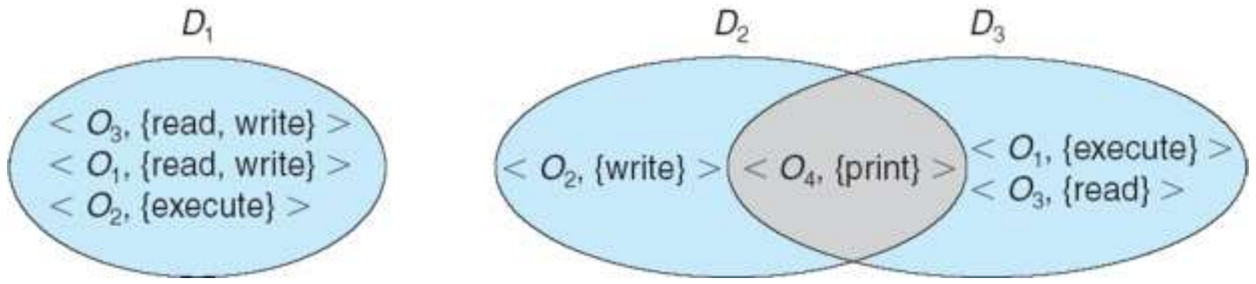
Goals of Protection

- The role of protection in a computer system is to provide a mechanism for the enforcement of the policies governing resource use. Note: **mechanisms are distinct from policies.**
- A computer 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 (or procedures or users) that are allowed to do so
- Guiding principle – **principle of least privilege:** programs, users and systems should be given just enough **privileges** to perform their tasks. This limits damage if entity has a bug, gets abused. Privileges can be static or dynamic.

Note: most systems do not have the granularity of privileges and permissions required to apply this principle precisely.

Protection domain structure

- **Access-right** = <*object-name*, *rights-set*>
  where *rights-set* is a subset of all valid operations that can be performed on the object
- **Protection domain** = set of access-rights



Protection domain implementation

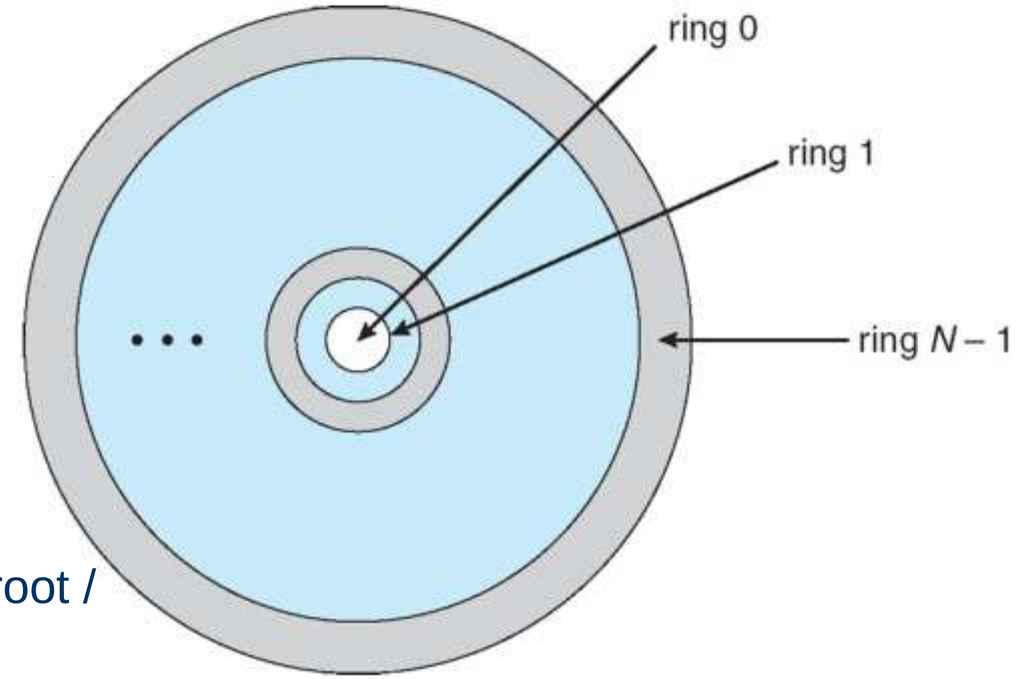
- Each **user** may be a domain. The sets of objects that can be accessed depends on the identity of the user. Domain switching occurs when the user is changed, e.g. when one user logs out and another user logs in.
- Each **process** may be a domain. The set of objects that can be accessed depends on 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. The sets of objects that can be accessed corresponds to the local variables defined within the procedure. Domain switching occurs when a procedure call is made.

Domain Implementation (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 switch accomplished via passwords
  - `su` command temporarily switches to another user’s domain when other domain’s password provided
- Domain switching via commands
  - `sudo` command prefix executes specified command in another domain (if original domain has privilege or password given)
  - change of current group can be used to approximate the principle of least privilege.

Domain Implementation (MULTICS)

- Let D_i and D_j be any two domain rings
- If $j < I \Rightarrow D_i \subseteq D_j$

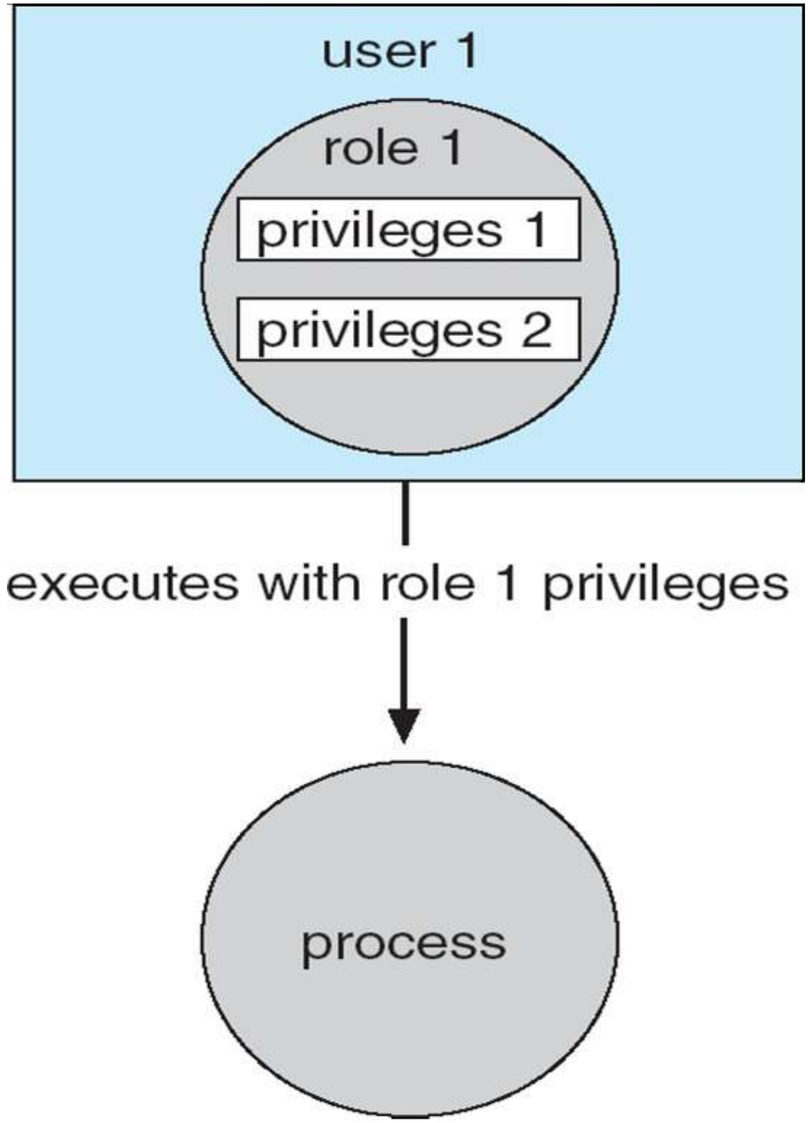


- Ring / hierarchical structure provided more than the basic kernel / user or root / normal user design
- Fairly complex -> more overhead
- But does not allow strict need-to-know
  - Object accessible in D_j but not in D_i, then ***j*** must be < ***i***
  - But then every segment accessible in D_i also accessible in D_j

The main disadvantage of the hierarchical structure – it doesn't allow to enforce the need-to-know principle.

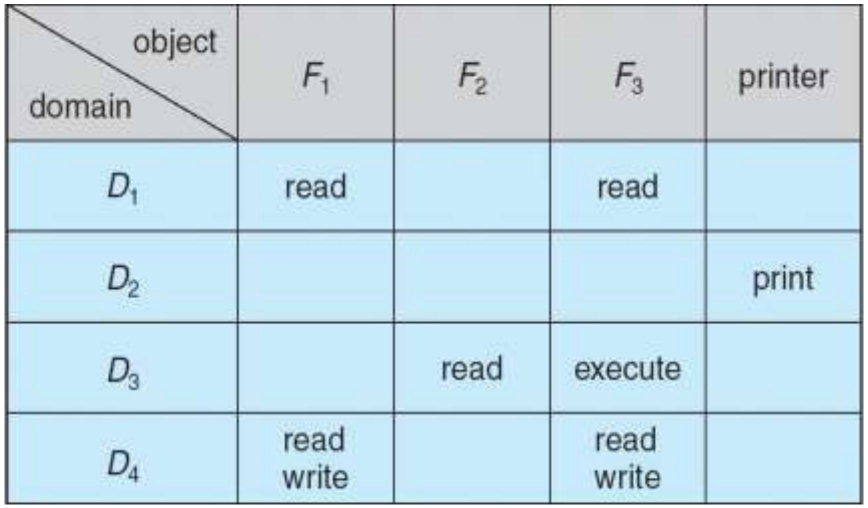
Domain Implementation – cont.

- Solaris 10 implemented the role-based access control (RBAC). It can be applied to system objects and also to activities (e.g. to the right to use a system function or an optional argument of the function)
- A role can be assigned to a process
- Users can take roles based on passwords to the roles. So a user can run a program requiring specific privileges without risks associated with superusers and setuid/setgid programs.
- RBAC is different from MAC and DAC access control frameworks (see p. 21), but it can enforce these policies.



A protection model - Access Matrix

- 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	

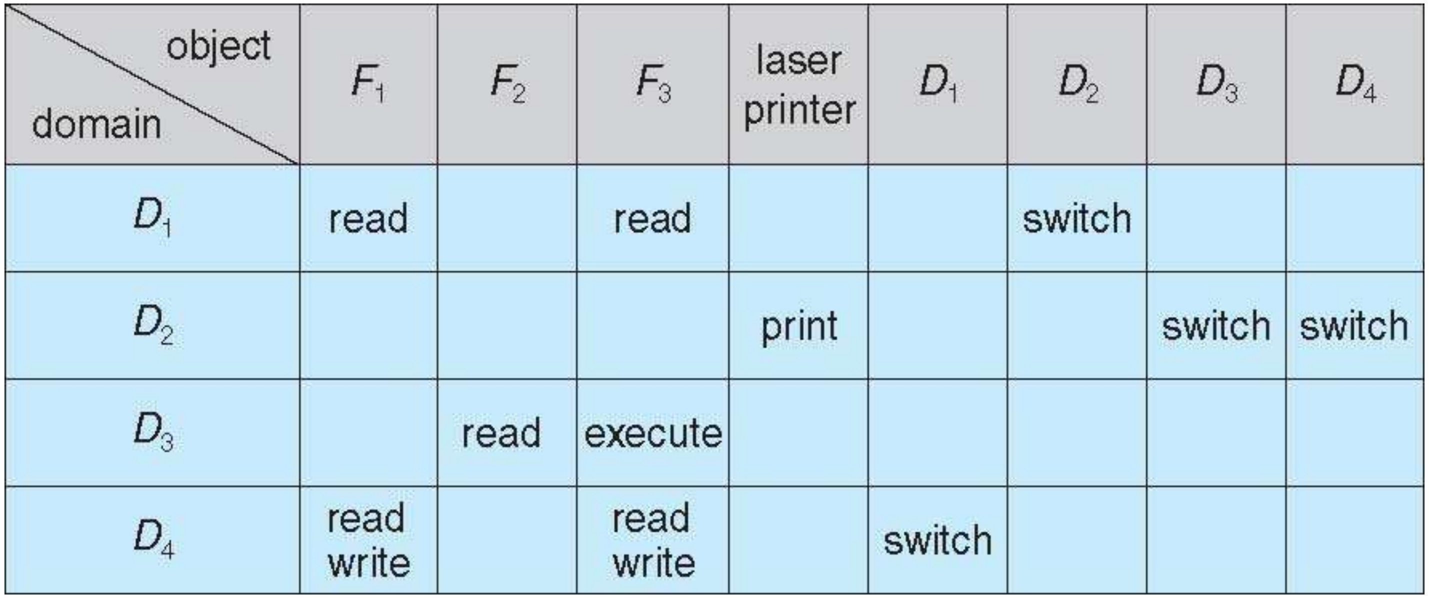
Use of Access Matrix

- **Access matrix** design separates mechanism from policy
  - Mechanism: operating system provides access-matrix + rules. It ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced
  - Policy: user dictates policy wo can access what object and in what mode

- 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 *(denoted by "*")*
    - ***control*** *–* D_i *can modify* D_j *access rights*
    - ***transfer*** *– switch from domain* D_i *to* D_j
  - ***Copy*** and ***Owner*** applicable to an object
  - ***Control*** applicable to domain object

- Note: the ***copy*** and ***owner*** operations provide a mechanism to limit the propagation of access rights, but they are not sufficient to prevent undesired propagation (or disclosure) of information. The **confinement problem** (how to guarantee that no information initially held in an object can migrate outside of its execution environment) is in general unsolvable.

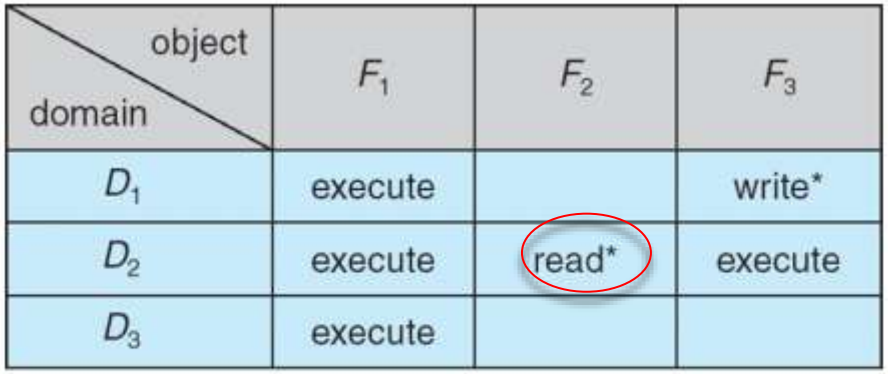
Example: Access Matrix with Domains as Objects



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			

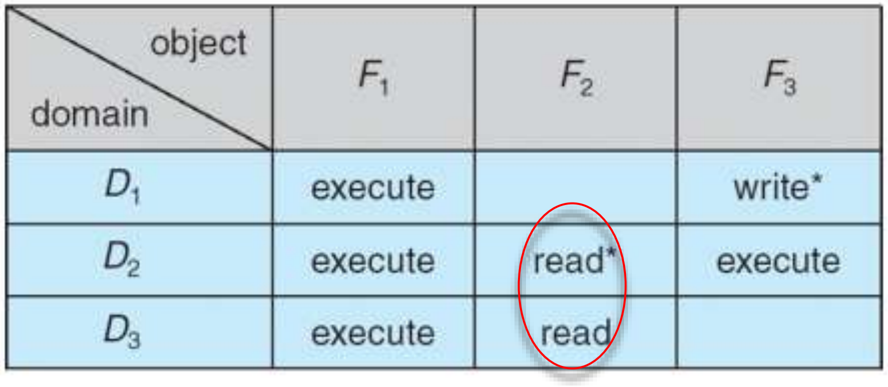
switch – enables change of process domain

Example: 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)

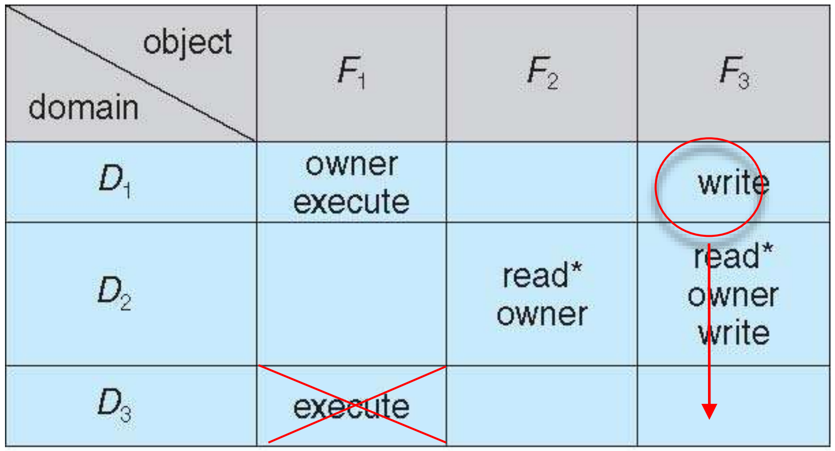
read*, write* - copying access rights (read, write) from one domain (row) of the access matrix to another.

Example: application of read* in column F2 to domain D3 in Fig. (a) resulted in the access matrix as shown in Fig. (b). Note that the copy does not allow a process in domain D3 to copy the rights to other domain. It is also possible to define other copy type operations:

- which enables copying the right to copy.

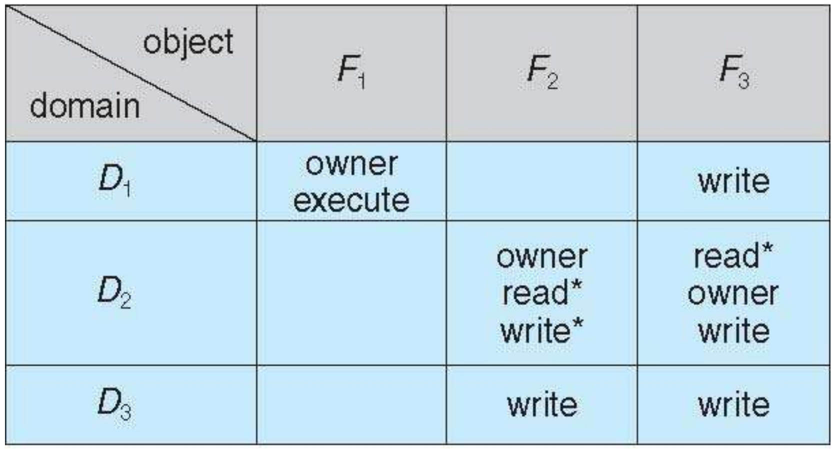
- a transfer of right, which removes the right from the copy source.

Example: Access Matrix With *Owner* Rights



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

(a)

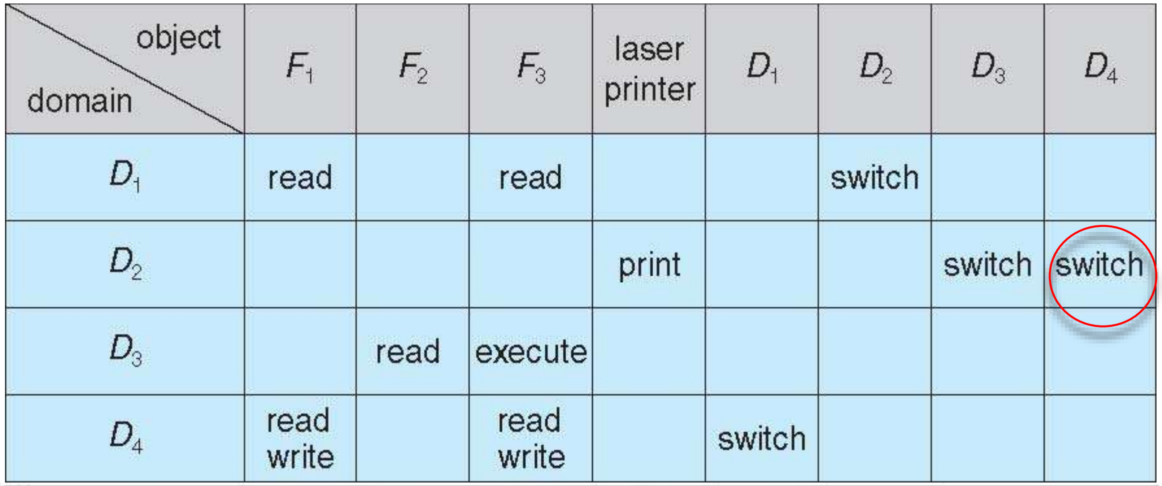


domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

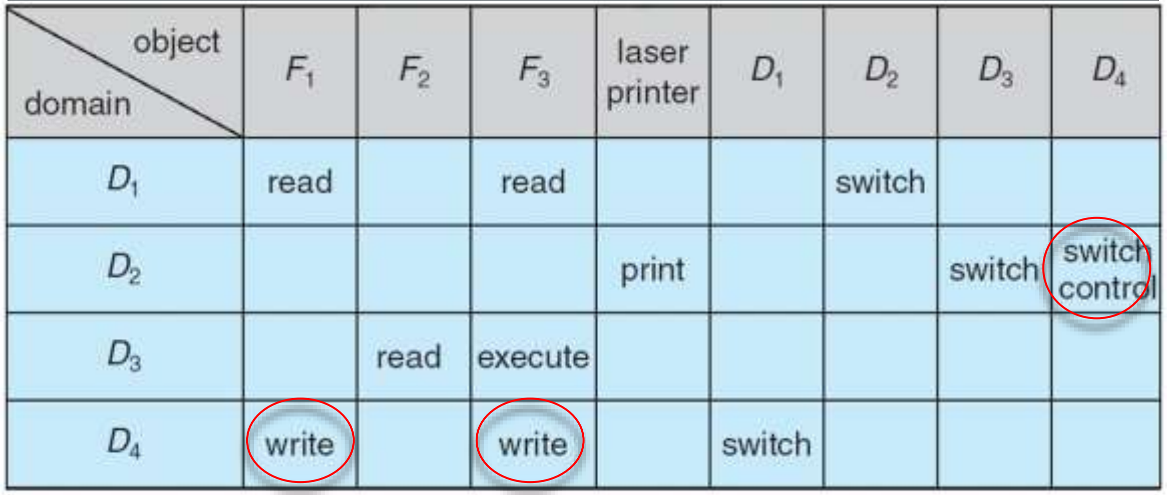
(b)

owner – controls addition of new rights to the owned object and removal of existing ones

Example: control operation



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			



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			

control – the right of a process belonging to one domain to remove access rights in another domain.

Example: adding control in the entry D2-D4 can lead to changes of the access matrix as shown below.

←\

Implementation of Access Matrix

- Generally, the access matrix is sparse
- Option 1 – **Global table**; represents ordered triples `<domain, object, rights-set>`
  The table could be large -> won't fit in main memory
- Option 2 – Access lists for objects
  - Each column = Access-control list for one object
    Resulting per-object list consists of ordered pairs `<domain, rights-set>` defining all domains with non-empty set of access rights for the object
  - Easily extended to contain default set -> If M ∊ default set, also allow access
- Option 3 – Capability list for domains
  - Each Row = **Capability list** (like a key)
    Object represented by its name or address, called a **capability.** Possession of capability means access is allowed
  - Capability list associated with domain but never directly accessible by domain
    - Rather, protected object, maintained by OS and accessed indirectly (like a "secure pointer")
- Option 4 – **Lock-key**. A compromise between access lists and capability lists
  - Each object has list of unique bit patterns, called **locks**
  - Each domain as list of unique bit patterns called **keys**
  - Process in a domain can only access object if domain has key that matches one of the locks

Revocation of Access Rights

- Various options to remove the access right of a domain to an object
  - **Immediate vs. delayed**
  - **Selective vs. general**
  - **Partial vs. total**
  - **Temporary vs. permanent**
- **Access List** – Delete access rights is **simple** – search access list and remove entry
- **Capability List** – Scheme required to locate capability in the system before capability can be revoked
  - **Reacquisition** – periodic delete, with require and denial if revoked
  - **Back-pointers** – set of pointers from each object to all capabilities of that object (Multics)
  - **Indirection** – capability points to global table entry which points to object – delete entry from global table, not selective (CAL)
  - **Keys** – unique bits associated with capability, generated when capability created
    - Master key associated with object, key matches master key for access
    - Revocation – create new master key
    - Policy decision of who can create and modify keys – object owner or others?

Comparison of Implementations

- Many trade-offs to consider
  - Global table is simple, but can be large
  - Access lists correspond to needs of users
    - Determining set of access rights for domain non-localized so difficult
    - Every access to an object must be checked
      - Many objects and access rights -> slow
  - Capability lists useful for localizing information for a given process
    - But revocation capabilities can be inefficient
  - Lock-key effective and flexible, keys can be passed freely from domain to domain, easy revocation
- Most systems use combination of access lists and capabilities
  - First access to an object -> access list searched
    - If allowed, capability created and attached to process – to be used during subsequent accesses (the system can easily reject improper access attempts, e.g. write attempt to a file opened for reading). Note: the right to access the object must still be checked on each access.
    - After last access, capability destroyed

Language-Based Protection

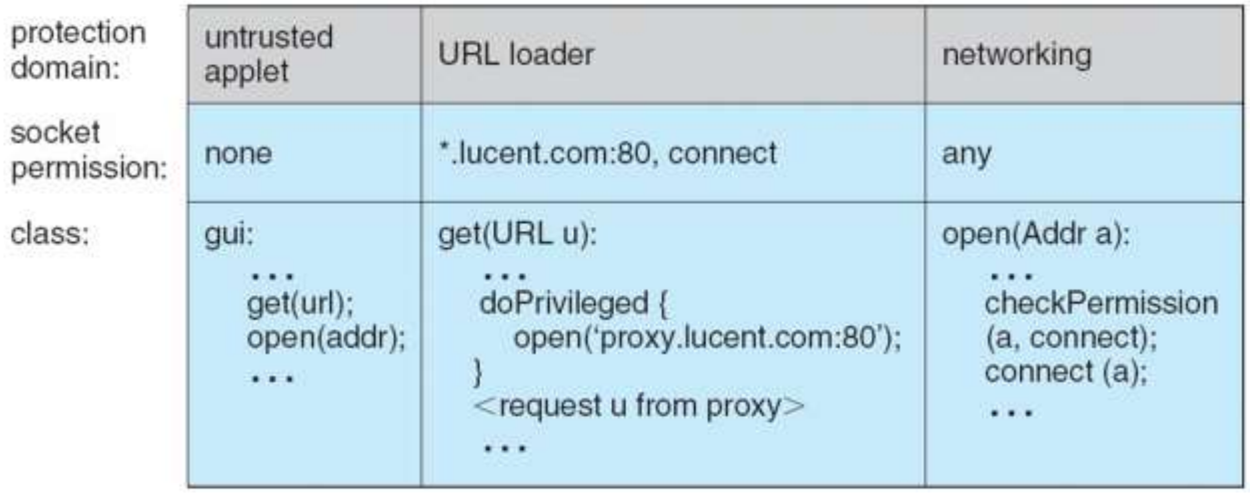
- Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources
- Language implementation can provide software for protection enforcement when automatic hardware-supported checking is unavailable
- Protection enforcement by kernel provides a greater degree of security of the protection system itself than des the generation of protection-checking code by a compiler. In a compiler-supported scheme security rests on correctness of translator and storage management.
- There are limits to the flexibility of a protection kernel in implementing a user-defined policy. Language-based protection is potentially more flexible.
- Language-based protection can perform static access enforcement in compile time - not in run-time.

Protection in Java 2

- Protection is handled by the Java Virtual Machine (JVM)
- A **class** is assigned a protection domain when it is loaded by the JVM
- The protection domain indicates what operations the class can (and cannot) perform
- Generally, Java's load-time and run-time checks enforce **type safety**
- Classes effectively **encapsulate** and protect data and methods from other classes
- If a library **method** is invoked that performs a privileged operation, the stack is **inspected** to ensure the operation can be performed by the library

Example of

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);
			. . .

The Security Problem

- Protection is an internal problem of a system. Security requires (besides appropriate protection mechanisms) consideration of external environment within which the system operates.
- System is ideally **secure** if resources used are accessed as intended (i.e. satisfy security policy) **under all circumstances**
- It is impossible to have absolute security, but it is possible to make cost to perpetrator sufficiently high to deter most intruders. Risk analysis is desirable.
- Security must occur at four levels to be effective:
  - **Physical -** Data centers, servers, connected terminals
  - **Human** Avoid **social engineering**, **phishing**, **dumpster diving**
  - **Operating System** Protection mechanisms, debugging
  - **Network** Intercepted communications, interruption, DOS
- Security is as weak as **the weakest link** in the chain
- But can too much security be a problem?
- **Threat** is potential security violation. **Attack** is an attempt to breach security.
- Attack can be accidental or malicious (unauthorized data access, data modification/ destruction). It is easier to protect against accidental than malicious misuse

OS security concepts

- **Identification** - refers to the naming of each user on the system
- **Authentication** – refers to the proof that the user is who s/he is.
- **Authorization** – a system mechanism which grants privileges to individual users
- **Access control** – mediates user access of system resources. **Discretionary Access Control** (DAC) is capable of enforcing access limitations decided upon by users, while **Mandatory Access Control** (MAC) refers to enforcement of system security policy centrally controlled by a security policy administrator. MAC can overwrite access permissions of DAC
- **Audit** can be configured to log events that users create in the system
- **Accountability** – requires identification, authentication and auditing
- **Object reuse** requirement states that all user accessible resources are initially cleared so that no lingering information cannot be extracted from them.
- **Assurance** – guarantee (provided by vendor and third parties) that the system's security features work as advertised.
- **Trusted Computing Base** – minimum hardware and software components of the computer system that security depends on. Note that the software of TCB has to protect itself against tampering to be of any effect.

Computer Security Classifications

- **Trusted Computer System Evaluation Criteria** (TCSEC) - a U.S. DoD standard ("Orange Book") introduced in 1983 and updated in 1985.
- TCSEC defined four major computer security: **A**, **B**, **C**, and **D**
  - **D** – Minimal security (fail to meet requirements for the higher sec. level)
  - **C** – Identification, authentication, discretionary access control
    - **C1** identifies cooperating users with the same level of protection
    - **C2** individual accountability, object reuse, resource isolation. C2 is a de-facto standard for commercial secure UNIX type systems
  - **B** – Mandatory protection
    - **B1** Labeled Security Protection (each object may have unique sensitivity labels, e.g. confidential, secret, top secret), Mandatory Access Control (MAC) over selected subjects and objects
    - **B2** – Structured Protection. **Operator != Administrator**
    - **B3** – Security Domains
  - **A** – Formal design and verification techniques to ensure security
- European standards: **ITSEC** (Information Technology Security Evaluation Criteria), distinguish functional requirements and guarantees. Rough equivalence: C2<-> F-C2. Additionally assurance level is specified, so typically C2 <-> F-C2,E-3
- Common Criteria (ISO/IEC 15408) – an international standard for computer security classification.

Protection and security in Linux

- Access control under UNIX systems, including Linux, is performed through the use of unique numeric identifiers (***uid*** and ***gid***)
- Access control is performed by assigning objects a *protections mask*, which specifies which access modes—read, write, or execute (search for directory)—are to be granted to processes with owner, group, or world access (UGO/RWX)
- Linux augments the standard UNIX ***setuid*** mechanism as it implements the POSIX specification's saved ***user-id*** mechanism, which allows a process to repeatedly drop and reacquire its effective uid

- Linux provides another mechanism (local socket message transfer) that allows a client to selectively pass access to a single file to some server process without granting it any other privileges
- Linux **control groups** (*cgroups*) - a kernel feature that limits, accounts for and isolates the resource use for a collection of processes. Processes from different cgroups can see different components of the physical system.
- **Namespace** – Specific view of file system hierarchy
  - Most processes share common namespace and operate on a shared file-system hierarchy
  - But each can have unique file-system hierarchy with its own root directory and set of mounted file systems (namespace isolation)

Access Control Lists in Linux

- **Access control list** (ACL) – can be associated with a file or catalogue. It provides a list of user identifiers and related specific access rights

- **setfacl** – a system command which makes possible to modify ACL for the specified file/directory. Example

```
$ setfacl -m user:student:r shared1.txt
```

- **getfacl** – a system command which displays actual ACL for the specified file/directory. Example:

```
$ getfacl shared1.txt
# file: shared1.txt
# owner: lopalski
# group: lopalski
user::rw-
user:student:r--
group::r--
mask::r--
other::---
```

C2 security in Unix/Linux

- UNIX/Linux systems provide C1 class security by default.

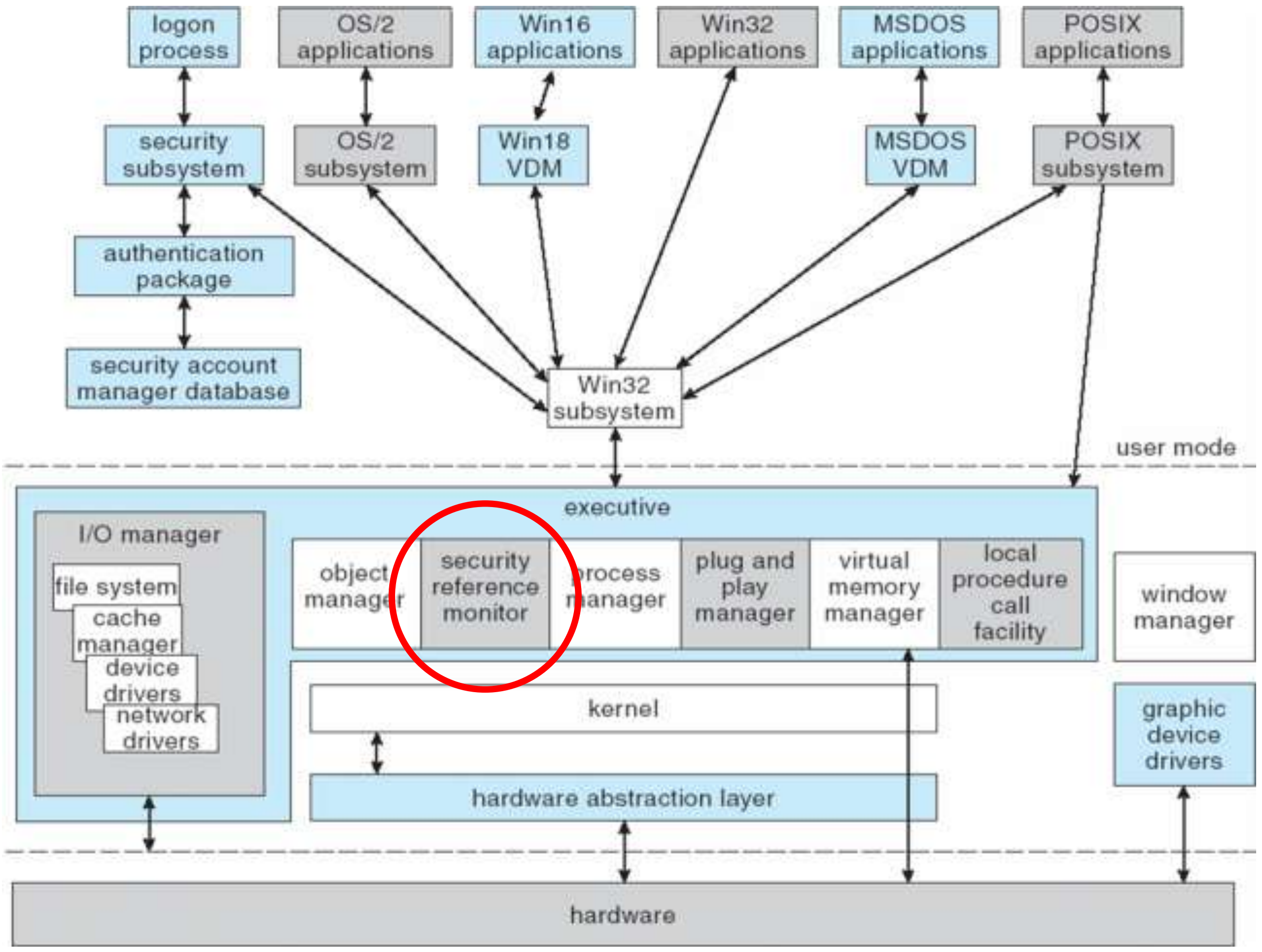
- C2 class security require specialized enhancements. Examples:
  - HP-UX (and other commercial UNIX systems) – can operate in Standard Model (legacy, standards-based UNIX) and Trusted Mode (C2 compliant)

- **Security-Enhanced Linux** – (SELinux) - a kernel module that provides Mandatory Access Control (MAC) policies, including those based on the concepts of type enforcement, role-based access control and multilevel security. SELinux is popular in systems based on Linux containers, to help further enforce isolation between containers and their host.

Protection and security in MSWin

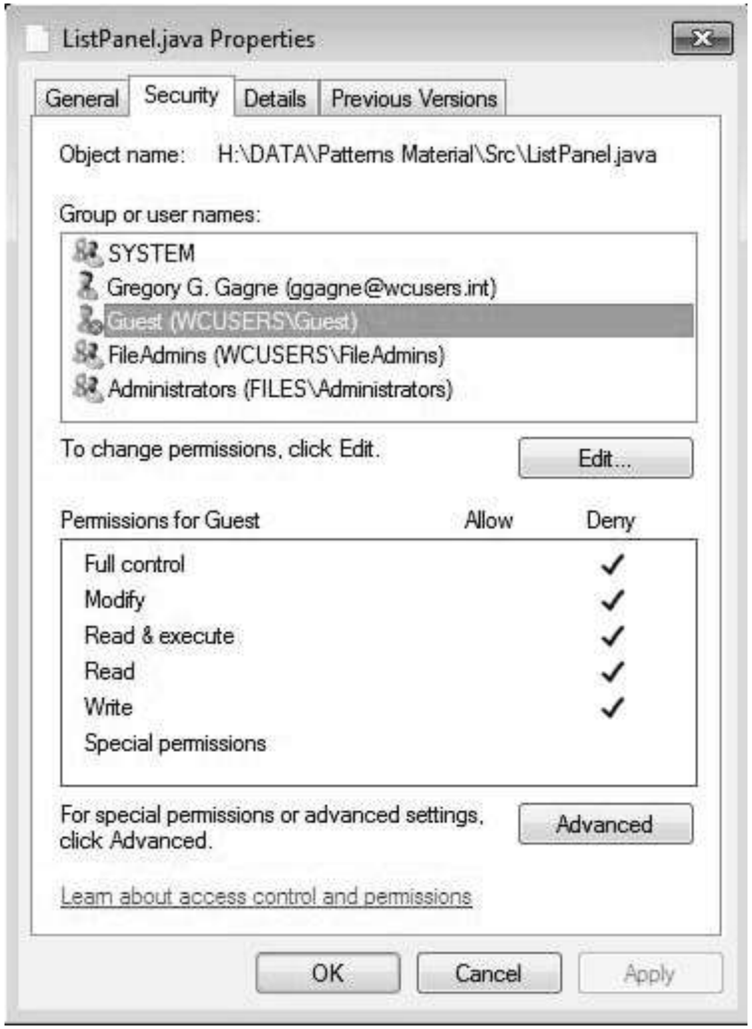
- Windows (NT family) uses objects for all its services and entities; the **object manager** supervises the use of all the objects
- Windows objects are manipulated by a standard set of methods, namely `create`, `open`, `close`, `delete`, `query name`, `parse` and `security`.
- Security is based on **user accounts**
  - Each user has unique **security ID** (SID)
  - Login to ID creates **security access token**
    - Includes security ID for user, for user's groups, and special privileges
    - Every process gets a copy of the token
- Simple subjects (user processes) inherit security settings of logged users; servers can use settings of clients (service requestors)
- Each object in Windows is associated with **security descriptor**. It contains SID of the owner and group, and access control lists: **Discretionary ACL** – set by the owner and **System ACL** – reflecting audit settings made by administrator.
- Each object is associated with a **security descriptor**; Whenever a process opens a handle to an object, the **security reference monitor** checks the process's security token and the object's access control list to see whether access is allowed or denied.
- Since Windows Vista and Windows Server 2008 Microsoft implements Mandatory Integrity Control (MIC) - to restrict the access permissions for potentially less trustworthy contexts. MIC allows classes of applications to be isolated.

Depiction of MSWin7 Architecture



NTFS — Security

- Security of an NTFS volume is derived from the Windows NT object model.
- Each file object has a security descriptor attribute stored in this MFT record.
- This attribute contains the access token of the owner of the file, and an access control list that states the access privileges that are granted to each user that has access to the file.



ListPanel.java Properties

Object name: H:\DATA\Patterns Material\Src\ListPanel.java

Group or user names:
- SYSTEM
- Gregory G. Gagne (ggagne@wcusers.int)
- Guest (WCUSERS\Guest)
- FileAdmins (WCUSERS\FileAdmins)
- Administrators (FILES\Administrators)

To change permissions, click Edit.

Permissions for Guest	Allow	Deny
Full control		✓
Modify		✓
Read & execute		✓
Read		✓
Write		✓
Special permissions		

For special permissions or advanced settings, click Advanced.

Learn about access control and permissions