



Operating Systems

Protection

Seyyed Ahmad Javadi

sajavadi@aut.ac.ir

Spring 2022

Chapter 17: Protection

- **Goals of Protection**
- **Principles of Protection**
- **Protection Rings**
- **Domain of Protection**
- **Access Matrix**
- **Implementation of Access Matrix**
- Revocation of Access Rights
- Role-based Access Control
- Mandatory Access Control (MAC)
- Capability-Based Systems
- Other Protection Implementation Methods
- Language-based Protection



Objectives

- 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.
- Describe how protection mechanisms can mitigate system attacks.



Goals of Protection

- In one protection model, computer 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.



Principles of Protection

- Guiding principle – **principle of least privilege**
 - Programs, users and systems should be given **just enough privileges** to **perform their tasks**
 - Properly set **permissions** can limit damage if entity has a bug, gets abused
 - Can be **static** (during life of system, during life of process)
 - Or **dynamic** (changed by process as needed)
 - ▶ **E.g., domain switching, privilege escalation**



Principles of Protection (cont.)

- Must consider “grain” aspect
 - Rough-grained privilege management easier, simpler, but least privilege now done in large chunks
 - ▶ For example, traditional Unix processes either have abilities of the associated user, or of root
 - Fine-grained management more complex, more overhead, but more protective
 - ▶ File ACL lists, RBAC



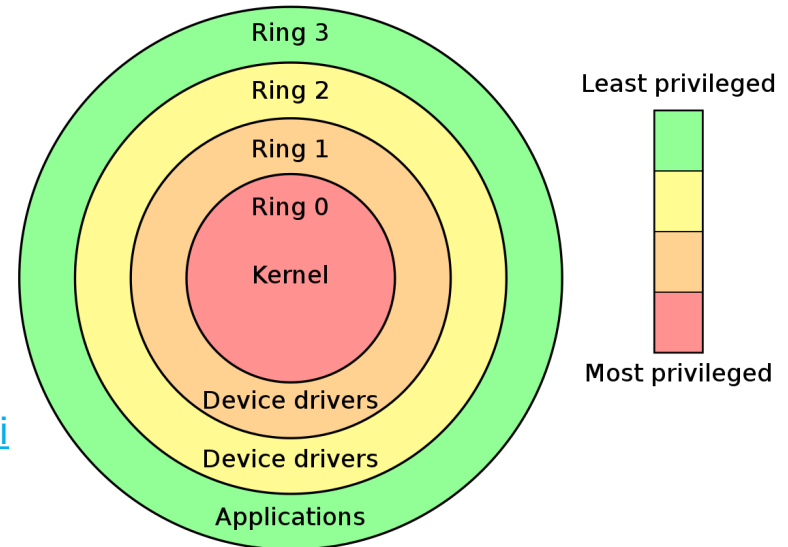
Principles of Protection (cont.)

- Domain can be user, process, procedure
- **Audit trail** – recording all protection-orientated activities, important to understanding what happened, why, and catching things that shouldn't
- No single principle is a panacea for security vulnerabilities
 - Need **defense in depth**



Protection Rings

- Components ordered by amount of privilege and protected from each other
 - For example, the kernel is in one ring and user applications in another
 - This privilege separation requires hardware support
 - Gates used to transfer between levels, for example the syscall Intel instruction
 - Also traps and interrupts



https://en.wikipedia.org/wiki/Protection_ring

Domain of Protection

- Rings of protection separate functions into domains and order them hierarchically
- Computer can be treated as **processes** and **objects**
 - **Hardware objects** (such as devices) and **software objects** (such as files, programs, semaphores)
- Process for example **should only have access to objects it currently requires to complete its task**
 - The **need-to-know** principle



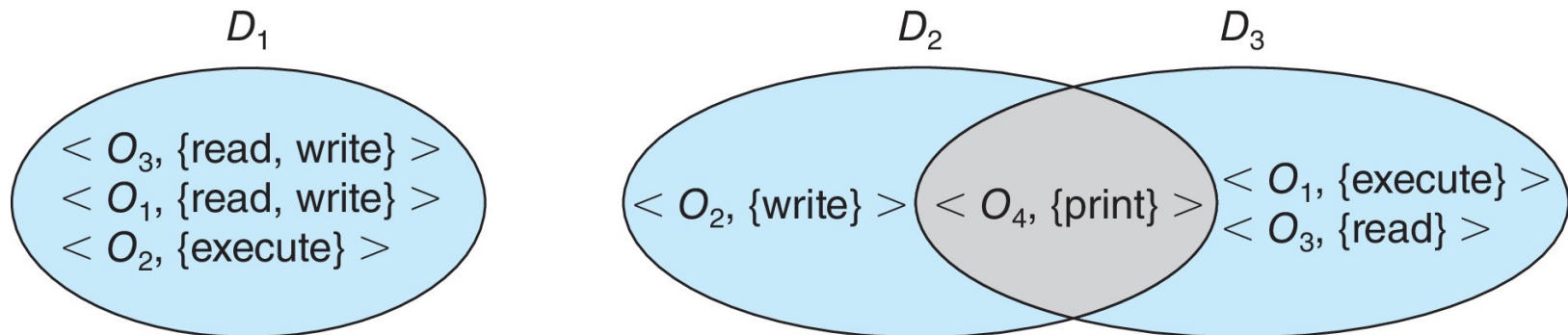
Domain of Protection (Cont.)

- Implementation can be via process operating in a **protection domain**
 - Specifies resources process may access
 - Each domain specifies set of objects and types of operations on them
 - Ability to execute an operation on an object is an **access right**
 - ▶ <object-name, rights-set>
 - Domains may share access rights
 - Associations can be **static** or **dynamic**
 - If dynamic, processes can **domain switch**



Domain Structure

- Access-right = $\langle \text{object-name}, \text{rights-set} \rangle$
 - *rights-set* is a subset of all valid operations that can be performed on the object
 -
- Domain = set of access-rights



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



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

- If a process in Domain D_i tries to do “op” on object O_j , then “op” must be in the access matrix.
- User who creates object can define access column for that object.

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

- 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

Use of Access Matrix (cont.)

- **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

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

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

- Generally, a *sparse* matrix
- **Option 1 – Global table**
 - Store ordered triples $\langle \text{domain}, \text{object}, \text{rights-set} \rangle$ in table
 - A requested operation M on object O_j within domain $D_i \rightarrow$ search table for $\langle D_i, O_j, R_k \rangle$
 - ▶ with $M \in R_k$
 - But table could be large \rightarrow won't fit in main memory
 - Difficult to group objects
 - ▶ E.g., consider an object that all domains can read.

Implementation of Access Matrix (cont.)

- Option 2 – Access control lists (ACL) for objects
 - Each column implemented as an access 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 $M \in$ default set, also allow access



Implementation of Access Matrix (cont.)

- Each column = Access-control list for one object

- Defines who can perform what operation

Domain 1 = Read, Write

Domain 2 = Read

Domain 3 = Read

- Each Row = Capability List (like a key)

- For each domain, what operations allowed on what objects

Object F1 – Read

Object F4 – Read, Write, Execute

Object F5 – Read, Write, Delete, Copy

Implementation of Access Matrix (cont.)

- Option 3 – Capability list for domains
 - Instead of object-based, list is domain based
 - **Capability list** for domain is list of objects together with operations allows on them
 - Object represented by its name or address, called a **capability**
 - Execute operation M on object O_j , process requests operation and specifies capability as parameter
 - ▶ Possession of capability means access is allowed



Implementation of Access Matrix (cont.)

- Option 3 – Capability list for domains
 - ...
 - Capability list associated with domain but never directly accessible by domain
 - ▶ Rather, protected object, maintained by OS and accessed indirectly
 - ▶ Like a “secure pointer”
 - ▶ Idea can be extended up to applications



Implementation of Access Matrix (cont.)

- Option 4 – Lock-key
 - 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



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



Comparison of Implementations (cont.)

- Many trade-offs to consider
 - ...
 - 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



Comparison of Implementations (cont.)

- 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
 - Additional accesses need not be checked
 - ▶ After last access, capability destroyed
 - ▶ Consider file system with ACLs per file

