OPERATING SYSTEMS

Protection

Goals

- Operating system consists of a collection of objects (hardware or software).
- Each object has a unique name and are accessible through some defined set of operations.
- Set policies to ensure authorized access of objects within the computer system.

Principles of Protection

Principle of Least Privilege

- Programs, users and systems should be given just enough privileges to perform their tasks.
- A component's failure or compromise causes the least amount of damage.

Mechanism vs Policy

- Mechanism is what is built into the OS for its protection.
- Policy defines what states are allowed (i.e. authorized) or not allowed (i.e. unauthorized) for a given system.

Domain of Protection

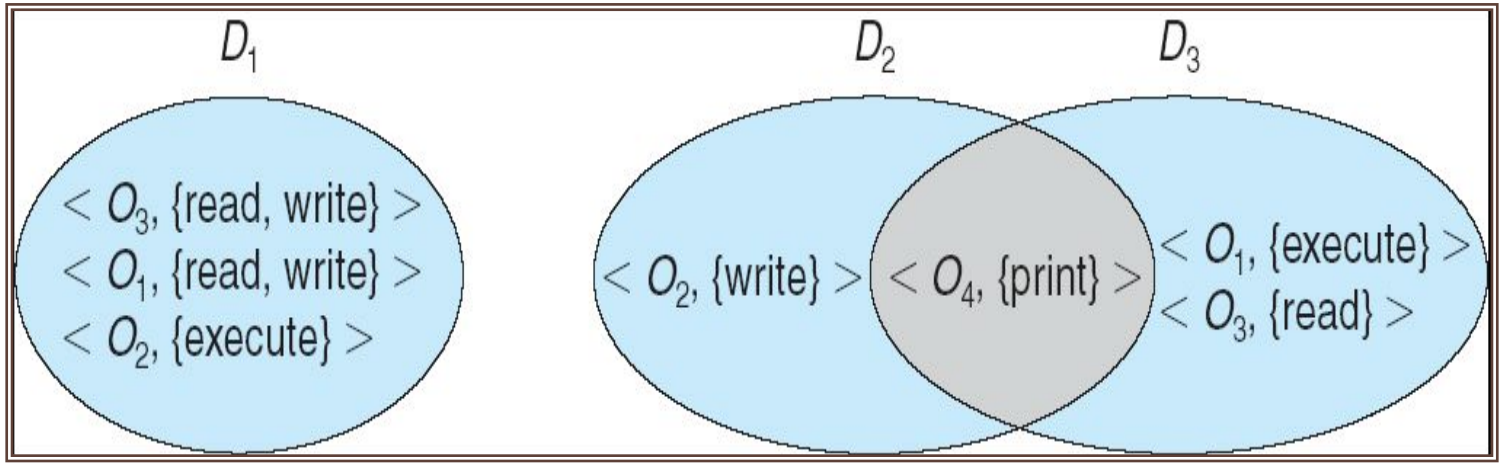
- Rings of protection separate functions into domains and order them hierarchically.
- Computer can be treated as processes and objects.
  - Hardware objects (such as devices) and software objects (such as files, programs, semaphores.
- Process for example should only have access to objects it currently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

Need-to-know-principle:

Processes should only be allowed to access resources which are necessary for completing tasks and for which they are authorized.

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



Domain Implementation in 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).

Domain Implementation in Android App IDs

- In Android, distinct user IDs are provided on a per-application basis.
- When an application is installed, the installd daemon assigns it a distinct user ID (UID) and group ID (GID), along with a private data directory (/data/data/<appname>) whose ownership is granted to this UID/GID combination alone.
- Applications on the device enjoy the same level of protection provided by UNIX systems to separate users.
- A quick and simple way to provide isolation, security, and privacy.
- The mechanism is extended by modifying the kernel to allow certain operations (such as networking sockets) only to members of a particular GID (for example, AID INET, 3003).
- A further enhancement by Android is to define certain UIDs as “isolated,” prevents them from initiating RPC requests to any but a bare minimum of services.

Access Matrix

- View protection as a matrix (access matrix).
- Columns => access-control list (ACL) for an object => represents objects.
- Rows => capability list (permissible operations on objects, per domain) => represents domains.
- Access(i, j) is the set of operations that a process executing in **Domain$_i$** can invoke on **Object$_j$**

domain \ object	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 $O_{\square}$, then "op" must be in the access matrix.
- User who creates object can define access column for that object.
- 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 $O_{\square}$ (denoted by "*")
    - control – D_i can modify $D_{\square}$ access rights
    - transfer – switch from domain D_i to $D_{\square}$
  - Copy and Owner applicable to an object.
  - Control applicable to domain object.
- 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.
- But doesn't solve the general confinement problem.

Access Matrix with Domains as Object

Switching Domains

- Domain switch from D_i to D_j is possible only if access right **switch** $\in$ **access**(**i, j**)

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			

Access Matrix with Copyrights

Copyrights

- **Asterisk** denotes that an access right can be copied within column.

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)

Process executing in domain D2 copies the read operation into another entry associated with file F2

Access Matrix with Owner Rights

Owner Rights

- **Ownership:** Can add or remove 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)

Access Matrix with Control Rights

Control Rights

- **Control:** Process executing in one domain can modify another domain.
- A process executing in domain D_2 could modify domain D_4

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			

Revocation of Access Rights

Access-List Scheme

- Search for right to be revoked, then delete.
- Immediate or Delayed.
- Selective or General.
- Partial or Total.
- Temporary or Permanent.

Capabilities

Identify capabilities before revoking them

Reacquisition • Try to reacquire after deletion	**Back-pointers:** point from object to capabilities • Expensive (used in MULTICS)
Indirection • Capability points to entry in table • Not selective	**Keys** • One key per capability • Check in global key table

Mandatory Access Control (MAC)

- Operating systems traditionally had discretionary access control (DAC) to limit access to files and other objects (for example UNIX file permissions and Windows access control lists (ACLs)).
  - Discretionary is a weakness – users / admins need to do something to increase protection.
- Stronger form is mandatory access control, which even root user can't circumvent.
  - Makes resources inaccessible except to their intended owners.
  - Modern systems implement both MAC and DAC, with MAC usually a more secure, optional configuration (Trusted Solaris, TrustedBSD (used in macOS), SELinux), Windows Vista MAC).
- At its heart, labels assigned to objects and subjects (including processes).
  - When a subject requests access to an object, policy checked to determine whether or not a given label-holding subject is allowed to perform the action on the object.

Capability-Based Systems

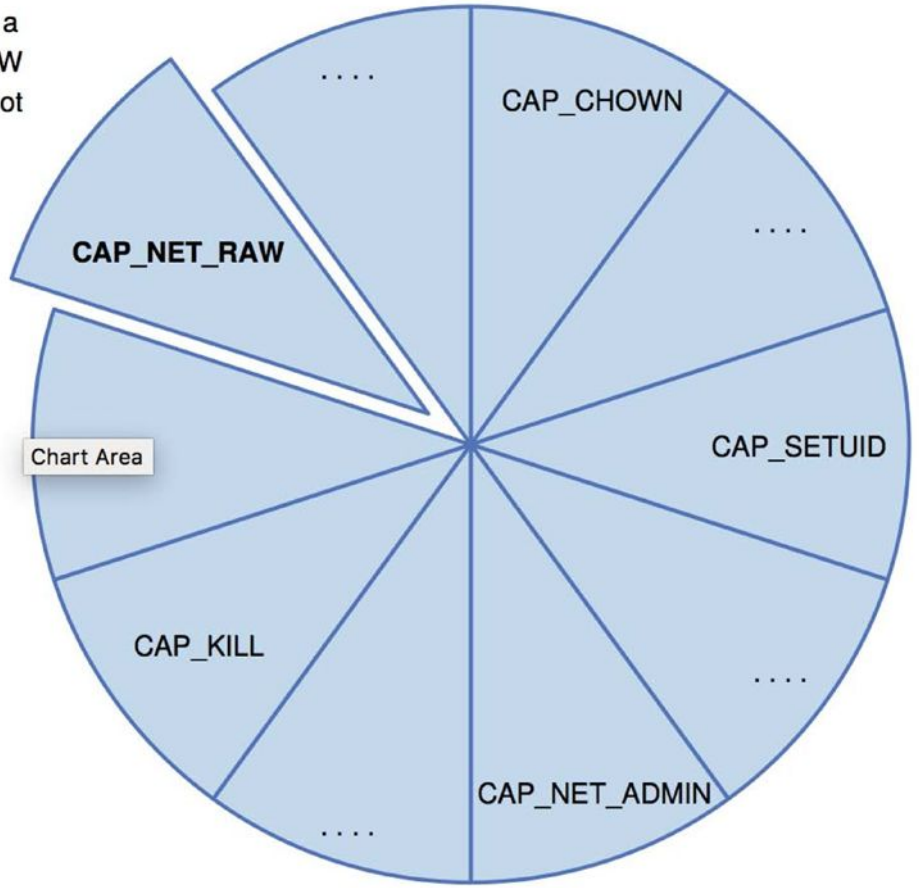
- Hydra and CAP were first capability-based systems.
- Now included in Linux, Android and others, based on POSIX.1e (that never became a standard).
  - Essentially slices up root powers into distinct areas, each represented by a bitmap bit.
  - Fine grain control over privileged operations can be achieved by setting or masking the bitmap.
  - Three sets of bitmaps – permitted, effective, and inheritable.
    - Can apply per process or per thread.
    - Once revoked, cannot be reacquired.
    - Process or thread starts with all privs, voluntarily decreases set during execution.
    - Essentially a direct implementation of the principle of least privilege.
- An improvement over root having all privileges but inflexible (adding new privilege difficult, etc.).

Capabilities in POSIX.1e

In the old model, even a simple `ping` utility would have required root privileges, because it opens a raw (ICMP) network socket

Capabilities can be thought of as "slicing up the powers of root" so that individual applications can "cut and choose" only those privileges they actually require

With capabilities, ping can run as a normal user, with CAP_NET_RAW set, allowing it to use ICMP but not other extra privileges



Other Protection Improvement Methods

- **System integrity protection (SIP):**
  - Introduced by Apple in macOS 10.11.
  - Restricts access to system files and resources, even by root.
  - Uses extended file attributes to mark a binary to restrict changes, disable debugging and scrutinizing.
  - Also, only code-signed kernel extensions allowed and configurable only code-signed apps.
- **System-call filtering:**
  - Like a firewall, for system calls.
  - Can also be deeper –inspecting all system call arguments.
  - Linux implements via SECCOMP-BPF (Berkeley packet filtering).
- **Sandboxing:**
  - Running process in limited environment.
  - Impose set of irremovable restrictions early in startup of process (before main()).
  - Process then unable to access any resources beyond its allowed set.
  - Java and .net implement at a virtual machine level.
  - Other systems use MAC to implement.
  - Apple was an early adopter, from macOS 10.5's "seatbelt" feature.
    - Dynamic profiles written in the Scheme language, managing system calls even at the argument level.
    - Apple now does SIP, a system-wide platform profile.

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.