Access Control

Introduction to Security (192.019)

Mauro **Tempesta**, Lorenzo **Veronese**

Security & **Privacy** Research Unit (192-06)
https://**secpriv.wien**

Access Control

The **decision** to permit or deny a **subject** access to system **objects** (network, data, application, service, etc.)

- **Subjects**: any entity capable of accessing resources (users, processes, …)
- **Objects**: resources that need to be protected (files, peripherals, …)
- **Access rights**: kind of access allowed to the object
  - Read / write information, execute (e.g., a program), create or delete

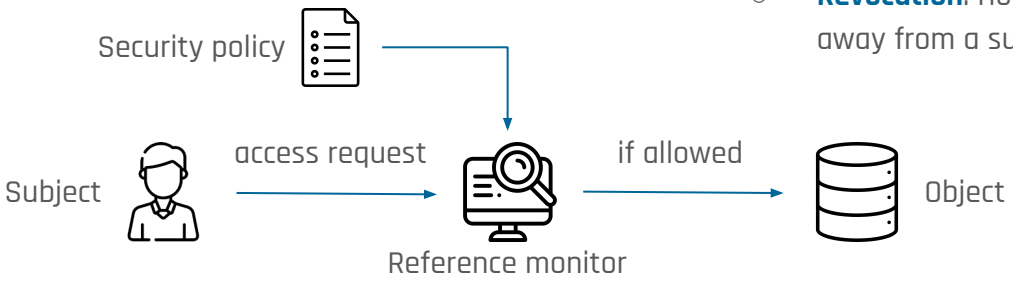
Enforcement of Access Control

Reference Monitor

- Component that authorizes / denies **access requests** to the system objects
  - Example: Component of the file system in an operating system to regulate access to files
- Required **properties**: non-bypassable, verifiable, tamper-proof

Security Policy

- **Set of rules** defining what which **access rights** subjects have on objects
- Various operational / administrative aspects:
  - **How** and **by whom** rules can be added / modified / deleted
  - **Delegation**: Can a subject with certain access rights propagate them to other subjects?
  - **Revocation**: How can access rights be taken away from a subject?

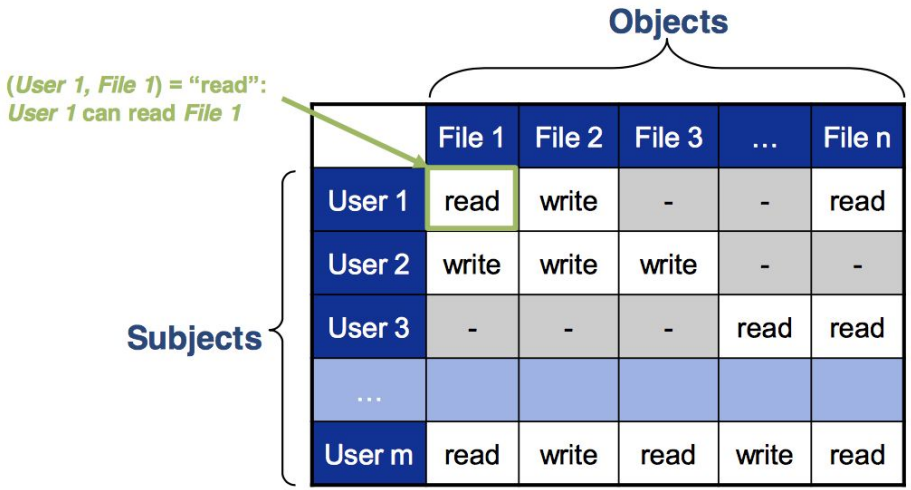


Properties of Reference Monitors

- **Non-bypassable**
  - All access requests to objects **must pass through** the reference monitor
  - Prevents unexpected violations of the enforced security policy
- **Verifiable**
  - Monitor should be amenable to analysis and tests
  - It should be verifiable that the monitor **correctly** enforces the security policy
- **Tamper-proof**
  - Correct functioning of the reference monitor **cannot be compromised** by an attacker

Storing Access Rights - Access Control Matrix

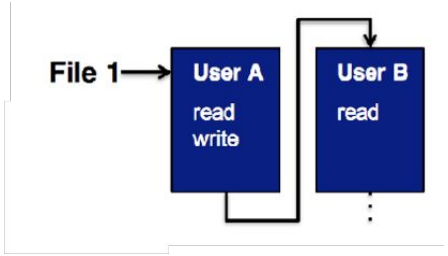
- Proposed by Lampson in 1971
- Rows represent **subjects**, columns represent **objects**
  - Intersection contains the **access rights** of the subject for the object



	File 1	File 2	File 3	…	File n
User 1	read	write	-	-	read
User 2	write	write	write	-	-
User 3	-	-	-	read	read
…					
User m	read	write	read	write	read

Storing Access Rights - ACLs and Capabilities

- **Access Control Lists (ACL)**
  - **Object-centered**: Associate each object with list
  - Reference monitor checks subject against list of accessed object
  - Relies on authentication: need to know user



- **Capabilities**
  - **Subject-centered**: Capability is unforgeable ticket (token) that defines the privileges of its holder
    - Usually implemented via random strings (cryptographically protected, e.g., via HMAC) or controlled by the operating system
    - Can be passed from one subject to another
  - Reference monitor checks only the validity of the token
    - No need to identify the subject

Delegation and Revocation in ACLs and Capability-based Systems

	ACL	Capabilities
Delegation	• Ask the owner / administrator to grant privileges to objects to the desired subject • In operating systems: Let specific processes run by one user to act with the privileges of another user	The capability can be passed to the desired subject at run time
Revocation	Modify the access rights stored in the ACLs associated to the resources to which access has to be revoked	Only possible in systems with appropriate bookkeeping • Reference monitor needs to track all revoked capabilities (until they expire) • If capability is used for multiple resources, have to revoke all or none

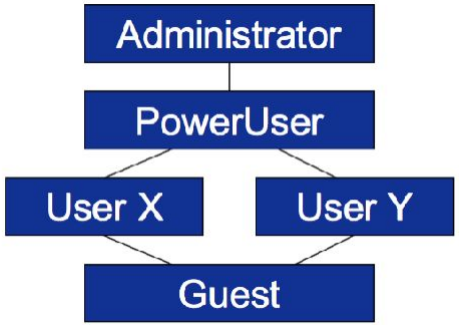
Mandatory (MAC) and Discretionary Access Control (DAC)

- **Mandatory Access Control (MAC)**
  - Security policy set and modified centrally by **trusted administrator**
  - Subjects **cannot override** the policy(e.g., delegate rights if not allowed by policy)

- **Discretionary Access Control (DAC)**
  - Subjects can freely **delegate / revoke / modify** access rights to objects for which they have certain access rights
    - Objects they own
    - Objects for which they have a capability

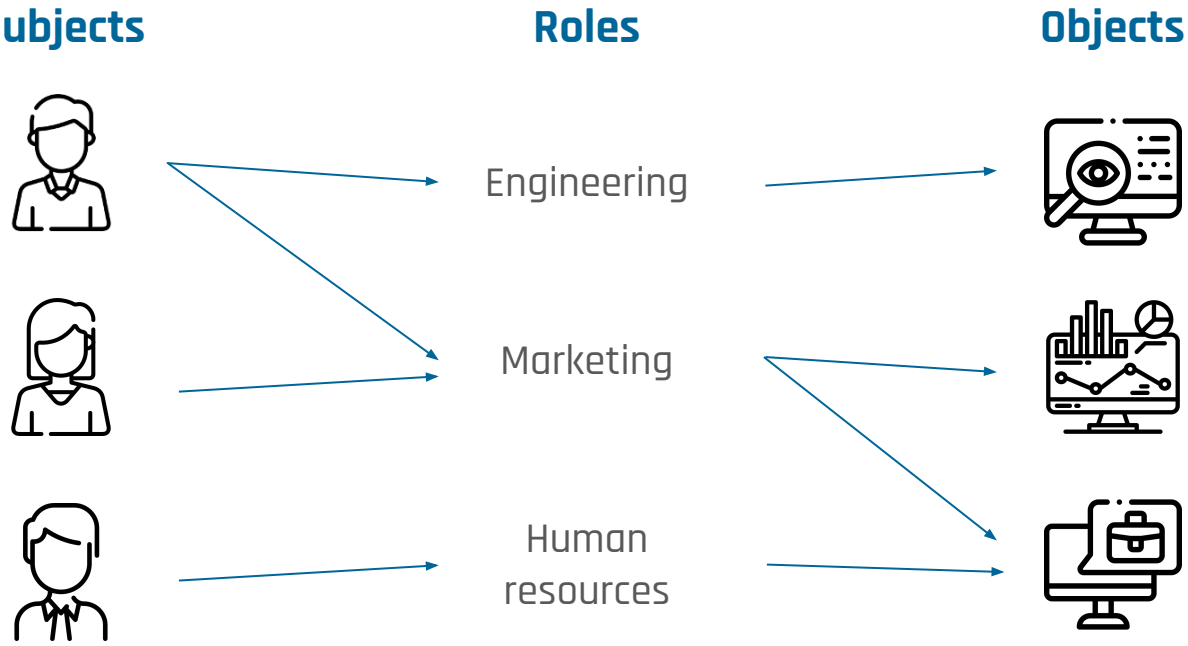
Role-Based Access Control

- Writing policies for systems with a large number of users can be difficult
  - Define **roles** (also known as **groups**)
    - A role represent **sets of subjects** with **similar access rights**
  - Assign **permissions** to **roles** and **roles** to **subjects**
    - Users obtain the access rights of the roles assigned to them



- Roles might be organized in a **hierarchy**
  - **Partial ordering** is defined over roles
  - Each role gets **all** access rights of roles below
  - Only **new** access rights are assigned to a role

Role-Based Access Control



Bell-LaPadula Model

- Model used in government and military applications
  - Developed to ensure **data confidentiality**

- Subjects and objects are assigned a **security level** and a set of **categories**
  - An **order** is defined over security levels
  - A subject can **read** an object if
    - the object has **the same or lower** security level compared to the user (“no read up”)
    - the object’s categories are a **subset** of the categories for which the subject is cleared
  - A subject can **write** an object if
    - the object has **the same or higher** security level compared to the user (“no write down”, prevents downgrade of information)
    - the object’s categories are a **superset** of the categories for which the subject is cleared

Bell-LaPadula Model - Example

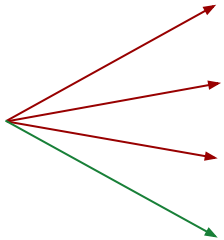
- Security Levels: Unclassified < Restricted < Confidential < Secret < Top Secret
- Categories: Planes, Troops, Submarines
- Who can read which file(s)?

User	Level	Cleared for
Sven	Secret	Submarines
Oliver	Top Secret	Planes

File	Level	Categories
warplan	Top Secret	Troops, Submarines, Planes
runway	Confidential	Planes
sonar	Top Secret	Submarines
torpedo	Secret	Submarines

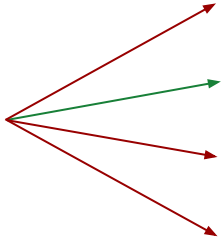
Solution

Person	Level	Cleared for
Sven	Secret	Submarines



File	Level	Categories
warplan	Top Secret	Troops, Submarines, Planes
runway	Confidential	Planes
sonar	Top Secret	Submarines
torpedo	Secret	Submarines

Person	Level	Cleared for
Oliver	Top Secret	Planes



File	Level	Categories
warplan	Top Secret	Troops, Submarines, Planes
runway	Confidential	Planes
sonar	Top Secret	Submarines
torpedo	Secret	Submarines

Other Types of Security Policies

- **Conditional Policies**
  - Temporal policies: Allow access between 10AM and 6PM
  - Context-aware policies: Allow access if current location is office

- **Biba Model**
  - Dual of Bell-LaPadula
  - Preservation of **data integrity**: “No read down, no write up”
    - Subject cannot read an object of lower security level
    - Subject cannot write an object of higher security level

Other Types of Security Policies

- **Separation of Duty**
  - Requires the involvement of multiple people to perform some task
  - Example: If amount is over $10,000, check is only valid if signed by two authorized people
    - Two people must be different
    - Policy involves role membership and inequality

- **Chinese Wall Policy**
  - Mitigate conflicts of interest in an organization
  - Prevents information flow between subjects and objects that would create a conflict of interest
  - Example: Lawyers L1, L2 in the same company
    - If company C1 sues C2,
      - L1 and L2 can each work for either C1 or C2
      - No lawyer can work for opposite sides in any case

Case study: Access Control in the UNIX File System

Access Control for UNIX Files

- Every file has an **owner** and a **group**
- File permissions represented using **three triads**
  - First triad applies to the **file owner**
  - Second triad applies to users belonging to the **file group**
  - Third triad applies to **all the other users**
- A triad specifies the **access rights** assigned to the corresponding subjects
  - Possible access rights: read (`r`), write (`w`), execute (`x`)
- Only the **file owner** and the **superuser** can change file permissions

Superuser

- Special account with **full privileges** on the system
  - Typically called `root` (user identifier = 0)
  - Can perform **any** operation on the file system, **irrespectively** of permissions

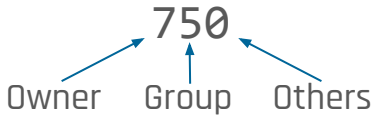
- Many systems disallow direct login as `root`
  - Authorized users can use the command `sudo` to execute programs with **`root` privileges**
  - File `/etc/sudoers` contains the list of **authorized users**
    - Possible to **restrict** the list of command that a specific user can run

Notation for Permissions

Symbolic Notation

- Standard notation used by many programs (e.g., command `ls -l`)
- Three characters for each permission triad
  1. `r` if reading is permitted, `-` if not
  2. `w` if writing is permitted, `-` if not
  3. `x` if execution is permitted, `-` if not

Octal Notation



- Typically used with the `chmod` command to set file permissions
- Each digit represents a permission triad
  - Sum of the numbers representing allowed permissions
  - 4 = read, 2 = write, 1 = execute

Which Permissions Apply?

- Resolution order for permissions
  - If the user is the file owner, apply **owner** permissions
  - Otherwise, if the user belongs to the file group, apply **group** permissions
  - Otherwise, apply the permission for **all the other users**

- If the user is the **owner** and belongs to the file **group**, owner permissions are used
  - Even if they are more restrictive than the group permissions!
  - However, the owner can change the permissions at their will

Example: Output of the `ls` Command

- Which access rights has the user `mauro` on the file `foo`?
  - Read, write, execute
- Which access rights has a user belonging to group `sp` (not `mauro`) on the file `foo`?
  - Read, execute
- Which access rights has a user who is neither `mauro` nor belongs to the group `sp` on the file `foo`?
  - No access rights

User Identifiers in UNIX

- Each process has three user identifiers (+ more under Linux)
  - **Real user ID (RUID)**
    - determines the user who **started** the process
    - initialized with the RUID of the parent process
  - **Effective user ID (EUID)**
    - determines the **permissions** of the process (used for access control checks)
    - initialized with the EUID of the parent, unless the **setuid** bit is set on the executed program
  - **Saved user ID (SUID)**
    - contains the previous EUID in case it is changed at runtime
    - initialized with the SUID of the parent process
- Each process has also three group IDs (real, effective, saved group ID) used in a similar way

`setuid` and `setgid` Bits

- `setuid` and `setgid` bits can be used to implement **delegation**
  - Bits are assigned to files containing **binary programs**
    - Cannot be set for scripts (cause of race conditions in old UNIX / Linux systems)
  - If `setuid` bit is set, the **effective user id** is set to the **owner** of the program file
  - If `setgid` bit is set, the **effective group id** is set to the **group** of the program file

- Representation in the symbolic notation
  - File has **`setuid`** bit if the third element of the **owner triad** is `s` or `S`
  - File has **`setgid`** bit if the third element of the **group triad** is `s` or `S`
  - `s` = bit is set + file is executable, `S` = bit is set, but file is not executable

Usage of `setuid` Bit: The Tool `passwd`

- Password file `/etc/shadow` can **only** be read and modified by `root`

```
-rw------- 1 root root 651 Jul 8 14:20 /etc/shadow
```

- However, users need to be able to modify their own passwords
  - This can be done with the tool `passwd`, which is owned by `root` and has the `setuid` bit set!

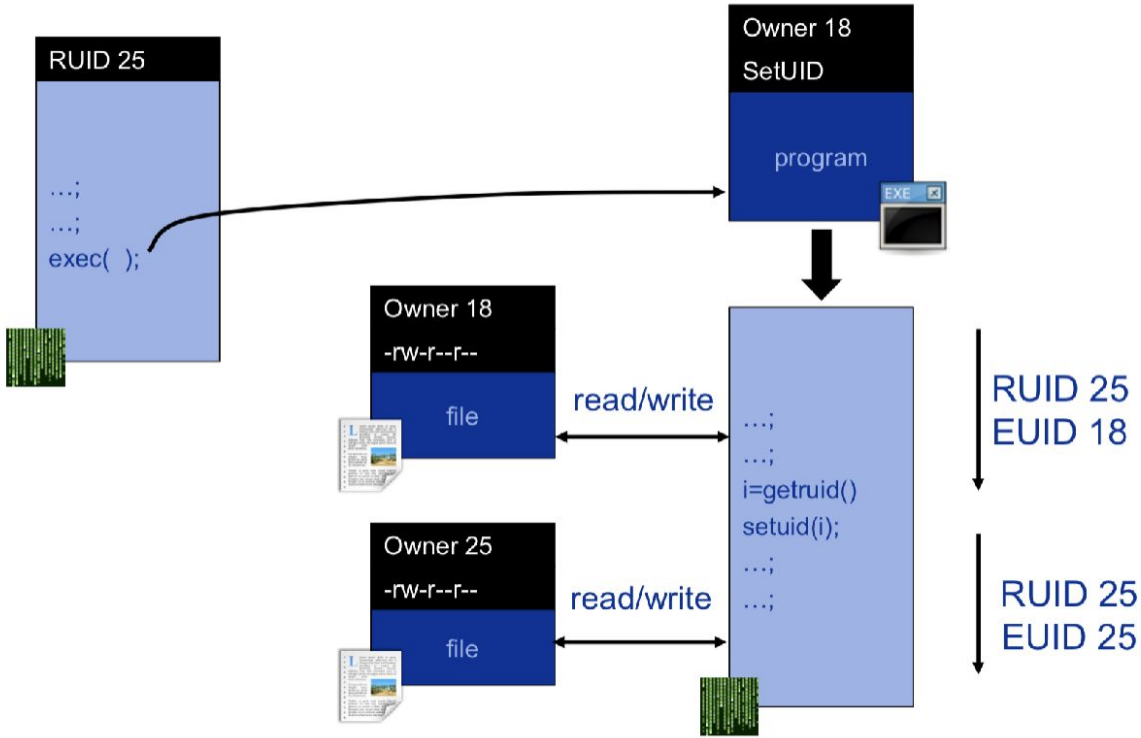
```
-rwsr-xr-x 1 root root 45604 May 10 15:24 /usr/bin/passwd
```

- Everyone can execute the program with `root` privileges and read / modify the file
  - `passwd` is designed in such a way that users can change **only their own password**
  - Exception is `root`, who can change the passwords of all users

Dynamically Changing User IDs via System Calls

- Processes can retrieve the current user IDs using various system calls
  - `getuid` (only RUID), `geteuid` (only EUID), `getresuid` (all three IDs)
- Processes can also change their user IDs
  - `seteuid(newId)` changes the effective user ID
    - For **privileged** processes (e.g., current EUID = 0), `newId` can have any value
    - For **unprivileged** processes, `newId` must be equal to current RUID, EUID or SUID
  - `setuid(newId)` similar to `seteuid`
    - For **privileged** processes, the RUID and SUID are also changed
    - Once `root` privileges are dropped, it is **not** possible to reacquire them
  - Other syscalls available: `setresuid`, `setreuid`
- Similar functions exist for group IDs (`getgid`, `setgid`, `setegid`, ...)

Example with setuid Bits and System Calls



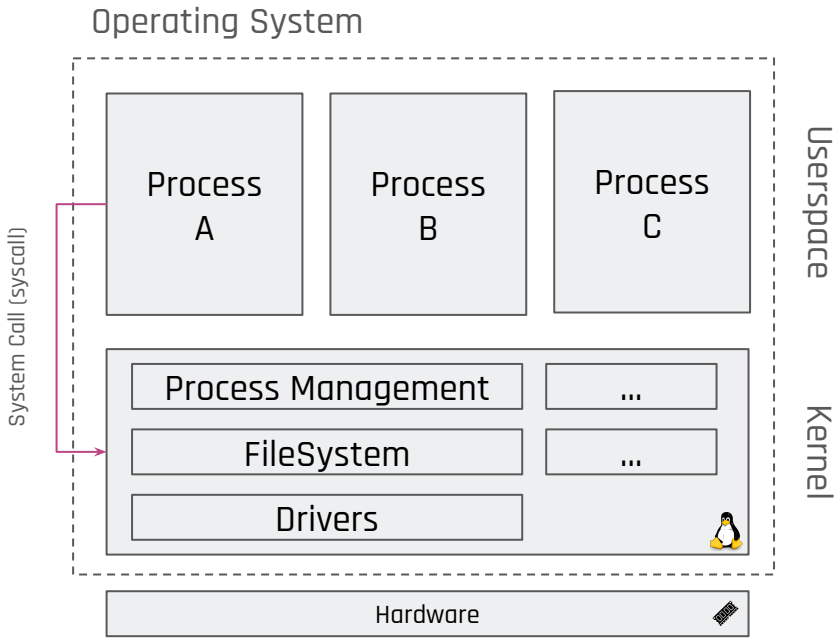
Summary of Permission Triads - Files vs. Directories

	Symbol	Effect on files	Effect on directories
Read (1st position)	r	File can be read	Directory's content can be shown (ls <dir> will print the contents)
Write (2nd position)	w	File can be modified	If x is also set, the directory's content can be modified (create, delete, rename files and directories)
Execute (3rd position)	x	File can be executed	Directory can be traversed (cd <dir> will work)
	s / S	Binary files can be executed with the permissions of the user or group owner (does not apply to scripts)	When set on the group triad (setgid), causes new directories and files to be created using the group id of the owner (instead of the group id of the current user)
	t	Sticky bit. "On Linux files in that directory may only be deleted or renamed by root or the directory owner or the file owner. The sticky bit is ignored on files"	

An Application of Access Control

Containers and Virtualization

Operating Systems tl;dr



The operating system manages hardware and software resources providing services to programs.

- For IO and memory allocation the OS acts as an intermediary between programs and hardware.
  - Programs can request OS operations using *system calls (syscalls)*
- Multi-tasking OSs allow multiple programs to run concurrently: the available processor time is divided between multiple processes (identified by a *process ID or PID*).

The OS is generally composed of:

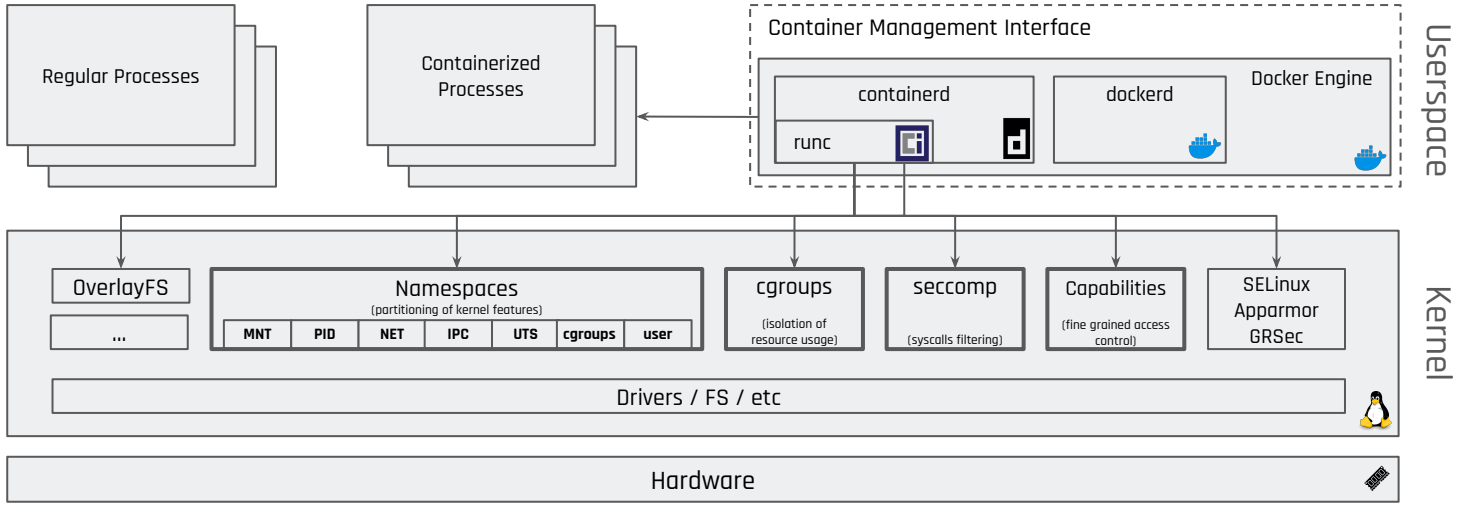
- A kernel that manages memory access, performs access control and mediates storage access.
- Userspace programs that provide the user interface to access the OS functionality (e.g., the shell).

Linux Containers

lightweight virtualization

Similar to regular processes, but:
- separate `namespace`
- separate root directory (`chroot`)
- resource constrained (`cgroups`)
- filtered syscalls (`seccomp`)
- potentially subject to additional restrictions (`capabilities`, `user namespaces`, `SELinux`, ...)

Docker
LXC
Systemd Nspawn
podman
...



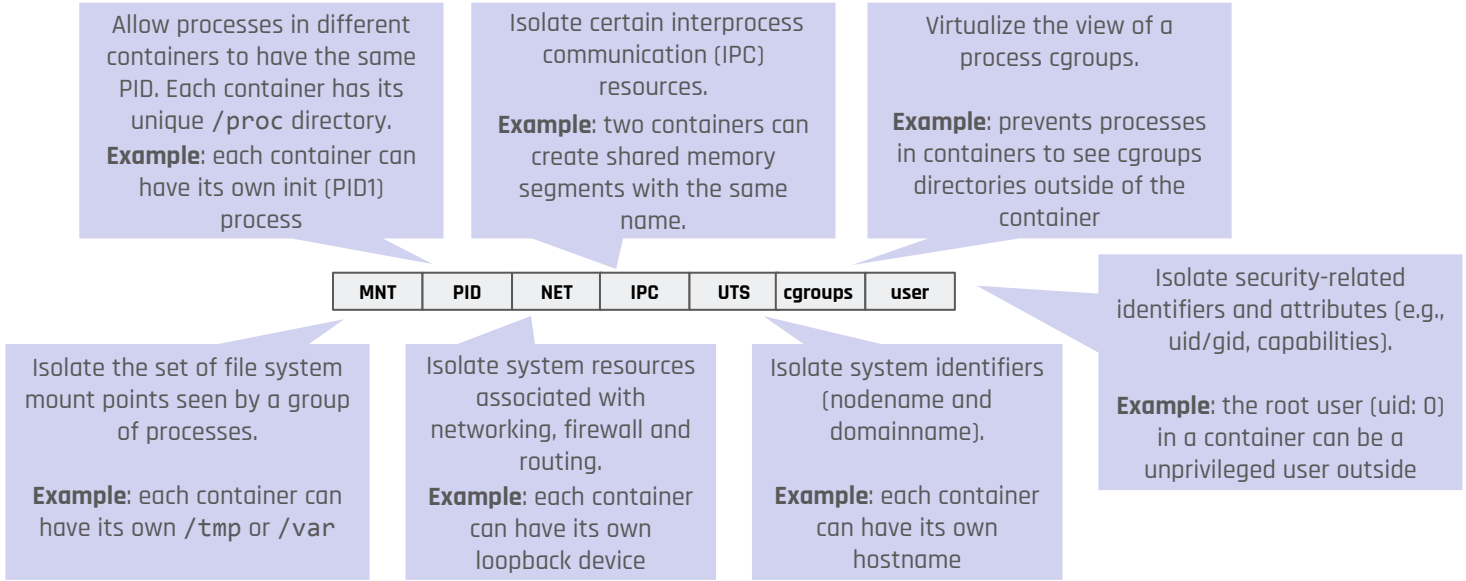
Namespaces

```
$ man 7 namespaces
```

A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource. Changes to the global resource are visible to other processes that are members of the namespace, but are invisible to other processes.

- Introduced in 2008 as an effort to reimplement OpenVZ in mainline kernel.
- The foundation of linux OS-level virtualization.
- The `unshare` command can be used to create new namespaces and execute a program in the newly created namespace.

Namespaces


Allow processes in different containers to have the same PID. Each container has its unique /proc directory.
Example: each container can have its own init (PID1) process
Isolate certain interprocess communication (IPC) resources.
Example: two containers can create shared memory segments with the same name.
Virtualize the view of a process cgroups.
Example: prevents processes in containers to see cgroups directories outside of the container
MNT
PID
NET
IPC
UTS
cgroups
user
Isolate security-related identifiers and attributes (e.g., uid/gid, capabilities).
Example: the root user (uid: 0) in a container can be a unprivileged user outside
Isolate the set of file system mount points seen by a group of processes.
Example: each container can have its own /tmp or /var
Isolate system resources associated with networking, firewall and routing.
Example: each container can have its own loopback device
Isolate system identifiers (nodename and domainname).
Example: each container can have its own hostname

Chroot

```
$ man 2 chroot
```

chroot changes the root directory of the calling process [...].
This directory will be used for pathnames beginning with /.
The root directory is inherited by all children of the calling process.

- Creates what is called a "`chroot` jail".
- The superuser can escape the `chroot` jail in various ways!
- This call does not close open file descriptors:
  - File descriptors opened before the `chroot` call may allow access to files outside the jail.

Demo

Create new mnt, pid, net, ipc and user namespace

Reset the program environment

Change the root of the `/bin/sh` program to the the `alpine` folder

```
$ wget https://dl-cdn.alpinelinux.org/alpine/v3.19/releases/x86_64/alpine-minirootfs-3.19.1-x86_64.tar.gz
$ mkdir alpine
$ tar -C alpine -xvf alpine-minirootfs-3.19.1-x86_64.tar.gz
$ unshare --map-root-user --fork --mount --pid --net --ipc --uts --user env - chroot alpine /bin/sh
# whoami
root

# mount -t proc none /proc
# ps aux
PID   USER     TIME  COMMAND
1 root         0:00 /bin/sh
5 root         0:00 ps aux

# ip link set lo up
# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
       link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
       inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
       inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
# ping -c1 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: seq=0 ttl=64 time=0.056 ms
```

After mounting the `/proc` folder, we can see that `/bin/sh` has PID 1

The new network namespace contains only the loopback interface

Cgroups

```
$ man 7 cgroups
```

> Control cgroups, usually referred to as cgroups, are a Linux kernel feature which allow processes to be organized into hierarchical groups whose usage of various types of resources can then be limited and monitored.

- Cgroups allocate CPU time, system memory, network bandwidth, or combinations of these among user-defined groups of tasks.
- Cgroups can be created by accessing the cgroup virtual filesystem (`cgroupfs`).
  - Or by using the utilities provided by `libcgroup` (`cgcreate`, `cgexec`, ...).

Seccomp

Seccomp (short for secure computing) allows a process to make a one-way transition into a secure state where it cannot make any system call except those that are explicitly allowed.

Strict Mode

The only system calls that the process is permitted to make are `read`, `write`, `exit` and `sigreturn`. Other system calls result in the termination of the process.

Filter Mode

The allowed system calls are defined by a BPF (Berkeley Packet Filter) program. The filter program returns an *action* to execute for each syscall. Actions can be, e.g., `SECCOMP_RET_KILL_PROCESS`, `SECCOMP_RET_ERRNO`, `SECCOMP_RET_ALLOW`.

Such mechanism can be abstracted by:

- The `libseccomp` library, which uses a more conventional function-call based filtering interface.
- Docker/kubernetes allow to specify a JSON profile that defines the allowed syscalls and the action to execute when other syscalls are executed.

Capabilities

```
$ man 7 capabilities
```

For the purpose of performing permission checks, traditional UNIX implementations distinguish two categories of processes: privileged processes, and unprivileged processes.
Privileged processes bypass all kernel permission checks, while unprivileged processes are subject to full permission checking based on the process's credentials (effective UID, etc).

Starting with kernel 2.2, Linux divides the privileges traditionally associated with superuser into distinct units, known as capabilities, which can be independently enabled and disabled.

- Allows to reduce the number of setuid binaries.
  - Example: the ping utility was deployed as setuid (e.g., in the `inetutils-ping` package) but is now a normal binary with only the **CAP_NET_RAW** capability in most distributions.
- Capabilities can be set and read with the `setcap` and `getcap` commands.

```
# getcap `which ping`
/usr/bin/ping cap_net_raw=ep
```

Capabilities

CAP_SYS_ADMIN	Perform a range of system administration operations, e.g. `mount`, `umount`, `sethostname`, `unshare`, etc.
CAP_NET_ADMIN	Perform various network-related operations, e.g., interface configuration, firewall configuration, modifying routing tables, etc.
CAP_NET_BIND_SERVICE	Bind a socket to Internet domain privileged ports (port numbers less than 1024).
CAP_SYS_CHROOT	Use `chroot`.
CAP_SYS_PTRACE	Trace arbitrary programs using `ptrace`.
CAP_SYS_TIME	Set system and hardware clock.
CAP_NET_RAW	Use RAW and PACKET sockets and bind to any address for transparent proxying.
...	See capabilities(7) for information of additional capabilities.

That is why Docker privileged mode is not recommended:
privileged containers run as root with all capabilities!

Note:

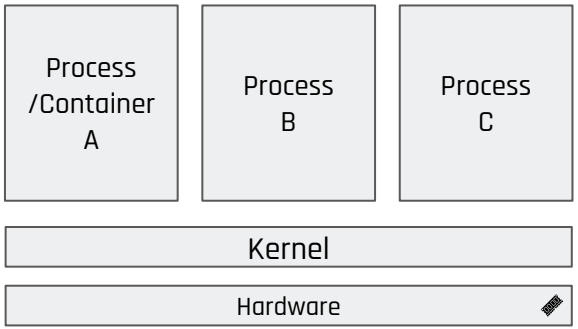
- Docker by default uses capabilities to limit the root user inside containers instead of using user namespaces.
- `gdb` does not work in Docker containers as by default containers do not have the `CAP_SYS_PTRACE` capability.

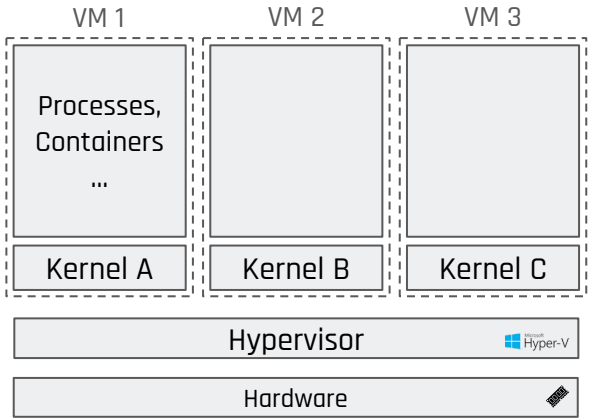
Virtual Machines

Virtualization

Virtualization allows to run multiple unmodified *guest* OSs in isolation.

- Virtual machine monitors (or hypervisors) simulate or mediate hardware access transparently.
- In hardware-assisted virtualization, the hardware (e.g., the CPU via custom instructions) provides facilities to support the hypervisor (Example: Intel VT-X or AMD-V).

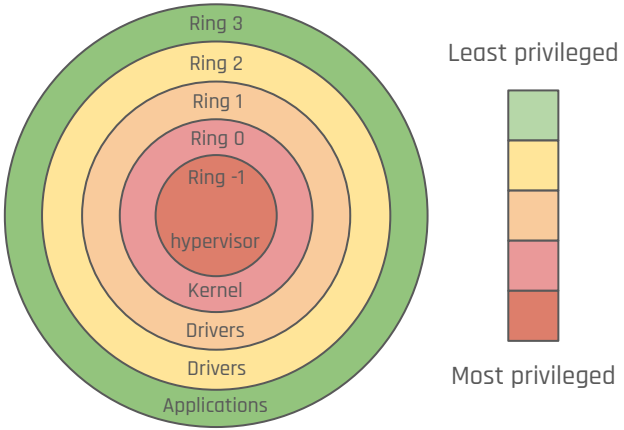




Hypervisors and Privilege Rings

Protection rings are hierarchical levels of privilege in the architecture of a computer system.

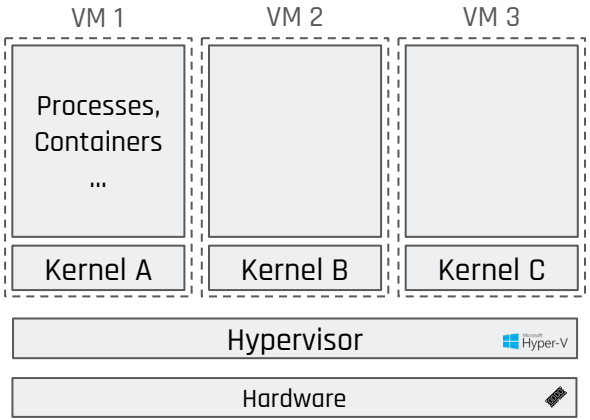
- Enforced at hardware-level by some CPU (e.g., x86)
- Ring 3 is the least privileged mode, used for userspace code.
- Ring 0 is the level with most privilege, allowing access to the physical hardware, e.g., CPU control registers and I/O controllers. This level is also called *supervisor* mode.
- Recent CPUs offer virtualization instructions for a *hypervisor* to control Ring 0 hardware access that only work at the higher privilege level of Ring -1.



Hypervisor Types

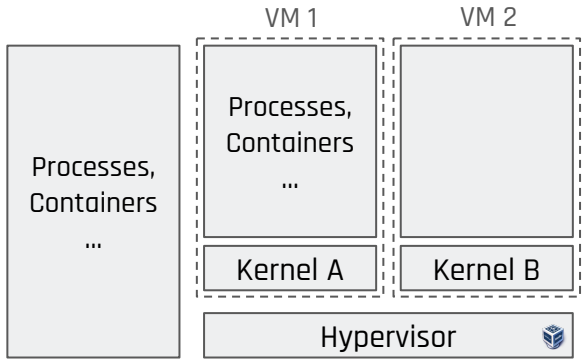
Also called bare-metal hypervisor, runs directly on the hardware and manages guest OS access.

Type 1 Hypervisor



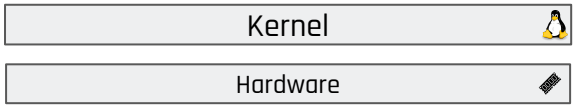
Also called hosted hypervisor, runs as a regular process on the host

Type 2 Hypervisor



Type 2 Hypervisors can be kernel modules

KVM



R .P. Goldberg "Architectural Principles for Virtual Computer Systems", 1974