Access Control

Chengyu Song

Slides modified from
John Mitch, David Wagner, and Dawn Song

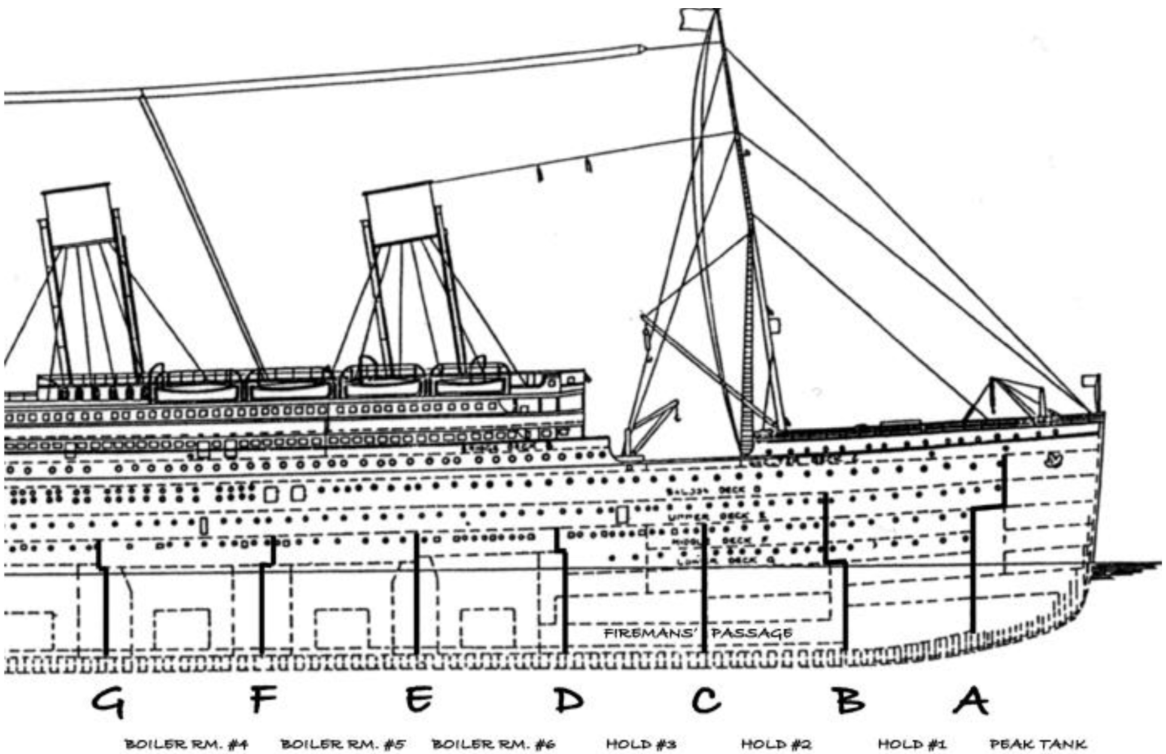
Overview of OS security

- Problems
  - An application may be a malware
  - A benign software can be compromised and becomes malicious
- Goal: how can we still protect the system?
- Approach: security system architecture that follows a set of **basic design principles**

Principles of secure design

- Compartmentation
  - Isolation
  - Principle of least privilege
- Access control
  - Complete mediation, tamper proof, correctness
- Defense in depth
- Keep it simple

First idea: compartmentation



Bulkheads & Compartments in the Bow Section

http://staff.imsa.edu/~esmith/treasurefleet/treasurefleet/watertight_compartments.htm

Compartmentation you have seen

- In ships
- In buildings
- At home
- What was the first concept you learn in the OS class?

Isolation

- Mechanisms to enforce the separation
  - **Reference monitor**
- Hardware reference monitors
  - Examples?
- Software reference monitors
  - Examples?

Software reference monitors

- Kernel access control mechanisms
- Inlined reference monitors
    - CFI, SFI, memory safety

OS access control

- Kernel reference monitor
  - Who (subject) gets to access what (object) and how (rights)
- Goal: protect the confidentiality and integrity of objects
- Older than computers, policies were created for accessing classified documents, e.g.,
  - Bell--LaPadula Model (confidentiality)
  - Biba model (integrity)

Access control in computer systems

- **Subject**: users (processes)
- **Object**: all other OS abstractions
    - Including other processes
- Rights
    - Unix: read, write, execute (`rwx`)
    - Windows: more complicated

Principles of access control

- **Complete mediation**: any access to any object should be checked by the access control system
- **Tamper proof**: data used by the access control system should be protected from illegal modification
- **Correctness**: the correctness of the access control system should be verifiable

Complete mediation

Access control matrix

Objects

Subjects

	/one	/two	/three
Alice	rw	-	rw
Bob	w	-	r
Charlie	w	r	rw

- Problem: sparse, many cells are empty

Access control lists (ACL)

- ACL: associate access rights with objects

Objects

Subjects

	/one	/two	/three
Alice	rw	-	rw
Bob	w	-	r
Charlie	w	r	rw

ACL

Capabilities

- Cap: associate access rights with subjects

Objects

Subjects

	/one	/two	/three
Alice	rw	-	rw
Bob	w	-	r
Charlie	w	r	rw

Capability

ACLs and capabilities

- Capabilities are easier to transfer
  - They are like keys, can handoff, does not depend on subject
- In practice, ACLs are easier to manage
  - Object-centric, easy to grant, revoke
  - To revoke capabilities, have to keep track of all subjects that have the capability – a challenging problem

ACLs and capabilities (cont.)

- ACL is the de-facto model for file protection. Why?
- Capabilities are widely used in mobile systems. Why?
- Hint: length of the list

Unix access control model

```
$ ll sec1.html
-rw-r--r--  1 csong  staff   3.4K Jun  2 11:31 sec1.html
```

- Unix uses ACLs: rights are associated with files
- Three sets of rights: owner, group, others

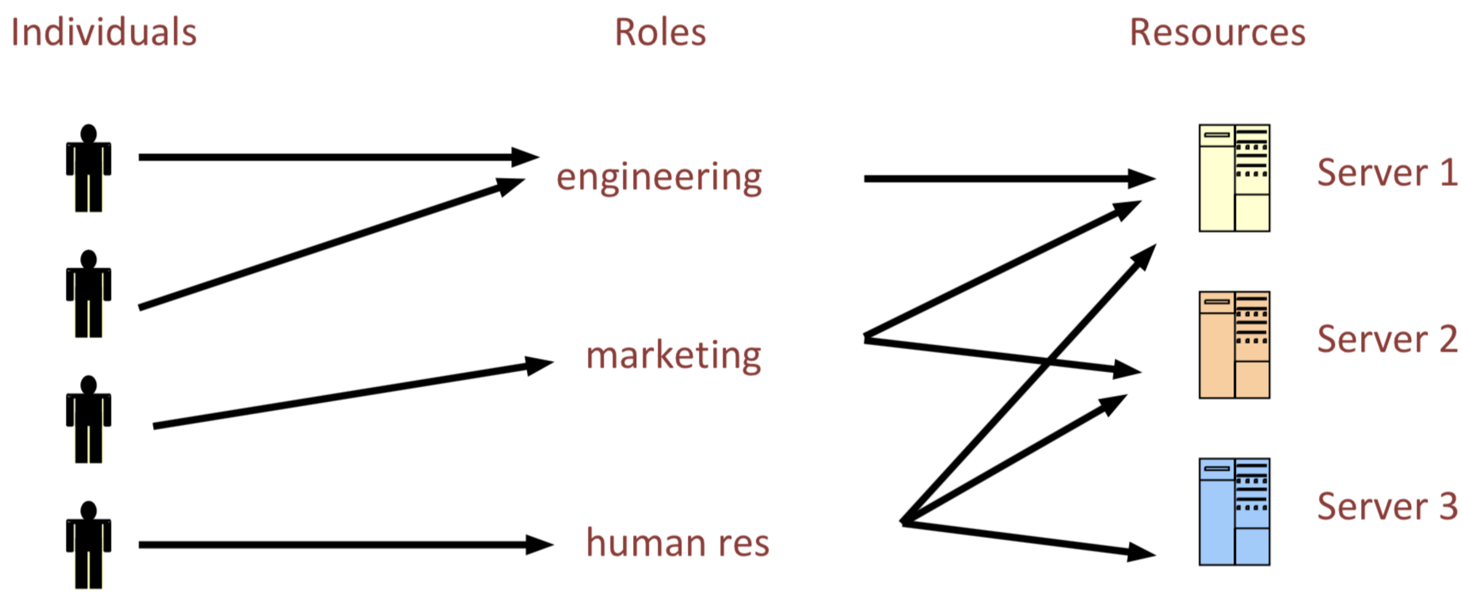
Owners

- Q: who gets to assign the access rights?
- A: the owner of the object

Groups

- Motivations: ACLs have a problem when objects are heavily shared the ACLs become very large
    - Hard to embed into inode
- Solution: put subject into **groups** (roles)
    - Administrator, PowerUser, User, Guest
    - Assign permissions to groups/roles; each user gets permission

Role-Based Access Control (RBAC)



Advantage: user's change more frequently than roles

Unix access control rights

- Files
    - Literal: read, write content and execute
- Directories
    - Read: read file names, but not attributes
    - Write: create, rename, and delete files
    - Execute: read file attributes, list files

DAC and Root

- The access control model we have discussed so far is called **Discretionary Access Control** (DAC)
  - Including both ACLs and capabilities
- Root is a special user in DAC who can **override** any existing policies e.g.,
  - Change the owner/group of any files
  - Change the access rights of any files
- This is the reason why attackers are after root

Users and processes

- **FACT**: although ACLs use *users* as subject, the OS actually treats *processes* as subjects
  - Processes act on behalf of the users, like a proxy
- Q: how to decide/change the identity (user id) of a process?
  - A1: inherited from its parent process unless
  - A2: changed by the process (via `setuid()` system call) or
  - A3: executed a setuid program

Process tree (Unix)

- The first process of the *nix system is `init`
    - Executed as root
    - Start daemons (services) and the login process
- After a successful login
    - A new shell is spawned and it changes its `uid` to the authenticated user

setuid programs

- Each process has three `uids`
  - `ruid`: real user id -> who starts the process
  - `euid`: effective user id -> used for access control
  - `suid`: saved user id -> so previous `euid` can be restored
- setuid programs
  - Once executed, changes the `euid` of the process to the owner of the file
  - Why is this useful?

Problems of DAC

- Root has unrestricted privileges
- Processes may be malicious

Mandatory Access Control (MAC)

- MAC = mandatory, so even root is checked against the policies
- Examples
  - Integrity levels on Windows
  - Capabilities on Linux
    - Same name, different meanings
    - Divide root's privileges into different capabilities so as to enforce the principle of **least privilege**
  - SELinux

Linux capabilities

- http://elixir.free-electrons.com/linux/v4.13.11/source/include/uapi/linux/capability.h#L96

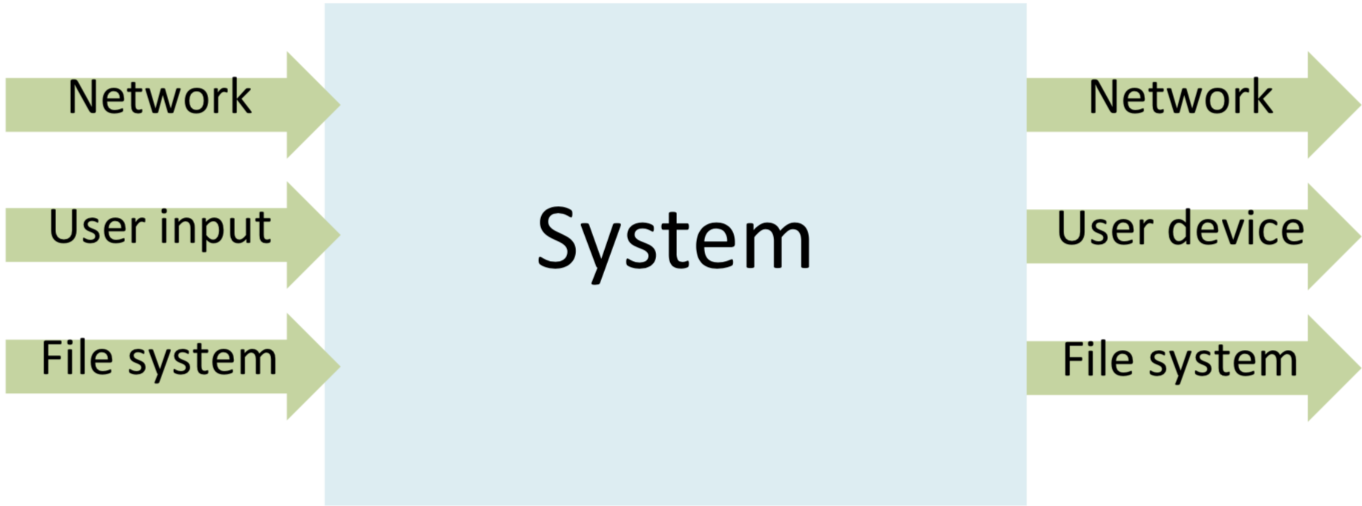
Mechanisms and Policies

- So far we've discussed are all *mechanisms*, but we still need policies to drive the mechanisms
- Q: **how to define access control policies?**

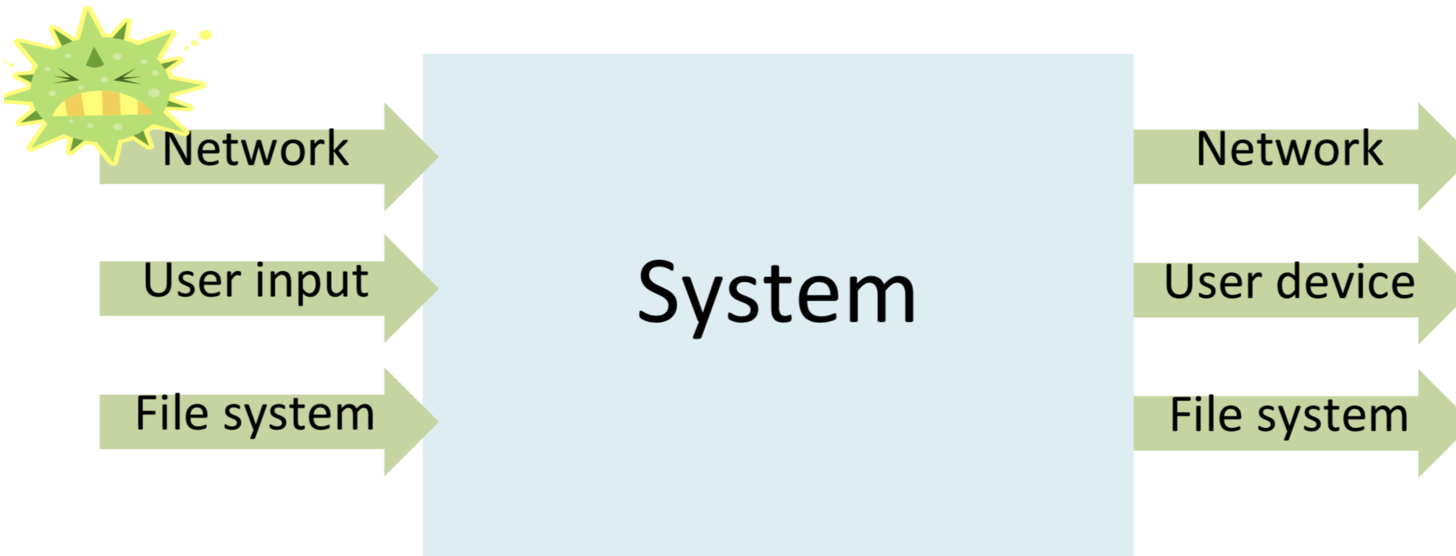
Principle of least privilege

- Privilege
    - Ability to access or modify a resource
- Principle of **Least Privilege**
    - A system module should only have the minimal privileges needed for intended purposes

Monolithic design



Problem of monolithic design



Problem of monolithic design

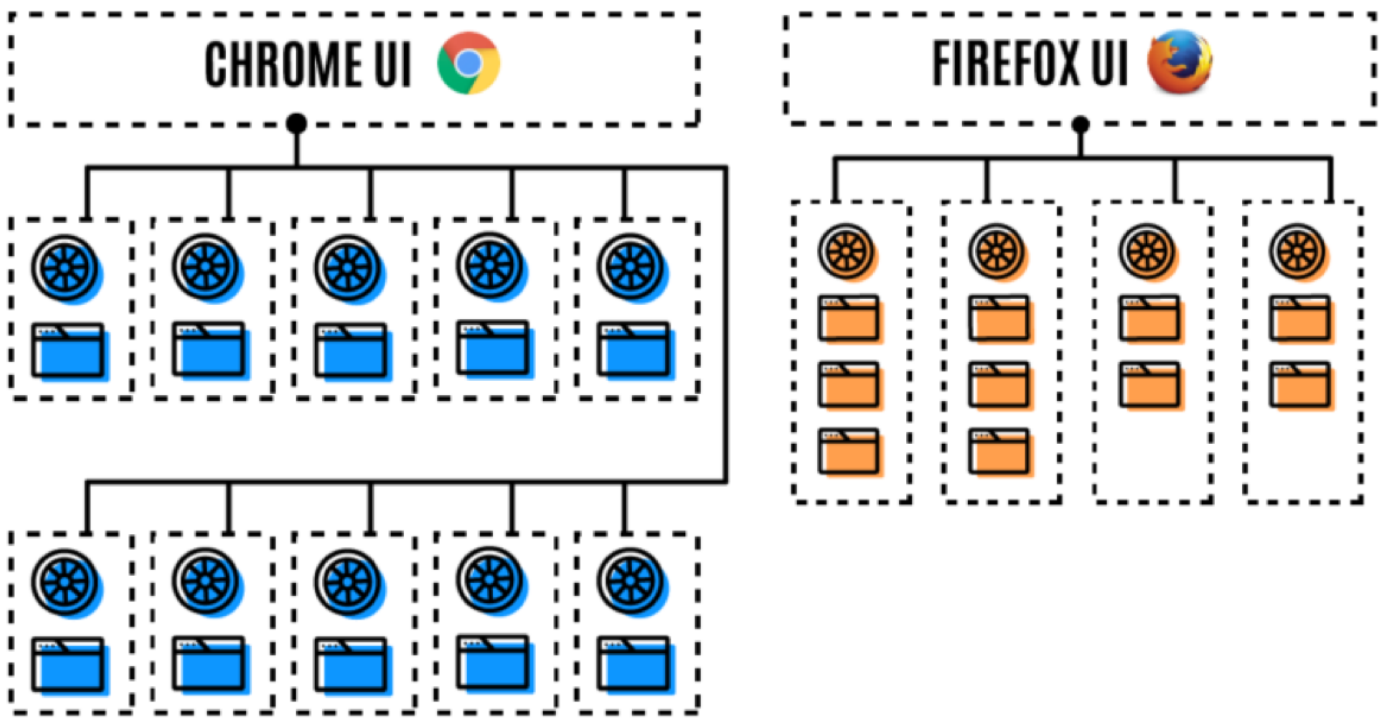
Software compartmentation

- Identify components of the program
- Identify privileges required by each component
- Group components based on privileges
- Separate and isolate
- Enforce least privilege

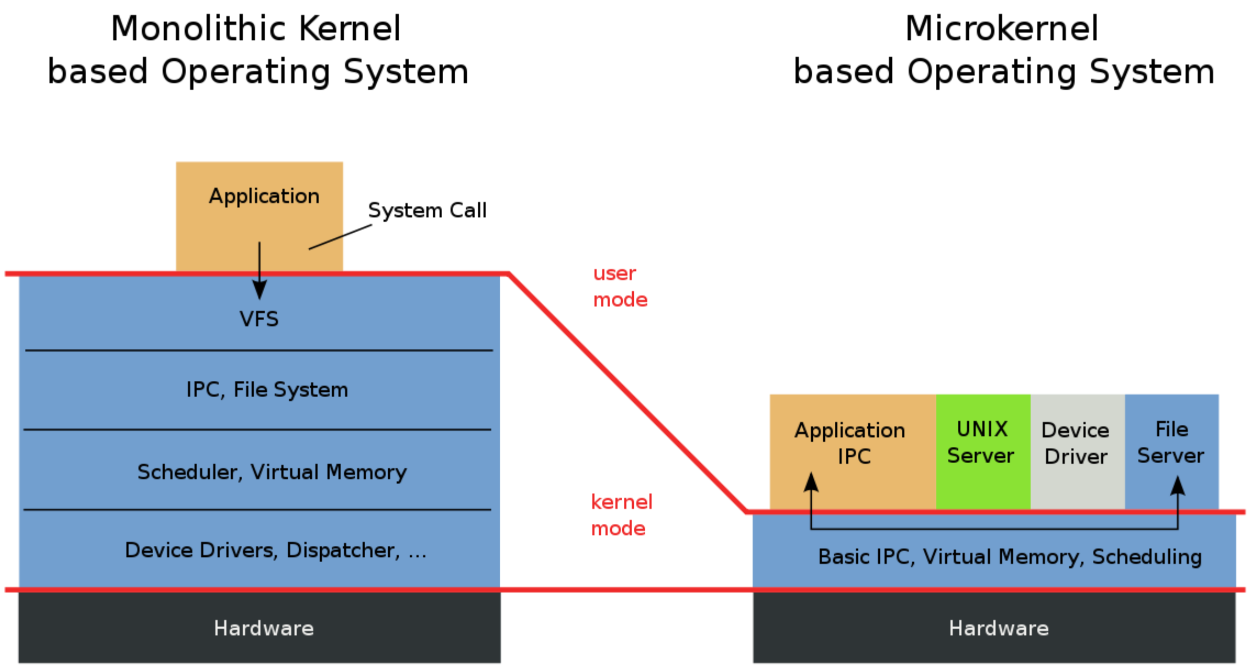
Examples

- Web Browsers
- Microkernels

Secure browser with compartmentation



Microkernel



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

Sandbox

- A controlled environment for untrusted applications, by limiting system resources they can access
    - File system
    - IPC
    - CPU and memory
    - Disk and I/O rates
    - Network access
    - Sensors (camera/microphone/GPS/...)

Mechanisms: chroot

chroot() changes the root directory of the calling process to that specified in path. This directory will be used for pathnames beginning with /. The root directory is inherited by all children of the calling process.

Only a privileged process (Linux: one with the CAP_SYS_CHROOT capability) may call chroot().

In the past, chroot() has been used by daemons to restrict themselves prior to passing paths supplied by untrusted users to system calls such as open(2).

chroot limitations

However, if a folder is moved out of the chroot directory, an attacker can exploit that to get out of the chroot directory as well. The easiest way to do that is to chdir(2) to the to-be-moved directory, wait for it to be moved out, then open a path like ../../../etc/passwd.

It is not intended to be used for any kind of security purpose, neither to fully sandbox a process nor to restrict filesystem system calls.

Mechanisms: Jail

- FreeBSD jail, an OS-level virtualization mechanism
  - Each jail is a virtual environment running on the host machine with its own files, processes, user and superuser accounts.
  - Each jail is sealed from the others
  - The limited scope of a jail allows system administrators to delegate several tasks which require superuser access without handing out complete control over the system
- https://www.freebsd.org/cgi/man.cgi?query=jail&format=html

Mechanisms: namespaces

- Linux namespaces (similar to jail): virtualization and isolation
  - Cgroup: Cgroup root directory (resources quota)
  - IPC: System V IPC, POSIX message queues
  - Network: Network devices, stacks, ports, etc.
  - Mount: Mount points
  - PID: Process IDs
  - User: User and group IDs
  - UTS: Hostname and NIS domain name

Mechanisms: more OS-level virtualization mechanisms

https://en.wikipedia.org/wiki/Operating-system-level_virtualization

- Docker, containers, LXC, etc

Mechanisms: reusing DAC & MAC

- Windows: security tokens, job object, desktop object, integrity level
- Linux: DAC, capabilities, SELinux

Examples: iOS sandbox

- Mandatory Access Control Framework (MACF)
- Apple Mobile File Integrity (AMFI)
- Entitlements
- Permission system

Examples: Android sandbox

- UID-based isolation
- SEAndroid
- Binder
- Permission system

Caveats

- **What is intended behavior?!!**
    - User's vs app's
- Performance degradation
    - WHy?
- Inter-component interface design
    - Why we need Inter-component communication?
    - What can go wrong?