CSci 4271W
Development of Secure Software Systems
Day 12: OS security: isolation and protection

Stephen McCamant (he/him)

University of Minnesota, Computer Science & Engineering

Operating systems

- The goal of an operating system is to provide a uniform platform for programs to access system resources.
- The security goal of an operating system is to prevent processes from inappropriately accessing resources used by other processes.
- In order to do this, the OS must also protect itself from the processes it manages.

Operating systems

An OS broadly provides three kinds of security functions:

- Authentication: linking processes to users
- Access Control: making decisions about access to resources
- Protection: enforcing access control policies

Outline

Isolation

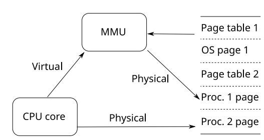
Isolation and protection, the basic mechanisms by which the OS implements security controls, usually rely on hardware mechanisms.

Hardware access control consists of two mechanisms:

- Address translation
- Supervisor mode/rings

Address translation

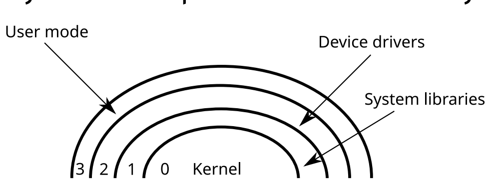
Processes access memory using virtual addresses that are translated into physical addresses by the MMU/page table:



The OS manages the translation so that each process sees only its own data.

Modes/rings

On modern processors, user programs are prevented from changing page tables or using physical addressing by using two or more protection "modes" or "rings"

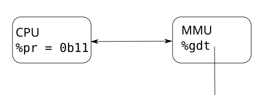


Using rings for isolation

The global descriptor table holds the ring number for each memory segment, sr.

The status register holds the current ring level pr. If request has sr < pr a fault is raised.

MMU translation tables, device buffers, etc. are ring 0.



segment	sr	base
0x001	00	0xFEBCDA09
0x0C3	11	0xF7A00000

Access to system resources requires a system call.

Moving inward

The OS maintains an interrupt table (exception vector) mapping interrupts to handlers

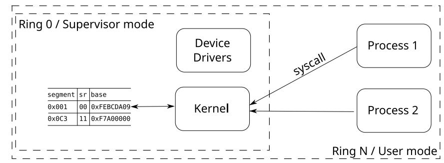
Typical interrupts include traps, I/O, timers, page faults, `sysenter/syscall/svc`.

IRQ	address
`0x0E`	`0xC032FF00`
`0x80`	`0xC0108328`

The interrupt causes the handler to be called at a lower ring level. The handler resets the status register on exit.

Rings for Unix

Typical Unix implementations make use of just two rings:



System calls use an exception to enter ring 0, check access, allocate resources, return to user mode.

Outline

- OS security overview
- Basic isolation mechanisms
- Announcements intermission
- More tools for isolation
- Midterm debrief, cont'd

Upcoming assignments

- Homework 3 due tonight, submission now available
- Section drafts for project 1 due Thursday
- Piazza has lab Q&A, project clarifications, find-a-teammate area

Outline

- OS security overview
- Basic isolation mechanisms
- Announcements intermission
- More tools for isolation
- Midterm debrief, cont'd

Mandatory access controls

- Operating systems provide or deny access to resources based on access control policies.
- Regular file permissions are discretionary access controls — they are set, and can be changed, by subjects (using `chmod`, etc.).
- Many OSes provide mechanisms for mandatory access controls which cannot be changed by subjects.

DAC vs. MAC

DAC	MAC
Controlled by owner	Controlled by admin
Users are trusted	Avoid trusting users
All processes with the same UID have the same access	Processes have user and file-specific labels

Examples of modern MAC frameworks: SELinux (used on Android/ChromeOS) and AppArmor (Ubuntu and others)

AppArmor example

AppArmor uses a "profile." For a (set of) executable(s):

- What files can be accessed?
- What permissions do child processes have?
- What capabilities can the process have?

```
/usr/sbin/tcpdump {
  capability net_raw,
  capability setuid,
  capability dac_override,
  network raw,
  network packet,
  capability sys_module, # for -D
  @{PROC}/bus/usb/ r,
  @{PROC}/bus/usb/** r,
  audit deny @{HOME}/.* mrwkl,
  audit deny @{HOME}/.*/ rw,
  audit deny @{HOME}/.*/** mrwkl,
  @{HOME}/ r,
  @{HOME}/** rw,
  /usr/sbin/tcpdump r,
}
```

Stronger isolation

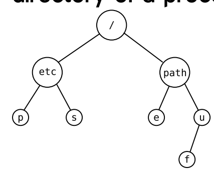
OSes have many features by which processes interact:

- The filesystem
- Interprocess communication
- Networking
- Devices
- Other kernel data structures

Other isolation mechanisms apply to some or all of these

Filesystem isolation

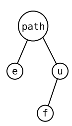
The system call `chroot("/path")` resets the root directory of a process file system to `/path`.



A chrooted process can't access files outside of its directory subtree.
All required libraries, config files, and binaries must be present in the subtree.

Filesystem isolation

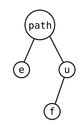
The system call `chroot("/path")` resets the root directory of a process file system to `/path`.



A chrooted process can't access files outside of its directory subtree.
All required libraries, config files, and binaries must be present in the subtree.

Filesystem isolation

The system call `chroot("/path")` resets the root directory of a process file system to `/path`.



A chrooted process can't access files outside of its directory subtree.
All required libraries, config files, and binaries must be present in the subtree.

Calls to `chroot` are only allowed if `euid` = 0: why?

`chroot` limitations

A chroot can be escaped if other processes with same UID are running, or open file descriptors refer to outside files, or directories are moved, ...

Partial mitigations:

- `setuid`: set UID of chrooted process to a new UID
- `rlimit`: limit the number of file descriptors, memory, etc., a process can access

But the man page no longer recommends it for security

System call isolation

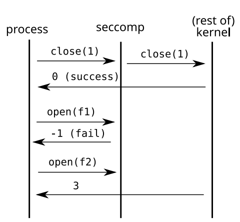
Linux, BSD, macOS, Windows all provide similar “sandboxing” frameworks that can inspect/alter system calls.

On Linux, `seccomp-bpf` uses BPF bytecode to manipulate syscalls:

```
#define Allow(syscall) \
BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, SYS_##syscall, 0, 1), \
BPF_STMT(BPF_RET+BPF_K, SECCOMP_RET_ALLOW)
struct sock_filter filter[] = {
BPF_STMT(BPF_LD+BPF_W+BPF_ABS, SYSCALL_NUM_OFFSET),
Allow(brk), // allow heap extension
Allow(close), // allow closing files!
Allow(openat), // to permit openat(config_dir), etc.
BPF_STMT(BPF_RET+BPF_K, SECCOMP_RET_TRAP), // or die
}
```

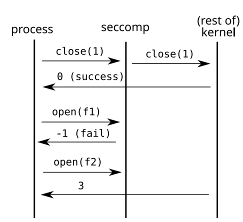
Syscall filtering pitfalls

`seccomp-bpf` has only limited support for filtering by arguments, but it would be hard to do so safely anyway



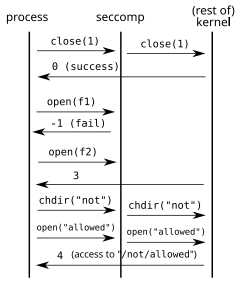
Syscall filtering pitfalls

`seccomp-bpf` has only limited support for filtering by arguments, but it would be hard to do so safely anyway



Race conditions:
Contents of `f2` can change after seccomp check;
If `c2 = "/tmp/foo"`, another process could link `/tmp/foo` → `/not/allowed`

Syscall filtering pitfalls, cont'd



Shadowing

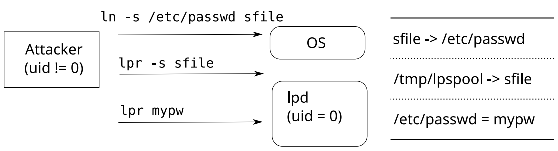
If allowed calls change the process state, the sandbox needs to "shadow" this state to make proper judgments.

Containers

- Linux "cgroup" and "namespace" features flexibly limit resource usage and visibility between groups of processes
  - Applies to filesystems, processes, memory, UIDs, networking, etc.
- "Container" describes systems that build on these mechanisms (LXC, Docker, etc.) and analogues on other systems
- Containers running in different namespaces ultimately share kernel code and devices but cannot directly interact.
  - Unless the kernel or containerization code have bugs!

Recall: confused deputies

When a process needs some privileges (e.g., of a UID), and can be confused into using other privileges the UID.



(How) could AppArmor, `seccomp`, or containers help with this specific example?

Outline

- OS security overview
- Basic isolation mechanisms
- Announcements intermission
- More tools for isolation
- Midterm debrief, cont'd

Q2: defensive programming

(Code shown outside slides)

Q3: memory corruption

(Code shown outside slides)