CSE 127: Introduction to Security

Isolation

George Obaido

UCSD

Winter 2022 Lecture 6

Some slides from Kirill Levchenko, Stefan Savage, Nadia Heninger, Stephen Checkoway, Hovav Shacham, David Wagner, Deian Stefan, Dan Boneh, and Zakir Durumeric

Principles of secure system design

1. Least privilege
2. Privilege separation
3. Complete mediation
4. Fail safe/secure
5. Defense in depth
6. Keep it simple

1. Principle of Least Privilege

1. Principle of Least Privilege

- Users should only have access to the data and resources needed to provide authorized tasks

Principle of Least Privilege

- Users should only have access to the data and resources needed to provide authorized tasks

- Examples:
  - Faculty can only change grades for classes they teach
  - Only employees with background checks have access to classified documents

2. Principle of privilege separation

2. Principle of privilege separation

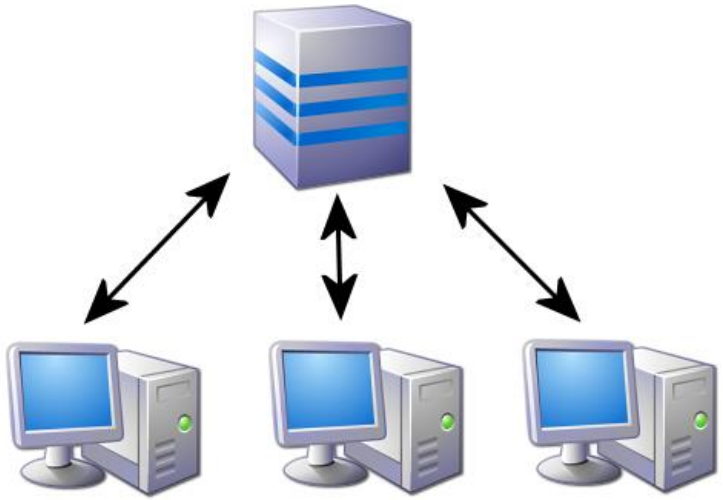
Least privilege requires dividing a system into parts to which we can limit access:

- Break system into compartments
- Ensure each compartment is isolated
- Ensure each compartment runs with least privilege
- Treat compartment interface as trust boundary

Example: Multi-user operating system

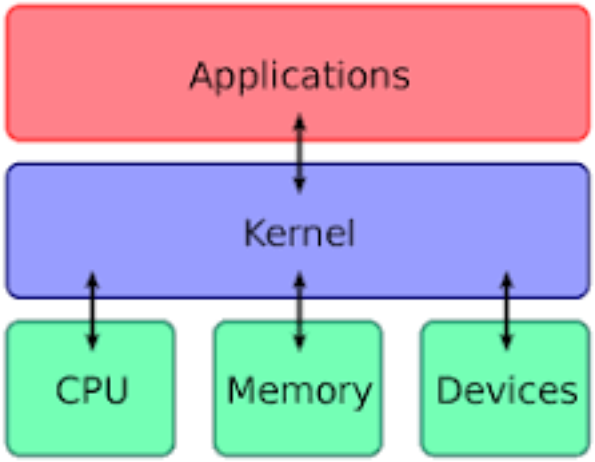
In this system:

- Users can execute programs/processes
- Processes can access resources



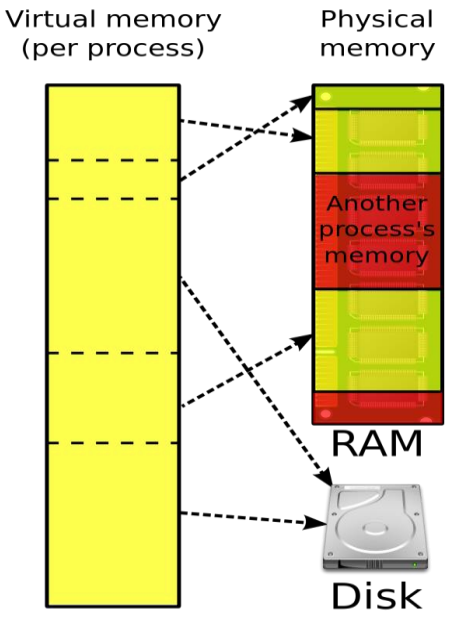
Multi-user OS security properties

- **Memory isolation**
  - Process should not be able to access another process's memory.
- **Resource isolation**
  - Process should only be able to access certain resources.



Process memory isolation

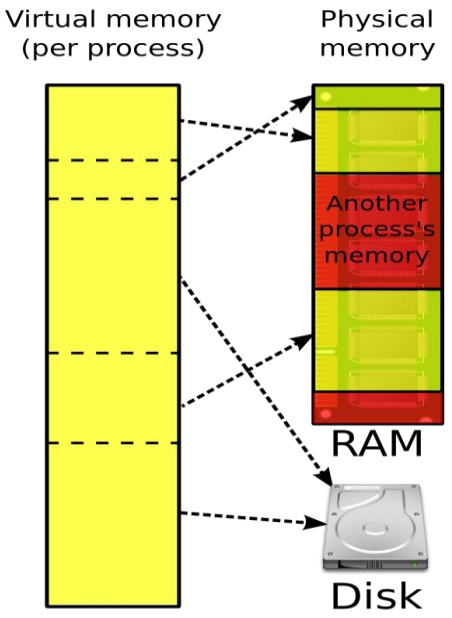
- How are individual processes memory-isolated from each other?
  - Each process gets its own virtual address space, managed by the operating system
- Memory addresses used by processes are virtual addresses (VAs) not physical addresses (PAs)
  - The CPU memory management unit (MMU) does the translation



3. Principle of complete mediation

3. Principle of complete mediation

- Every memory access goes through address translation
  - Load, store, instruction fetch
  - Virtual memory allows address space much larger than physical memory
  - Also means that operating system mediates all process memory accesses and enforces access control policy



Resource isolation in the Unix security model

In Unix, everything is a file: files, sockets, pipes, hardware devices...

- **Permissions** to access files are granted based on user IDs
  - Every user has a unique UID
- **Access Operations**: Read, Write, Execute
- Each file has an access control list (ACL)
  - Grants permissions to users based on UIDs and roles (owner, group, other)
  - root (UID 0) can access everything

Role-Based Access Control

In a general access control system we can specify permissions in a matrix:

	hw/	exams/	grades/	lectures/
cse127-instr	r/w	r/w	r/w	r/w
cse127-tas	r/w	read	-	r/w
cse127-students	read	-	-	read
cse-students	-	-	-	read

ACLs vs. Capabilities

ACL: System checks where subject is on list of users with access to the object.

- Permissions stored by column of access control matrix

Capabilities: Subject presents an unforgeable ticket that grants access to an object. System doesn't care who subject is, just that they have access.

- Row of access control matrix

Unix file permissions are a simplified ACL

```
rabeshi@login:/cse/htdocs/classes/wi21/cse127-a$ ls -l total 32
-rw-rw-r-- 1 rabeshi cse127-a-wi 18660 Jan 14 00:34 index.html

drwxrwxr-x 2 rabeshi cse127-a-sp  4096 April 13 08:42 pa
drwxrwxr-x 2 rabeshi cse127-a-sp  4096 April 13 19:57 resources
drwxrwsr-x 3 rabeshi cse127-a-sp  4096 April 14 00:34 slides
```

- Permissions grouped by user owner, group owner, other
- **Operations:** read, write, execute

Process UIDs

Process permissions are determined by UID of user who runs it unless changed.

- Real user ID (RUID)
  - Used to determine which user started the process
  - Typically **same** as the user ID of parent process
- Effective user ID (EUID)
  - Determines the permissions for process
  - Can be different from RUID (e.g. because `setuid` bit on the file being executed)
- Saved user ID (SUID)
  - EUID prior to change

Demo:
https://linuxhint.com/difference-between-real-effective-user-id-in-linux-os/

setuid

- A program can have a setuid bit set in its permissions
- This impacts fork and exec
  - Typically inherit three IDs of parent
  - If setuid bit set: use UID of file owner as EUID

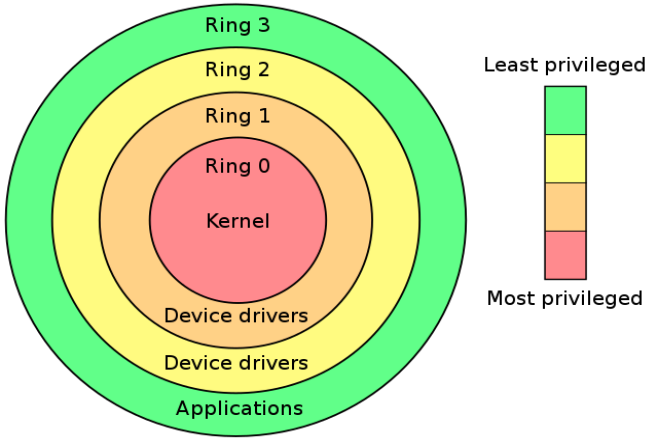
```
-rwsr-xr-x 1 root root 54256 Mar 26  2019 $ ls -ltr /usr/bin/passwd
-rwsr-xr-x 1 root root 54256 Mar 26  2019 $ passwd
```

Overview of Unix file security mechanism

- **Pro**: Simple and flexible
- **Con**:
  - Nearly all system operations require root access.
  - In practice, common to run many services as root. This violates principle of least privilege and increases attack surface.

Kernel isolation

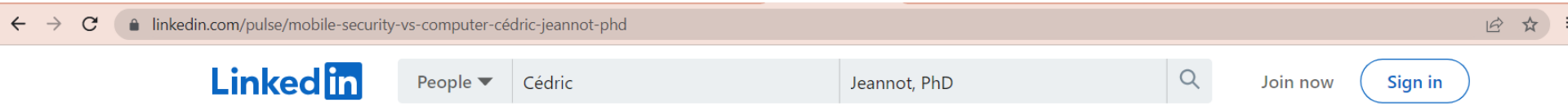
- Kernel is isolated from user processes
  - Separate page tables
  - Processor privilege levels ensure userspace code cannot use privileged instructions
- Interface between userspace and kernel: system calls



Example: Smartphone OS design

Does the threat model for a smartphone differ from a desktop?

Mobile Security vs Computer Security

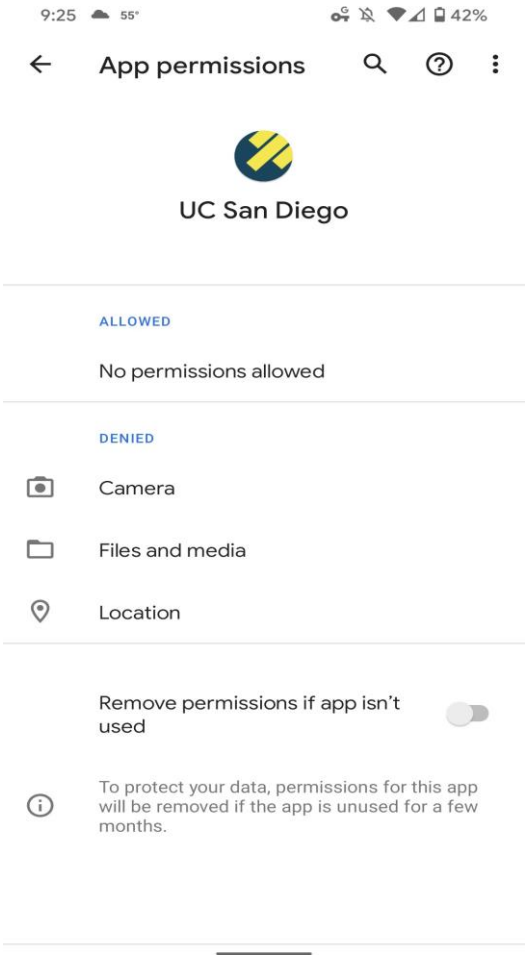


One reason why the use of smartphones has become so commonplace is thanks to their striking similarity to computers: mobile phones are now just mini portable computers, with similar processing power and with telephone capabilities built into them. People use smartphones as they would computers to access their Favorited applications and web services. They're just smaller and more convenient to access data whenever and wherever you want. Without proper security, though, a smartphone is a honeypot of personal or corporate information making them an attractive target for thieves or hackers.

One of the biggest threats regarding mobile security is dangerous user behavior. Either failure to recognize the dangers or just through laziness, people often fail to take basic measures to protect their devices. A recent study from Consumer Reports found that over a third of users did not implement any mobile security, with 36% using a 4-digit PIN and only 11% using more complex passwords. With the sheer number of phones being lost and stolen, many millions are falling into the hands of unauthorized users. Devices can also be "jailbroken" in order to remove built-in restrictions and allow consumers to add unapproved features and services. Jailbreaking bypasses the basic protections provided by the device manufacturer and often leads to weakened protection against malware and hackers' bad intentions.

Android process isolation

- Android uses Linux and sandboxing for isolation.
- Each app runs under its own UID.
- Apps can request permissions, which are basically capabilities.
- Android has a well-developed notion of app/app access control.
- Heavily relies on user's consent, for better or worse.



Software fault isolation (SFI)

Placing untrusted components in their own address space provides isolation, but comes with overhead.

Software fault isolation wants to partition apps running in the same address space.

- Kernel modules should not corrupt kernel
- Native libraries should not corrupt JVM

Software fault isolation (SFI)

Placing untrusted components in their own address space provides isolation, but comes with overhead.

Software fault isolation wants to partition apps running in the same address space.

- Kernel modules should not corrupt kernel
- Native libraries should not corrupt JVM

SFI approach: Partition process memory into segments

- Memory isolation: Instrument all loads and stores
- Control flow integrity: Ensure all control flow is restricted to CFG that instruments loads/stores
- Complete mediation: Disallow privileged instructions
- Syscall-like interface between isolated code

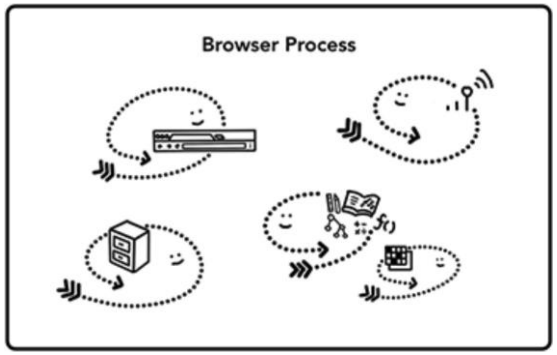
Example: Browser design

What's the threat model?

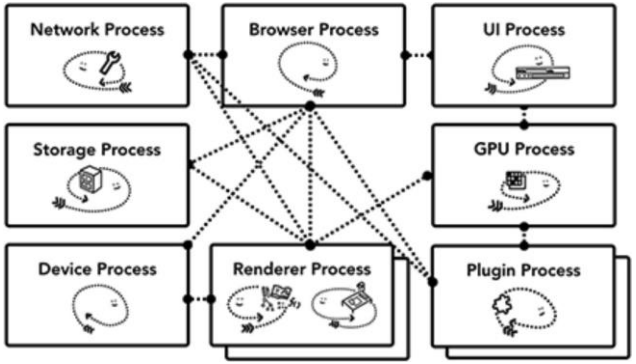
What are the assets?

What security properties do we want to preserve?

Chrome Security Architecture



Pre-2006



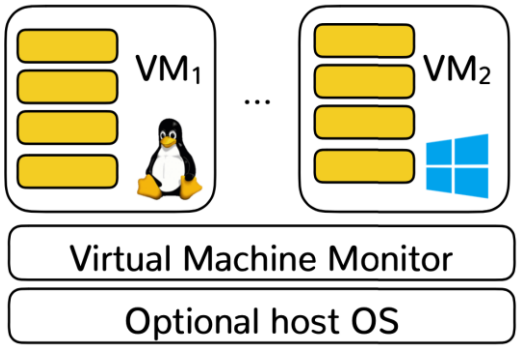
Modern

Modern Browser Security Model

- Browser process
  - Handles the privileged parts of browser (network requests, address bar, bookmarks)
- Renderer process
  - Handles untrusted attacker content: JS engine, DOM, etc.
  - Communication restricted to remote procedure calls
- Many other processes (GPU, plugin, etc.)

Virtual Machines

- Virtual machines allow a single piece of hardware to emulate multiple machines
- Useful for cloud computing and also for isolation
- Intel has hardware support for x86 virtualization: VMM support in hardware so that operating system can be run in ring 0 without requiring VMM intervention for syscalls



VMs and Isolation

VM Isolation for the cloud:

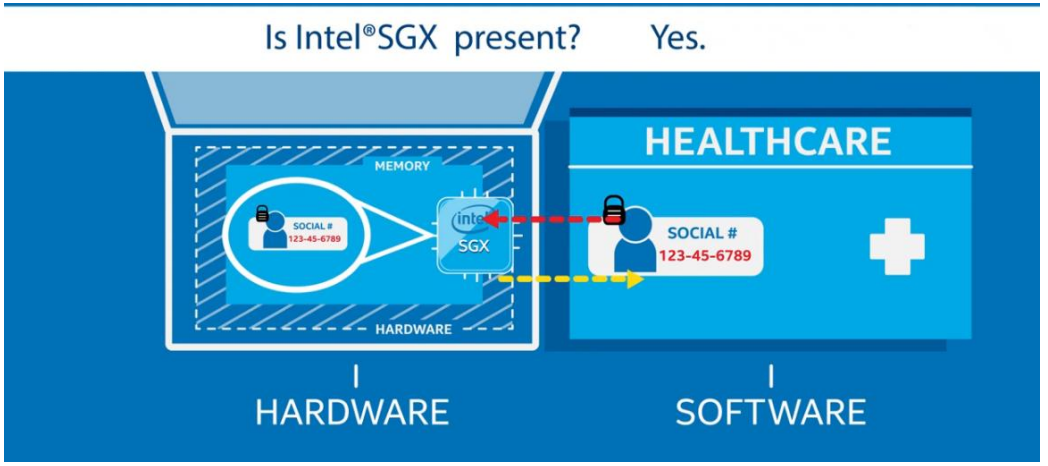
- VMs from different customers may run on the same machine
- Hypervisor tries to isolate VMs to minimize information leaks

VM Isolation for the end user:

- Qubes OS: A desktop OS where everything is a VM.
- Every window frame UI identifies VM source.

Hardware isolation: Secure enclaves

- Intel Software Guard eXtensons (SGX)
  - Runs trusted code in an *enclave*
  - Enclave memory encrypted and only decrypted in the CPU
  - Can't be read even by malicious OS
- Why do we want to protect a program against a malicious OS?



Hardware isolation: Secure enclaves

- Intel Software Guard eXtensons (SGX)
  - Runs trusted code in an *enclave*
  - Enclave memory encrypted and only decrypted in the CPU
  - Can't be read even by malicious OS

- Why do we want to protect a program against a malicious OS?

  Example applications:
  - DRM (Digital Rights Management)
  - Secure remote computation
  - Protecting crypto keys or sensitive information

iOS Secure Boot

- Apple devices use a secure enclave coprocessor as part of its boot chain.
- Hardware-based root of trust: code and code-verifying keys baked into boot ROM (read-only memory).
- Each step of the boot process verifies that the bootloader, kernel are signed by Apple.
- What are the positives and negatives of this kind of design?

Physical isolation: Air gap

- To ensure that a misbehaving app cannot harm the rest of the system, you could run it on physically isolated system.

- What kinds of systems would you do this for? What are the merits/demerits?

4. Principles: Fail-safe and Fail-secure

4. Principles: Fail-safe and Fail-secure

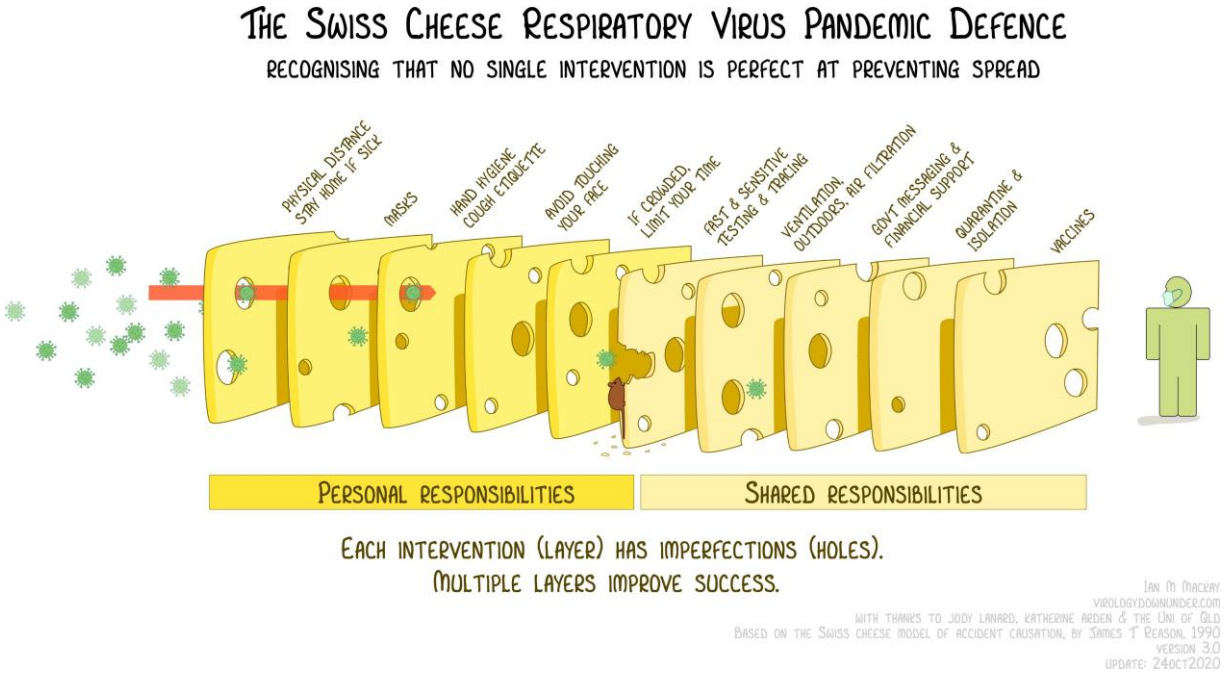
- A **fail-secure system** is one that, in the event of a specific type of failure, responds in a way such that access or data are denied.
    - **Related:** a fail-safe system, in the event of failure, causes no harm, or at least a minimum of harm, to other systems or to personnel.

- **Fail-secure** and **fail-safe** may suggest different outcomes
    - For example, if a building catches fire, fail-safe systems would unlock doors to ensure quick escape and allow firefighters inside, while fail-secure would lock doors to prevent unauthorized access to the building.

5. Principles: Defense in depth

5. Principles: Defense in depth

We do not expect any of our defenses to be perfect.



Last. Principles: Keep it simple

6. Principles: Keep it simple

We *have* to trust some components of our system.

In general keeping the Trusted Computing Base small and simple makes it easier to verify.

- In theory a hypervisor can be less complex than a full host operating system.
- A small OS kernel has less attack surface than one with many features.

Software and hardware isolation techniques

- Memory isolation
- Resource isolation and access control
- System call interposition
- Sandboxing
- Containers
- Virtualization
- Secure enclaves
- Physical air gap

Lesson: Complete isolation is often inappropriate;
applications need to communicate through regulated interfaces

Principles of secure system design

1. Least privilege
2. Privilege separation
3. Complete mediation
4. Fail safe/closed
5. Defense in depth
6. Keep it simple