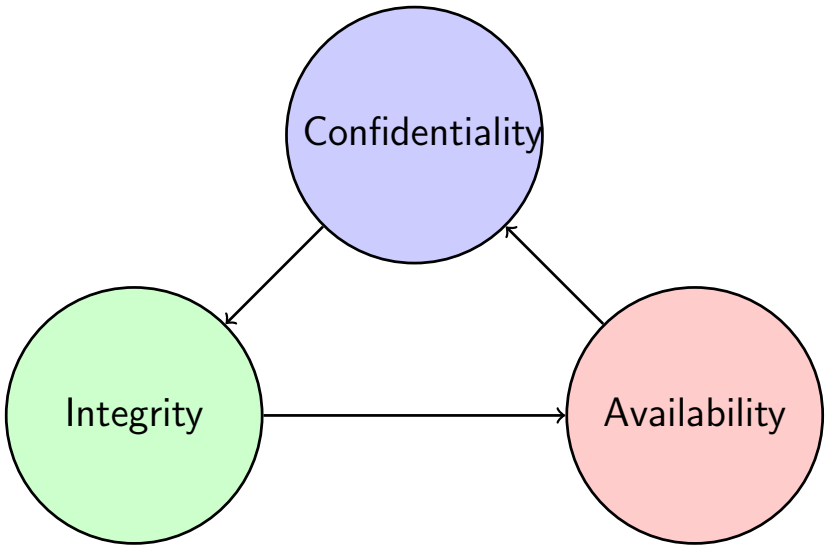


The CIA Triad

CSL 301

OPERATING SYSTEMS

Lecture 27

Introduction to OS Security

Instructor

Dr. Dhiman Saha

Security

Why Study OS Security?

- **The Foundation:** Pretty much everything runs on top of an operating system.
- If the OS is insecure, everything running on it is also insecure.
- It's like building a house on sand.
- **Broad Impact:** Security flaws in a widely-used OS affect a massive number of users and applications.
- **Ultimate Control:** The OS has ultimate control of all hardware resources:
  - The processor
  - Main memory
  - Peripheral devices

The Core Problem

What if an adversary could control the OS?

Imagine an attacker could force OS components to behave maliciously:

- **Memory Management:** Read or modify any process's memory.
- **Scheduling:** Starve legitimate processes of CPU time.
- **File Systems:** Ignore permissions to read, write, or delete any file.
- **Synchronization:** Create deadlocks or race conditions at will.

"Our computing lives depend on our operating systems behaving as they have been defined to behave."

Challenges in Securing an OS

- **Complexity:** Modern operating systems are vast and complex.
  - More code and more complexity lead to more bugs.
  - Security failures often arise from these flaws.
- **Concurrency:** OSs are designed to support multiple processes at once.
  - These processes are not equally trusted.
  - The OS must segregate processes and protect shared hardware from misuse.

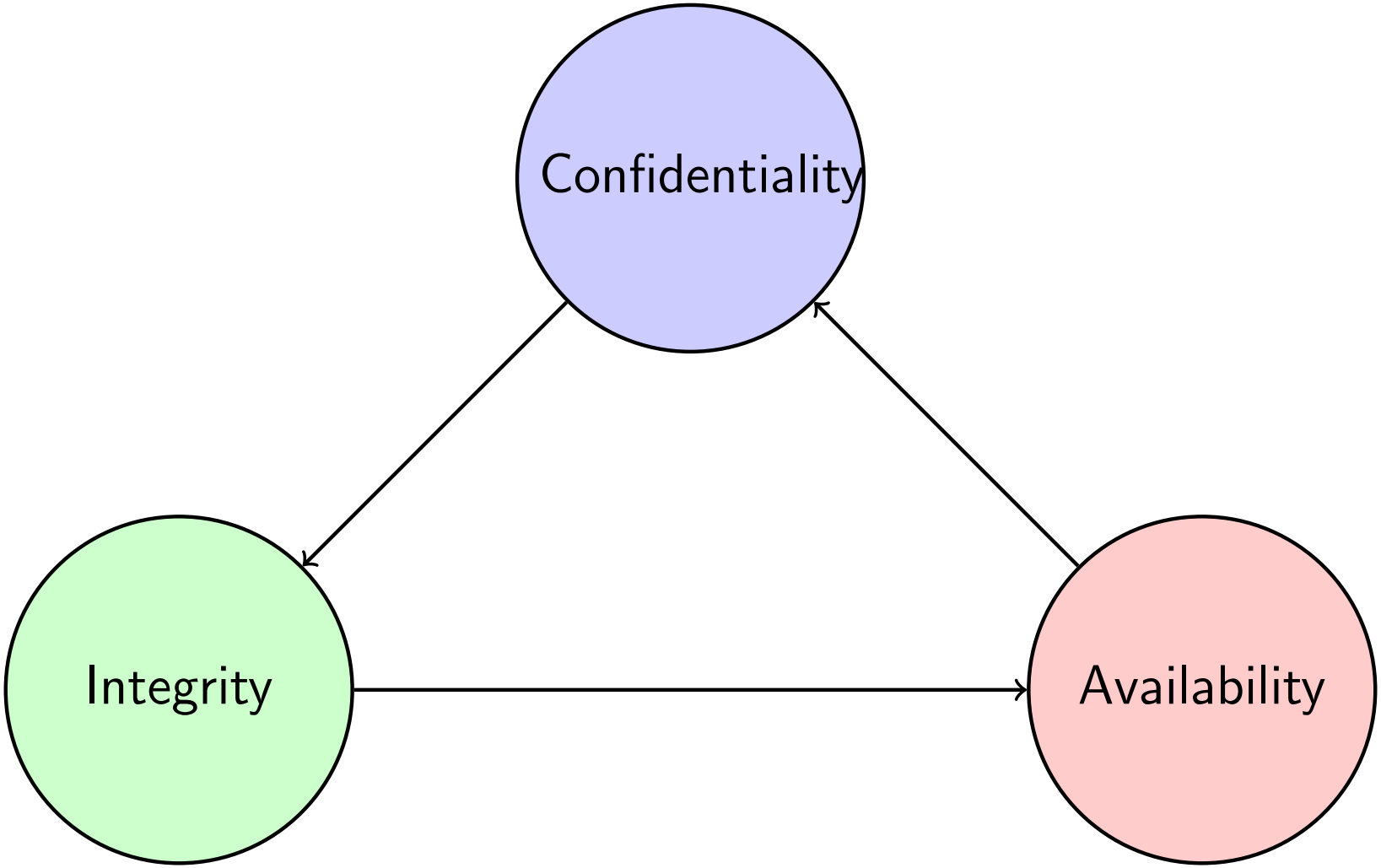
What Are We Protecting?

At a high level, the answer is simple: **Everything**.
A compromised OS can:

- Examine or alter any process's memory.
- Read, write, delete, or corrupt any file on any storage device.
- Change the scheduling or halt any process.
- Send any message to anywhere.
- Enable or disable any peripheral device.

Processes are at the mercy of the operating system.

The CIA Triad: Core Security Goals



These three pillars work together to ensure robust information security.

Confidentiality

Goal: Prevent unauthorized disclosure of information.

- **Analogy:** Keep your credit card number secret.
- **OS Context:**
  - One process must not be able to read the private memory of another process.
  - File permissions must prevent unauthorized users from reading sensitive files.

Integrity

Goal: Prevent unauthorized modification of information.

- **Analogy:** An online order for 1 pizza should not be changed to 1000 anchovy pizzas.
- **OS Context:**
  - A user who should not be able to write to a file cannot change its contents.
  - An OS must protect its own code and data structures from modification by user processes.
- Includes **Authenticity**: Verifying that data was created by a specific, trusted party.

Availability

Goal: Ensure information and services are accessible when needed.

- **Analogy:** A competitor shouldn't be able to block the streets to your store during a big sale.
- **OS Context:**
  - One process cannot hog the CPU and prevent all other processes from running (Denial of Service).
  - The file system must remain accessible.

From Goals to Policies

The CIA goals are too general for direct implementation.

- We must convert vague goals into highly specific **security policies**.
- A **policy** defines the security-relevant behavior of a system.
  - *Example: "Users Alice and Bob may read and write to file `report.docx`, but no other user may access it."*
- The OS provides general **mechanisms** (e.g., access control lists, system call checks) to enforce many different policies.

Designing Secure Systems

Saltzer and Schroeder (1975)

A set of design principles that have proven to lead to more secure systems.

"While neither the original authors nor later commentators would claim that following them will guarantee that your system is secure, paying attention to them has proven to lead to more secure systems, while you ignore them at your own peril."

Design Principles (1/2)

- **Economy of Mechanism**
  - Keep the design as simple and small as possible. Complex systems are harder to secure.
- **Fail-safe Defaults**
  - Default to denying access. Grant permission only explicitly.
- **Complete Mediation**
  - Every access to every object must be checked for authority. Every single time.
- **Open Design**
  - Do not depend on the secrecy of the design (security by obscurity). Assume the attacker knows how it works.

Design Principles (2/2)

- **Separation of Privilege**
  - Require multiple conditions to grant permission (e.g., two-factor authentication).
- **Least Privilege**
  - A user or process should only have the minimum privileges necessary to perform its task.
- **Least Common Mechanism**
  - Minimize mechanisms shared among users. Shared state is a potential information path.
- **Acceptability**
  - The security mechanism must be usable. If it's too cumbersome, users will bypass it.

How the OS Enforces Security

The OS acts as a gatekeeper for all system resources.

- **Process Isolation:**
  - The OS virtualizes resources (especially memory) to create a "container" for each process.
  - Hardware support (e.g., MMU) prevents a process from even naming a memory address outside its own space.
- **System Calls:**
  - The primary mechanism for a process to request a service from the OS (e.g., open a file, send a network packet).
  - This provides a critical point for the OS to check permissions.

System Calls as Checkpoints

1. A process executes a **system call** to request a service.
2. The hardware switches from *user mode* to *supervisor (kernel) mode*.
3. The OS handler is invoked. It can now identify:
   - Which process is making the request?
   - What service is being requested?
4. The OS uses **access control mechanisms** to check if the process is authorized according to the system's security policy.
5. If authorized, the OS performs the action. If not, it returns an error.

The Weakest Link

Tip: Be Careful of the Weakest Link

Attackers are smart and lazy. They will go for the easiest way to overcome your system's security.

- Often, a system's weakest link is not its software, but its **human users**.
- An attacker might try to fool a legitimate user into misusing the system (**social engineering**).
- This is why the **Principle of Least Privilege** is so important.
  - If an attacker fools a user who has complete privileges, the damage can be total.
  - If the user has only limited privileges, the damage is contained.

Summary

- OS security is the foundation for all software security.
- Security goals are centered on **Confidentiality**, **Integrity**, and **Availability**.
- We translate these goals into specific **policies**.
- The OS provides **mechanisms** (like process isolation and system call checks) to enforce policies.
- Following established **design principles** (e.g., Least Privilege) is crucial for building secure systems.