

# **Subject Name: Information System**

Unit No:04 Unit Name: 4: Systems Design

Faculty: Mrs. Bhavana Alte

Mr. Prathmesh Gunjgur

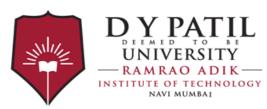
Unit No: 4 Unit Name: Systems Design

# Security Design Principles



# **Simplicity in Security Design**

- •Keeping things simple reduces the chances of errors and makes security easier to manage.
- •Example: Imagine a vending machine with only one button to select a drink. There's little chance of pressing the wrong button. But if the machine had 20 buttons, it could be confusing, leading to mistakes.
- •Real-world example (Sendmail): The program sendmail had multiple steps to process configuration files, and one step was often overlooked. This led to security vulnerabilities, as sendmail assumed earlier steps had done checks properly, but they didn't always.

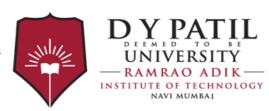


# 2. Policy Conflicts Due to Complexity

- •Simple rules help avoid conflicting policies.
- •Example: A school has two rules:
- 1.Students must report cheating.
- 2.Student files must remain private.

If a teaching assistant (TA) finds two identical student files and reports it, they might be accused of violating privacy while following the rule to report cheating.

•Lesson: Security policies should be designed to prevent such conflicts.

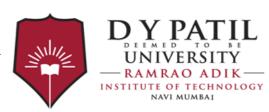


# 3. Restriction for Security

•People (or systems) should only have access to what they absolutely need.

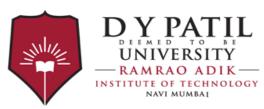
#### •Example:

- •In an exam, students aren't allowed to bring notes to prevent cheating.
- •Government officials only access information they *need to know* to reduce the risk of leaks.



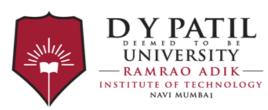
# 4. Limiting Communication as a Security Measure

- Limiting Communication for Security
- Why? To prevent information from leaking to unauthorized people.
- How? By controlling who can talk to whom and in what ways.
- Example 1: Prisoners
- Prisoners can only talk to approved people through monitored visits or letters.
- This prevents them from planning escapes or getting illegal items like weapons.
- **Exception:** They can talk privately with their lawyer because that's a legal right.
- OUTCOME
- Even silence can be a form of communication.
- Security systems must consider both direct and indirect ways people can share information.



# **Principles of Secure Design**

- Secure design principles originate in computer security but extend beyond technology into policy-making, governance, and international security.
- These principles serve as guidelines to build robust, resilient systems that prevent vulnerabilities and ensure operational stability.
- Security design aims to restrict unnecessary access to protect systems.
- Two closely related principles:
  - Least Privilege (PoLP) Focuses on permissions (explicit access rights).
  - Least Authority (PoLA) Focuses on authority (both direct and indirect effects).
- They are often used interchangeably but have a key distinction.



# **Principle of Least Privilege**

#### Definition:

A subject (user, process, system) should be granted only the **minimal privileges** necessary to perform its task—nothing more.

- Example 1: System Administrator Accounts (Violation of PoLP)
- In UNIX/Linux, the root user has unrestricted system access.
- This violates PoLP because even if an admin only needs to configure the network, they still have the power to delete critical files.
- A better approach is role-based access control (RBAC), where admins only get privileges for specific tasks (e.g., network admin vs. file system admin).
- Example 2: Database Access Control (Good Implementation of PoLP)
- A web application connects to a database to read user profiles but does not need to modify them.
- Instead of granting full read-write privileges, PoLP suggests granting only read access.
- This reduces the risk of accidental modifications or SQL injection attacks.

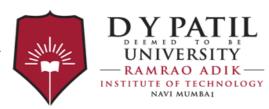


# **Principle of Least Authority (PoLA)**

#### Definition:

A subject should be granted only the **minimal authority** needed to complete a task—including both **direct and indirect** effects.

- Goes beyond explicit permissions and considers indirect interactions between processes.
- Example 1: Take-Grant Protection Model (Illustration of PoLA)
- Scenario:
  - Process A does not have direct permission to modify a critical file.
  - But it can pass the data to Process B, which has the required permission.
  - Even though Process A did not directly edit the file, it influenced the change.
- Security Concern: If indirect access is not controlled, PoLA is violated, even if PoLP is enforced.



# **Principle of Least Authority (PoLA)**

- Example 2: Web Browser Extensions (Real-World Violation of PoLA)
- A browser extension requests permission to read webpage data.
- But it can exploit **other extensions** that have access to **sensitive user data** (e.g., passwords).
- Even though the extension was not directly granted access to the sensitive data, it gains authority through interactions with other extensions.



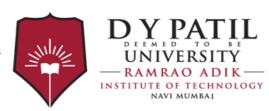
# **Principle of Fail-Safe Defaults**

- •If access is not **explicitly granted**, it should be **denied** by default.
- •If a process fails, it should not leave the system vulnerable.
- Example 1: Mail Server Access Control (Good Implementation of Fail-Safe Defaults)
- A mail server writes messages to a spool directory.
- If it fails to create a file, it should:
  - Close the network connection.
  - Issue an error message.
  - Stop processing instead of trying alternative storage.
- Why?
  - An attacker could exploit unrestricted write permissions to overwrite files or cause denial-of-service (DoS) attacks by filling up disk space.



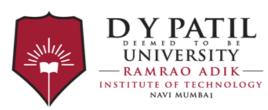
# **Principle of Fail-Safe Defaults**

- Example 2: Default Access Control in Operating Systems (Violation of Fail-Safe Defaults)
- Many systems grant full access to newly created users or applications.
- Secure systems should apply minimal privileges by default until explicitly changed by an administrator.



# **Principle of Economy of Mechanism**

- Definition:
- Security mechanisms should be simple to minimize errors and vulnerabilities.
- Complex security designs increase risk due to hidden flaws.
- Example 1: Ident Protocol (Violation of Economy of Mechanism)
- The Ident protocol sends the username of a process holding a TCP connection.
- The protocol assumes the originating host is trustworthy.
- Attack scenario:
  - A compromised system can fake identity information.
  - If a system grants access based on Ident response, an attacker can spoof credentials and gain unauthorized access.



# **Principle of Economy of Mechanism**

- Example 2: Overly Complex Firewall Rules (Violation of Economy of Mechanism)
- Firewalls with thousands of rules are hard to audit.
- Attackers can exploit overlooked loopholes in rule combinations.
- A simpler rule structure ensures better security without sacrificing protection.



# **Principle of Complete Mediation**

- Definition:
- Every access request should be checked every time instead of using cached permissions.
- Example 1: UNIX File Permissions (Violation of Complete Mediation)
- A UNIX process requests access to a file.
- The OS checks the file permissions and grants a file descriptor.
- If the file owner revokes permissions, the process still has access through the cached file descriptor.
- This violates complete mediation since subsequent accesses are not rechecked.



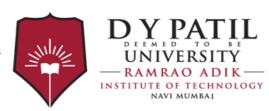
# **Principle of Complete Mediation**

- Example 2: DNS Cache Poisoning (Violation of Complete Mediation)
- The Domain Name System (DNS) maps hostnames to IP addresses.
- DNS caches the responses to speed up lookups.
- Attack scenario:
  - An attacker poisons the DNS cache by injecting a malicious IP for a trusted website.
  - Users are redirected to a fake site (e.g., a phishing page).
- Solution:
  - Always verify DNS entries instead of blindly using cached values.



# **Principle of Open Design**

- The Principle of Open Design states that the security of a system should not rely
  on secrecy of its design or implementation. Instead, security should depend on
  strong cryptographic principles, rigorous testing, and proper key
  management.
- Case Study: The Content Scrambling System (CSS) Flaw
- How CSS Works?
- The Content Scrambling System (CSS) was designed to prevent unauthorized copying of DVD movies. It worked as follows:

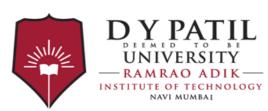


#### DVD Encryption Mechanism

- •A disk key is stored on the DVD, encrypted multiple times using different player keys.
- •When a DVD is inserted into a **DVD player**, the player decrypts the disk key using its **unique player key**.
- •Once the **disk key is decrypted**, it is used to decrypt the **title key**, which finally decrypts the movie.

#### Security Measure in Place

- •The encryption keys were **not stored in the movie file** itself, so simply copying the movie file would not allow playback.
- •This was intended to prevent piracy by ensuring that only **licensed DVD players** with valid keys could decrypt and play the DVD.



# **How the System Was Broken**

Despite these measures, CSS was cracked in 1999 due to a flawed security **approach** that relied on secrecy:

#### 1. Discovery of an Unprotected Key

- A software-based DVD player in Norway contained an unencrypted CSS decryption key.
- Hackers extracted this key and reverse-engineered the entire CSS algorithm.

#### 2. Creation of Decryption Software

- Once the algorithm was understood, hackers created free software that could decrypt any DVD movie.
- This software quickly spread across the Internet, allowing anyone to bypass DVD **encryption** and copy movies freely.



# **How the System Was Broken**

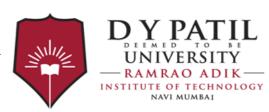
#### 3. Legal Battle and Irony

- The DVD Copyright Control Association (DVD CCA) sued to remove this software from the Internet.
- However, in their court filing, they accidentally included the full source code of the CSS algorithm!
- Before they realized the mistake, over 21,000 copies of the document were downloaded.
- This made the algorithm permanently available to the public, further accelerating DVD piracy



# **Principle of Least Common Mechanism – Explanation**

- The Principle of Least Common Mechanism states that mechanisms used to access resources should not be shared because shared mechanisms create security vulnerabilities by providing channels for information leakage or exploitation.
- This principle is particularly important in securing operating systems, networks, and applications.
- Key Idea: Minimizing Shared Mechanisms Enhances Security
- When multiple processes, users, or systems share a common mechanism, an attacker can exploit that mechanism to gain unauthorized access or disrupt operations.
- Reducing the number of shared mechanisms limits attack surfaces and improves security.



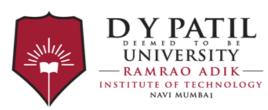
# Example 1: Denial of Service (DoS) Attack on an E-Commerce Website

#### Scenario:

A major e-commerce website provides online shopping services. Attackers flood
the website with excessive traffic (a Denial of Service (DoS) attack) to overload
the system. As a result, legitimate customers cannot access the website,
leading to revenue loss for the company.

#### Cause of the Attack:

 The attack was successful because both legitimate users and attackers share the same Internet connection to access the website. This shared mechanism enabled attackers to disrupt normal operations.



# **Example 1: Denial of Service (DoS) Attack on an E-Commerce Website**

- Countermeasures:
- Restrict the attackers' access to the website by blocking their traffic.
- Use a proxy server like the Purdue SYN intermediary, which filters suspicious connections and only allows legitimate traffic.
- Implement traffic throttling, which limits the number of requests a user can send in a given time frame to prevent overload.
- By minimizing the shared Internet access mechanism, the attack impact is reduced.



# **Example 2: Software Diversity to Prevent System-wide Attacks**

#### Scenario:

 A company uses identical software versions on all its computers. If an attacker finds a vulnerability in this software, they can exploit all computers simultaneously, causing a widespread system compromise.

#### Cause of the Attack:

 The shared mechanism in this case is the identical software configuration across all systems. Since every computer runs the same program, attacking one system means attacking them all.



# **Principle of Least Astonishment (PoLA) – Presentation Explanation**

- This principle considers the human element in security design.
- It states that security mechanisms should be designed so that users understand them intuitively and do not cause confusion or frustration.
- A security system that is difficult to understand or use may lead to misconfigurations, errors, or users bypassing security controls.

#### KeyPoints:

- Security should be simple and intuitive.
- ♦ Users should **understand why** a security mechanism works the way it does.
- **♦** Complicated security leads to **misconfigurations and security failures**.



#### **Definition:**

#### **Definition:**

The Principle of Least Astonishment (PoLA) states that security mechanisms should be simple and intuitive, preventing user confusion and misconfiguration.

- Why is this important?
- If a system's security doesn't align with user expectations, users may unknowingly bypass or disable security features.
- Clarity in error messages and system behavior improves overall security.



# **Example – Secure Shell (SSH) Configuration**

- Good Example of PoLA: SSH Key-Based Authentication
- SSH allows public key authentication for secure remote logins.
- UNIX versions of SSH let users store public keys locally without password protection.
- Users can connect securely without repeatedly entering a password.
- ✓ Aligns with user expectations simple yet secure.



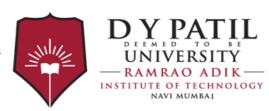
# **Example – Poor Error Messages in Login Systems**

- **Bad Example of PoLA:** Login Error Messages
- **Incorrect Implementation:** 
  - When a user enters a **wrong password**, the system displays:
    - X "Incorrect password."
  - This confirms that the **username** is valid, which is valuable information for attackers.
- **Better Approach:** 
  - Display a **generic failure message**:
  - This prevents attackers from learning whether the username exists.



# **Balancing Security & Usability**

- Security must not overwhelm users.
- •Error messages should be informative but not exploitable.
- •Security mechanisms should match the environment and user workflow.





# Thank You