5.4. SECURITY MECHANISMS

Introduction:

- Security mechanisms in an Operating System (OS) are techniques used to **protect system resources, data, and programs** from unauthorized access, misuse, or attacks.
- They ensure the **confidentiality, integrity, and availability (CIA)** of the system.

Goals of OS Security:

- **Confidentiality** – Prevent unauthorized data access.
- **Integrity** – Prevent unauthorized data modification.
- **Availability** – Ensure resources are accessible when needed.

Types of Security Mechanisms

1. **Authentication** – Verifies the identity of a user before granting system access.
2. **Authorization** – Determines what actions an authenticated user is allowed to perform.
3. **Access Control** – Regulates access to system resources based on policies and permissions.
4. **Encryption** – Converts data into a secure format to prevent unauthorized access.
5. **Sandboxing** – Executes programs in an isolated environment to prevent system damage.
6. **Process Isolation** – Ensures processes run independently without interfering with each other.
7. **Memory Protection** – Prevents unauthorized access to a process's memory space.
8. **File System Security** – Controls access to files using permissions and access control lists.
9. **Network Security** – Protects the system from network-based threats using firewalls and protocols.
10. **Auditing and Logging** – Records system activities for monitoring and security analysis.
11. **Intrusion Detection System (IDS)** – Monitors and detects suspicious or malicious activities.

12. **Antivirus/Malware Protection** – Detects and removes malicious software from the system.
13. **Virtualization Security** – Uses virtual machines to isolate and secure different environments.
14. **Mandatory Access Control (MAC)** – Enforces system-defined access policies (e.g., SELinux).
15. **Discretionary Access Control (DAC)** – Allows resource owners to define access permissions.

5.4.1. SANDBOXING:

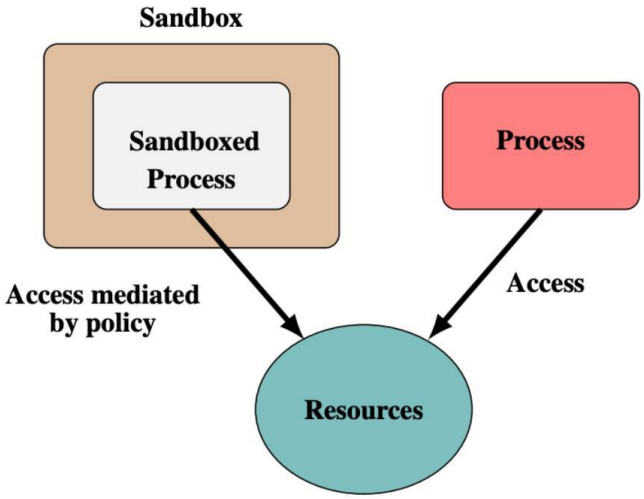
- Sandboxing is an advanced **security mechanism in operating systems and applications** that provides a **safe, isolated environment** for executing programs.
- It ensures that even if a program is malicious or faulty, it **cannot affect the host system or other applications**.
- In modern computing (cloud, mobile, browsers), sandboxing is a core security principle.

Definition:

A sandbox is a **restricted execution environment** where:

- Applications run with **limited permissions**
- Access to system resources is **controlled and monitored**
- Unauthorized actions are **blocked**

☞ It acts like a **protective boundary** between the application and the system.



Need for Sandboxing:

Sandboxing is required due to increasing security threats:

- **Malware Protection:** Prevents viruses from accessing system files.

- **Safe Execution of Untrusted Code:** Used when running downloaded software or scripts.
- **Application Isolation:** Prevents one app from interfering with another.
- **Secure Web Browsing:** Each browser tab runs in isolation.
- **Testing & Development:** Developers test software without risking the main system.

Working of Sandboxing:

Sandboxing works using a combination of **security policies, isolation techniques, and monitoring tools**.

Step-by-Step Process:

1. **Environment Creation:** A virtual or restricted environment is created.
2. **Policy Enforcement:** Rules define what the application can access (e.g., read-only files, no system calls).
3. **Resource Restriction:** Limits are applied to:
    - CPU usage
    - Memory
    - Disk access
    - Network
4. **Execution Monitoring:** System continuously tracks application behaviour.
5. **Blocking Malicious Actions:** Unauthorized actions are denied immediately.
6. **Logging and Analysis:** Activities are recorded for security review.

Types of Sandboxing:

a) Application-Level Sandboxing:

- Each application runs in its own sandbox.
- Common in browsers and mobile apps.
- Example: A browser tab cannot access another tab's data.

☞ Uses process isolation and permission models.

b) OS-Level Sandboxing:

- Implemented using **kernel features**.
- Uses:
    - Namespaces
    - Control groups (cgroups)
    - Access control lists

☞ Stronger and more efficient than app-level sandboxing.

c) Virtual Machine (VM) Sandboxing:

- Uses a **hypervisor** to create a virtual system.
- Each VM runs a separate OS.

☞ Provides **complete isolation**, but:

- High memory usage.
- Slower performance.

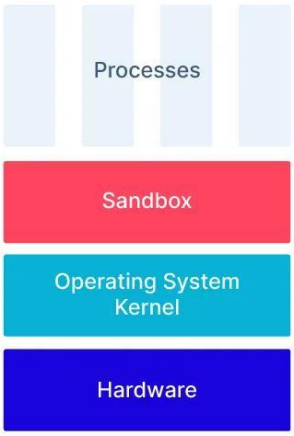
d) Container-Based Sandboxing:

- Lightweight alternative to VMs.
- Applications share OS kernel but remain isolated.

☞ Faster and widely used in cloud systems.

Key Features:

- **Isolation:** Programs cannot interact directly with the host system.



Sandbox Based Isolation

- **Least Privilege Principle:** Applications get only the permissions they need.
- **Controlled Execution:** All actions go through security checks.
- **Behavior Monitoring:** Suspicious activity is detected in real-time.
- **Fault Containment:** Errors do not spread beyond the sandbox.

Advantages:

- **Improved Security:** Protects against malware and exploits.
- **Safe Testing Environment:** Ideal for developers and researchers.
- **System Stability:** Prevents crashes from affecting entire system.
- **Data Protection:** Sensitive files remain inaccessible.
- **Supports Cloud & Virtualization:** Essential for modern computing environments.

Limitations:

- **Performance Overhead:** Especially in VM-based sandboxing.

- **Limited Functionality:** Some applications may not work fully.
- **Sandbox Escape Attacks:** Advanced malware may bypass restrictions.
- **Complex Implementation:** Requires careful configuration.

Real-World Examples:

- **Web Browsers:** Each tab runs in a sandbox.
- **Mobile OS (Android, iOS):** Apps cannot access each other's data.
- **Cloud Platforms:** Containers isolate microservices.
- **Cybersecurity:** Malware is tested safely in sandbox environments.

5.4.2. SELinux (Security-Enhanced Linux)

- **SELinux (Security-Enhanced Linux)** is an advanced **Linux security mechanism** that provides **strong access control policies** beyond traditional file permissions.
- It was originally developed by the National Security Agency and later integrated into the Linux kernel.
- SELinux enforces **Mandatory Access Control (MAC)** to enhance system security.

Definition:

SELinux is a **kernel-level security module** that controls how users, applications, and processes access system resources using **security policies**.

Need for SELinux:

Traditional Linux security uses **Discretionary Access Control (DAC)**:

- Users can change permissions
- Vulnerable to privilege misuse

☞ SELinux solves this by:

- Enforcing strict policies
- Limiting even root user access
- Preventing unauthorized system access

Key Concept: Mandatory Access Control (MAC):

- Access decisions are made by the **system**, not users
- Policies are **centrally defined and enforced**
- Even administrators cannot bypass rules easily

Working of SELinux:

SELinux works by assigning **security labels (contexts)** to:

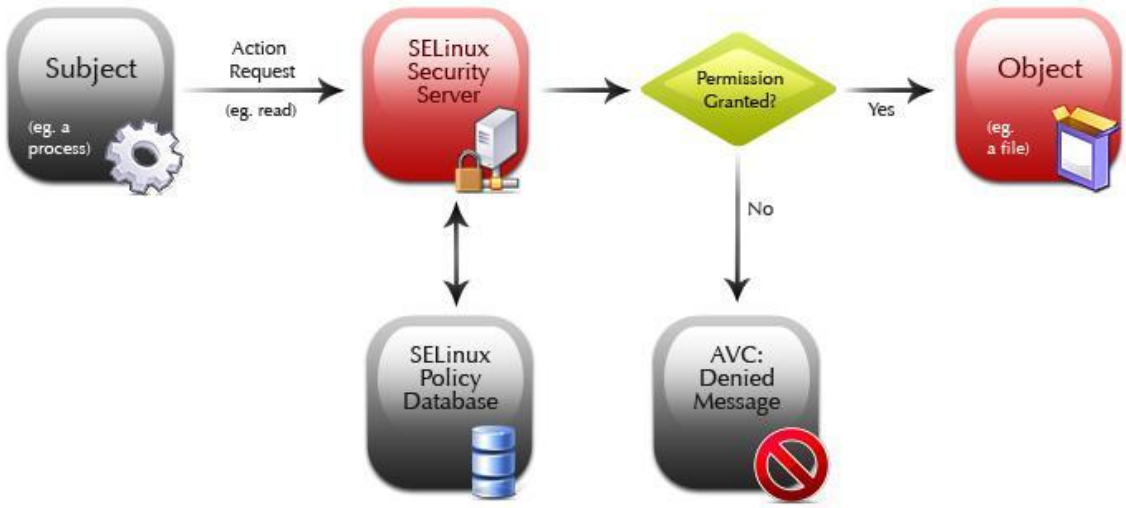
- Files

- Processes
- Users

Access Decision Process:

1. Every object (file/process) gets a **security label**
2. Policy rules define allowed interactions
3. When a process tries to access a resource:
    - SELinux checks policy
    - Grants or denies access

SELinux Architecture:



Components:

- **Security Server** – Makes access decisions.
- **Access Vector Cache (AVC)** – Stores previous decisions.
- **Policy Database** – Contains rules.
- **Kernel Enforcement** – Applies decisions.

Security Context:

Each object has a **security context**:

Format:

user : role : type : level

Example:

- system_u:object_r:httpd_sys_content_t:s0.

Explanation:

- **User** – SELinux user identity.
- **Role** – Defines access roles.
- **Type** – Most important (Type Enforcement).
- **Level** – Security level (MLS/MCS).

SELinux Modes:

1. **Enforcing Mode:**
    - Policies are enforced.
    - Violations are blocked.
2. **Permissive Mode:**
    - Violations are logged.
    - No blocking.
3. **Disabled Mode:**
    - SELinux is turned off.

Types of SELinux Policies:

- **Targeted Policy:**
    - Protects selected services.
    - Most commonly used.
- **Strict Policy:**
    - Enforces rules on all processes.
- **MLS (Multi-Level Security):**
    - Used in military/security systems.

Advantages:

- Strong security enforcement
- Protects against privilege escalation
- Limits damage from compromised applications
- Fine-grained access control

Limitations:

- Complex configuration.
- Difficult for beginners.
- May block legitimate operations if misconfigured.

Real-Time Example:

- A web server process (**httpd**) cannot access user home directories.
- Even if hacked, attacker cannot access sensitive files.

Comparison: DAC vs MAC (SELinux):

Feature	DAC	MAC (SELinux)
Control	User-based	System-based
Flexibility	High	Restricted
Security	Moderate	Very High
Root Access	Full	Limited

5.4.3. PERMISSION MODELS:

- Permission models (or access control models) define **how users and processes are granted or denied access** to system resources such as files, memory, and devices.
- They are essential for ensuring **security, privacy, and controlled resource sharing** in an Operating System.

Types of Permission Models:

1. Discretionary Access Control (DAC):

DAC is a model where the **owner of a resource decides who can access it**.

- Each file/resource has an owner.
- Owner sets permissions (read, write, execute).
- Common in UNIX/Linux systems.

Example:

- File owner gives read permission to another user

Advantages:

- Flexible and easy to implement

Disadvantages:

- Less secure (permissions can be misused)

2. Mandatory Access Control (MAC):

MAC is a model where **access is controlled by system-enforced policies**, not users.

- Every object and subject has a **security label.**
- Access is based on classification levels.
- Used in high-security environments.

Example:

- Only users with "Top Secret" clearance can access classified files

Advantages:

- Very high security

Disadvantages:

- Rigid and difficult to manage

3. Role-Based Access Control (RBAC):

Access is granted based on the **role assigned to a user**.

- Users are assigned roles (Admin, Manager, Student).
- Roles have predefined permissions.

- Users inherit permissions from roles.

Example:

- Admin → Full access
- Student → Limited access

Advantages:

- Easy to manage in large organizations.
- Reduces complexity.

Disadvantages

- Not flexible for dynamic conditions.

4. Attribute-Based Access Control (ABAC):

Access is based on **attributes of user, resource, and environment**.

- Attributes include:
  - User role
  - Time
  - Location
  - Device
- Policies evaluate multiple conditions

Example:

- Access allowed only during office hours

Advantages:

- Highly flexible
- Fine-grained control

Disadvantages:

- Complex to implement

5. Rule-Based Access Control:

Access decisions are made based on **predefined system rules**.

- Rules are enforced globally
- Often used in firewalls and security systems

Example:

- Block access after 3 failed login attempts

Advantages:

- Simple and automated

Disadvantages:

- Limited flexibility

6. Lattice-Based Access Control (LBAC):

Access is based on a **hierarchical structure of security levels**.

- Uses levels like:
  - Confidential
  - Secret
  - Top Secret
- Follows:
  - **No Read Up**
  - **No Write Down**

Advantages:

- Strong data confidentiality

Disadvantages:

- Complex structure

7. Capability-Based Access Control:

Access is granted through **capability tokens** assigned to users/processes.

- A capability is like a **key**
- It specifies allowed operations

Example:

- A process has a token allowing read-only access to a file

Advantages:

- Efficient and secure

Disadvantages:

- Difficult to manage capabilities

8. Access Control Lists (ACL):

Each resource maintains a list specifying **which users can access it and how**.

- Attached to objects (files, directories)
- Lists users and permissions

Example:

- File allows:
  - User A → Read
  - User B → Write

Advantages:

- Fine-grained control

Disadvantages: Hard to manage for large systems

Comparison Table:

Model	Control Type	Flexibility	Security Level
DAC	User-based	High	Low
MAC	System-based	Low	Very High
RBAC	Role-based	Medium	High
ABAC	Attribute-based	Very High	High
ACL	Object-based	Medium	Medium