System Security

Q. Explain memory and address protection in OS security.

Memory and address protection are techniques used by an operating system (OS) to **prevent unauthorized access** to memory spaces of processes and the OS itself.

The goal is to **isolate processes**, **protect critical system data**, and prevent **malicious code** from reading or overwriting sensitive memory.

Why Is Memory Protection Needed?

Reason	Description
Process Isolation	Prevents one program from reading/writing another's memory
Protect OS Kernel Memory	Stops user processes from corrupting system code
Defend Against Attacks	Blocks buffer overflows, code injection, and privilege escalation
Reliability & Stability	Faulty programs can't crash others or the entire OS

Key Techniques for Memory and Address Protection

A. Virtual Memory and Address Translation

- Each process is given its own **virtual address space**.
- The **Memory Management Unit (MMU)** maps virtual addresses to physical addresses.
- This **isolates processes**, even if they use the same address values.

B. Segmentation

- Memory is divided into **segments** (code, data, stack).
- Each segment has **base and limit values**.
- Access beyond limits triggers a **segmentation fault**.

Analogy: Each program has its own "room," and segmentation makes sure no one steps beyond their assigned area.

C. Paging with Protection Bits

- Memory is split into **fixed-size pages**.

- Each page has **access control bits**:
    - **Read**, **Write**, **Execute** permissions
- Unauthorized access causes a **page fault**.

For example, **code pages are non-writable**, and **stack pages are non-executable** (helps prevent exploits like buffer overflow attacks).

D. User Mode vs Kernel Mode

- **User mode**: Programs can only access their memory.
- **Kernel mode**: Only the OS can access hardware and core memory.
- Prevents user applications from executing privileged instructions or accessing protected memory.

E. Address Space Layout Randomization (ASLR)

- OS **randomly arranges memory locations** (like stack, heap, libraries).
- Makes it harder for attackers to predict memory locations during an exploit.

Q. Explain user authentication methods in operating systems.

User authentication is the process of verifying the **identity of a user** before granting access to a system.
It ensures that only **authorized users** can access system resources, protecting against **unauthorized access**, **data theft**, and **misuse**.

Common User Authentication Methods in Operating Systems

A. Password-Based Authentication

- **User enters a username and password**
- The OS compares the password with the **stored hash** in the system (e.g., /etc/shadow in Linux)
- **Weak passwords** are vulnerable to guessing, brute-force, and dictionary attacks

Best practices: Enforce **strong passwords**, use **salting** and **hashing** (e.g., SHA-256)

B. Token-Based Authentication

- Uses a **physical device** or **digital token** (e.g., USB keys, smart cards)
- The token generates a **one-time password (OTP)** or contains a **digital certificate**

Example: RSA SecureID tokens, YubiKeys, or government-issued smart cards

C. Biometric Authentication

- Uses **physical characteristics** like:
    - Fingerprint
    - Face recognition
    - Iris scan
- OS verifies the user against stored biometric templates

Very user-friendly, but needs **secure storage** and **anti-spoofing measures**

D. Multifactor Authentication (MFA)

- Combines **two or more** of the following:
    1. **Something you know** (e.g., password)
    2. **Something you have** (e.g., token or phone)
    3. **Something you are** (e.g., fingerprint)

Example: Logging in with a password + OTP from mobile app

E. Certificate-Based Authentication

- User presents a **digital certificate** issued by a trusted **Certificate Authority (CA)**
- OS verifies the certificate's signature and validity

Common in corporate environments, VPNs, and secure networks

F. Pluggable Authentication Modules (PAM)

- Used in **Linux/Unix systems**
- A **modular authentication system** that supports:
    - Passwords
    - Biometrics
    - Certificates
    - Tokens
- Allows admins to configure authentication methods for different services

Q. Explain the file protection.

File protection is a mechanism used by operating systems to **control access to files** and prevent:

- Unauthorized **read**, **write**, or **execute** operations

- **Accidental or malicious modification or deletion**
- **Security breaches** like data leakage or privilege escalation

The goal is to ensure **only authorized users or processes** can access or modify files.

Why Is File Protection Necessary?

Need	Explanation
Confidentiality	Prevents unauthorized users from reading sensitive files
Integrity	Ensures only authorized modifications can occur
Access Control	Restricts what users/processes can do with a file
Data Isolation	Protects users from interfering with each other's files
Prevents malware	Stops malicious code from tampering with system or user files

---

File Protection Techniques

A. Access Control Lists (ACLs)

- Every file has a list of **users** and their **permissions**
- Common permissions: **read**, **write**, **execute**
- Example in UNIX: rwxr-xr–

User can **read/write/execute**, group can **read/execute**, others can **read**

B. User Groups and Ownership

- Each file has an **owner** and belongs to a **group**
- The OS applies permissions based on whether the user is:
    - The **owner**
    - A member of the **group**
    - **Another** (everyone else)

C. File System Permissions

- Enforced by the **file system** (e.g., NTFS, ext4)
- Define actions allowed per file/directory:
    - Create, delete, rename, execute, etc.

D. Encryption

- Files can be stored in encrypted form

- Only authorized users with the **decryption key** can access them

Example: Windows EFS (Encrypted File System)

E. Mandatory Access Control (MAC)

- Files and users are labeled with **security levels** (e.g., Top Secret, Confidential)
- The OS enforces strict rules — even root users may be restricted

Used in **military-grade** security systems (e.g., SELinux, AppArmor)

F. Audit and Logging

- The OS records access attempts and actions on sensitive files
- Helps in detecting **unauthorized access** or **file tampering**

Real-World Examples

OS	File Protection Features
Windows	NTFS permissions, EFS, Group Policy
Linux	chmod, chown, umask, ACLs, SELinux
MacOS	Sandbox, Keychain access control, ACLs

Q. What is file system security? Compare Linux and Windows mechanisms.

File system security refers to the methods used by an **operating system** to:

- Control **access to files and directories**
- Prevent **unauthorized access**, **modification**, or **deletion**
- Ensure **confidentiality, integrity, and availability** of file data

It is an essential part of **OS security** and involves both **permissions** and **encryption mechanisms**.

File System Security in Linux vs Windows

Feature	Linux (e.g., ext4, XFS)	Windows (NTFS)
Permission Model	Based on **User/Group/Other (UGO)** with rwx bits	Based on **Access Control Lists (ACLs)**

Feature	Linux (e.g., ext4, XFS)	Windows (NTFS)
Default Command Tools	chmod, chown, umask, setfacl	GUI + icacls, takeown, Group Policy Editor
Granular Access Control	Limited in default POSIX (rwx only), extended with ACLs	Highly granular: full control, read, write, execute, delete
File Encryption	Not built-in (requires tools like ecryptfs, gpg)	Built-in **Encrypting File System (EFS)**
Mandatory Access Control (MAC)	Available (SELinux, AppArmor)	Not standard; available in enterprise tools
Auditing & Logging	auditd, syslog, audit rules for file access	Advanced auditing via Event Viewer & NTFS logging
Hidden/System Files	Hidden by dot prefix (e.g., .bashrc)	Hidden attribute and system flag on file
Filesystem Support	ext3/ext4, Btrfs, XFS, etc.	NTFS, exFAT, ReFS

Q. List and explain file protection mechanisms in Linux/Windows.

File protection mechanisms are techniques used by **operating systems** to control how users and applications **access, modify, or execute** files.
Both **Linux** and **Windows** have built-in tools and systems to enforce **access control, auditing, and encryption** to secure file data.

File Protection Mechanisms in Linux

Mechanism	Explanation
UGO Permissions	Every file has permissions for **User**, **Group**, and **Others**: • r = read, w = write, x = execute Example: chmod 755 file.txt (user: all, group: read/execute, others: read/execute)
chmod & chown	chmod changes permission bits (like rwx), chown changes the file's owner or group
umask	Sets **default permissions** for new files when they're created
ACL (Access Control List)	Provides more detailed control than UGO Commands: getfacl, setfacl

Mechanism	Explanation
SELinux / AppArmor	**Mandatory Access Control (MAC)** systems Enforce strict security rules using labels/policies
File Encryption	Encrypt files/directories using tools like **eCryptfs**, **GnuPG**, or **LUKS**
Audit Logs	Use auditd to **log access**, permission changes, and **unauthorized attempts**

File Protection Mechanisms in Windows

Mechanism	Explanation
NTFS Permissions	Supports permissions like **Read, Write, Execute, Delete, Modify, Full Control** Set via **file Properties → Security tab**
ACL (Access Control List)	Grants specific file/folder access to users or groups Can be managed via **GUI** or icacls command
Ownership & Inheritance	Files have **owners**, and permissions can **inherit** from parent folders
EFS (Encrypting File System)	Built-in **file-level encryption** for NTFS Enabled via right-click → **Encrypt**
BitLocker	Provides **full-disk encryption**; available on **Windows Pro/Enterprise editions**
Group Policy	Admins enforce **system-wide or network-wide** access rules and restrictions
File Auditing	Tracks access, changes, or deletions using **Event Viewer → Security logs**

Q. What are common Linux and Windows vulnerabilities?

Both **Linux** and **Windows** operating systems can suffer from **vulnerabilities**, which are **security weaknesses** that attackers can exploit to:

- Gain **unauthorized access**
- Escalate privileges
- **Steal or corrupt data**
- Disrupt system operations (DoS)

Though both OSes aim for security, their **design, usage, and attack surfaces** differ — leading to **different types of common vulnerabilities**.

Common Vulnerabilities in Linux

Vulnerability	Description
Privilege Escalation	Attackers exploit misconfigured sudo rules or kernel bugs to become root
Unpatched Kernel/Packages	Outdated software may have known exploits (buffer overflows, CVEs)
Weak File Permissions	Misconfigured chmod, chown, or umask can allow unauthorized access to sensitive files
SUID/SGID Misuse	Executables with the SUID bit can be misused to run as root
Shell Injection	Poorly secured scripts or services may allow code injection through shell commands
Exposed Services	Services like SSH, FTP, Apache running with default configs may be vulnerable to brute-force, DoS, or misconfig attacks
Improper Input Validation	Web apps on Linux (e.g., PHP apps on Apache) can be vulnerable to SQL injection or command injection
Weak Authentication	No password policies, open SSH, or improper use of PAM modules

---

Common Vulnerabilities in Windows

Vulnerability	Description
DLL Hijacking	Malicious DLLs are loaded in place of legitimate ones
Unpatched OS/Software	Exploits like EternalBlue target outdated systems (e.g., used in WannaCry ransomware)
Weak Authentication	Poor password policies or stored credentials in plaintext
Disabled UAC / Defender	Attackers disable security controls like UAC or antivirus to persist
ActiveX / COM Exploits	Browser or Office-based vulnerabilities through scripting and ActiveX controls
Registry Exploits	Malware can modify registry keys to auto-start or escalate privileges

Vulnerability	Description
Exposed Admin Shares	Default shared folders (like C$) can be abused in network attacks
Lack of Proper ACLs	Incorrect NTFS permissions allow unauthorized access or privilege abuse

Q. Explain database security.

Database security refers to the set of **tools, controls, and policies** designed to **protect the confidentiality, integrity, and availability** of data stored in a **Database Management System (DBMS)**.

It protects the database against:

- **Unauthorized access**
- **Malicious attacks**
- **Accidental data loss**
- **Data corruption**
- **Internal misuse**

Why Is Database Security Important?

Reason	Explanation
Protect sensitive data	Prevent leakage of PII, passwords, business secrets, etc.
Prevent unauthorized access	Stop users or hackers from viewing or changing restricted data
Ensure integrity	Prevent unauthorized modifications to records
Ensure compliance	Meet legal requirements like GDPR, HIPAA, PCI-DSS
Preserve availability	Ensure data and systems are available even under attack or failure

Q. List and explain database security requirements.

Database security requirements define the necessary safeguards to ensure that the **confidentiality, integrity, and availability** of data stored in a database are maintained.

These requirements are essential for protecting sensitive information from:

- Unauthorized access
- Tampering
- Data loss
- Insider threats
- External attacks

Database Security Requirements

---

1. Access Control

- Only **authorized users** should be able to **access or modify** the database.
- Based on:
  - **User roles**
  - **Privileges** (e.g., SELECT, INSERT, DELETE)
  - **Authentication** (passwords, tokens, biometrics)

Prevents unauthorized access or actions.

---

2. Authentication

- Verifies the **identity** of users before granting access.
- Can use:
  - Username/password
  - Multi-factor authentication (MFA)
  - Certificates or biometric credentials

Ensures only legitimate users interact with the database.

---

3. Data Integrity

- Ensures that data is **accurate**, **consistent**, and **not modified** improperly.
- Mechanisms include:
  - Constraints (e.g., foreign keys)

- Transaction controls (ACID properties)
- Digital signatures or checksums

Prevents unauthorized or accidental data corruption.

---

4. Data Confidentiality

- Protects sensitive data from being **disclosed to unauthorized users**.
- Techniques:
  - **Encryption** (at rest and in transit)
  - **Data masking** or **tokenization**
  - Least privilege access

Keeps private data hidden from unauthorized viewers.

---

5. Auditing and Monitoring

- Tracks all access and changes to the database.
- Maintains logs of:
  - Login attempts
  - Data queries or updates
  - Security policy violations

Helps detect and investigate suspicious activity.

---

6. Backup and Recovery

- Protects against **data loss** from crashes, attacks, or hardware failure.
- Includes:
  - Regular backups
  - Redundancy
  - Disaster recovery plans

Ensures data is restorable and always available.

---

7. Protection Against SQL Injection

- Prevents malicious SQL code from altering or stealing data.

- Use:
    - **Parameterized queries**
    - **Input validation**
    - **Stored procedures**

Defends against a common and dangerous attack vector.

---

8. Secure Configuration and Patch Management

- Default settings must be **hardened** (e.g., disabling unused services).
- Keep database software **up to date** with patches for known vulnerabilities.

Prevents exploitation of known flaws.

Q. What are inference attacks? How are they prevented? Give an example.

An **inference attack** occurs when an attacker uses **legally permitted queries** to **indirectly infer sensitive information** from a database — **without directly accessing the restricted data**.

It exploits **patterns, statistical queries, and logic** to deduce protected or private values.

How Inference Attacks Work

Even if direct access is denied, a user may ask **seemingly innocent aggregate or filtered queries**, and from their results, **infer confidential data**.

Types of Inference Attacks

Type	Description
Direct inference	Data is revealed via specific combinations of query results
Indirect inference	Data is deduced from statistical, aggregate, or pattern-based analysis
Tracker attacks	Use logical identities to bypass query restrictions

---

How to Prevent Inference Attacks

Method	Description
Query restriction	Limit queries that return very few records (e.g., < 3 rows)
Noise addition	Introduce small random noise to aggregate results
Data suppression	Suppress output when results could reveal individual records
Access control and views	Restrict sensitive fields or rows using **views or roles**
Audit and monitoring	Detect suspicious query patterns or repeated attempts
Inference detection systems	AI or rule-based systems to flag risky query combinations

Q. Explain multilevel database security model.

A **Multilevel Database Security Model** is designed to **store data with different sensitivity levels** (e.g., Confidential, Secret, Top Secret) in the **same database** — while ensuring that **users only access data they are authorized to see**.

It combines **data classification** with **user clearance levels** to enforce **mandatory access control (MAC)**.

Key Concepts

Term	Meaning
Subjects	Users or processes requesting access
Objects	Data items (tables, rows, fields)
Clearance	Security level assigned to a subject (e.g., Secret)
Classification	Security level assigned to a data object

Each access is permitted only if:

Subject Clearance ≥ Object Classification

Example Security Levels

Level	Description
Public	No restrictions
Confidential	Sensitive internal data

Level	Description
Secret	Important internal data
Top Secret	Highly sensitive, limited access

Example Scenario

Imagine a military payroll database:

Name	Salary	Clearance Level
Alice	$50,000	Secret
Bob	$80,000	Confidential

- A user with **Confidential clearance** can:
  - See Bob's salary
  - **Cannot see** Alice's salary
- A user with **Secret clearance** can see both.

Bell–LaPadula Model (Read/Write Rules)

The multilevel model is often implemented using **Bell–LaPadula (BLP)** security principles:

Rule	Description
No Read Up	A user **cannot read** data above their clearance level
No Write Down	A user **cannot write** to a lower level (to prevent leaks)

This ensures **confidentiality** is not breached.

Challenges in Multilevel Databases

Challenge	Description
Inference attacks	Low-level users may infer high-level data from allowed queries
Polyinstantiation	Same record may exist at different levels with different values to prevent data leakage
Complexity	Managing labels and enforcing consistent access is hard
Performance overhead	Additional checks may slow down access

Techniques Used

Technique	Function
Labeling	Assigns security labels to data and users
Views	Creates filtered subsets of data for each clearance level
Query modification	Alters user queries to return only authorized data
Polyinstantiation	Allows multiple versions of the same row for different users

Q. Explain SQL injection and preventive mechanisms.

SQL Injection is a type of **code injection attack** where an attacker manipulates input fields in a web application to **inject malicious SQL commands** into the backend database.

The goal is to **bypass authentication**, **steal data**, **modify records**, or even **delete entire databases**.

How SQL Injection Works

In many web applications, user input is passed directly into SQL queries without proper validation.

Example (Vulnerable Code):

SELECT * FROM users WHERE username = 'admin' AND password = '1234';

If an attacker enters:

username: ' OR '1'='1

password: ' OR '1'='1

The resulting query becomes:

SELECT * FROM users WHERE username = '' OR '1'='1' AND password = '' OR '1'='1';

This always returns **true**, giving the attacker **unauthorized access**.

Types of SQL Injection

Type	Description
Classic SQLi	Injects raw SQL via user input

Type	Description
Blind SQLi	No error messages, but attacker infers info by observing response
Time-based Blind SQLi	Uses SLEEP() or delays to infer true/false conditions
Error-based SQLi	Relies on database error messages for information
Union-based SQLi	Injects UNION SELECT to combine attacker-controlled data with legitimate output

---

Effects of SQL Injection

Impact	Description
Bypass authentication	Log in without valid credentials
Destroy data	DROP tables or delete records
Data theft	Extract usernames, passwords, credit card numbers
Privilege escalation	Gain admin rights in the DBMS
Remote code execution	In some DBs, execute system commands (advanced SQLi)

Prevention Techniques

A. Parameterized Queries / Prepared Statements (Best Practice)

- Separate SQL code from data input
- Supported in all major languages (e.g., Python, Java, PHP)

Example (safe):

cursor.execute("SELECT * FROM users WHERE username = ? AND password = ?", (user, pwd))

B. Stored Procedures

- Use pre-defined queries on the server
- Reduces exposure to injected input

C. Input Validation and Sanitization

- Reject inputs with special characters (', ", --, ;)
- Use **whitelisting**, not just blacklisting

D. Least Privilege Access

- Application accounts should have only **necessary DB permissions**
- Never allow direct access to DROP/DELETE by low-privilege users

E. Web Application Firewall (WAF)

- Monitors and filters out suspicious SQL traffic

F. Error Handling and Logging

- Disable detailed DB error messages from being shown to users
- Log all failed queries and suspicious activity

Q. How does encryption protect sensitive data in a database?

Databases often store **sensitive information** such as:

- Passwords
- Credit card numbers
- Personal details (PII)
- Medical or financial records

If attackers gain access to the database, **unencrypted data is immediately exposed**.

Encryption ensures that even if attackers access the data, they can't read it without the decryption key.

Types of Database Encryption

Type	**Description**
Data-at-Rest Encryption	Encrypts data stored on disk (e.g., tables, files, backups)
Data-in-Transit Encryption	Encrypts data as it moves between client and database server (e.g., via TLS/SSL)
Column-level Encryption	Encrypts specific sensitive columns (e.g., credit_card, password)
Transparent Data Encryption (TDE)	Automatically encrypts the entire database storage layer without changes to the application
Application-level Encryption	Data is encrypted/decrypted in the application before reaching the database

---

How Encryption Protects Data

Scenario	Protection Provided
Database breach	Attackers cannot read encrypted values without keys
Lost or stolen backups	Encrypted backups are useless without decryption
Eavesdropping during transmission	Encrypted channels (e.g., TLS) prevent data leakage
Insider misuse	Even internal users can't read encrypted fields without authorization

---

Best Practices for Secure Encryption

Practice	Explanation
Use Strong Algorithms	e.g., AES-256, RSA-2048
Store Keys Securely	Use hardware security modules (HSMs) or key vaults
Rotate Keys Periodically	Prevent long-term exposure in case of key compromise
Audit Decryption Access	Monitor who accesses decrypted data
Combine with Access Control	Encryption works best with user authentication and role-based access

Q. What are common techniques for hardening an operating system?

Operating System (OS) hardening is the process of **securing an OS** by reducing its **vulnerabilities** and **attack surface**.

It involves configuring the system to make it **more secure** against **malware**, **intrusions**, and **unauthorized access**.

Common Techniques for OS Hardening

Technique	Purpose
Disable Unused Services	Reduce attack surface
Apply Security Patches	Fix known vulnerabilities

Technique	Purpose
Enable Firewall	Block unwanted traffic
Use Strong Passwords	Prevent unauthorized login
Limit Privileges	Minimize damage if compromised
Log and Audit	Track system activity
Secure File Permissions	Protect sensitive files
Use Antivirus	Detect malware
Use Security Modules	Enforce strict access control
Harden Default Accounts	Prevent misuse of built-in users

Q.

Q.