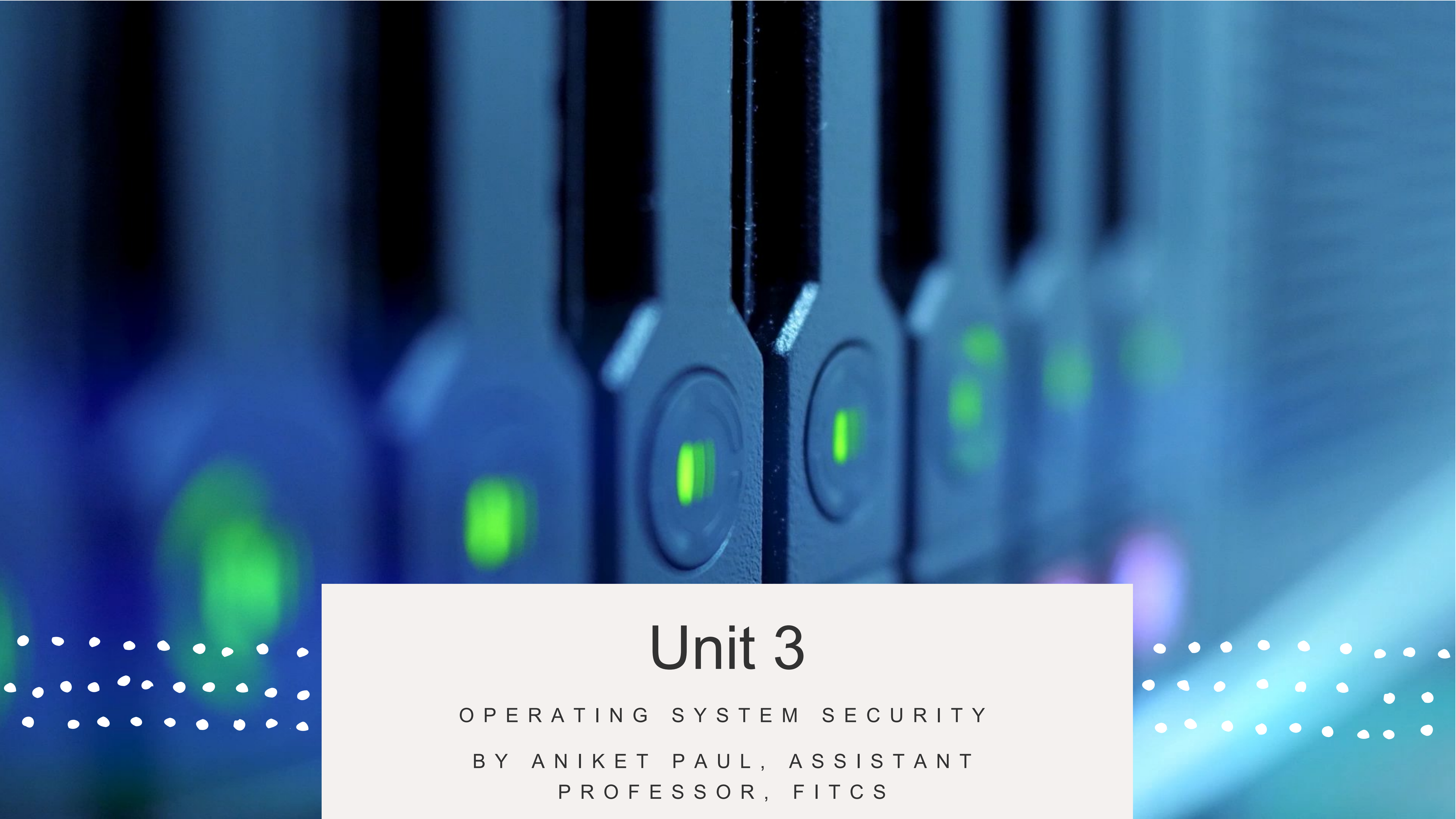


Unit 3

OPERATING SYSTEM SECURITY

BY ANIKET PAUL, ASSISTANT PROFESSOR, FITCS

1: Role of Operating Systems in Information Systems Applications

Definition of an Operating System

An **Operating System (OS)** is system software that manages computer hardware and software resources, providing essential services to applications. It ensures efficient execution of tasks, manages memory, and enables user interaction with hardware components.

Importance of OS in Cybersecurity

- The OS is responsible for enforcing security policies, managing user access, and protecting system resources from malicious activities. Some of the critical security functions of an OS include:
- **User Authentication** – Ensuring that only authorized users can access the system.
- **Process Isolation** – Preventing unauthorized processes from interfering with others.
- **System Monitoring and Logging** – Recording system activities to detect and investigate security incidents.
- **Access Control** – Regulating which users and programs can access specific resources.
- **Software Execution Control** – Managing applications to prevent malicious software from executing.

OS as a Critical Component in Information Systems

Key responsibilities include:

- **Memory Management:** Allocating memory to running processes while preventing memory leaks.
- **Process Management:** Scheduling and controlling the execution of programs.
- **Storage Management:** Handling file storage, retrieval, and security.
- **Network Management:** Facilitating secure communication between devices and networks.

OS as a Security Enforcer

The OS plays a significant role in **enforcing security policies** within an organization's information system. It ensures compliance with security guidelines and implements protective mechanisms to safeguard against cyber threats.

Key Security Responsibilities of an OS

1.User Authentication & Access Control

1. Uses authentication mechanisms like passwords, biometrics, and multi-factor authentication (MFA).
2. Assigns user roles and permissions based on access control models (DAC, MAC, RBAC).

2.Process Isolation

1. Ensures that programs run independently without interfering with each other.
2. Uses sandboxing techniques to confine potentially malicious applications.

3.System Monitoring & Logging

1. Records system activities to track unauthorized access and detect anomalies.
2. Uses log management tools like SIEM (Security Information and Event Management) for threat detection.

4.Secure Software Execution

1. Implements measures like digital signatures and application whitelisting to prevent execution of malicious software.
2. Uses mechanisms such as **Control Flow Integrity (CFI)** and **Runtime Application Self-Protection (RASP)** to enhance security.

Security Challenges in Operating Systems

Common OS Security Threats and Vulnerabilities

Despite security implementations, operating systems remain **prime targets** for cyberattacks. Below are some of the most common OS security threats:

1. **Malware Attacks**
   1. **Viruses** – Self-replicating programs that corrupt files and slow down the system.
   2. **Trojans** – Malicious programs disguised as legitimate software.
   3. **Rootkits** – Stealthy malware that gains deep-level system access.
2. **Insider Threats**
   1. Employees or administrators misusing their privileges to access sensitive data.
   2. Weak password policies leading to unauthorized access.
3. **Privilege Escalation Attacks**
   1. Exploiting OS vulnerabilities to gain administrative control.
   2. Example: A **buffer overflow attack** that allows an attacker to execute arbitrary code.
4. **Denial-of-Service (DoS) Attacks**
   1. Overloading system resources to crash or slow down operations.
   2. Example: An attacker sending excessive requests to an OS service, making it unavailable to legitimate users.
5. **Zero-Day Exploits**
   1. Attacks that take advantage of unknown OS vulnerabilities before patches are released.
   2. Example: Windows and Linux Kernel vulnerabilities exploited in cyber espionage campaigns.

Key Security Mechanisms in Operating Systems

To mitigate these security threats, modern operating systems implement **robust security measures**, including:

1. Authentication Mechanisms

- **Passwords** – Traditional authentication method but vulnerable to attacks like brute-force and phishing.
- **Biometric Authentication** – Uses fingerprints, facial recognition, or iris scans.
- **Multi-Factor Authentication (MFA)** – Combines two or more authentication factors for added security.

2. Access Control Models

Operating systems use different **Access Control Models (ACMs)** to regulate user access:

- **Discretionary Access Control (DAC)** – Users have control over file permissions (e.g., Windows file permissions).
- **Mandatory Access Control (MAC)** – Strict access policies enforced by the OS (e.g., SELinux in Linux).
- **Role-Based Access Control (RBAC)** – Users are assigned roles with predefined permissions (e.g., Enterprise Active Directory).

3. Secure Boot and Trusted Platform Module (TPM)

- **Secure Boot** ensures only trusted operating system components are loaded during startup, preventing rootkits.
- **Trusted Platform Module (TPM)** is a security chip that stores cryptographic keys securely, enhancing OS security.

4. Patch Management and System Updates

- Regular updates ensure OS security by fixing vulnerabilities and patching security flaws. **Automated patch management tools** like Windows Update and Linux package managers (APT, YUM) help streamline security updates.

2: Operating System Security

Introduction

Operating System (OS) security is a critical aspect of cybersecurity that ensures the integrity, confidentiality, and availability of system resources. A secure OS protects user data, prevents unauthorized access, and defends against cyber threats such as malware, privilege escalation, and insider attacks.

In this lecture, we will explore:

- Security features in modern operating systems
- The importance of security updates and patch management
- OS security models and enforcement mechanisms
- OS security testing and auditing
- A well-configured and regularly updated OS can significantly reduce security risks and act as the first line of defense against cyber threats.

Security Features in Modern Operating Systems

1.1 Secure Kernel and Microkernel Architecture

The **kernel** is the core component of an operating system that manages system resources and enforces security policies. There are two primary kernel architectures:

- **Monolithic Kernel:** The entire OS runs in a single address space, making it faster but more vulnerable to security flaws. (e.g., Linux, Windows)
- **Microkernel:** Only essential OS functions run in the kernel, while other components run in user space, enhancing security. (e.g., MINIX, QNX)

1.2 OS Hardening Techniques

OS hardening refers to the process of securing an operating system by reducing its attack surface. Best practices include:

- **Disabling Unnecessary Services:** Unused services can introduce vulnerabilities. Disabling them reduces potential attack vectors.
- **Applying the Principle of Least Privilege (PoLP):** Users and applications should only have the permissions necessary to perform their tasks.
- **Restricting Remote Access:** Remote desktop protocols (RDP) should be restricted or secured with multi-factor authentication (MFA).
- **Implementing Firewall Rules:** A properly configured firewall blocks unauthorized network traffic.
- **Enabling Secure Boot:** Ensures that only trusted software loads during system startup, preventing boot-level malware.

Security Features in Modern Operating Systems

1.3 Secure Configuration Guidelines (CIS Benchmarks)

The **Center for Internet Security (CIS)** provides security configuration benchmarks for operating systems such as Windows, Linux, and macOS. These benchmarks define best practices for:

- Password policies
- File system permissions
- Service configurations
- Network security settings
- Following CIS guidelines enhances OS security by minimizing misconfigurations that could lead to vulnerabilities.

Security Updates and Patch Management

2.1 Importance of OS Patching in Security

Operating systems frequently release security patches to fix vulnerabilities, address security flaws, and enhance system stability. Failing to apply updates exposes systems to cyberattacks, including ransomware and zero-day exploits.

2.2 Types of OS Patches

- **Security Patches:** Fix vulnerabilities that could be exploited by attackers.
- **Feature Updates:** Introduce new OS functionalities and enhancements.
- **Hotfixes:** Quick updates that address specific critical issues.

2.3 Challenges in Patch Management

While patching is essential, it presents several challenges:

- **Compatibility Issues:** Updates may cause software conflicts, leading to system instability.
- **Zero-Day Vulnerabilities:** Cybercriminals exploit unknown flaws before patches are released.
- **Patch Delays:** Organizations may delay updates due to operational concerns, increasing security risks.

2.4 Best Practices for Patch Management

- **Automated Updates:** Use centralized patch management tools like **Windows Update Services (WSUS)** or **Linux package managers (APT/YUM)**.
- **Patch Testing:** Test patches in a controlled environment before deployment.
- **Regular Security Audits:** Assess systems for missing updates and vulnerabilities.
- **Emergency Patch Deployment:** Immediately apply critical security patches to mitigate zero-day threats.

OS Security Models and Enforcement Mechanisms

Operating systems implement different security models to enforce access control and data protection. These models define how security policies are applied within the system.

3.1 Bell-LaPadula Model (Confidentiality)

- Developed for government and military applications.
- **Ensures confidentiality** by restricting users from accessing higher security levels.
- Uses two main rules:
    - **No Read Up (NRU):** Users cannot read data at higher security levels.
    - **No Write Down (NWD):** Users cannot write data to lower security levels.
- Example: A government agency using classification levels (Top Secret, Secret, Confidential).

3.2 Biba Model (Integrity)

- Focuses on data integrity by preventing unauthorized modification.
- Uses the **No Write Up, No Read Down** principle:
    - Users **cannot write** to higher integrity levels.
    - Users **cannot read** from lower integrity levels.
- Example: Ensuring financial transactions maintain data integrity without unauthorized modifications.

OS Security Models and Enforcement Mechanisms

3.3 Clark-Wilson Model (Well-Formed Transactions)

- Designed for commercial environments, ensuring data integrity through **well-formed transactions**.
- Requires strict separation of duties (e.g., only authorized personnel can approve financial transactions).

Example: Preventing fraud in banking systems by ensuring two-person authentication for money transfers.

3.4 Reference Monitor Concept

A **Reference Monitor** is a security component that mediates all access requests between users and system resources. It must be:

- **Tamper-proof** – Cannot be altered by unauthorized users.
- **Always Invoked** – Enforces security policies on every access attempt.
- **Verifiable** – Must be tested to ensure security compliance.

Example: Mandatory Access Control (MAC) in **SELinux**, which strictly enforces security policies based on predefined rules.

OS Security Testing and Auditing

To ensure an OS remains secure, organizations perform **security testing and auditing** using various tools and techniques.

4.1 Security Testing Tools

- **Microsoft Baseline Security Analyzer (MBSA):** Assesses Windows security configurations and missing patches.
- **Lynis (Linux Security Auditor):** Scans Linux systems for security misconfigurations.
- **Nessus & OpenVAS:** Conduct vulnerability scans to identify security weaknesses.

4.2 Log Analysis and Intrusion Detection

- **System Logs:** OS logs record user activities, login attempts, and system errors. Reviewing logs helps detect unauthorized access.
- **Security Information and Event Management (SIEM):** SIEM tools aggregate log data from multiple systems to detect anomalies.
- **Host-Based Intrusion Detection System (HIDS):** Monitors OS behavior for signs of cyber threats.

4.3 OS Forensic Analysis

- **Memory Forensics:** Analyzes RAM for evidence of malware or unauthorized access.
- **File System Analysis:** Examines file access patterns and unauthorized modifications.
- **Registry Analysis (Windows):** Detects suspicious changes in system configurations.

4.4 Incident Response for OS Security

Organizations must have an **Incident Response Plan (IRP)** to handle OS security breaches. Key steps include:

- **Detection:** Identify suspicious activities using security monitoring tools.
- **Containment:** Isolate compromised systems to prevent further damage.
- **Eradication:** Remove malware and fix vulnerabilities.
- **Recovery:** Restore affected systems from secure backups.
- **Lessons Learned:** Analyze the incident and improve security policies.

3: Protected Objects and Methods of Protection

In any modern computing environment, operating systems must protect critical resources such as files, memory, and system processes from unauthorized access. This protection ensures system integrity, data confidentiality, and proper functionality.

We will focus on:

- **What are protected objects in an OS?**
- **Methods of protection for OS resources**
- **Access control mechanisms**
- **User authentication and authorization techniques**
- **Real-world examples of OS protection mechanisms**

By understanding these topics, we can ensure that operating systems maintain robust security measures against cyber threats and unauthorized access.

What Are Protected Objects in an Operating System?

Definition of Protected Objects

A **protected object** in an operating system refers to any **resource** that requires controlled access. These resources must be safeguarded to prevent unauthorized use or modification.

Examples of **protected objects** in an OS include:

- **Files and Directories:** Documents, system files, and logs.
- **Memory Segments:** RAM and virtual memory.
- **Processes and Threads:** Running applications and system services.
- **Devices:** Hardware components such as printers, USB drives, and network interfaces.
- **User Accounts and Sessions:** Authentication credentials and active user sessions.

Each of these objects needs **specific security controls** to ensure that only authorized users and processes can access them.

Methods of Protection in an OS

Operating systems employ multiple mechanisms to protect these critical objects. The key protection methods include:

2.1 Discretionary Access Control (DAC)

- **Definition:** DAC allows resource owners (users) to define who can access their files and applications.
- **Example:** In Windows, users can modify file permissions using **NTFS security settings**.
- **Security Concern:** Since users have full control over permissions, they might **accidentally** or **intentionally** grant access to unauthorized users, creating security risks.

2.2 Mandatory Access Control (MAC)

- **Definition:** MAC enforces strict access rules defined by the OS, rather than user discretion.
- **Example:** Security-Enhanced Linux (SELinux) and Windows Mandatory Integrity Control (MIC) restrict access based on security policies.
- **Use Case:** Military and government environments where data confidentiality is paramount.

2.3 Role-Based Access Control (RBAC)

- **Definition:** Access is granted based on user roles rather than individual permissions.
- **Example:** A company might define roles such as "Admin," "Manager," and "Employee," each with different levels of access.
- **Advantage:** Simplifies permission management in enterprise environments.

2.4 Attribute-Based Access Control (ABAC)

- **Definition:** Uses dynamic attributes (such as user location, device type, or time of access) to grant or deny access.
- **Example:** A banking system might allow transactions only if the request originates from an approved location or during business hours.
- **Security Benefit:** Offers **fine-grained access control** based on multiple factors.

File Protection Mechanisms

3.1 File Permissions and Access Control Lists (ACLs)

File permissions: Define who can **read, write, or execute** a file.

Access Control Lists (ACLs): Provide more granular permissions than simple file access rules.

Example in Linux:

- chmod 644 filename: Grants **read and write** access to the owner, but **read-only** access to others.
- chown user:group filename: Changes the file owner and group.

3.2 Encryption for File Security

Encryption protects files by converting them into an unreadable format unless decrypted by an authorized user.

Example: BitLocker (Windows), FileVault (Mac), and LUKS (Linux) provide **full disk encryption (FDE)** to prevent unauthorized data access.

Use Case: Encryption is crucial for protecting **sensitive data on lost or stolen devices**.

3.3 Secure File Deletion

Even after deleting a file, it can often be **recovered** using forensic tools.

Solution: Secure deletion methods like:

- shred -u filename (Linux)
- **CCleaner** for secure file erasure on Windows
- **DoD 5220.22-M** (Department of Defense standard for data wiping)

Memory Address Protection

Memory protection ensures that processes cannot **interfere with each other** or **access unauthorized memory segments**. Key protection mechanisms include:

4.1 Virtual Memory and Address Space Isolation

- **Virtual memory:** Separates a program's memory from others, preventing conflicts.
- **Page Tables:** Map logical memory addresses to physical memory.
- **Use Case:** Prevents one faulty application from crashing the entire OS.

4.2 Buffer Overflow Protection

What is it? Attackers exploit buffer overflow vulnerabilities to inject malicious code.

Protection Techniques:

- **Stack Canaries:** Small security tokens placed before return addresses to detect overflow attempts.
- **Address Space Layout Randomization (ASLR):** Randomizes memory locations to prevent attacks.
- **Data Execution Prevention (DEP):** Blocks execution of malicious code in protected memory regions.

4.3 Memory Access Control (Kernel vs. User Mode)

- **Kernel Mode:** Full access to system memory and hardware.
- **User Mode:** Restricted access to prevent accidental system modifications.
- **Benefit:** Separation prevents malware from easily gaining full system control.

Process and Application Protection

5.1 Process Isolation

•Ensures that one process **cannot interfere** with another.

•**Techniques:**

- •**Virtualization:** Running applications in separate virtual machines (VMs).
- •**Containers (Docker, Kubernetes):** Isolating applications at the OS level.

5.2 Sandboxing

•**Definition:** Runs applications in a restricted environment to prevent system-wide damage.

•**Example:** Web browsers (Chrome, Edge) run tabs in **isolated sandboxes** to prevent malware infections.

5.3 Secure Process Termination

•Ensures that terminated processes **do not leave security holes** (e.g., zombie processes).

•**Best Practice:**

- •Use kill -9 PID in Linux to forcefully terminate processes.
- •Implement **timeout policies** for inactive processes

Authentication and Authorization in OS Protection

6.1 Authentication Methods	**Passwords:** Traditional method but vulnerable to brute force attacks.
	Multi-Factor Authentication (MFA): Adds extra security (e.g., biometrics, OTPs).
	Public Key Infrastructure (PKI): Uses cryptographic keys for secure authentication.
6.2 Authorization Techniques	**Single Sign-On (SSO):** Allows users to authenticate once and access multiple services securely.
	OAuth & OpenID: Secure methods for delegating access without exposing credentials.
	Role-Based Access Control (RBAC): Ensures **only authorized users** can access certain system resources.

4: Memory Address Protection and File Protection Mechanisms

Introduction

Operating systems handle a variety of critical security tasks, including **memory protection and file security**. These two areas are essential for preventing unauthorized access, data corruption, and malicious attacks.

This lecture will cover:

- **Memory address protection** and its role in OS security
- **Techniques used to safeguard memory from unauthorized access**
- **Common memory security threats and mitigations**
- **File protection mechanisms in modern operating systems**
- **Access control, encryption, and secure file deletion**

Memory Address Protection

1.1 What Is Memory Address Protection?

Memory protection is a security feature that prevents one process from **accessing another process's memory space**. Without proper memory protection, a malicious or faulty program could:

- Read or modify another program's data.
- Overwrite system files stored in memory.
- Cause crashes or system instability.

Operating systems enforce **memory address protection** using specialized hardware and software techniques to ensure that each program runs securely within its designated memory space.

Techniques for Memory Address Protection

2.1 Virtual Memory and Address Space Isolation

Modern operating systems use **virtual memory** to isolate each program's memory space.

Definition: Virtual memory allows each process to operate as if it has exclusive access to the entire system memory, even though it is sharing resources.

Key Components:

- **Page Tables:** Map virtual addresses to physical memory locations.
- **Memory Segmentation:** Divides memory into segments (code, data, stack, heap).
- **Logical Addressing:** Ensures a program cannot directly access physical memory.

Example:
When a program requests memory, the OS assigns it a **logical address**, which is translated into a **physical address** using page tables. If a process tries to access memory outside its range, the OS blocks it, preventing unauthorized access.

2.2 Buffer Overflow Protection

Buffer overflow attacks occur when a program writes more data into a memory buffer than it can hold, potentially overwriting adjacent memory.

Mitigation Techniques:

- **Stack Canaries:** A small security value placed before return addresses to detect corruption.
- **Address Space Layout Randomization (ASLR):** Randomizes memory locations of processes to prevent attackers from predicting memory addresses.
- **Data Execution Prevention (DEP):** Prevents execution of malicious code in data memory segments.

Real-World Example:

Microsoft Windows and Linux use **ASLR** to randomize memory addresses, making it harder for attackers to execute **return-oriented programming (ROP) attacks**.

2.3 Kernel vs. User Mode Memory Protection

Operating systems differentiate between **Kernel Mode** and **User Mode** to enforce memory protection.

Kernel Mode:

- Has full access to system memory and hardware.
- Used by the operating system core (kernel).

User Mode:

- Has restricted access.
- Prevents user applications from interfering with system processes.

Example:
If a normal application attempts to access kernel memory, the OS **terminates** it immediately to prevent system crashes.

File Protection Mechanisms

What Is File Protection?

File protection mechanisms ensure that **only authorized users and processes** can access, modify, or delete files. Without proper file security, attackers could:

- Steal or modify sensitive documents.
- Execute **ransomware attacks** that encrypt user files.
- Delete critical system files, leading to system failures.

Operating systems implement various security techniques to **protect file integrity, confidentiality, and availability**.

Permission	Symbol	Numeric Representation
Read	r	4
Write	w	2
Execute	x	1

File Access Control Mechanisms

4.1 File Permissions (DAC - Discretionary Access Control)

Most operating systems use a **permission model** to define which users can read, write, or execute files.

UNIX/Linux File Permissions:
Each file is assigned:

- **Owner (user)**
- **Group**
- **Other (everyone else)**

Example:

- chmod 755 filename → Grants full access to the owner and read/execute access to others.
- ls -l filename → Displays file permissions in Linux.

Windows NTFS Permissions:

- **Full Control**
- **Modify**
- **Read & Execute**
- **Read Only**
- **Write**

These permissions can be managed using the **Windows Security Tab** in file properties.a

- **4.2 Access Control Lists (ACLs)**
- ACLs allow for **fine-grained access control** beyond simple file permissions.
- **Definition:** An ACL is a list of rules that define who can access a file and what actions they can perform.
- **Example:** A network administrator can configure ACLs to allow **read access** for some users while restricting **write access**.

File Encryption for Security

5.1 Importance of File Encryption

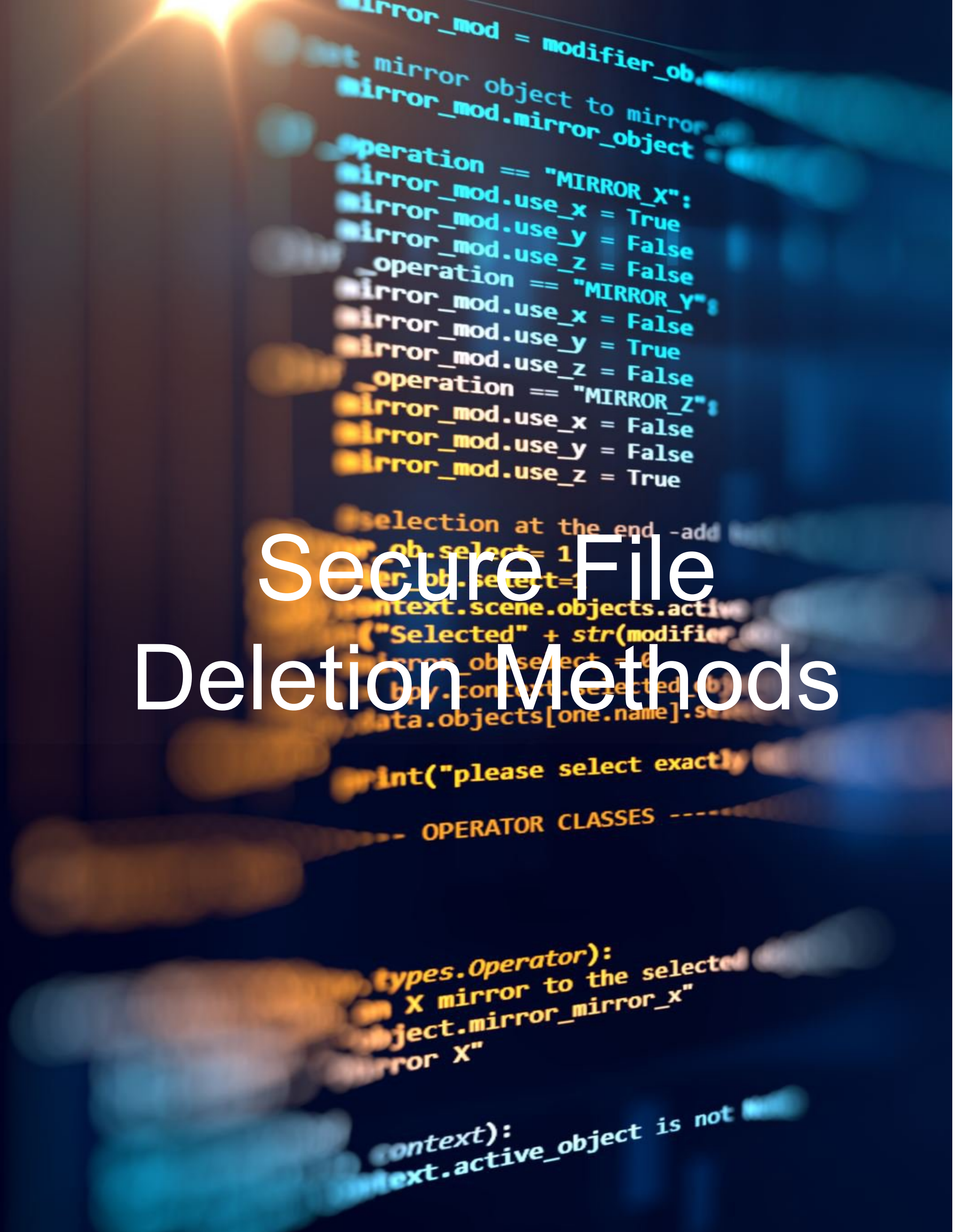
Encryption ensures that even if an unauthorized user gains access to a file, they **cannot read its contents** without the correct decryption key.

5.2 Types of File Encryption

1. **Full Disk Encryption (FDE):** Encrypts the entire disk, protecting all files.
    1. **Examples:** BitLocker (Windows), FileVault (Mac), LUKS (Linux).
2. **File-Level Encryption:** Encrypts specific files or folders.
    1. **Examples:** OpenPGP, AES-based tools.

Example Use Case:

- A **corporate laptop** that stores confidential documents should use **BitLocker** to ensure data remains safe even if the device is stolen.



Secure File Deletion Methods

Even after deleting a file, it can often be recovered using forensic tools. To ensure complete removal, use **secure file deletion techniques**:

6.1 Secure Deletion Commands

•**Linux:** shred -u filename (Overwrites a file multiple times before deleting it).

•**Windows:** Use **cipher /w:C:\folder** to overwrite deleted data.

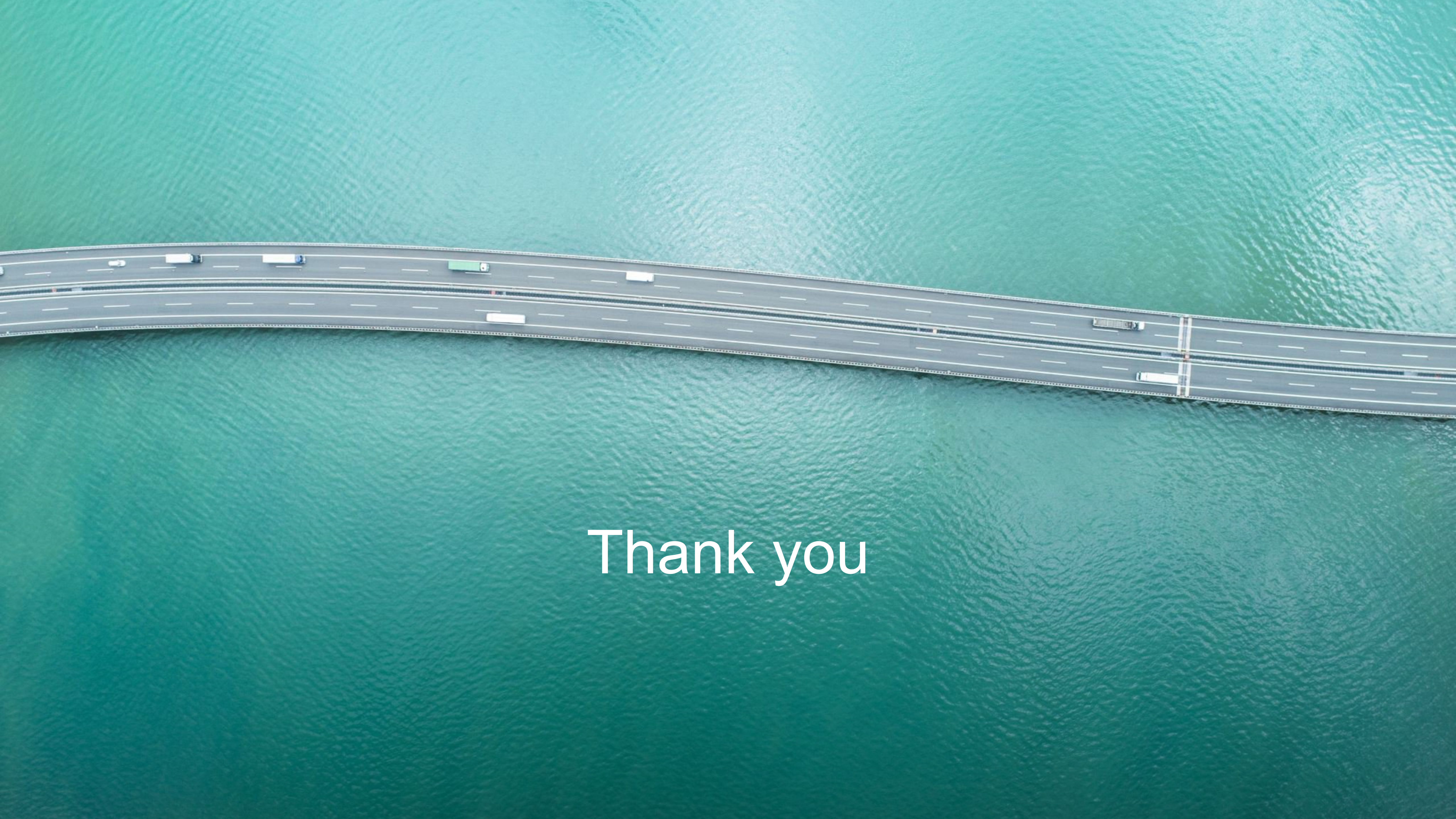
6.2 Disk Wiping Tools

•**DBAN (Darik's Boot and Nuke)** → Used for securely wiping entire hard drives.

•**CCleaner** → Offers secure deletion options.

•**DoD 5220.22-M Standard** → U.S. Department of Defense standard for wiping data.

Example Use Case:
Before selling a laptop, use **DBAN** to wipe the hard drive and prevent **data recovery by attackers**.



Thank you