Software Security

Fundamental Principles of Software Security

Dr. Zeyad Safaa Younus Saffawi

Fundamental Principles of Software Security

01

Isolation

Separating components to limit damage

02

Least Privilege

Minimal permissions required

03

Compartmentalization

Dividing systems into secure zones

04

Threat Modeling

Identifying and analyzing risks

05

Bug vs Vulnerability

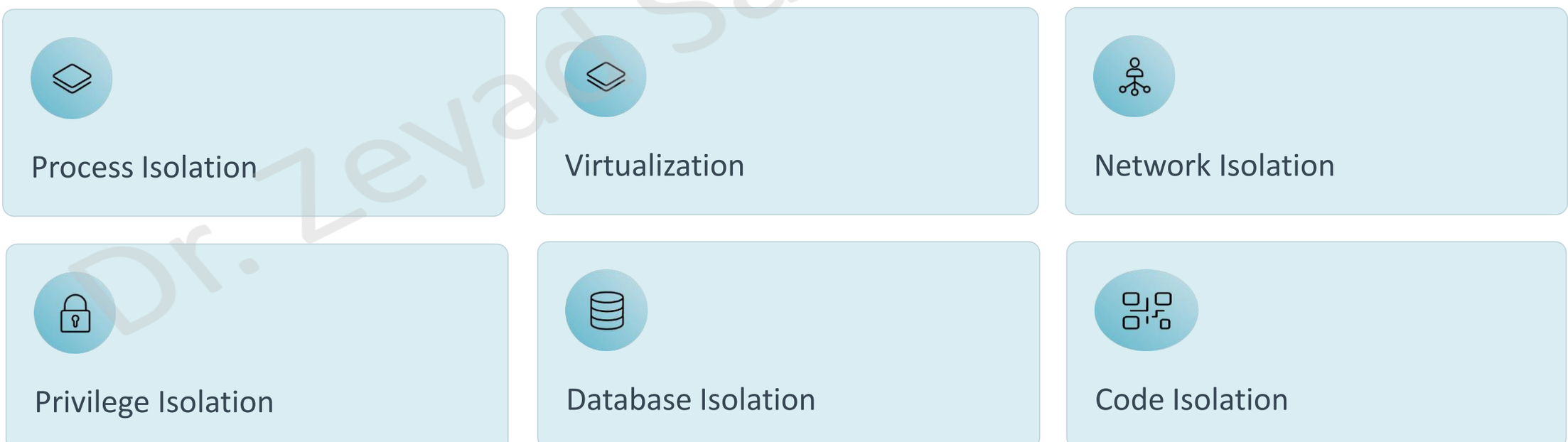
Understanding security impact

Fundamental Principles of Software Security

1. **Isolation** is a fundamental software security principle that **involves** the **practice** of **separating different components**, **processes**, or **data** within a system in order **to enhance security** and **prevent security risks** from **spreading across the system**.

- In software systems, **multiple applications and processes may run simultaneously**. **Without proper isolation**, a **vulnerability in one component could affect the entire system**. Therefore, **isolation** is a key strategy **helps contain potential security risks and preventing them from spreading to other parts of the system** to **reduce the overall attack surface** and **minimizing the impact of security breaches**.
- By implementing **isolation mechanisms**, organizations can ensure that **if one part of the system is compromised**, the **damage remains limited** and **does not affect other components**.

Several **types of isolation** are commonly used **to enhance system as explain below:**



Types of Isolation in Software Security

1. **Process Isolation** ensures that each **running process operates within its own memory space**, **separate from other processes**.

- This **prevents one process** from:
    - **Accessing the memory of another process**
    - **Modifying another program's data**
    - **Interfering with system operations**

2. Virtualization allows multiple **Virtual Machines (VMs)** to run on a **single physical machine**.

Each Virtual Machines VM:

- **Runs its own operating system instance**
- **Hypervisors manage the isolation between VMs.**
- **Isolated from other virtual machines VMs on the same physical host**

This **isolation is managed** by **a software layer** known as a **hypervisor**, which **controls and separates** the **virtual environments**.

Virtualization is widely used in:

- **Cloud computing**
- **Data centers**
- **Security sandboxes**

Types of Isolation in Software Security

3. Network Isolation divides a network into **smaller**, **isolated segments** or **zones** in order **to control traffic flow and limit the spread of potential attacks.**

For example:

- Isolating **internal corporate networks** from **public internet-facing servers**.
- Creating **DMZ (Demilitarized Zone)** networks for web servers.

This approach **prevents attackers** from **easily moving** within a **compromised network**.

4. Privilege Isolation is based on the **Principle of Least Privilege (PoLP).**

- This principle states that **users and processes** are granted the **minimum level of access rights** necessary to **perform their tasks**.
- By **restricting privileges**, the system ensures that **higher-privileged functions** are **isolated** from **lower-privileged ones**

Types of Isolation in Software Security

5. Database Isolation ensures that **data from different users** or **tenants** is kept **separate**, which means **data belonging** to different users or organizations **remains separate**.

- This is particularly important in **multi-tenant environments like cloud services**, where multiple customers share the same infrastructure.

Database isolation prevents:

- **Data leakage**
- **Unauthorized data access**
- **Cross-tenant attacks**

6. Code Isolation separates different parts of software into independent modules or services.

- **Module Isolation:** organizing the code into isolated modules or components with well-defined interfaces, meaning if a vulnerability occurs in one module, it will not easily affect the entire system.
- **Micro services Architecture:** a software architecture where an applications are built as a collection of loosely coupled, independently deployable services.

Each service:

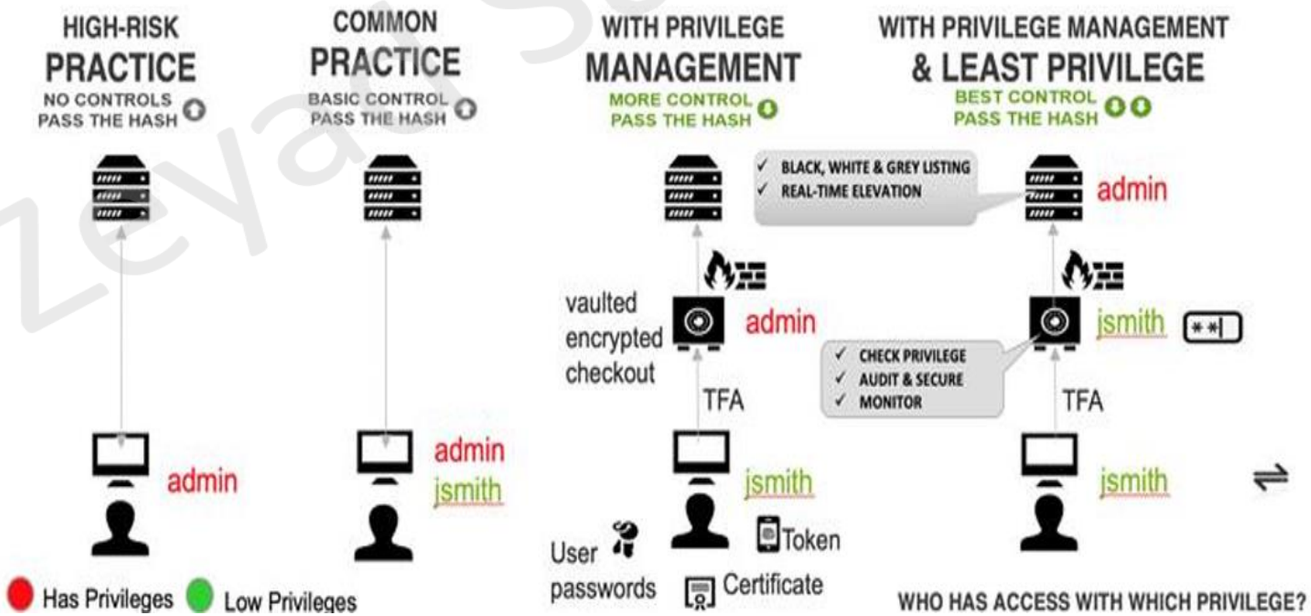
- Runs in its own isolated environment
- Performs a specific function
- Communicates through APIs

Benefit: This architecture enhances maintainability and security by isolating potential vulnerabilities and containing vulnerabilities within specific components.

Fundamental Principles of Software Security

2. **Principle of Least Privilege (PoLP)** is a fundamental security concept that **states** that **users, systems, processes, and applications** should be granted only the **minimum level of access** or **privileges necessary required to perform their** required **tasks**.

- **Limiting access rights** helps **reduce the attack surface** and **minimizes the risk of misuse of accidental or intentional misuse of privileges**.
- The **primary goal** of Least Privilege is to **reduce the risks by limiting the potential damage** caused from **security breaches or software errors**. This **minimizes** the opportunities **for malicious actors or malware** to **exploit elevated privileges**.



Applying of Least privilege in Software Security

Least privilege can be implemented in several contexts.

1. **User Accounts**

Regular Users should only **have access to the files, applications and systems needed for their job**. **Administrative privileges** should be **restricted to trusted individuals** who need them to **perform specific tasks**.

2. **System Processes**

System services and applications should **run under dedicated service accounts** with **minimal privileges, rather than using accounts with full administrative rights**. This **reduces the risk of system compromise if a service is exploited**.

3. **Applications**

Applications should request only the **permissions necessary to perform their functions**. For example, a **messaging application** should not require camera access unless it supports video calls.

4. **Network Access**

Network resources should be **segmented**, and **access** should be **restricted** based on the **principle of least privilege**, **ensuring that users or systems can only access the parts of the network they need for their operations.**

Fundamental Principles of Software Security

3. Compartmentalization is the practice of **dividing a system** into **distinct**, **isolated sections** or **compartments**, each with **specific functions, data and access controls**.

- The purpose of compartmentalization is to **limit the impact of a security breach**. **If one compartment is compromised, the other compartments remain unaffected**.
- This **strategy enhances security by reducing the attack surface and containing potential threats**.
- **Examples of Compartmentalization:**
  - **Government and Military Systems: s**ensitive information in military and government systems is compartmentalized based on classification levels (ex: **Confidential, Secret, Top Secret)** with strict access controls and isolation between compartments.
  - **Cloud Computing Environments:** Cloud systems often use compartmentalization in **multi-tenant architectures** to ensure that each tenant's data and applications are isolated from others, protecting against cross-tenant data breaches.
  - **Web Applications:** often compartmentalize user sessions, isolating them from each other to prevent session hijacking and cross-site attacks.

Fundamental Principles of Software Security

4. Threat Modeling is a structured process used to **identify, evaluate, and address potential security threats** and **vulnerabilities** that could harm a software system**, assessing their potential impact, and planning mitigations to protect the system**.

- It allows developers to **understand the security risks to a system**, **prioritize those risks, and develop strategies to mitigate them**.
- This process is crucial in **building secure software**, as it **helps in expecting and countering potential attacks before they occur**.
- The **primary goal** is to improve the **security posture** of a system by **proactively identifying and addressing potential threats**, ensuring that **security** is **integrated** throughout the **Software Development Lifecycle (SDLC)**.

Fundamental Components of Threat Modeling

1. Assets: are the **valuable components of the system that need protection**, such as:

- **Sensitive data**
- **User credentials**
- **System infrastructure**

Components of Threat Modeling in Software Security

2. Threats: are **potential actions or events that could compromise the confidentiality, integrity, or availability of assets,** such as:

- **Unauthorized access**
- **Data theft**
- **Service disruption**

3. Vulnerabilities: are weaknesses or flaws in the system that could be exploited by a threat to cause harm, such as:

- **Poor input validation**
- **Weak authentication mechanisms**

4. Attack Vectors: are the **methods** or pathways **attackers use to exploit vulnerabilities and implementing threats**, such as.

- **Malicious input**
- **Phishing attacks**
- **Exploiting exposed services**

5. Mitigations: are the security controls or countermeasures implemented to reduce or eliminate the risks posed by threats, such as.

- **Encryption**
- **Access control**
- **Input validation**

Fundamental Principles of Software Security

5. Bug vs Vulnerability

- In software security, the terms **bug** and **vulnerability** refer to **different types of software flaws**. While they are related, they have distinct meanings and implications.

➢ **Bug: is a flaw, mistake, or unintended behavior** in the **software code** that causes the **software to operate incorrectly or produce unexpected results**.

- Bugs can **arise from errors in logic, incorrect assumptions, or programming mistakes**.
- **Scope**: **Bugs can affect the functionality, performance, or usability of software.** They are not necessarily security-related and may not have any impact on the security of the system.
- **Examples**
  - A calculation error in a financial application that leads to incorrect results
  - A typo in a user interface string that displays incorrect information to the user.
  - A crash in a software program due to improper memory management.

➢ **Vulnerability** is a **specific type of bugs or flaw** that can be **exploited by attackers to compromise the security of the system.**

- Vulnerabilities create opportunities for **unauthorized access, data breaches, or other malicious activities**.
- **Scope**: **Vulnerabilities** directly **impact** the **confidentiality, integrity, or availability of a system**.
- They can be exploited to perform actions that should not be possible, such as **gaining unauthorized access, executing arbitrary code, or causing a denial of service.**
- **Examples**
  - A **buffer overflow vulnerability** that allows an attacker to **execute arbitrary code** on a system
  - An **SQL injection vulnerability** that enables an attacker to **manipulate a database.**
  - An **authentication bypass vulnerability** that lets an **attacker access a system without proper credentials.**

References

- Payer, M. (2021). *Software security: Principles, policies, and protection* (Version 0.37).