جامـــــعـة المـــــسـتـقـبـل
AL MUSTAQBAL UNIVERSITY

قـســـــم الامـــــــــــن الــــــــــــــــــــــسيبرانــــــــــــــــــــي

Department of Cyber Security

Subject:

Software Security

Class:

second

Lecturer:

Dr. Suha Alhussieny

Lecture: (3)

Attack Vectors , Denial of Service (DoS)

Section two: Attack Vectors

An attack vector, or threat vector, is a way for attackers to enter a network or system. Common attack vectors include social engineering attacks, credential theft, vulnerability exploits, and insufficient protection against insider threats.

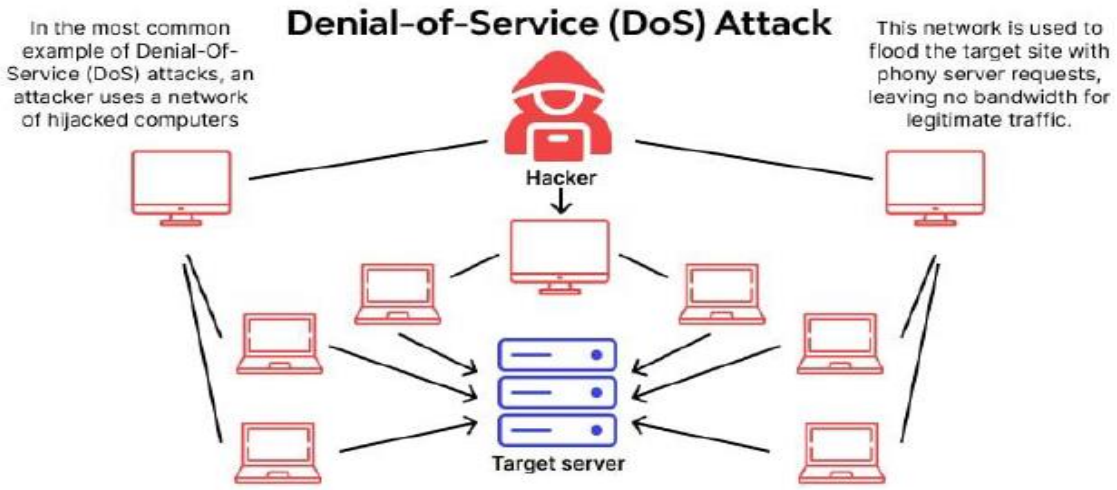
Suppose a security firm is tasked with guarding a rare painting that hangs in a museum. There are several ways that a thief could enter and exit the museum — front doors, back doors, elevators, and windows. A thief could enter the museum in some other way too, perhaps by posing as a member of the museum's staff. All of these methods represent attack vectors, and the security firm may try to eliminate them by placing security guards at all doors, putting locks on windows, and regularly screening museum staff to confirm their identity.

Similarly, digital systems all have areas attackers can use as entry points. Because modern computing systems and application environments are so complex, closing off all attack vectors is typically not possible. But strong security practices and safeguards can eliminate most attack vectors, making it far more difficult for attackers to find and use them.

Denial of Service (DoS)

A denial-of-service (DoS) attack occurs when legitimate users are unable to access information systems, devices, or other network resources due to the actions of a malicious cyber threat actor. Services affected may include email, websites, online accounts (e.g., banking), or other services that rely on the affected computer or network.

A denial-of-service condition is accomplished by flooding the targeted host or network with traffic until the target cannot respond or simply crashes, preventing access for legitimate users. DoS attacks can cost an organization both time and money while their resources and services are inaccessible.

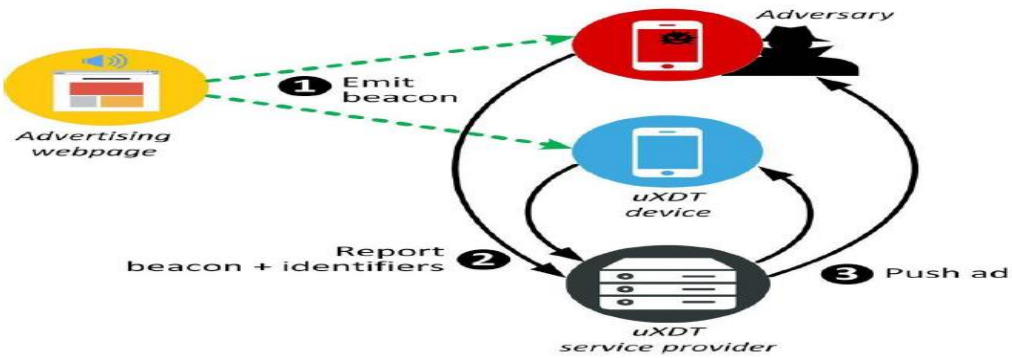


Information Leakage

Information leakage is the sharing of sensitive information with unauthorized parties. The leakage can be either;

1. Accidental, such as an employee sharing confidential information with an external party via email, or malicious, such as the exfiltration of data through phishing scams.
2. Regardless of the intent, however, the information shared is valuable to hackers and can be used to execute attacks on your organization's infrastructure, services or applications.

While information leaks originate from within an organization, data breaches are a result of actions that take place from unauthorized users from outside of the organization. Encryption, implementing security controls and classifying sensitive data are all strategies organizations use to prevent data loss. In addition, many organizations have various data leak prevention strategies and technology in place to defend against data breaches.



Confused Deputy

The "Confused Deputy" problem is a security issue that arises when a program (the "deputy") is tricked into performing actions on behalf of an attacker with more privileges than the attacker should have. This problem often occurs in systems where one program (the deputy) performs tasks on behalf of another program or user, and the deputy is tricked into executing actions it would not normally perform.

Here's a simplified example of how the Confused Deputy problem might manifest:

1. **Scenario**: Imagine a web application that allows users to upload files. The application processes these files on behalf of users, using a server-side script with permissions to read and write files in a specific directory.

2. **Exploitation**: An attacker might upload a file with a name that includes a path traversal attack (e.g., ../sensitive_file.txt). The application's script, operating with the server's higher privileges, might process this file and inadvertently read or write to sensitive files outside the intended directory.
3. **Result**: The attacker successfully accesses or modifies sensitive files by exploiting the higher privileges of the server-side script.

Privilege Escalation

Privilege escalation is a type of security vulnerability where an attacker gains elevated access to resources or permissions beyond what they are normally authorized to have. This can occur through various methods, leading to unauthorized access to sensitive data, system control, or administrative functions. Privilege escalation can be categorized into two main types:

Vertical privilege escalation occurs when a user with lower-level permissions gains access to higher-level privileges or administrative rights. For example:

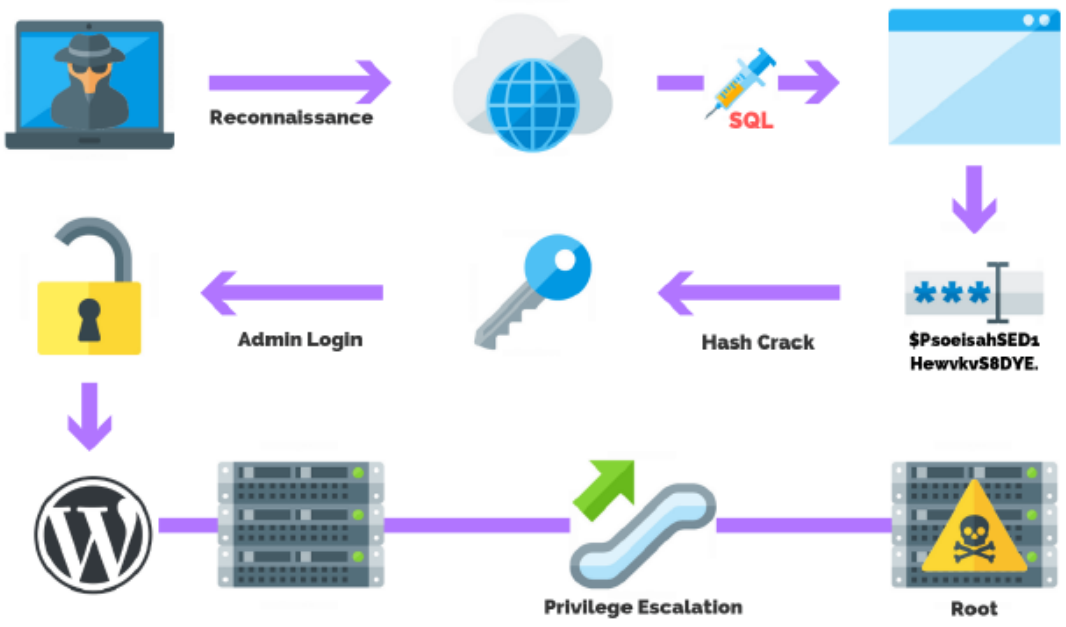
- **Exploiting Vulnerabilities**: Attackers might exploit software vulnerabilities to gain administrative access. For instance, vulnerability in an application might allow a user to execute

commands with root privileges.

- **Weak Authentication**: Weak or misconfigured authentication mechanisms can allow an attacker to impersonate an administrative user and gain elevated access.
- **Misconfigured Permissions**: Incorrectly configured file or directory permissions can enable a user to access or modify files they shouldn't.

Horizontal privilege escalation happens when a user gains access to resources or permissions of another user with the same level of privilege. For example:

- **Session Fixation**: An attacker might hijack a user's session to access resources that the user can access, even though the attacker shouldn't have access to those resources.
- **Insecure APIs**: APIs that do not enforce proper access controls may allow an attacker to perform actions intended for other users with similar permissions.

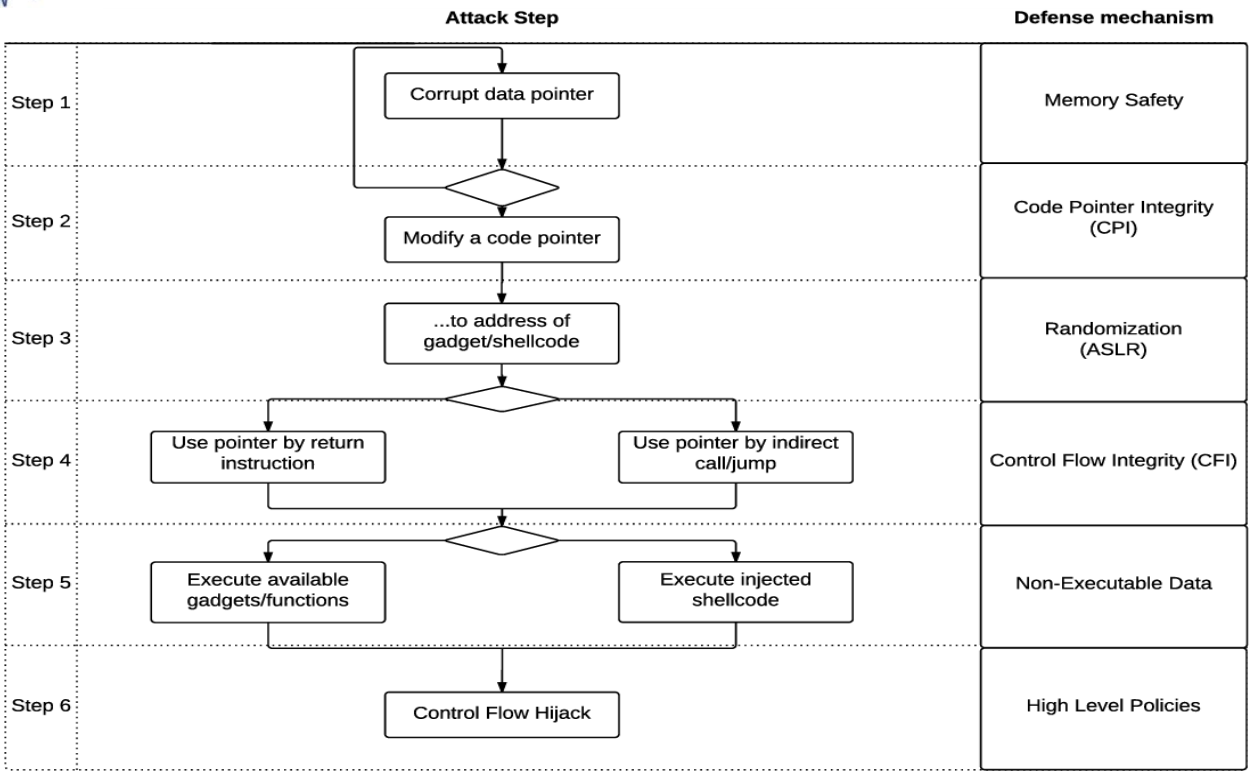


Control-Flow Hijacking

Control-flow hijacking is a type of security vulnerability that allows an attacker to alter the flow of execution in a program. This usually involves redirecting the program's execution to a location of the attacker's choosing. It's often used to exploit software vulnerabilities such as buffer overflows, where an attacker can overwrite parts of memory and redirect the execution flow to execute malicious code.

Here are some common types of control-flow hijacking techniques:

1. **Buffer Overflow Attacks**: By overwriting a buffer, an attacker can alter the return address on the stack, redirecting the program's execution to their own code.
2. **Return-Oriented Programming (ROP)**: An attacker uses existing code snippets (gadgets) that end in return instructions to execute malicious code without injecting new code into the process.
3. **Jump-Oriented Programming (JOP)**: Similar to ROP but uses jump instructions instead of return instructions to build a chain of gadgets for malicious purposes.
4. **Code Injection Attacks**: Injecting malicious code into the process's memory space and redirecting execution to it.
5. **Function Pointer Overwriting**: Manipulating function pointers to redirect the program's execution to malicious code.



Code Injection

Code injection is a type of security vulnerability where an attacker is able to insert or "inject" malicious code into a program or system. This injected code is then executed by the system, leading to unauthorized actions or compromise. Code injection attacks can affect various types of applications and systems, including web applications, databases, and even local software.

Common Types of Code Injection

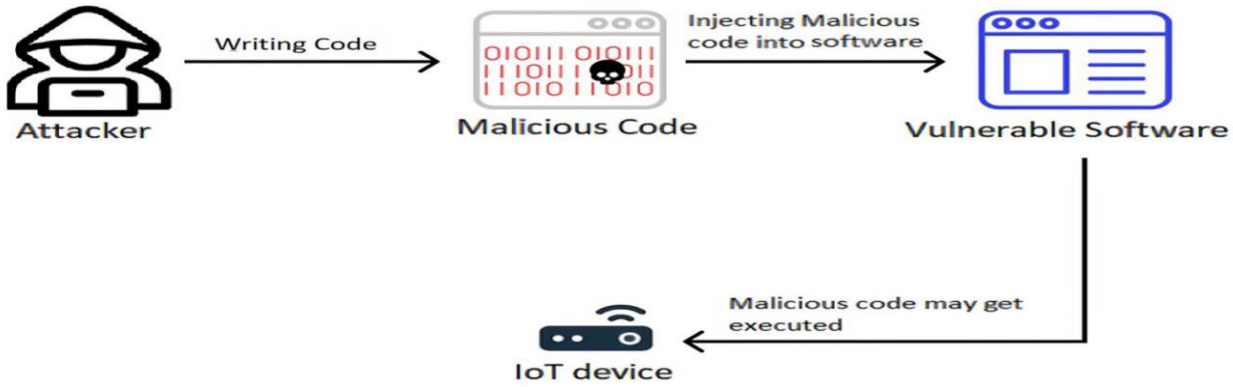
1.**SQL Injection**: An attacker inserts malicious SQL queries into a form input or URL parameter to manipulate or gain unauthorized access to a database.

2.**Cross-Site Scripting (XSS)**: Malicious scripts are injected into web pages viewed by other users, potentially leading to session hijacking, data theft, or other malicious actions.

3.**Command Injection**: Malicious commands are injected into an application's input fields, which are then executed by the server's command-line interface.

4.**Code Injection in Programming Languages**: Attacks that exploit weaknesses in how programming languages handle code execution, such as JavaScript eval(), PHP eval(), or Python exec().

5.**Script Injection**: Similar to XSS but more broadly involves injecting scripts into any environment where the script might be executed.



Code Reuse

A code reuse attack is a type of security exploit where an attacker leverages existing code within a program or system to perform malicious actions. Instead of injecting new code, the attacker reuses legitimate code already present in the program, often to bypass security mechanisms. These attacks are typically sophisticated and require an understanding of the program's structure and available code snippets.

Common Types of Code Reuse Attacks

1.**Return-Oriented Programming (ROP)**: An attacker uses sequences of instructions (gadgets) ending in return instructions to build a chain of operations

that perform malicious actions. This technique is used to exploit vulnerabilities like buffer overflows, where the attacker manipulates the program's control flow to execute their chosen sequence of gadgets.

2.**Jump-Oriented Programming (JOP)**: Similar to ROP, but instead of return instructions, JOP relies on jump instructions to control the execution flow. JOP can be used to bypass security defenses by exploiting existing code that uses jump instructions.

3.**Call-Oriented Programming (COP)**: This technique involves chaining together existing code that contains call instructions. By manipulating these calls, an attacker can execute a series of operations in a controlled manner.

Characteristics of Code Reuse Attacks

• **No Code Injection**: These attacks do not require injecting new code into the memory but instead exploit existing code, which can be harder to detect and mitigate.

• **Bypassing Security Mechanisms**: Code reuse attacks can bypass security measures like Data Execution Prevention (DEP) or Address Space Layout Randomization (ASLR) by avoiding the need to execute injected code.

• **Exploitation of Code Patterns**: Attackers exploit predictable patterns or behaviors in the existing code to achieve their goals.