Muhammad Tayab’ Report

CORVIT SYSTEMS

Email: muhammadt624@gmail.com

Website: CORVIT.COM

Tel: +923246308588

Address: Multan, Pakistan

CORVIT
Reaching end to end

TABLE OF CONTENTS

1. CODE INJECTION

Code injection is a broad term that encompasses various attack techniques where an attacker inserts or injects malicious code into a vulnerable program to alter its execution flow. The injected code can perform actions such as accessing unauthorized data, compromising system integrity, or executing harmful operations.

How It Works?

Input Handling: Code injection exploits flaws in the way a program processes input. When a program does not properly sanitize user input or other external data, it may mistakenly execute the injected code as if it were part of the program's legitimate instructions.

Execution Context: The injected code executes in the context of the vulnerable application, which means it can inherit the application's privileges, making it especially dangerous if the application runs with elevated privileges.

Examples:

1. **SQL Injection**:

Scenario: Consider a web application that allows users to log in by entering their username and password. The application might use a SQL query like this:

```
SELECT * FROM users WHERE username = 'input_username' AND password = 'input_password';
```

Attack: If an attacker enters the following into the username field:

```
' OR '1'='1
```

The query becomes:

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

This always returns true, potentially granting the attacker unauthorized access.

2. **Command Injection**:

Scenario: A web application might allow users to ping an IP address by entering it into a form. The application might execute a shell command like:

```
ping -c 4 user_input
```

Attack: If the user enters `127.0.0.1; rm -rf /`, the command becomes:

```
ping -c 4 127.0.0.1; rm -rf /
```

This could execute a destructive command that deletes files on the server.

Types of Code Injection:

1. **SQL Injection**: Inserting malicious SQL queries to manipulate database operations.
2. **Command Injection**: Executing arbitrary system commands by injecting them into the input processed by a shell or system command interpreter.
3. **LDAP Injection**: Manipulating LDAP queries to access unauthorized information in a directory service.
4. **XML Injection**: Inserting malicious XML content to alter the behavior of an application that processes XML data.

Mitigation:

- **Input Validation**: Strictly validate and sanitize all user inputs to ensure they conform to expected formats.
- **Parameterized Queries**: Use prepared statements and parameterized queries to prevent SQL injection by ensuring that user inputs are treated as data, not executable code.
- **Use of Security Libraries**: Implement security libraries or frameworks that provide built-in protection against common injection attacks.

Real-World Example:

Equifax Data Breach (2017): One of the most notorious examples of a code injection attack is the Equifax data breach, where attackers exploited a vulnerability in a web application framework (Apache Struts) to inject malicious code, resulting in the theft of sensitive information from millions of users.

2. HOOKING TECHNIQUES

Hooking refers to a method used by developers or attackers to intercept and possibly alter function calls, messages, or events that are passed within an operating system or between software components. This allows the interceptor to modify the behavior of the software dynamically.

How It Works?

Interception Point: A hook is typically placed at a strategic location where function calls or messages are passed. This could be at the API level, within system libraries, or even within the application code itself.
Modification: Once hooked, the interceptor can modify the data, change the flow of execution, or inject additional behavior. The original function call can either be passed through unaltered, modified, or completely replaced with custom code.

Examples:

1. API Hooking:

Scenario: A developer might want to monitor all file access operations performed by an application. By hooking the `CreateFile` API in Windows, the developer can intercept calls to this function and log details about each file opened by the application.
Attack: A malicious actor could hook the same API to intercept and modify file contents as they are read or written, potentially inserting malware or stealing sensitive data.

2. Inline Hooking:

Scenario: An antivirus program may use inline hooking to monitor calls to dangerous functions like `VirtualAlloc` (used to allocate memory) to detect if a process is attempting to allocate memory for malicious code execution.
Attack: An attacker might use inline hooking to intercept and modify the behavior of a security function, bypassing its checks and allowing the malicious code to execute.

3. Import Address Table (IAT) Hooking:

Scenario: A developer might use IAT hooking to modify the behavior of a specific library function without altering the application's binary code. For example, redirecting a network function to a custom implementation that logs network traffic for debugging purposes.
Attack: Attackers can use IAT hooking to replace critical system functions with malicious versions, leading to a wide range of potential exploits, such as redirecting network traffic to an attacker's server.

Tools:

i. **Detours by Microsoft**: A popular library for intercepting and modifying Windows API calls, often used for debugging or monitoring purposes.
ii. **Frida**: A powerful dynamic instrumentation toolkit that allows developers and researchers to hook into and modify the behavior of running applications on Windows, macOS, Linux, iOS, and Android.

Mitigation:

- **Code Integrity Checks**: Implement integrity checks to ensure that the code or functions have not been tampered with. This can involve verifying the hash or signature of critical binaries or libraries before they are loaded.

- **Anti-Hooking Techniques**: Use obfuscation or anti-tampering techniques to make hooking more difficult. This could include encrypting function pointers or employing complex control flow to prevent easy interception.

Real-World Example:

Stuxnet Worm (2010): The Stuxnet worm used advanced hooking techniques to evade detection by security software. It hooked system functions to hide its presence and alter the behavior of industrial control systems, leading to the physical destruction of centrifuges in an Iranian nuclear facility.

3. DLL INJECTION

DLL Injection is a technique where an attacker injects a malicious Dynamic Link Library (DLL) into the memory space of a running process, forcing the target process to execute the code contained within the DLL. This technique is often used to execute arbitrary code in the context of another process, allowing the attacker to manipulate the process's behavior.

How It Works?

I. **Target Process**: The attacker identifies a running process that they wish to target, typically one with higher privileges or access to sensitive resources.

II. **Injecting the DLL**: The attacker injects the DLL into the target process using various methods, such as creating a remote thread in the target process that loads the DLL, or by exploiting a vulnerability in the process to force it to load the DLL.

III. **Execution Context**: Once injected, the DLL executes in the context of the target process, allowing the attacker to perform actions such as keylogging, data exfiltration, or code modification.

Examples:

I. Remote Thread Injection:

Scenario: An attacker uses the **`CreateRemoteThread`** and **`LoadLibrary`** APIs in Windows to inject a DLL into a running process like `explorer.exe`. The DLL could contain code to monitor user activity or steal credentials.

Attack: This method is commonly used by malware to inject malicious code into system processes, making it harder to detect and remove.

II. **Reflective DLL Injection**:

Scenario: Reflective DLL injection is a stealthier form of injection where the DLL is loaded directly from memory rather than being written to disk. This technique avoids leaving traces on the file system, making detection more challenging.
Attack: Advanced persistent threats (APTs) often use reflective DLL injection to maintain a covert presence on infected systems, avoiding detection by traditional antivirus software.

III. **Manual Mapping**:

Scenario: Instead of using the Windows loader, an attacker manually maps a DLL into a process's memory space. This bypasses many security mechanisms that rely on monitoring standard DLL loading procedures.
Attack: This method is used in sophisticated malware to avoid detection by security software that monitors for suspicious DLL loads.

Mitigation:

- **Code Signing**: Ensure that only digitally signed and trusted DLLs can be loaded into critical processes. This reduces the risk of unauthorized DLL injection.
- **User Account Control (UAC)**: Limit the privileges of processes and ensure that they run with the least privileges necessary. This can prevent low-privilege processes from injecting code into higher-privilege processes.
- **Behavioral Monitoring**: Use security software that monitors for suspicious behavior indicative of DLL injection, such as the creation of remote threads in unrelated processes.

Real-World Example:

Zeus Trojan (2007-2010): The Zeus Trojan, a notorious banking malware, used DLL injection to inject itself into web browsers like Internet Explorer and Firefox. Once injected, it would monitor and capture sensitive information such as login credentials and banking details, sending them to the attacker’s server.

4. API HOOKING

API Hooking is a technique where an attacker or developer intercepts calls to an Application Programming Interface (API) to monitor, modify, or redirect them. This can be used for both legitimate purposes (such as debugging and enhancing software functionality) and malicious purposes (such as intercepting sensitive data or altering the behavior of security functions).

How It Works:

i. **Hook Placement**: To hook an API, the interceptor typically modifies the function pointers that are used to call the API. This can be done by directly altering the memory where the function is loaded, or by modifying the Import Address Table (IAT) which the application uses to resolve API calls.

ii. **Execution Flow**: Once the hook is in place, the execution flow is diverted from the original API to the hook function. The hook function can then execute its code before optionally calling the original API or entirely replacing its functionality.

Examples:

Example 1: Debugging:

Scenario: A developer might use API hooking to intercept network-related API calls (like **`send`** or **`recv`** in socket programming) to log data being sent and received over the network. This is useful for debugging or analyzing network traffic.

Implementation: The developer hooks the `send` function, adds logging code, and then calls the original `send` function to allow the data to be transmitted normally.

Example 2: Malware:

Scenario: A piece of malware could hook the **`ReadFile`** API to intercept file read operations, checking if the file being read contains sensitive information (e.g., passwords or cryptographic keys) and exfiltrating this data to an attacker-controlled server.

Implementation: The malware hooks `ReadFile`, adds code to scan the file contents for sensitive data, and either allows or modifies the data being read based on what it finds.

Techniques:

i. **Inline Hooking**: The attacker modifies the machine code of the API function directly, usually by placing a jump instruction at the beginning of the function that redirects execution to the hook function.

ii. **Detour Hooking**: Similar to inline hooking, but more structured. This technique involves redirecting the API calls to a different code path (the detour), where the interceptor's code can execute before or instead of the original API

iii. **IAT Hooking**: The Import Address Table (IAT) of a module is modified to point to a different function. This approach is less invasive because it doesn't require changing the code of the API itself. When the application resolves the API function, it will call the hook function instead of the original.

iv. **User-Mode Hooking**: Involves hooking APIs in user-mode applications. This is commonly done to modify the behavior of applications running in user space without requiring kernel-level privileges.

v. **Kernel-Mode Hooking**: Involves hooking APIs at the kernel level. This is more powerful and can be more difficult to detect or prevent but typically requires higher privileges.

Real-World Applications:

- **Security Software**: Antivirus and intrusion detection systems often use API hooking to monitor suspicious activity within a system, such as unauthorized file access or network connections.
- **Malware**: Many forms of malware, like keyloggers or data exfiltration tools, use API hooking to intercept sensitive data, such as keystrokes or credit card information, before it reaches its intended destination.

- **Game Hacking**: In the gaming world, cheat developers often use API hooking to modify game behavior, such as creating aimbots or wallhacks, by intercepting and altering the game's API calls.

Mitigation:

Code Integrity: Use code signing and integrity checks to ensure that critical system files and libraries haven't been tampered with. This makes it harder for attackers to introduce hooks.
Behavior Monitoring: Deploy security solutions that monitor for the unusual behavior indicative of API hooking, such as unexpected changes in application behavior or modifications to system libraries.
Address Space Layout Randomization (ASLR): Implement ASLR to make it more difficult for attackers to predict the memory locations of APIs, thus complicating the hooking process.

Example in Practice:

GameGuard: A well-known anti-cheat program, GameGuard, uses API hooking to monitor and block suspicious calls in games, preventing cheating by intercepting unauthorized modifications to the game's processes.

5. MALWARE ANALYSIS

Malware analysis is the process of dissecting and understanding malware to determine its functionality, origin, and potential impact. The goal is to extract as much information as possible to mitigate the threat, develop detection mechanisms, and understand the attacker's methods. Malware analysis can be categorized into static and dynamic analysis, each with different techniques and tools.

Types of Malware Analysis:

1. Static Analysis

Static analysis involves examining the malware file without executing it. This type of analysis is useful for identifying the basic characteristics of the malware, such as file type, strings, and embedded resources.

Steps in Static Analysis:

I. **File Identification**: Determine the file type (e.g., PE file for Windows, ELF for Linux).
II. **Hashing**: Compute file hashes (MD5, SHA-1, SHA-256) to compare with known malware databases.
III. **String Analysis**: Extract and analyze readable strings from the binary to find URLs, IP addresses, or commands used by the malware.
IV. **Disassembly**: Use a disassembler to convert the binary into assembly code, allowing you to study its instructions.

Tools for Static Analysis:

- **PEiD**: Identifies packers, cryptors, and compilers used in PE files.
- **Binwalk**: A tool for analyzing and extracting files and data from binary images.
- **Ghidra**: A software reverse engineering suite developed by the NSA that includes a disassembler and decompiler.
- **IDA Pro**: An interactive disassembler widely used for reverse engineering malware.

Example for Static Analysis:

Suppose you have a suspicious file `malware.exe`. Using `strings`, you extract readable text and find URLs pointing to command-and-control (C2) servers. By analyzing these strings, you can block these URLs at the network level to prevent further communication.

2. Dynamic Analysis

Dynamic analysis involves executing the malware in a controlled environment (sandbox) to observe its behavior. This approach helps in understanding the malware's runtime behavior, network activity, and interactions with the system.

Steps in Dynamic Analysis:

i. **Setting up the Environment**: Use a virtual machine or sandbox environment that mimics the target operating system.
ii. **Behavior Monitoring**: Observe file system changes, registry modifications, network traffic, and process creation.
iii. **Network Analysis**: Capture and analyze network traffic using tools like Wireshark to understand communication with external servers.

Tools for Dynamic Analysis:

- **Cuckoo Sandbox**: An automated malware analysis system that provides detailed reports on the malware's behavior.
- **Process Monitor (Procmon)**: Monitors file system, registry, process, and thread activity in real-time.
- **Wireshark**: A network protocol analyzer used to capture and analyze network traffic.
- **Regshot**: A registry comparison tool that allows you to track registry changes made by malware.

Example for Dynamic Analysis:

When executing `malware.exe` in a sandbox environment, you observe that it tries to connect to an external IP address and modifies several registry keys. This behavior suggests that the malware is trying to establish persistence and communicate with a remote C2 server.

Malware Analysis Tools

1. Ghidra:

Ghidra is a free and open-source reverse engineering tool developed by the NSA. It provides features for analyzing and understanding binary files.

Key Features:

- **Disassembly and Decompilation**: Converts binary code into human-readable assembly and decompiled code.
- **Scripting**: Supports scripting in Java and Python to automate analysis tasks.
- **Cross-Platform**: Works on Windows, macOS, and Linux.

Use Case:

Example: You have a piece of malware targeting Windows systems. Using Ghidra, you disassemble the binary and find a function responsible for decrypting C2 server URLs. By analyzing this function, you extract the URLs without executing the malware.

2. Cuckoo Sandbox

Cuckoo Sandbox is an open-source automated malware analysis system that runs the malware in an isolated environment and provides detailed reports on its behavior.

Key Features:

- **Automated Analysis**: Automatically generates detailed reports on file system changes, network activity, and API calls.
- **Modular**: Supports custom modules to extend its functionality.
- **Web Interface**: Provides a web interface for easy submission and analysis of samples.

Use Case:

Example: After submitting a suspicious PDF file to Cuckoo Sandbox, the analysis reveals that the file contains an embedded exploit that downloads a secondary payload from a remote server.

3. IDA Pro

IDA Pro is a powerful disassembler and debugger widely used for reverse engineering malware. It provides a graphical interface for analyzing binary code and understanding the flow of execution.

Key Features:

- **Interactive Disassembly**: Allows users to interact with the disassembled code, annotate it, and explore different execution paths.
- **Debugging**: Supports debugging binaries to step through the code and observe its behavior.
- **Extensibility**: Supports plugins and scripting for automating repetitive tasks.

Use Case:

Example: You encounter an obfuscated malware sample. Using IDA Pro, you analyze the control flow graph to identify the decryption routine, enabling you to recover the original malicious code.

4. Wireshark

Wireshark is a network protocol analyzer that captures and analyzes network traffic in real-time. It is widely used in malware analysis to observe the network behavior of malicious samples.

Key Features:

- **Packet Capture**: Captures all network traffic, including TCP/IP, DNS, HTTP, and more.
- **Filtering**: Supports advanced filtering options to focus on specific traffic of interest.
- **Protocol Analysis**: Decodes various network protocols to provide insights into the communication between the malware and remote servers.

Use Case:

Example: You analyze network traffic from a suspected infected machine. Using Wireshark, you identify an HTTP request to a suspicious domain, indicating that the malware is exfiltrating data to a remote server.

---THE END ---