Introduction to Command Injection

Concepts and Practices

Team 11

Jeongmin Lee, Jonghyeok Lee, Chanhang Lee, Nurnazihah Intan, Pedro Vidal Villalba

Outline

1. What is a Command Injection attack?
2. Command Injection vs. SQL Injection.
3. Types of Command Injection.
4. The impact and severity of an attack.
5. Prevention and defense strategies.
6. Practical examples from PortSwigger.
7. Conclusion
8. QnA

1. What is Command Injection?

A security vulnerability allowing an attacker to execute arbitrary system commands on a host operating system.

Occurs when a web application fails to properly validate or filter user-provided input.

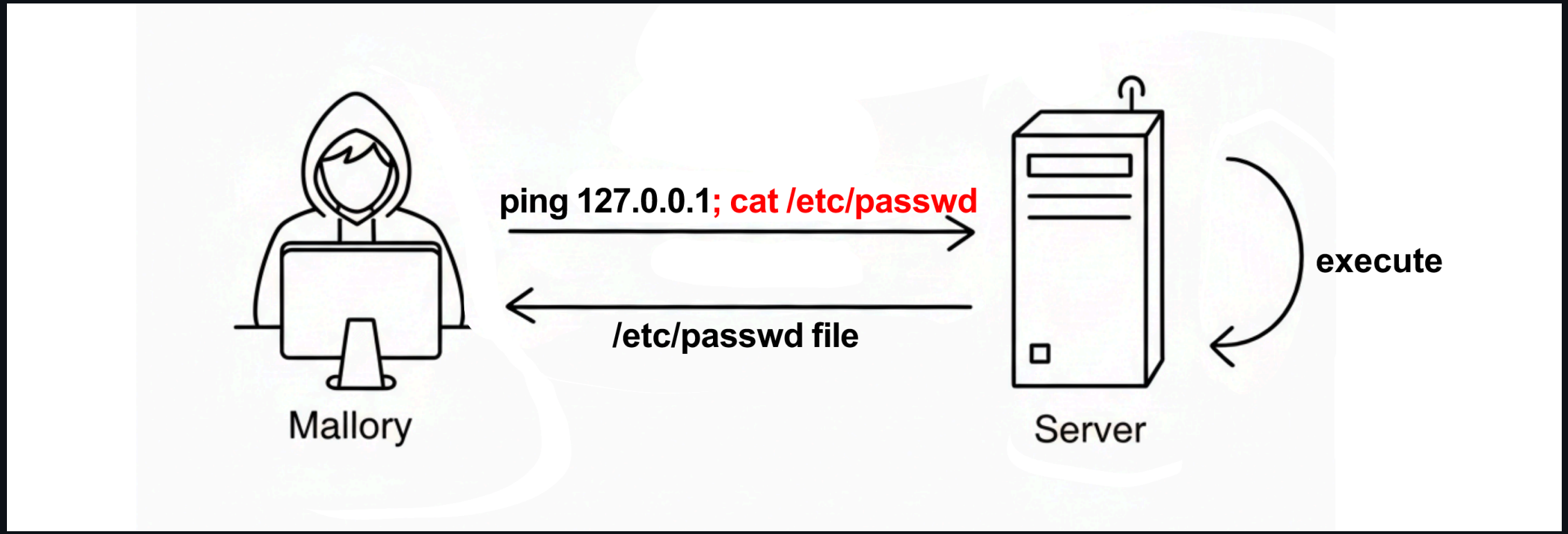
Attackers can perform unintended actions or steal sensitive information.

Worst Case: Remote Code Execution (RCE), leading to full server control.

A Simple Example

Normal Input: ping 127.0.0.1

Malicious Input: ping 127.0.0.1; cat /etc/passwd



Vulnerable Code Example

```
user_input = request.args.get("ip")
cmd = "ping " + user_input
os.system(cmd) // Executes the entire string via shell
```

2. Command Injection vs. SQL Injection

- Target

Host Operating System vs. Application Database

- Mechanism

Executes OS commands (ping, cat) vs. Manipulates database queries (SELECT, DROP)

3. Types of Command Injection Attacks

- In-band Command Injection

The output of the injected command is returned directly in the HTTP response. The attacker sees the results immediately.

- Blind Command Injection

The command executes, but the output is not returned. Attackers use indirect methods (e.g., time delays) to confirm execution.

- Out-of-Band (OOB) Command Injection

The command output is sent to a separate, attacker-controlled channel (e.g., a DNS or HTTP callback).

4. The Impact and Severity

A successful attack can have severe consequences.

- Sensitive Information Disclosure (/etc/passwd, config files).
- Remote Code Execution (RCE) (install malware, create backdoors).
- Denial of Service (DoS) (crash the server with resource-heavy commands).
- Data Compromise (delete or modify critical files).
- Stepping Stone for further network attacks.

5. Prevention and defense strategies.

"Avoid directly invoking operating system shell commands whenever possible"

Instead, use safe, language-specific APIs or functions

5. Prevention and defense strategies.

Input Validation and Sanitization

Strictly validate all user input against a whitelist of allowed characters or patterns. For example, an IPv4 address should only contain digits and dots.

Use Safe APIs and Library Functions

Instead of building command strings, use dedicated functions.

Vulnerable: os.system("mkdir " + user_input)

Safe: os.mkdir(user_input)

5. Prevention and defense strategies.

Principle of Least Privilege

Run applications with the minimum privileges necessary. Avoid root.

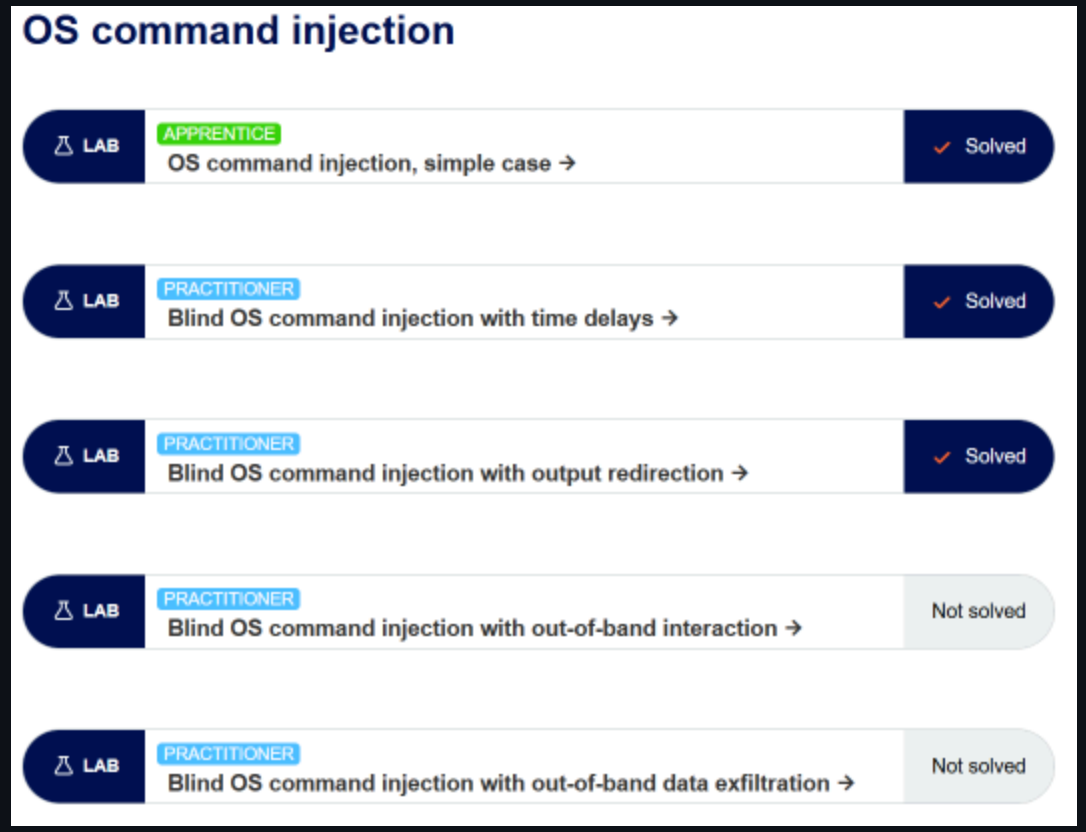
Use Security Options and Frameworks

In Python, use subprocess with the shell=False option.

Output Encoding

Properly encode any output returned to the user to prevent other vulnerabilities like Cross-Site Scripting (XSS).

6. Practical examples from PortSwigger.



*We need a **Burp Suite** to solve these Lab!

*And maybe additional Linux commands too...

Lab 1: Simple Case

Objective: Execute whoami and view its output.

Vulnerability: A stock checker includes user input directly in a shell command.

Attack:

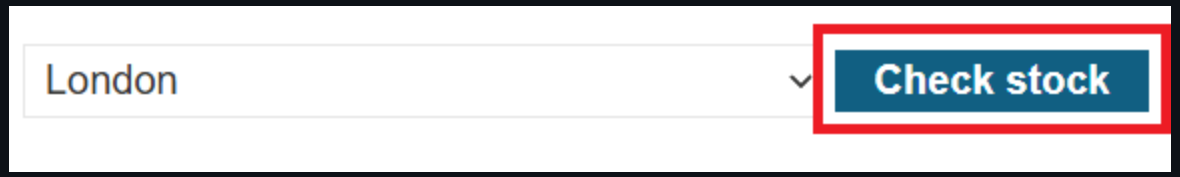
Intercept the POST request using Burp Suite.

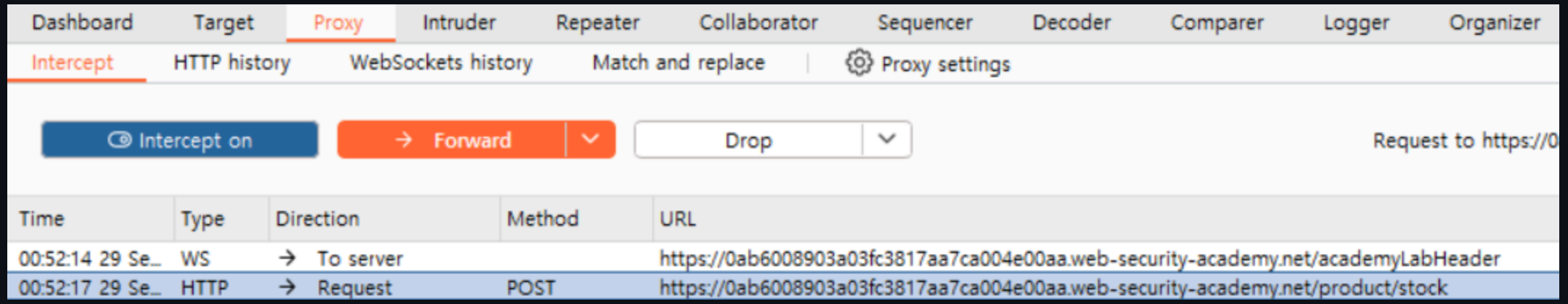
Modify the storeId parameter.

Payload: productId=1&storeId=1 | whoami

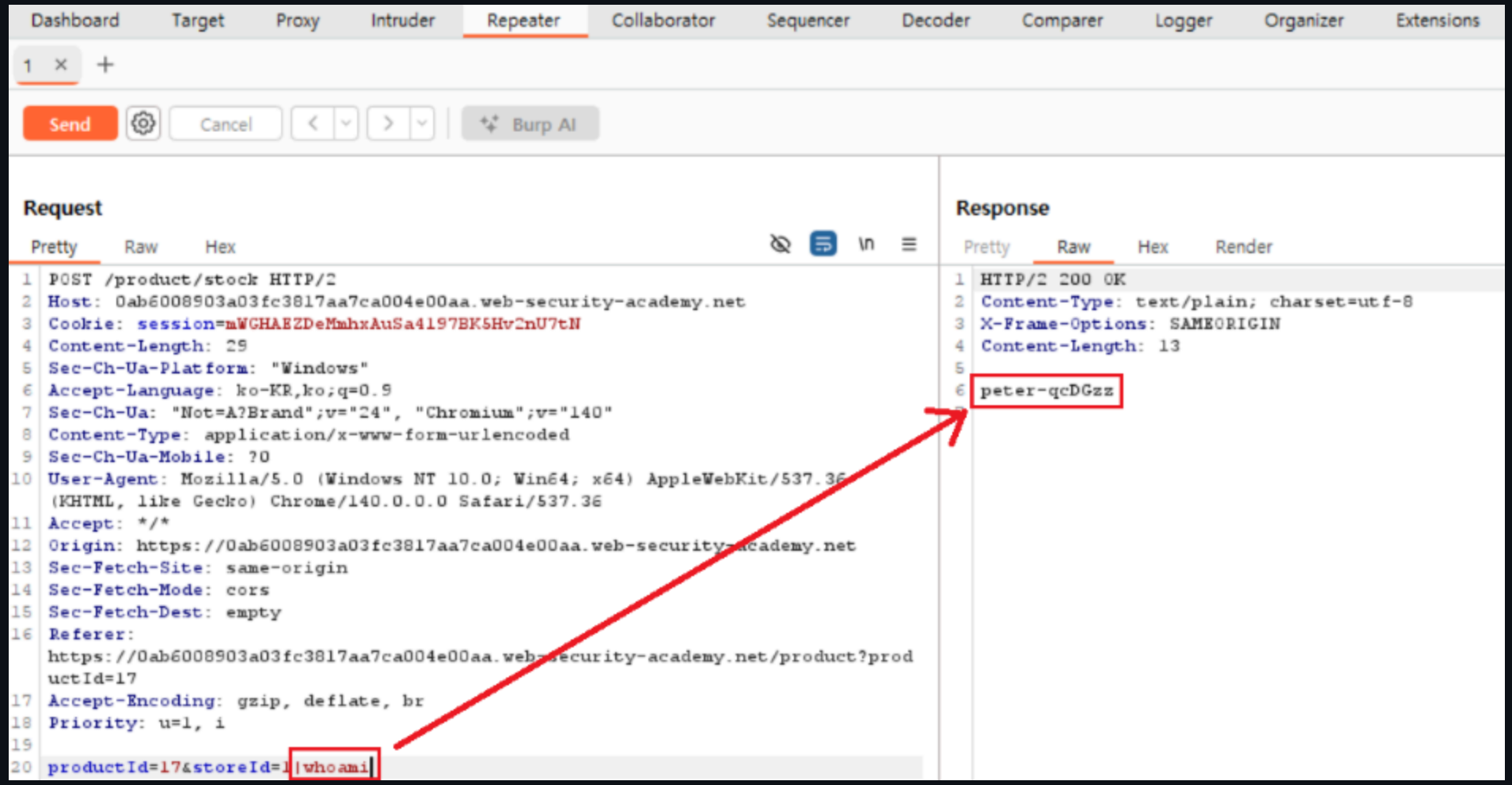
Result: The username (peter-PUEgeP) is returned directly in the HTTP response.

Lab 1: Simple Case





Lab 1: Simple Case



Lab 2: Blind Injection with Time Delays

Objective: Cause a 10-second delay to confirm a blind vulnerability.

Vulnerability: A feedback form executes a command but returns no output.

Attack:

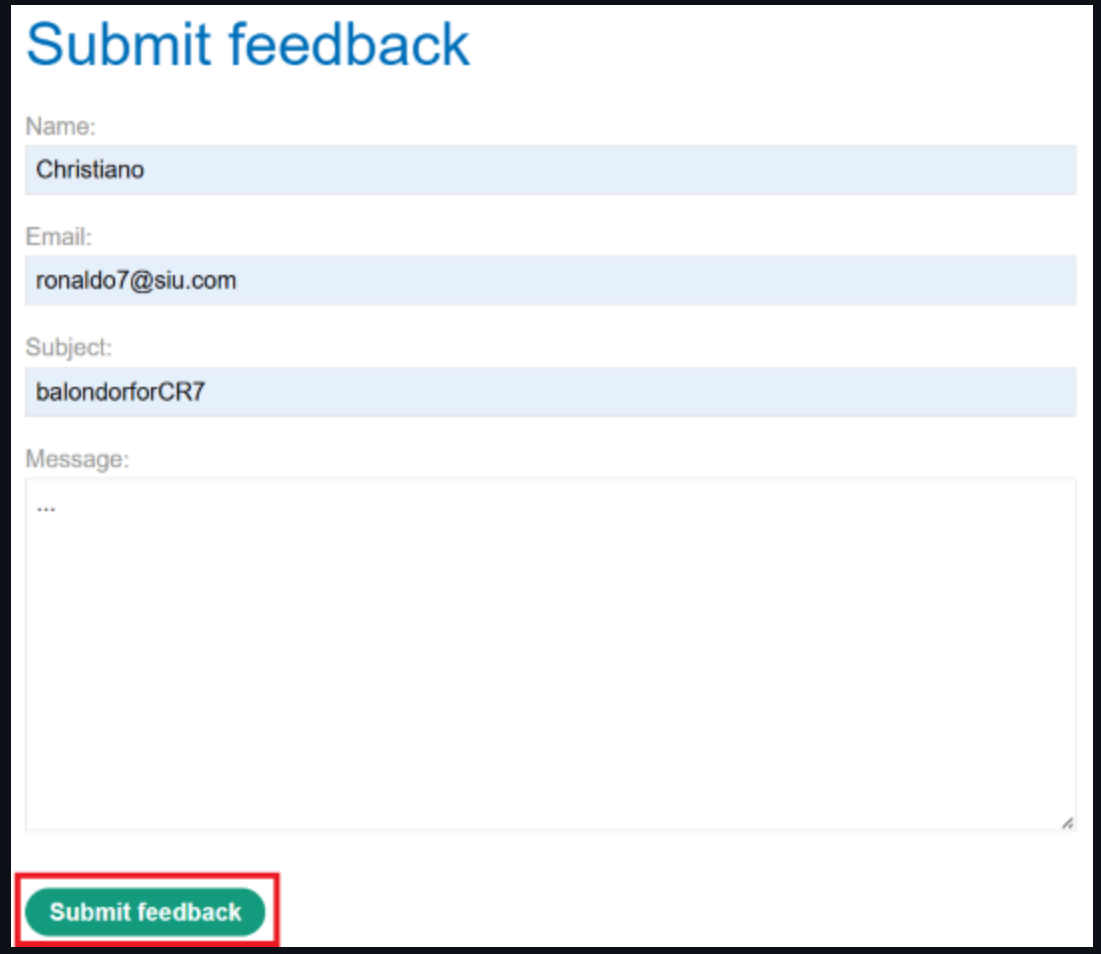
Intercept the feedback submission POST request.

Inject a time-based command into the email parameter.

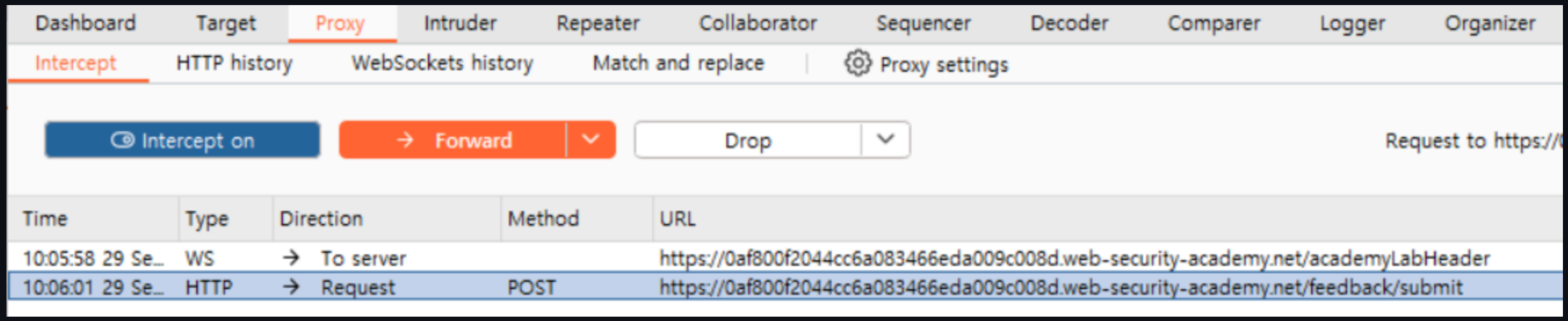
Payload: email=x || ping -c 10 127.0.0.1 ||

Result: The server's response is delayed by ~10 seconds, confirming execution.

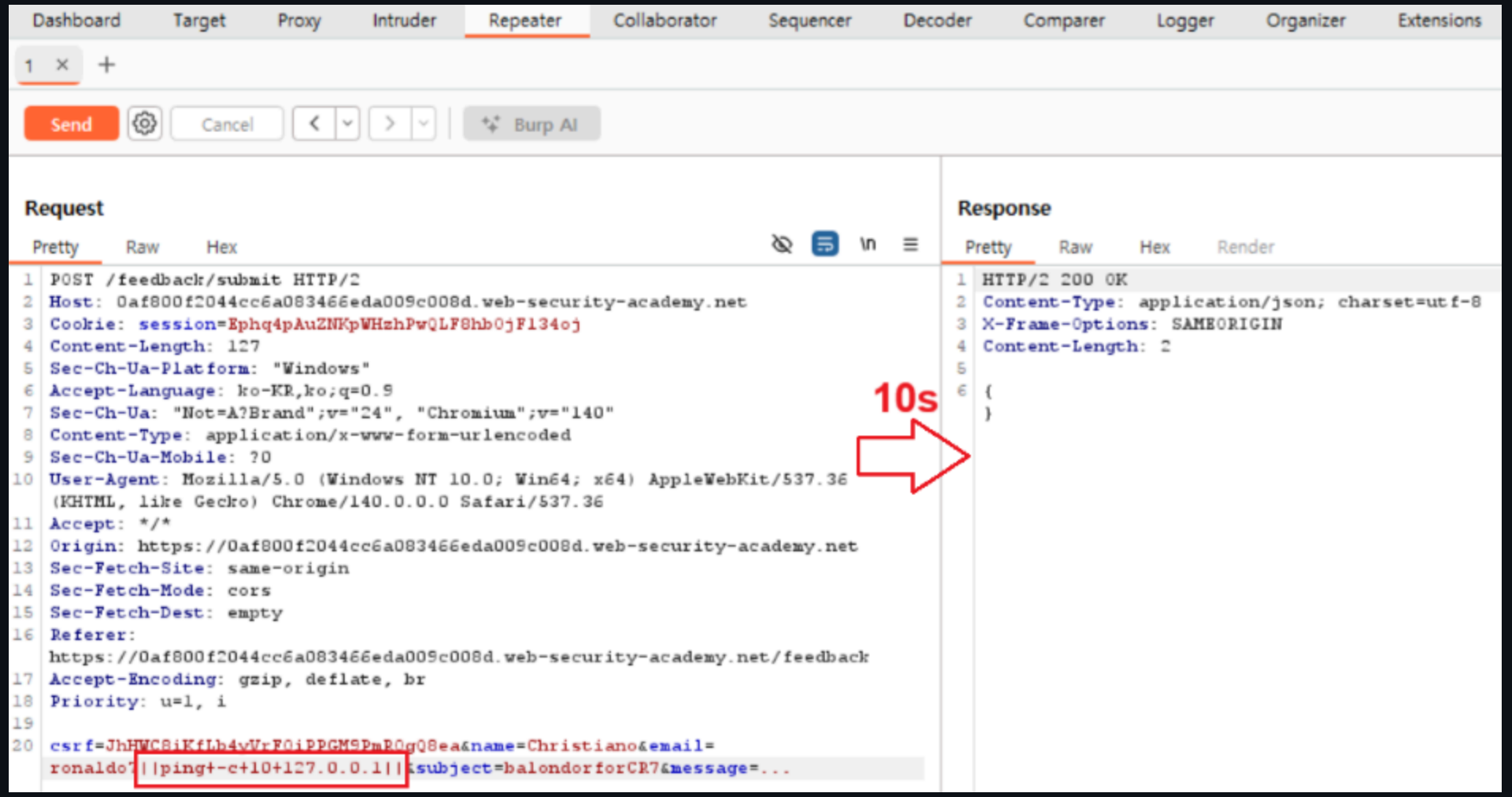
Lab 2: Blind Injection with Time Delays



Lab 2: Blind Injection with Time Delays



Lab 2: Blind Injection with Time Delays



Lab 3: Blind Injection with Output Redirection

Objective: Execute whoami, redirect its output to a file, and retrieve it.

Vulnerability: A blind injection vulnerability plus a publicly writable web directory (/var/www/images/).

Attack:

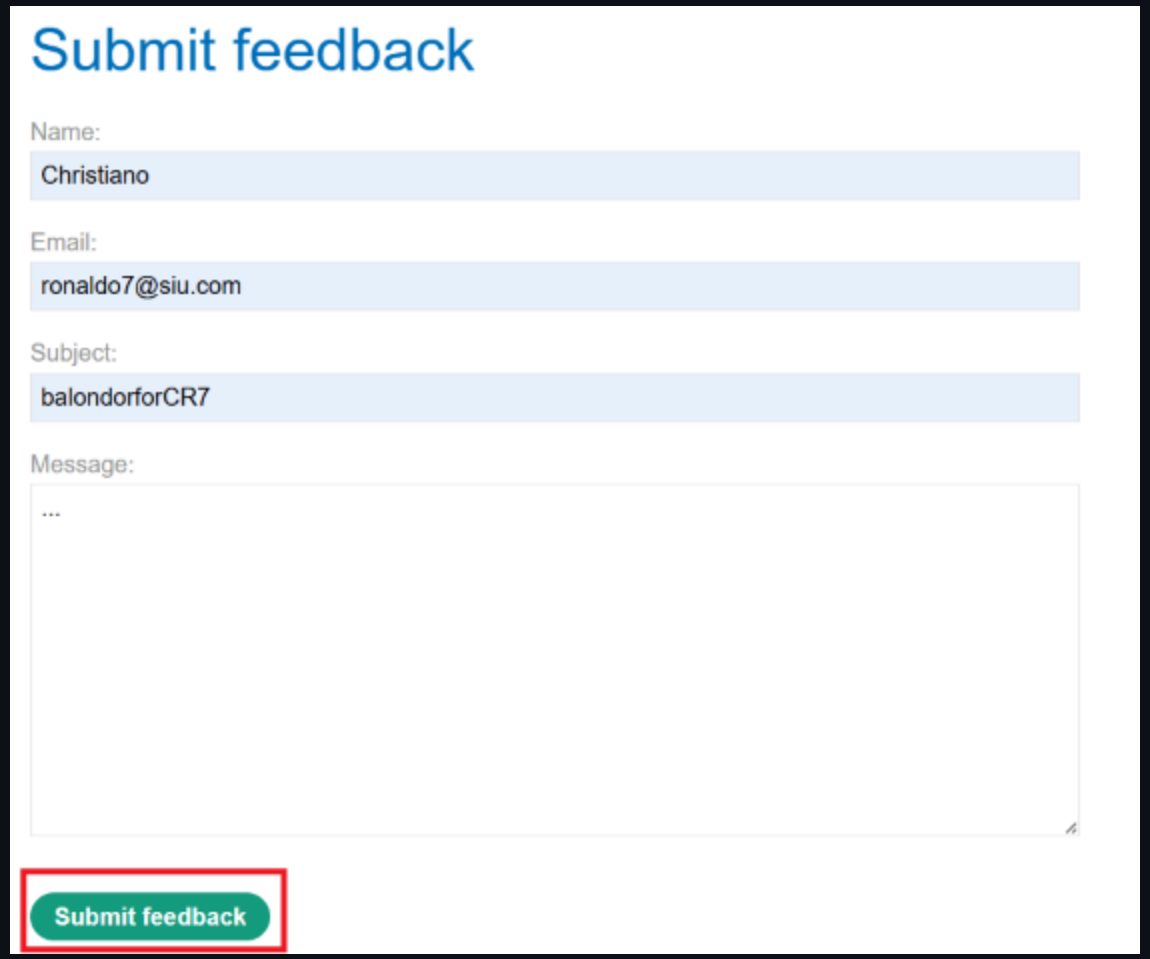
Inject a command into the email field that redirects its output.

Payload: email=x || whoami > /var/www/images/output.txt ||

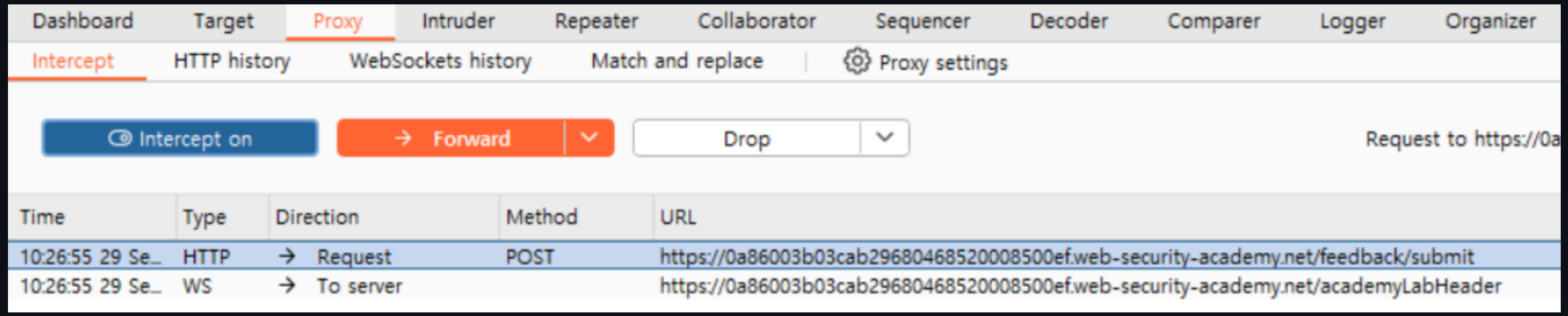
Access the file via its URL: /image?filename=output.txt.

Result: The contents of output.txt are displayed, revealing the whoami result.

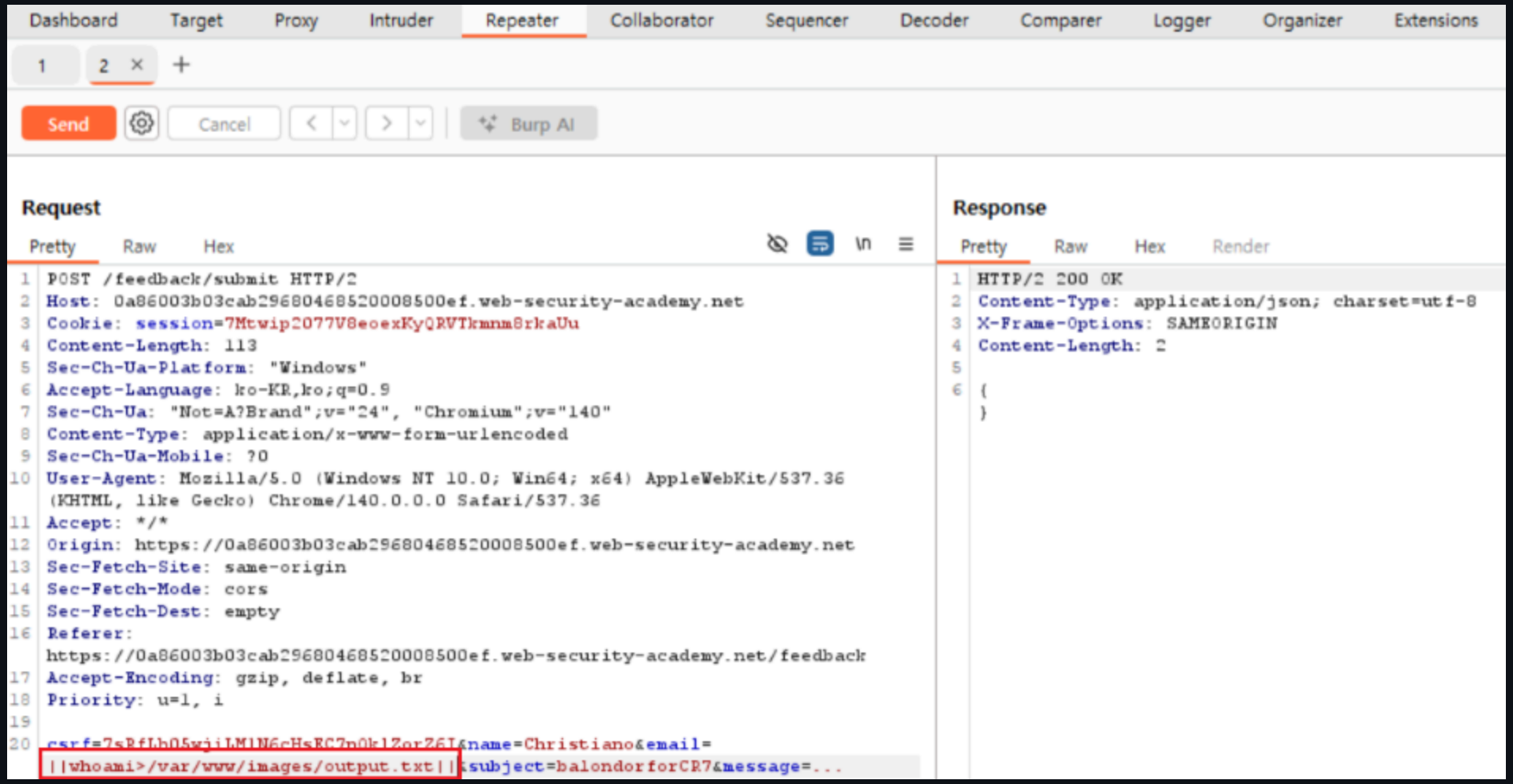
Lab 3: Blind Injection with Output Redirection



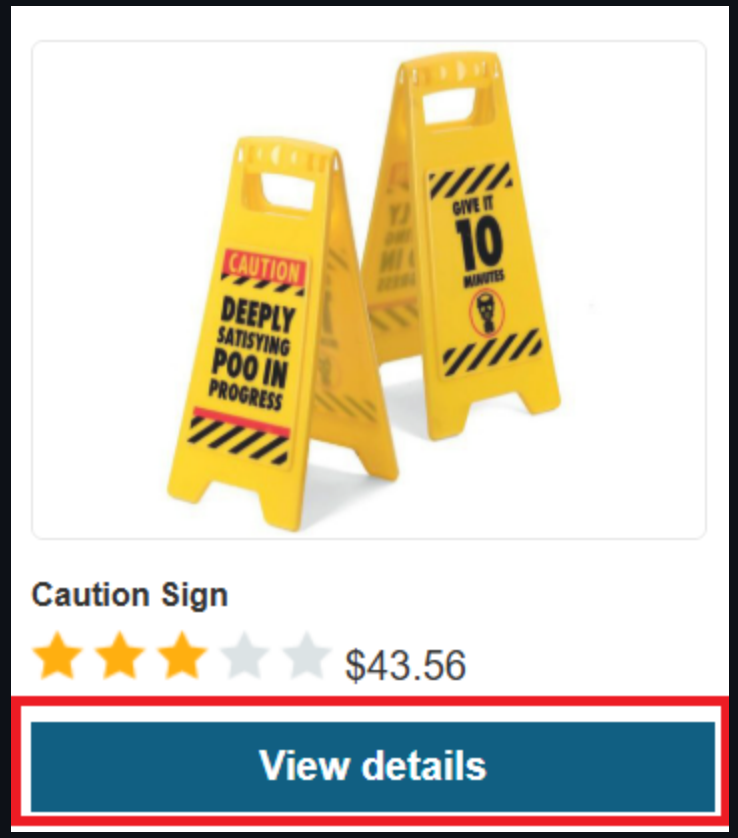
Lab 3: Blind Injection with Output Redirection



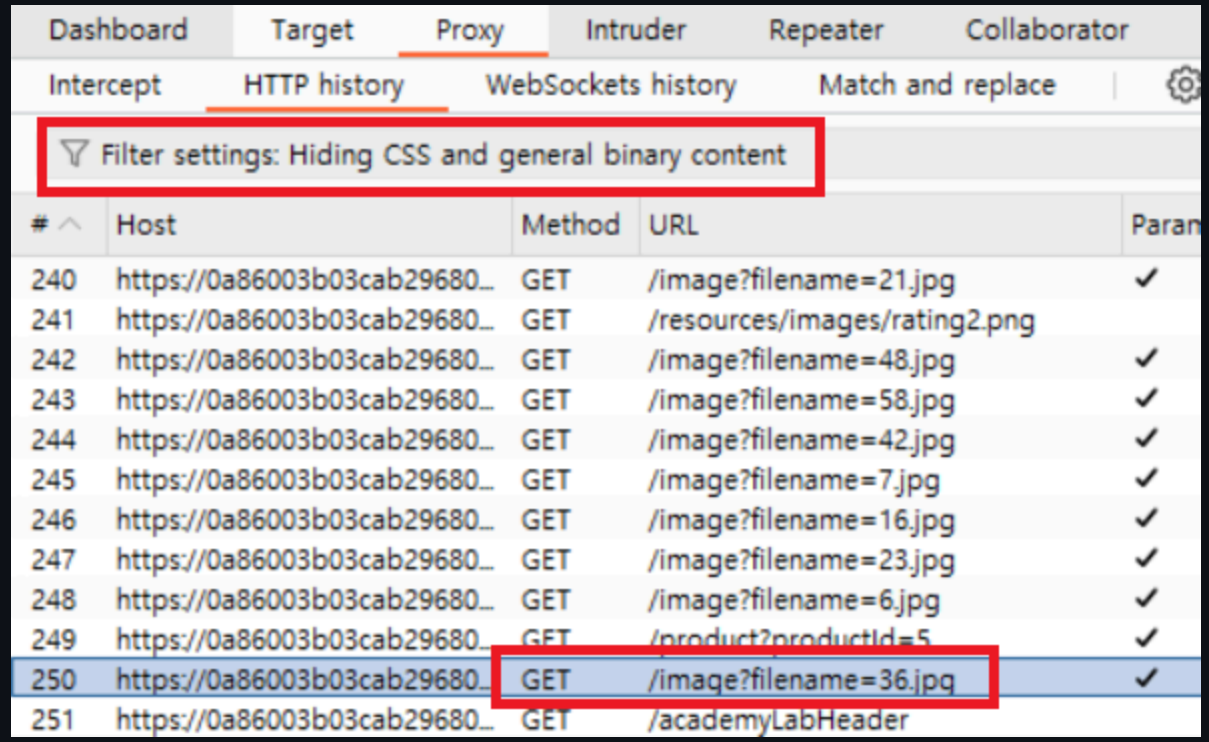
Lab 3: Blind Injection with Output Redirection



Lab 3: Blind Injection with Output Redirection

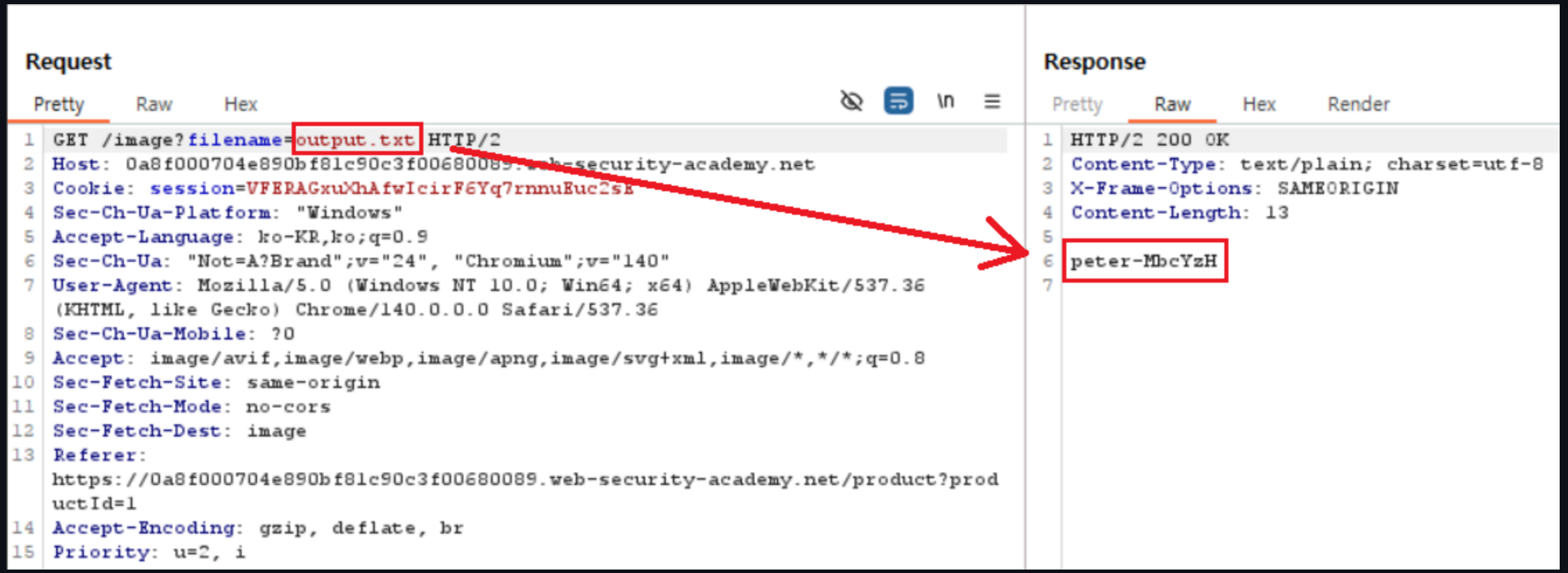


Lab 3: Blind Injection with Output Redirection



- Check the filter settings if you cannot found the packet

Lab 3: Blind Injection with Output Redirection



7. Conclusion

Command injection is a critical vulnerability that allows direct execution of OS-level commands.

"Never trust user input and avoid calling the system shell directly."

8. Q&A

Any questions?