**Template walkthrough**

# **Index**

[Index 1](#_Toc204706054)

[List of pictures 1](#_Toc204706055)

[Disclaimer 2](#_Toc204706056)

[Reconnaissance 2](#_Toc204706057)

[Initial foothold 2](#_Toc204706058)

[User flag 3](#_Toc204706059)

[Privilege escalation 8](#_Toc204706060)

[Personal comments 12](#_Toc204706061)

[References 13](#_Toc204706062)

# **List of pictures**

[Figure 1 - nMap scan results 2](#_Toc204706029)

[Figure 2 - SMB shares found 2](#_Toc204706030)

[Figure 3 - ffuf scan results 3](#_Toc204706031)

[Figure 4 - SQL Injection found via SQLMap 3](#_Toc204706032)

[Figure 5 - DB user privileges 4](#_Toc204706033)

[Figure 6 - Reading of 000-default.conf file 4](#_Toc204706034)

[Figure 7 - writer.esgi file 5](#_Toc204706035)

[Figure 8 - \_\_init\_\_.py file 5](#_Toc204706036)

[Figure 9 - Malicious fake image file 6](#_Toc204706037)

[Figure 10 - Reverse shell invoking 6](#_Toc204706038)

[Figure 11 - Database credentials found 7](#_Toc204706039)

[Figure 12 - Kyle credentials 7](#_Toc204706040)

[Figure 13 - Kyle credentials cracked 7](#_Toc204706041)

[Figure 14 - User flag 8](#_Toc204706042)

[Figure 15 - LinPEAS output 8](#_Toc204706043)

[Figure 16 - Postfix version 9](#_Toc204706044)

[Figure 17 - Disclaimer file permissions 9](#_Toc204706045)

[Figure 18 - User info 9](#_Toc204706046)

[Figure 19 - First shell as john user 10](#_Toc204706047)

[Figure 20 - John SSH key 10](#_Toc204706048)

[Figure 21 - Login as john via SSH 11](#_Toc204706049)

[Figure 22 - Folder permission 11](#_Toc204706050)

[Figure 23 - APT task files 11](#_Toc204706051)

[Figure 24 - Cronjob scheduled 12](#_Toc204706052)

[Figure 25 - root shell and root flag 12](#_Toc204706053)

# **Disclaimer**

I do this box to learn things and challenge myself. I’m not a kind of penetration tester guru who always knows where to look for the right answer. Use it as a guide or support. Remember that it is always better to try it by yourself. All data and information provided on my walkthrough are for informational and educational purpose only. The tutorial and demo provided here is only for those who are willing and curious to know and learn about Ethical Hacking, Security and Penetration Testing.

Just to say: I am not an English native person, so sorry if I did some grammatical and syntax mistakes.

# **Reconnaissance**

The results of an initial nMap scan are the following:

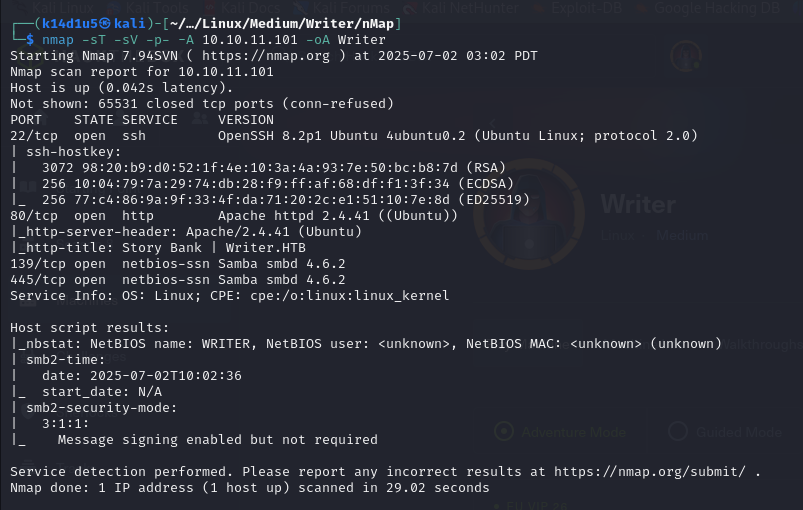


Figure 1 - nMap scan results

Open ports are 22, 80, 139 and 445. Therefore, enabled services are SSH (22) and SMB (139, 445). Also, a web application is running on port 80. Lastly, nMap tool recognized Linux as operative system.

# **Initial foothold**

First of all, I investigated SMB service and I looked for some interesting shares:

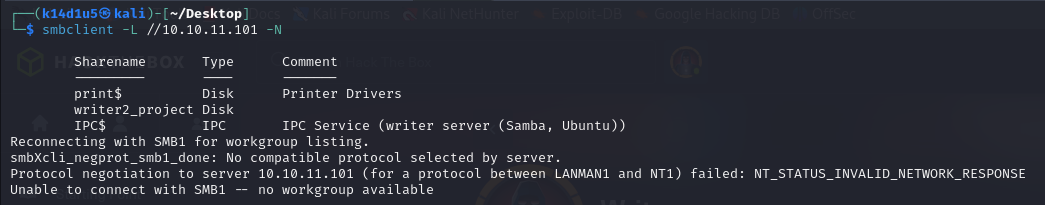


Figure 2 - SMB shares found

However, I was not able to access to them because I hadn’t credentials. Therefore, I investigated the web application running on port 80. Luckily, I found two interesting new paths:

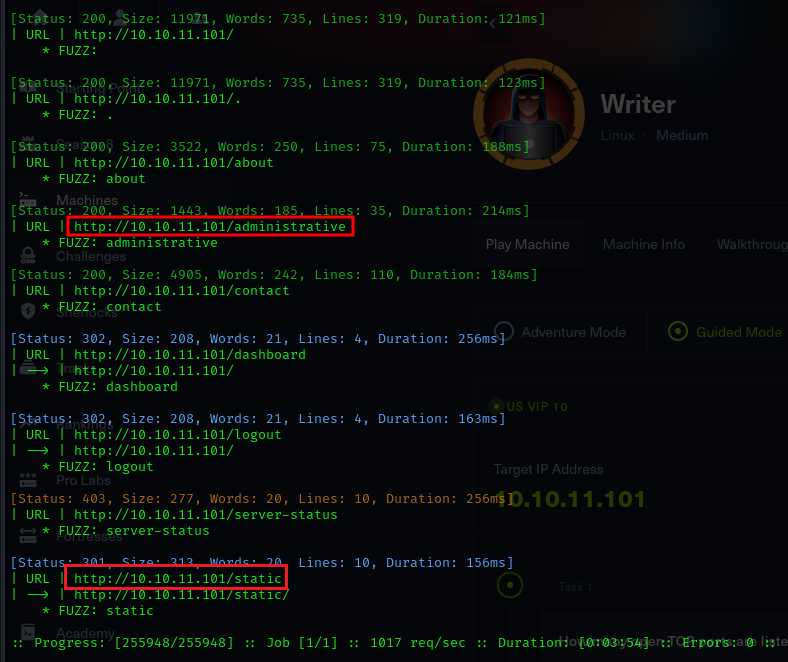


Figure 3 - ffuf scan results

# **User flag**

Since I found a login form on the new path, I tried to exploit it via SQL Injection. It was possible and I used SQLMap to accomplish this goal:

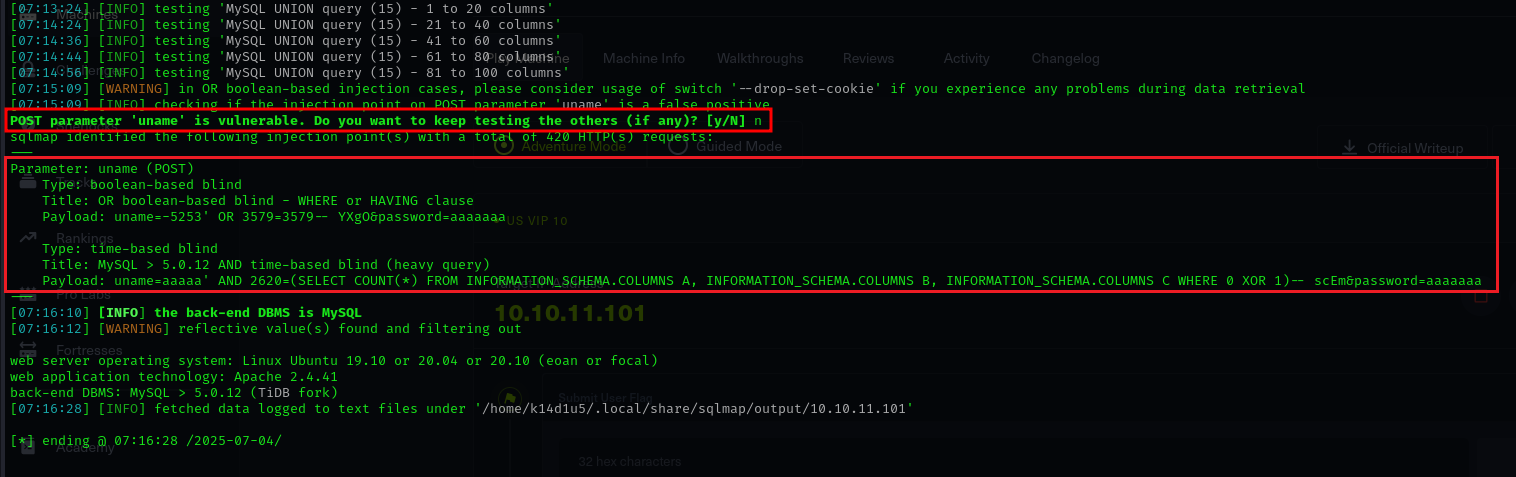


Figure 4 - SQL Injection found via SQLMap

Investigating deeper the SQLInjection, I found out that the user has interesting privileges:

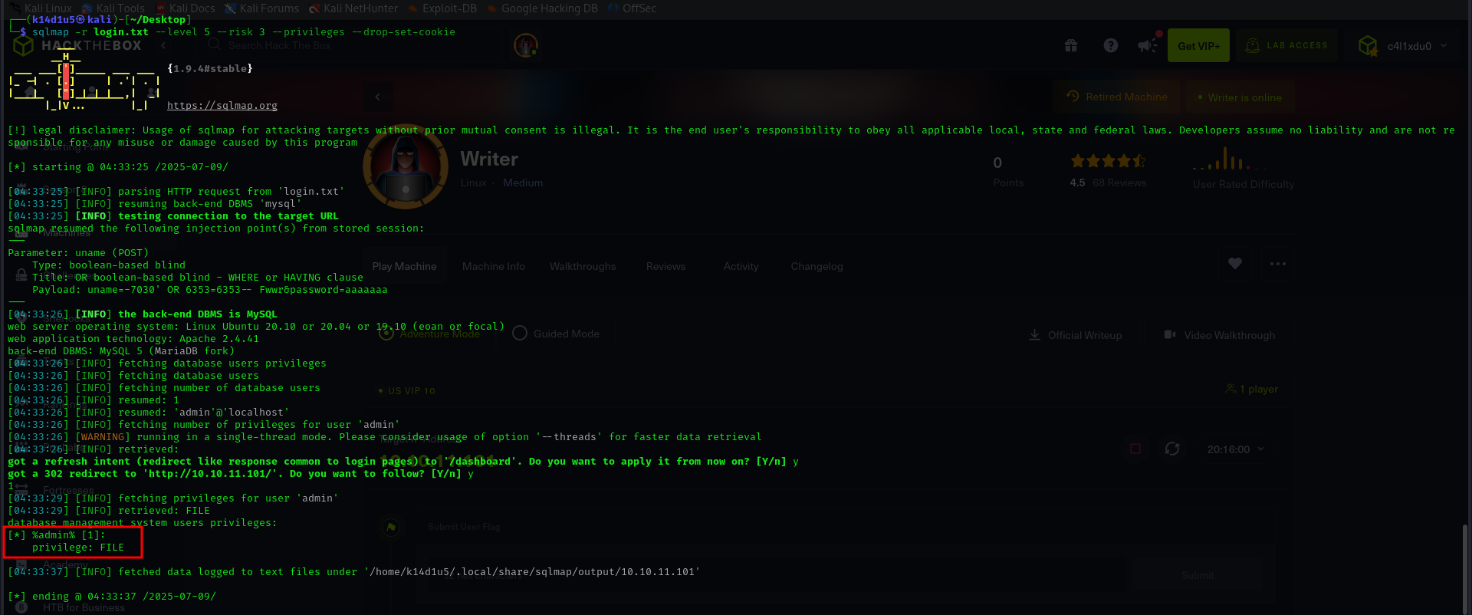


Figure 5 - DB user privileges

Due to this privilege, I was able to read some interesting files. One of these, allowed me to find a new virtual host, as shown I the following picture:

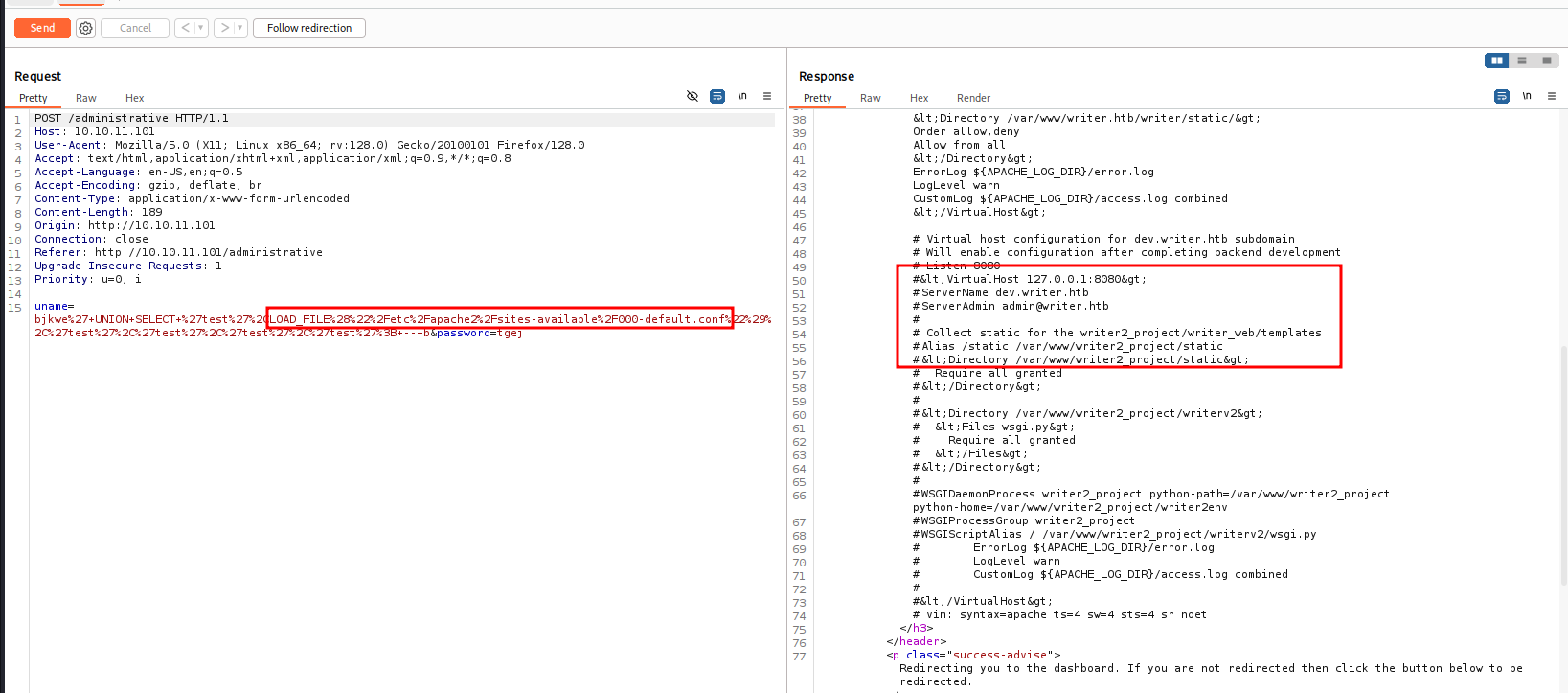


Figure 6 - Reading of 000-default.conf file

Also, I found a file named :

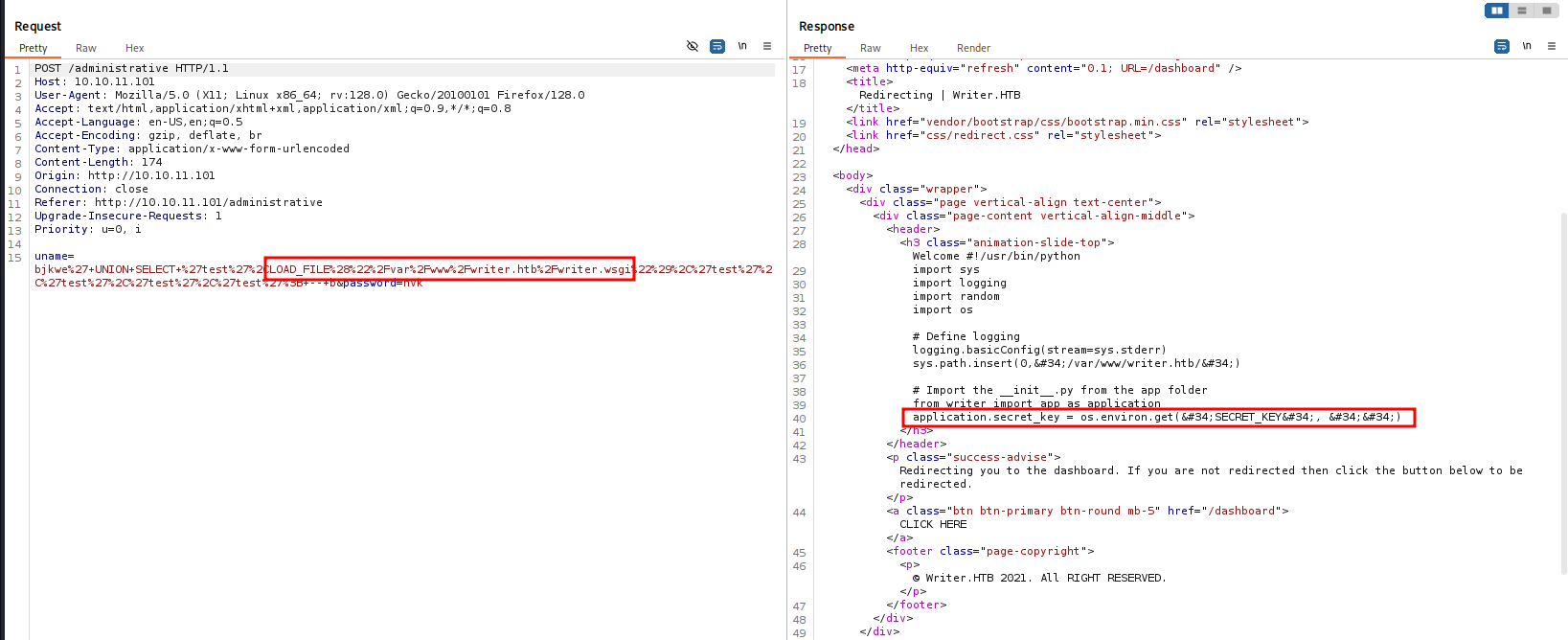


Figure 7 - writer.esgi file

The interesting thing about this file is that it is write in Python. Usually, a python project has a file. Therefore, I tried to read this file in the same way I just did. Luckily, this file existed and I downloaded it. Analyzing this file, I found the implementation of web application file upload functionality:

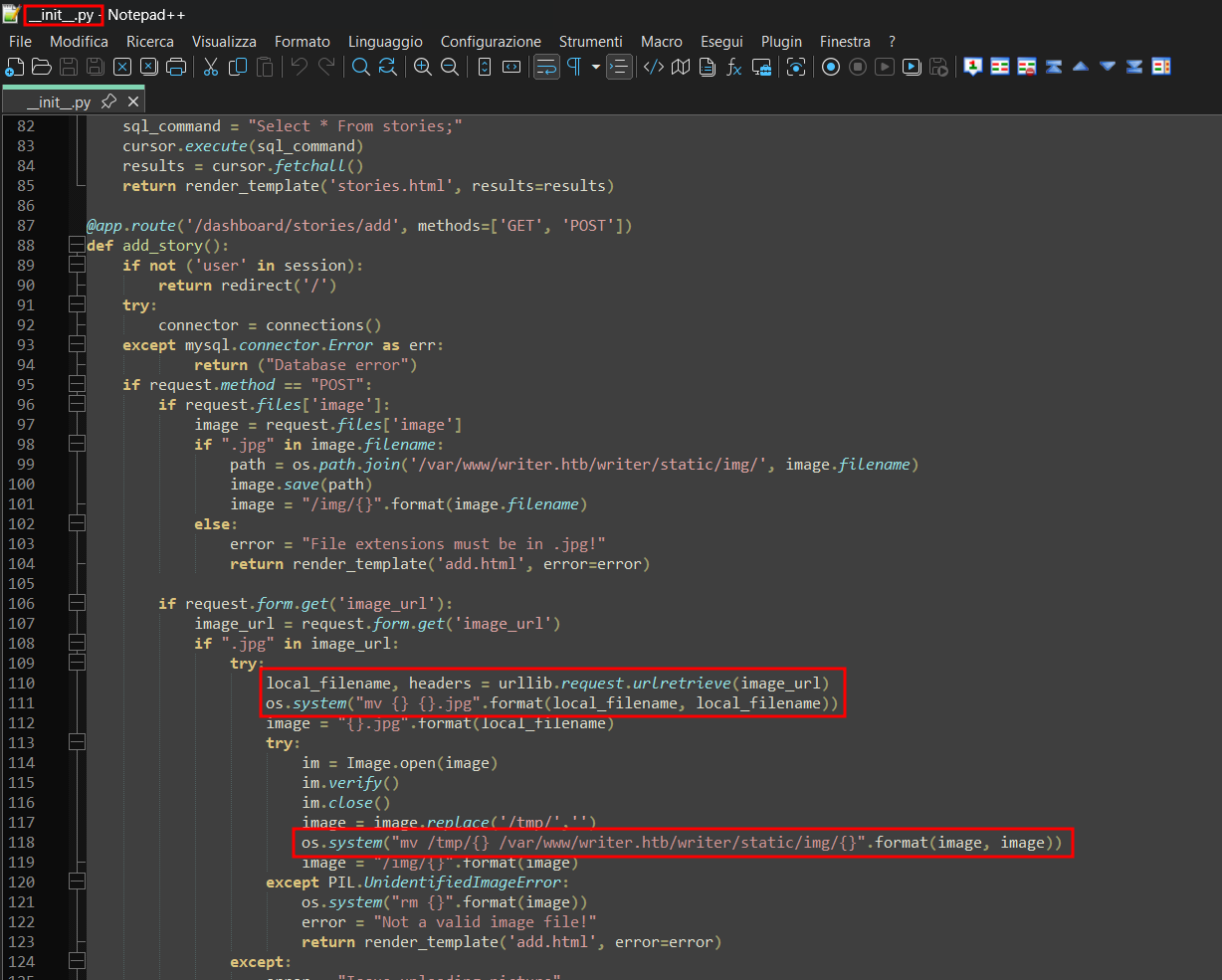


Figure 8 - \_\_init\_\_.py file

In particular, I learnt that the uploaded image was moved in the path. Also, it used the function that allow the user to provide a local file path, using the schema. Therefore, I tried to login in the web application. Using one of the SQLInjection payload used by SQLMap, I was able to access to the web application administrative panel. At this point, I tried to exploit the file upload functionality and I exploited the SSRF vulnerability. To do it, I created a fake image file which name has a reverse shell command encoded in base64, as shown in the following picture:

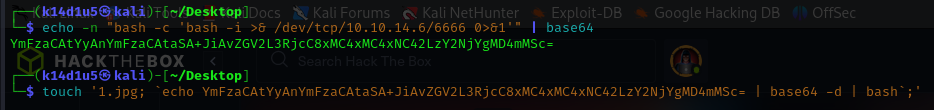


Figure 9 - Malicious fake image file

After I uploaded it, I was able to invoke the payload uploading it again. This time, I uploaded the file from the file system, as shown in the following picture:

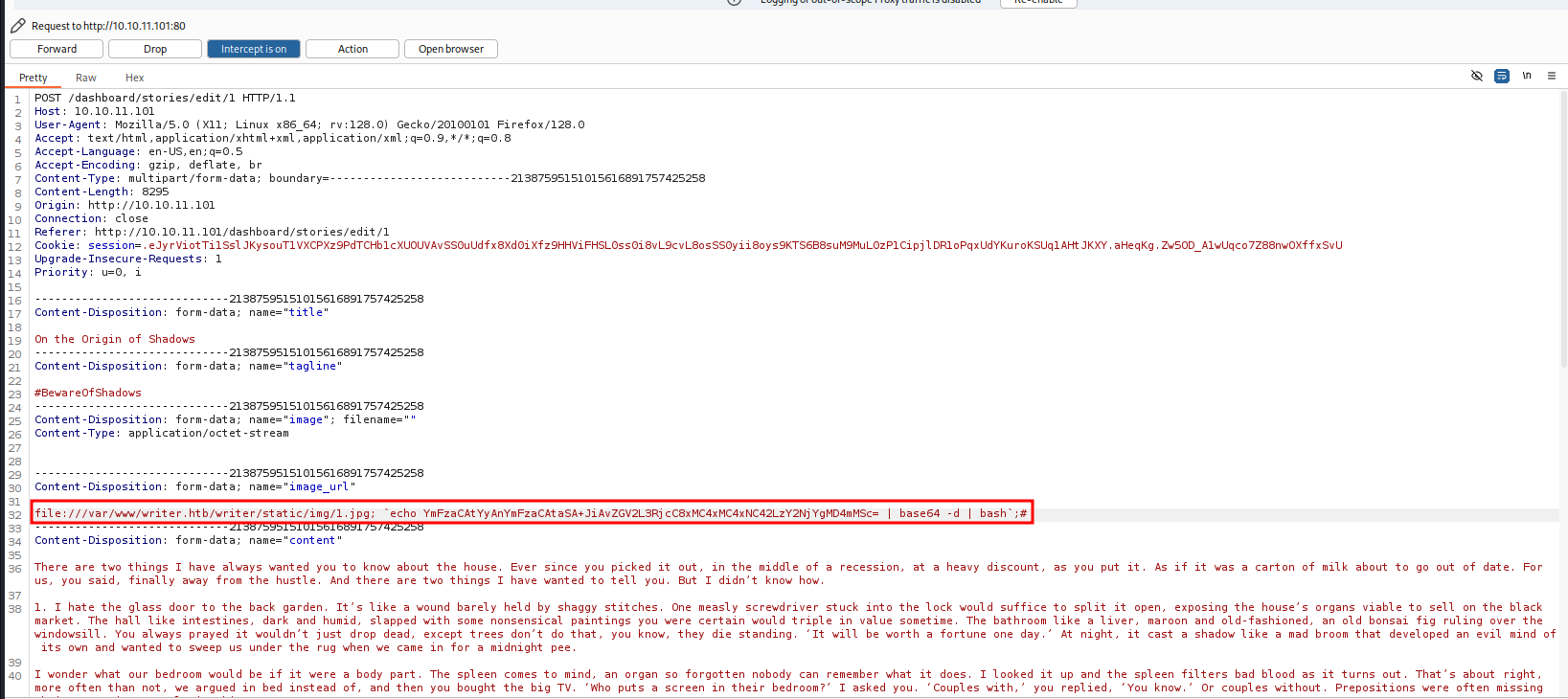


Figure 10 - Reverse shell invoking

In this way, I was able to obtain a shell as user. Therefore, I started to look for some interesting information. Luckily, I found out some database credentials:

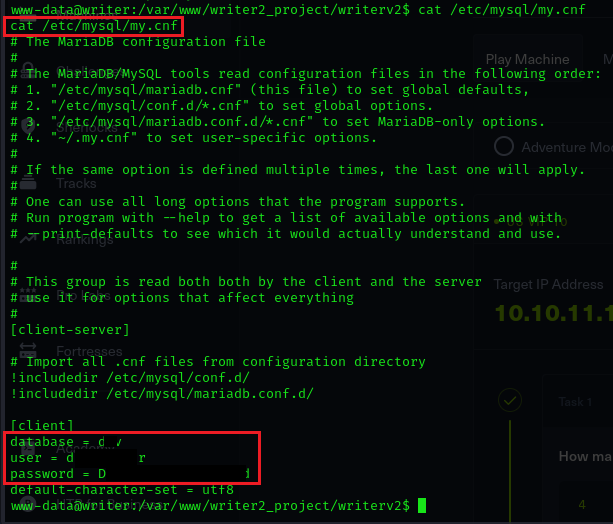


Figure 11 - Database credentials found

Next, I just connected to the database and I explored it. I found new credentials regarding the user:

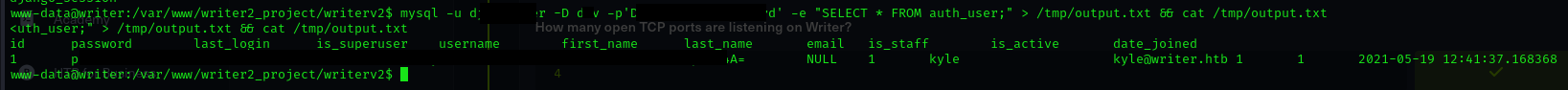


Figure 12 - Kyle credentials

Obviously, credentials are hashed. Next step was cracking it and, luckily, I was successful:

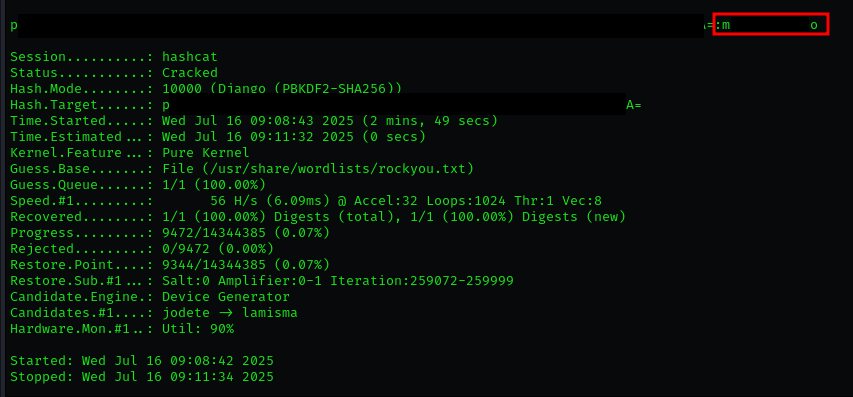


Figure 13 - Kyle credentials cracked

Finally, I was able to connect to the target via SSH as user and I retrieved the user flag:

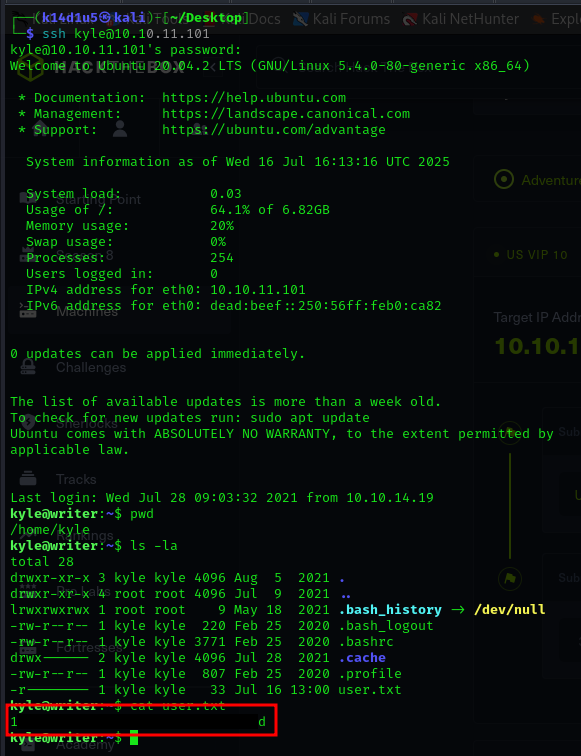


Figure 14 - User flag

# **Privilege escalation**

After all the effort I put until now, I started to find a way to escalate my privileges. First of all, I run LinPEAS tool. From its output, I found a Postfix interesting file used by user:

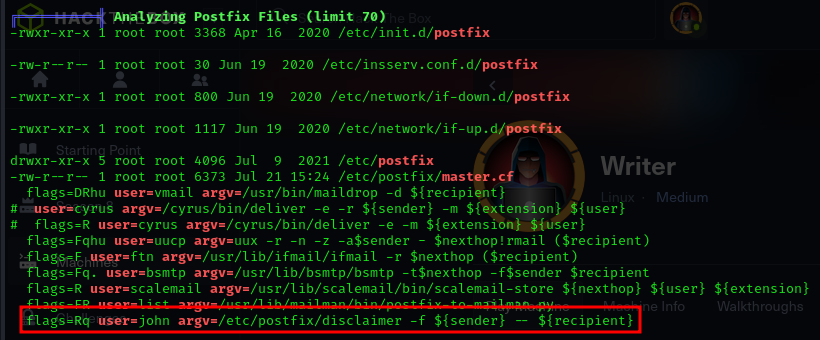


Figure 15 - LinPEAS output

At this point, I verified which Postfix version was installed:

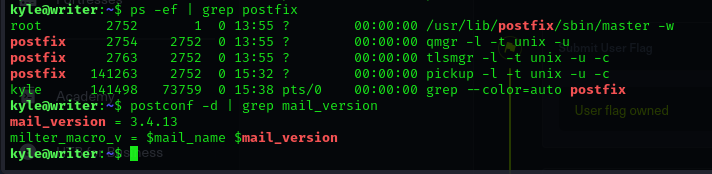


Figure 16 - Postfix version

Also, I investigated the disclaimer file. In particular, I noted that it was writable by group:

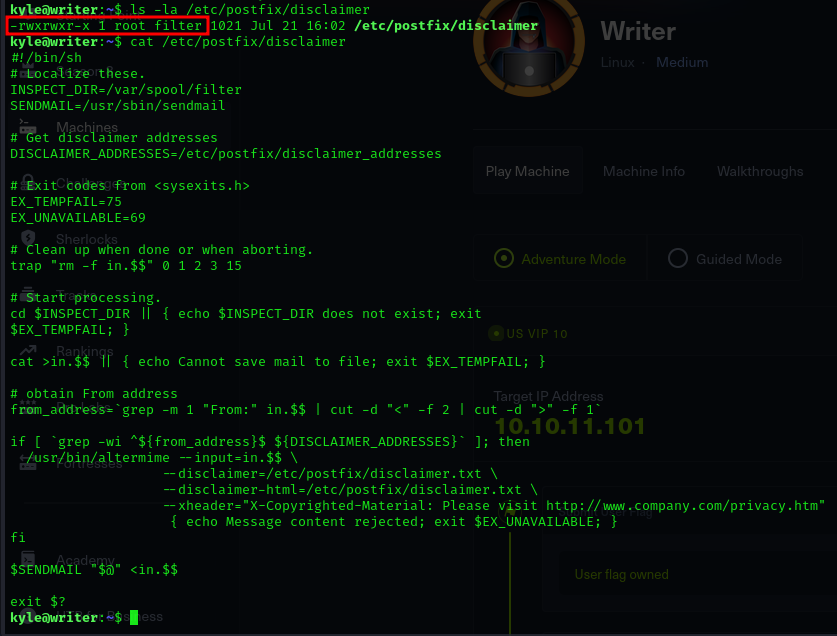


Figure 17 - Disclaimer file permissions

Therefore, I checked my user information. Luckily, my user was in the group:



Figure 18 - User info

At this point, I looked for an interesting exploit on the Internet and I performed it. In this way, I was able to obtain a first shell as user, as shown in the following picture:

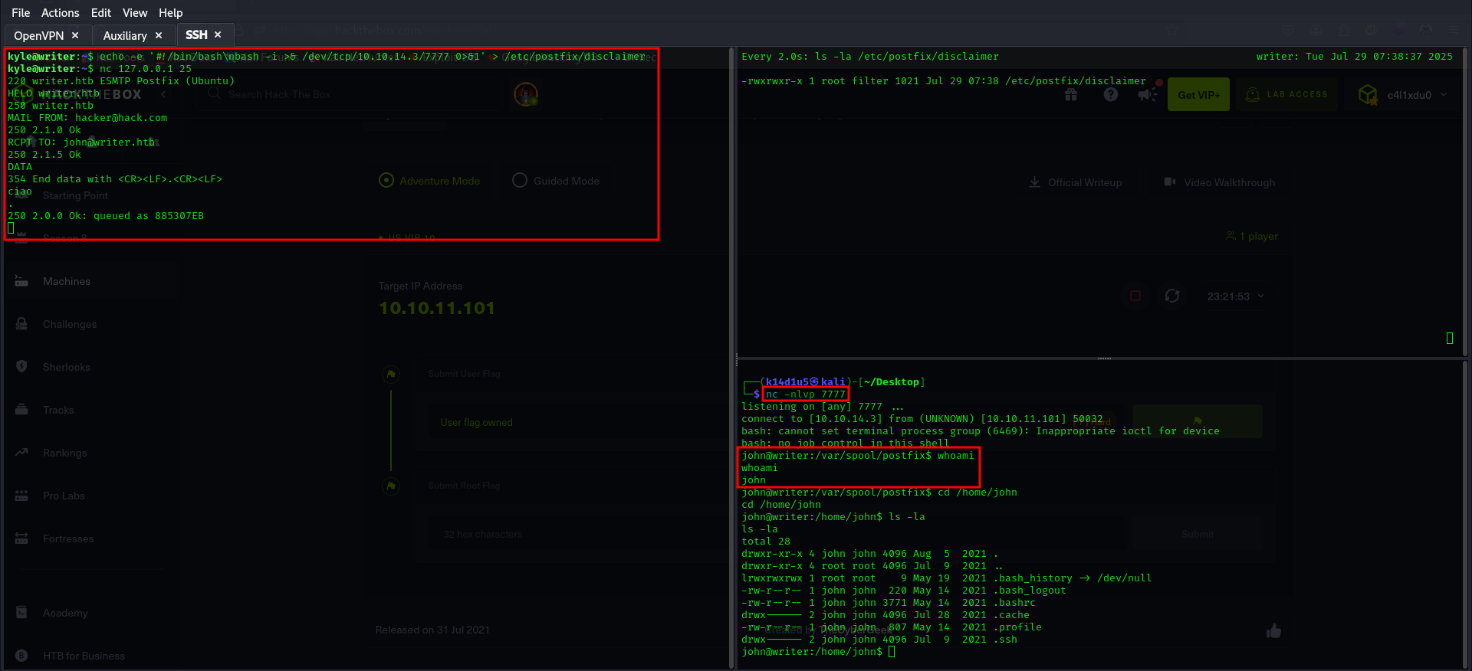


Figure 19 - First shell as john user

Sadly, this shell has not the environment variable set. This means that I was a little limited. Anyway, I looked for some interesting information and I found the ssh key:

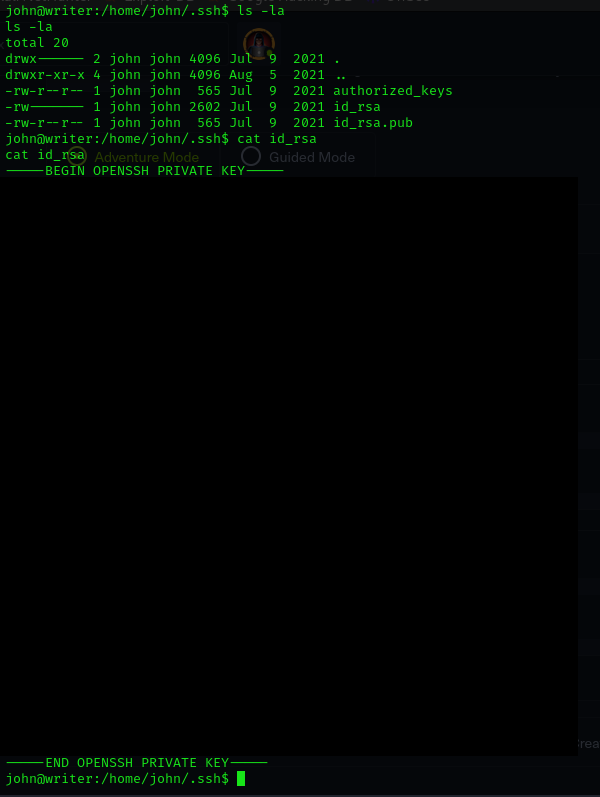


Figure 20 - John SSH key

I just used it to log in on the target as john and have a “complete” shell. Also, I found out that user is in the group:

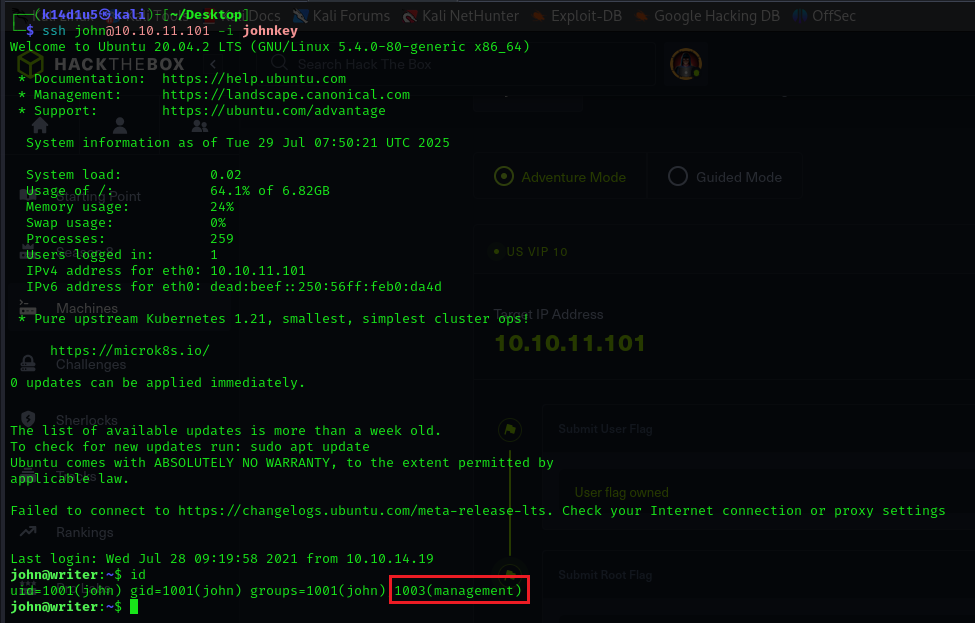


Figure 21 - Login as john via SSH

Therefore, I looked for some usable file from that group. I found out some apt tool relative file. These files are located in a folder in which the group can do anything:

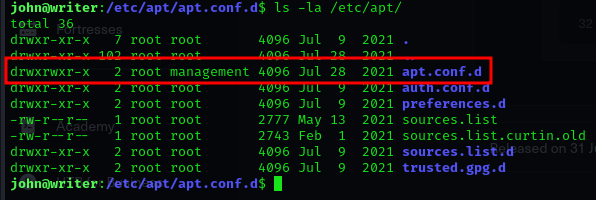


Figure - Folder permission

However, this group can’t modify the specific file:

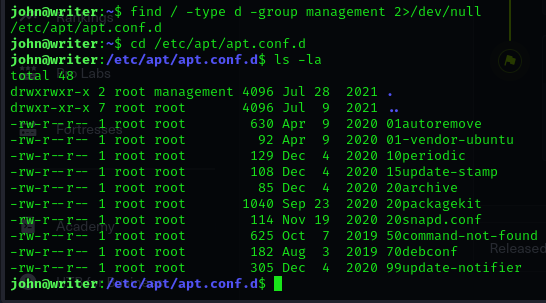


Figure 23 - APT task files

Also, I found out that all files in this folder run as root and are scheduled by cronjob. I found these information running the tool:

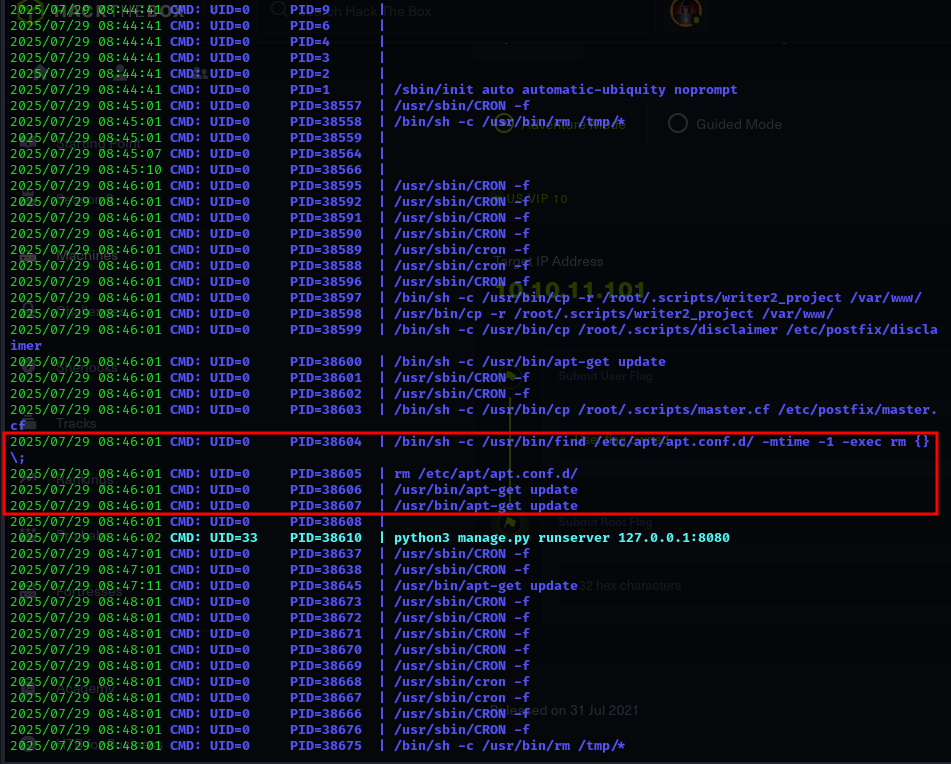


Figure 24 - Cronjob scheduled

I identify them as a cronjob because these commands were executed periodically. Analyzing some of those files, I tried to create a new one to obtain a reverse shell. Also, looking possible exploit on the Internet, I learnt that these files were executed in alphabetic order. Lastly, I created my file to be executed as first one. In this way, I obtained the root shell and I retrieved the root flag:

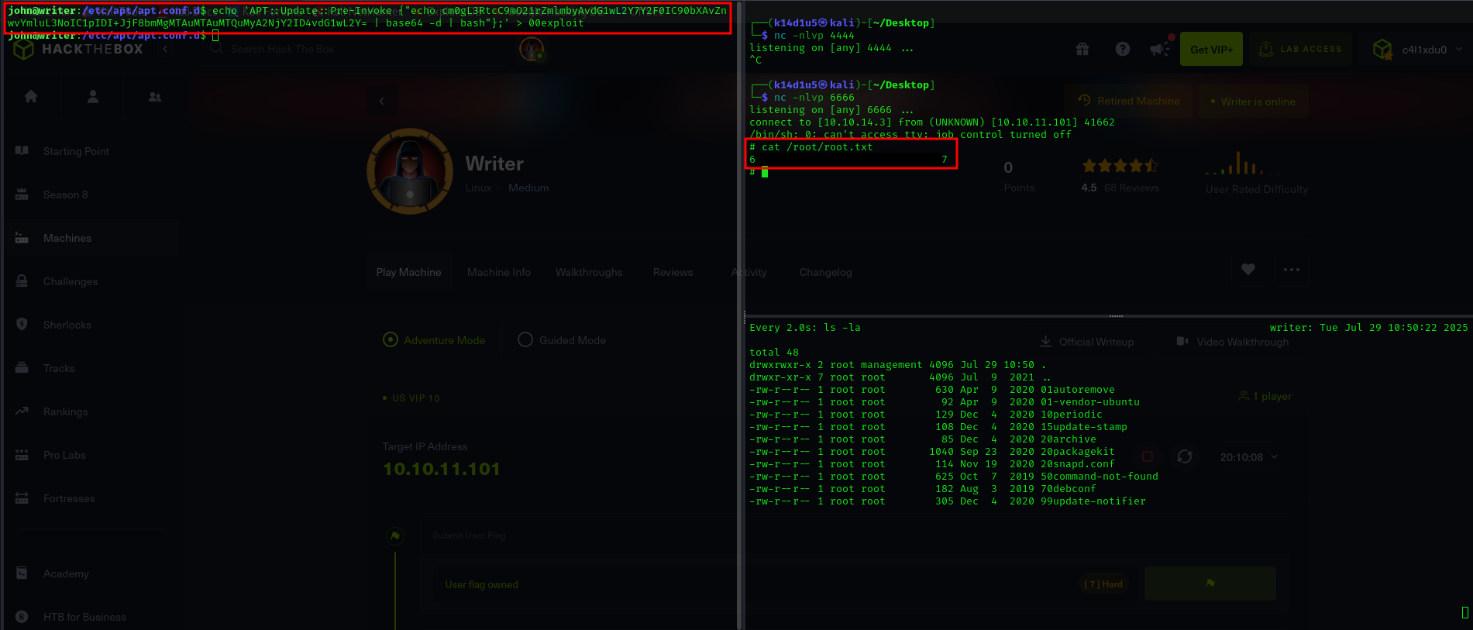


Figure 25 - root shell and root flag

# **Personal comments**

This box was literally crazy. There were several points that make this box both interesting and challenging. I needed a very uncommon flag () to let SQLMap work. At least, it was uncommon to me. Another uncommon task was reading file using SQLInjection. This task is very specific and it was pretty new to me. Also, it was needed to check database user permission to verify it was able to read files, another check I never seen before. Keep going on, I needed to hypothesize the existence of file just because I found a python file. In my opinion, this is a forced thought. Lastly, the first shell obtained as hadn’t the environment variable set. It was a new situation I never found before and I found out that it is limiting. Therefore, I needed to understand that with a different and “complete” shell, I was able to retrieve more details. In conclusion, this box is very interesting and very instructive, but it can’t be evaluated as Medium in my opinion. I evaluate it Hard or at least very close to this difficulty.

# **References**

1. Privilege escalation via APT tool: <https://www.hackingarticles.in/linux-for-pentester-apt-privilege-escalation/>.