**Frolic walkthrough**

# **Index**

[Index 1](#_Toc197628327)

[List of pictures 1](#_Toc197628328)

[Disclaimer 2](#_Toc197628329)

[Reconnaissance 2](#_Toc197628330)

[Initial foothold 3](#_Toc197628331)

[User flag 4](#_Toc197628332)

[Privilege escalation 7](#_Toc197628333)

[Personal comments 9](#_Toc197628334)

[References 9](#_Toc197628335)

# **List of pictures**

[Figure 1 - nMap scan results 2](#_Toc197628304)

[Figure 2 - Admin portal uses login.js script 3](#_Toc197628305)

[Figure 3 - Hardcoded credentials found 3](#_Toc197628306)

[Figure 4 - Decoded first code 4](#_Toc197628307)

[Figure 5 - Second code found 4](#_Toc197628308)

[Figure 6 - Creating zip file 4](#_Toc197628309)

[Figure 7 - Preparing to JohnTheRipper tool 4](#_Toc197628310)

[Figure 8 - Zip password cracked 5](#_Toc197628311)

[Figure 9 - Decoding third code: step 1 5](#_Toc197628312)

[Figure 10 - Decoding third code: step 2 5](#_Toc197628313)

[Figure 11 - Decoding third code: step 3 6](#_Toc197628314)

[Figure 12 - Second FFUF run 6](#_Toc197628315)

[Figure 13 - New path found 6](#_Toc197628316)

[Figure 14 - PlaySMS exploit 7](#_Toc197628317)

[Figure 15 - First shell obtained 7](#_Toc197628318)

[Figure 16 - User flag 7](#_Toc197628319)

[Figure 17 - Interesting custom executable file 8](#_Toc197628320)

[Figure 18 - ASLR check 8](#_Toc197628321)

[Figure 19 - Other checks related to buffer overflow 8](#_Toc197628322)

[Figure 20 - Offset to the return address 8](#_Toc197628323)

[Figure 21 - List of libraries used by rop executable 9](#_Toc197628324)

[Figure 22 - Function to use to exploit the buffer overflow 9](#_Toc197628325)

[Figure 23 - Privilege escalation and root flag 9](#_Toc197628326)

# **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:

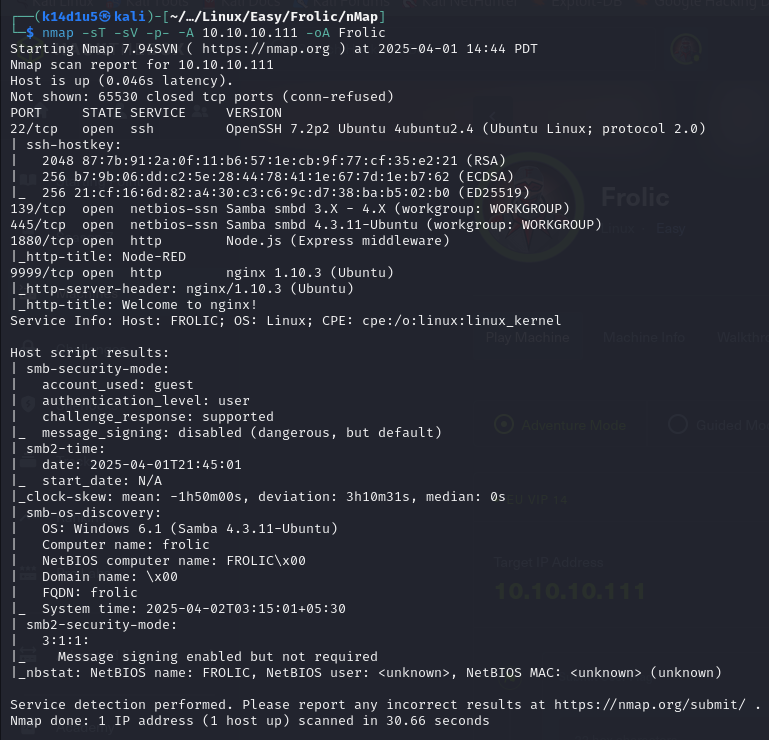


Figure 1 - nMap scan results

Open ports are 22, 139, 445, 1880 and 9999. Therefore, enabled services are SSH (22) and NetBIOS (139 and 445). Also, two web applications are running on 1880 and 9999 ports. Astly, nMap identified Linux as OS (probably Ubuntu).

# **Initial foothold**

First of all, I browsed to the two web application I found on port 1880 and 9999. On port 1880, I found a Node-RED login. On the web application running on port 9999 I found the nginx default page. After that, I run FFUF on both web application. In particular, I found two paths on the web application on port 9999. These paths are and . I analyzed the path and I found that it uses a JavaScript script to perform the login, as shown in the following picture:

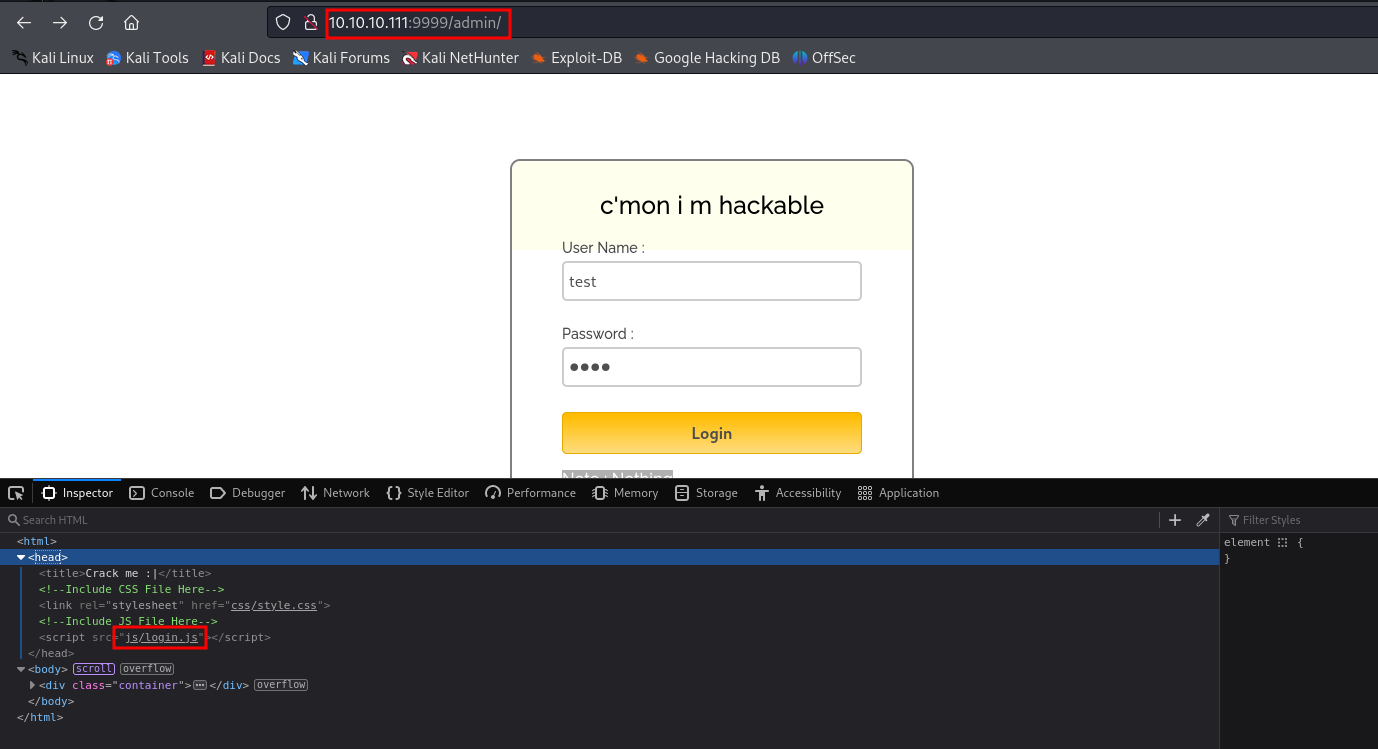


Figure 2 - Admin portal uses login.js script

I investigated this script and I luckily found hardcoded credentials:

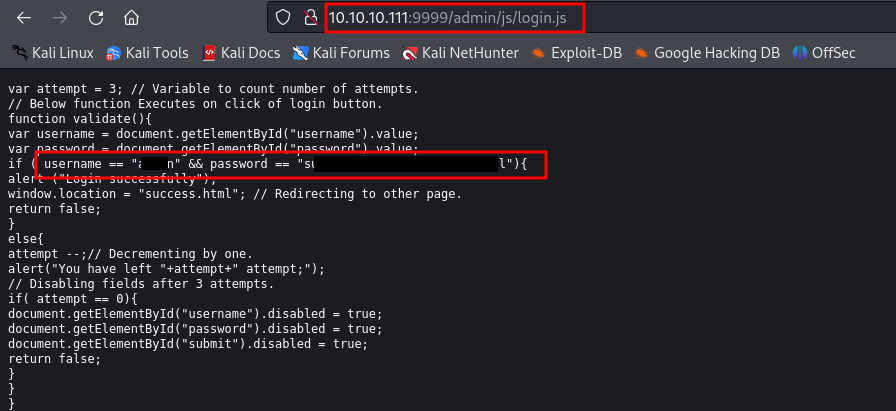


Figure 3 - Hardcoded credentials found

# **User flag**

These credentials allowed me to login on the web application. After login, I found in the web application a coded message. After a search on the Internet, I found a useful site to decode it:



Figure 4 - Decoded first code

Note that I needed to use the “Ook! Short code (to text)” method. In this way, I found a new path on the web application. Browsing to it, I found a new code. It was similar to base64, as shown in the following:

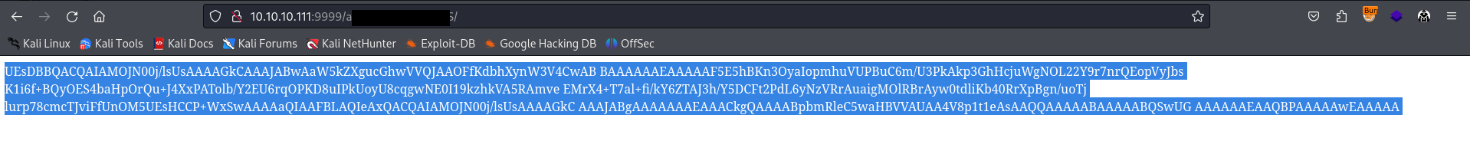


Figure 5 - Second code found

Trying to decode this string as base64 stored in a file, I found out it is representative of a zip file. Therefore, I created the zip file using the following command:

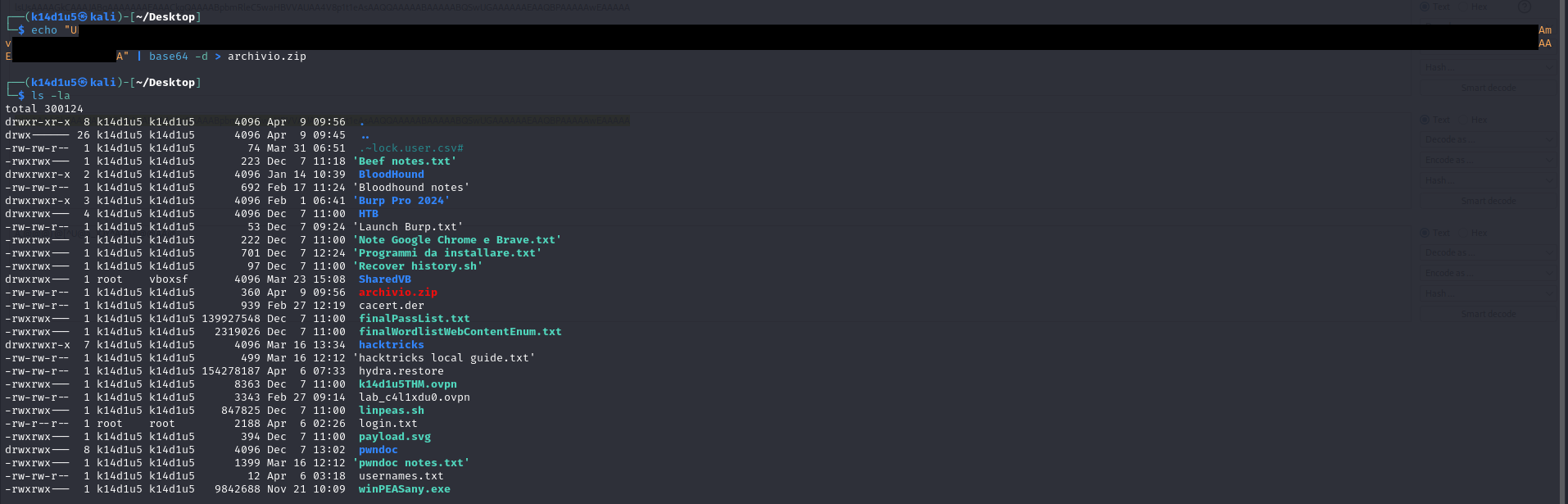


Figure 6 - Creating zip file

I tried to open this file, but I needed a password. At this point, I used JohnTheRipper tool to try to decode the password. To achieve this goal, I prepared the file to pass it to JohnTheRipper tool:

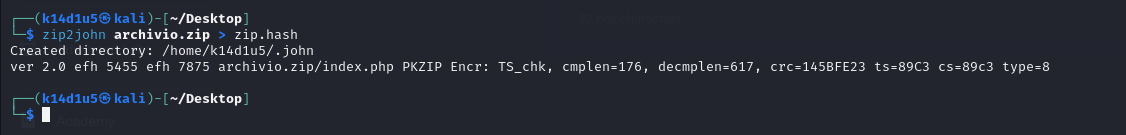


Figure 7 - Preparing to JohnTheRipper tool

Finally, I tried to crack it and I was successful, as shown in the following:

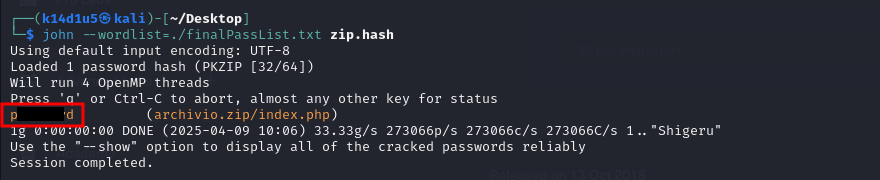


Figure 8 - Zip password cracked

In the archive, I found a new code. To decode it, I needed three steps, as shown in the following pictures:

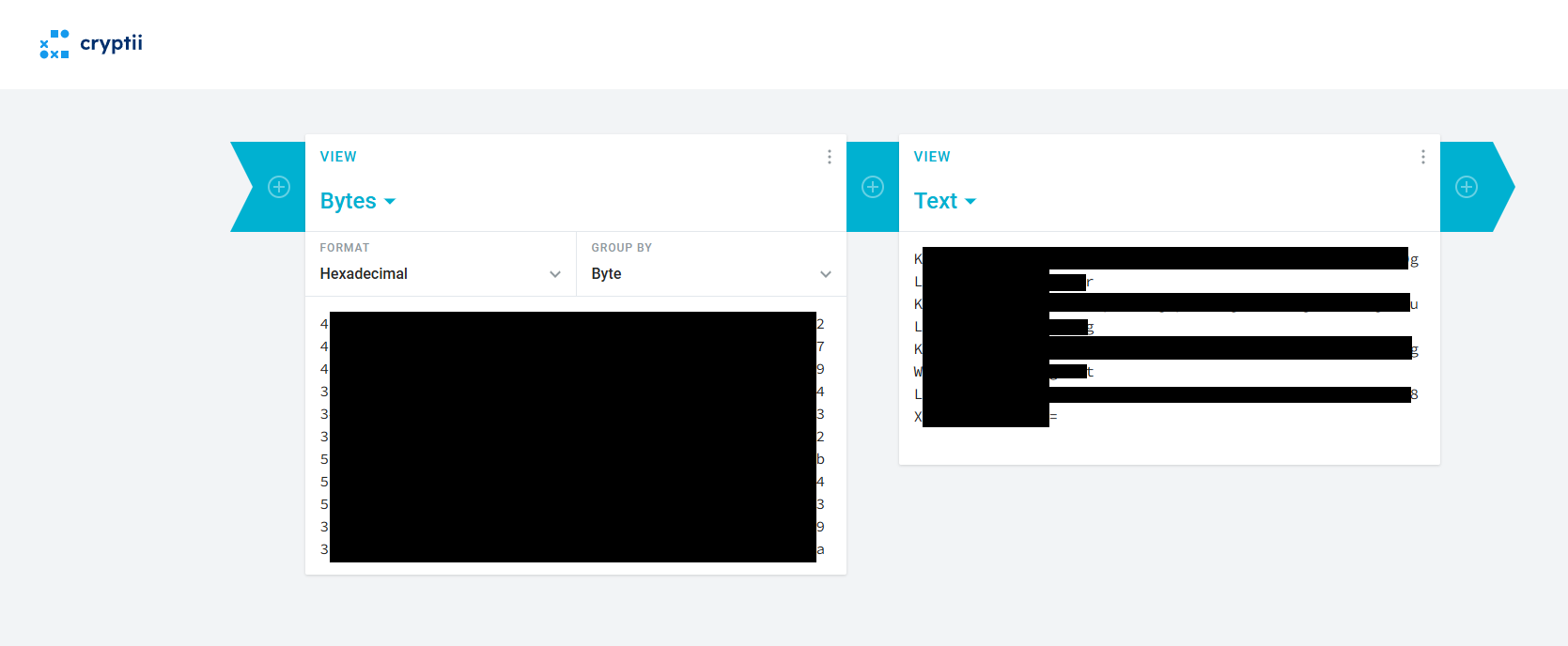


Figure 9 - Decoding third code: step 1

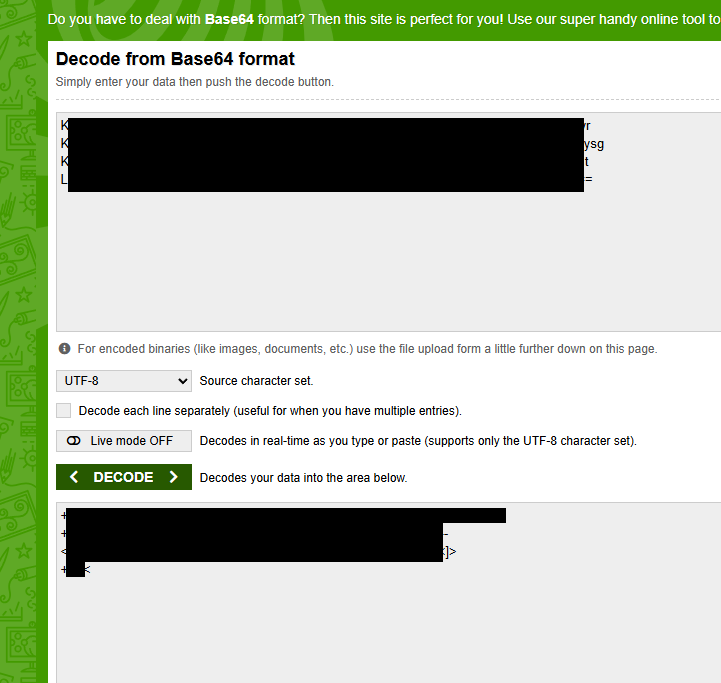


Figure 10 - Decoding third code: step 2

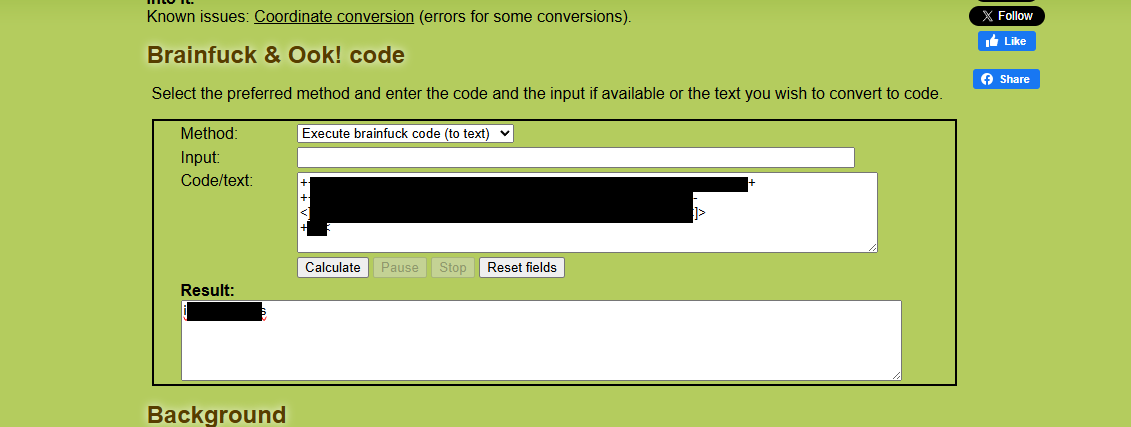


Figure 11 - Decoding third code: step 3

At this point I probably found a password. Therefore, I run again FFUF to keep to search new path. I run it on the path I preciously found. In this way, I found a new backup path, as shown in the following picture:

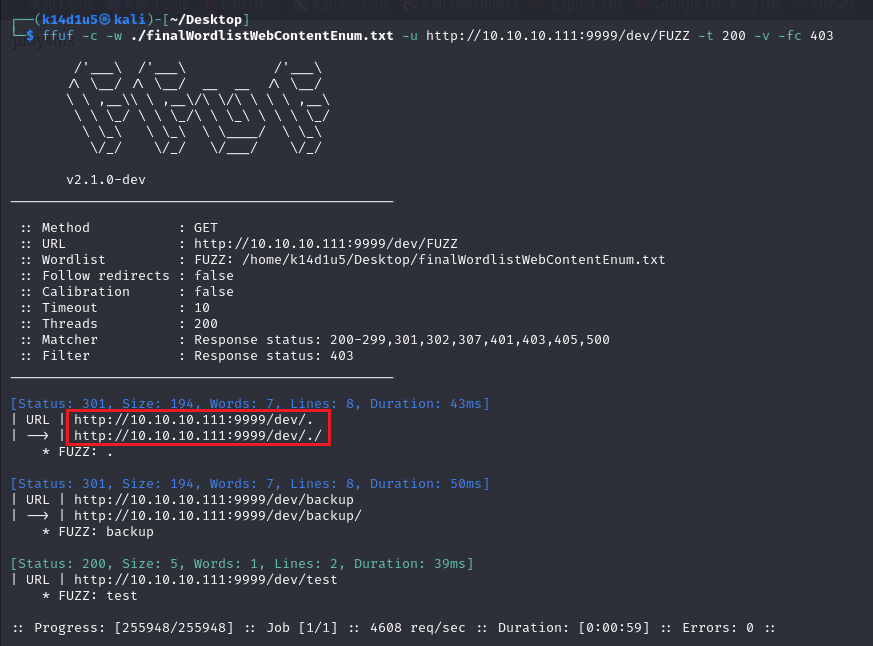


Figure 12 - Second FFUF run

When I browsed to this path, I found a new one, as shown in the following:

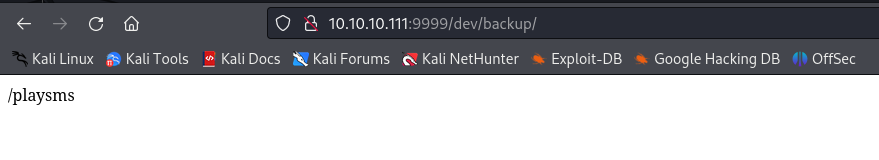


Figure 13 - New path found

On the new one, I found a new login. Luckily, I can log in here using the user and the password I decoded from the zip archive. On this last path, I found an application named PlaySMS and its version. Therefore, I looked for some exploit on the Internet. Luckily, I found an interesting one. At this point, I leverage it as shown in the following picture:

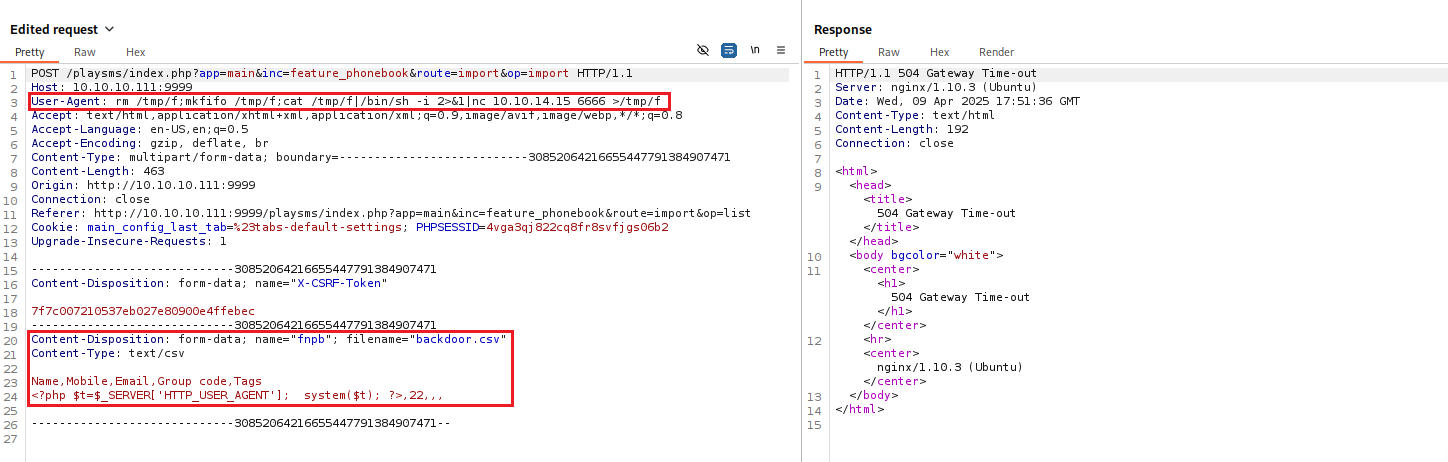


Figure 14 - PlaySMS exploit

In this way I obtained the first shell, as shown in the following:

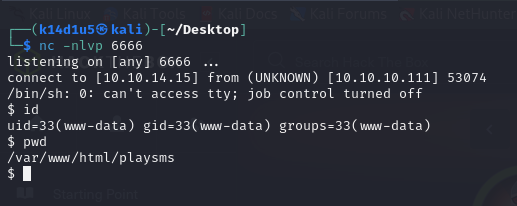


Figure 15 - First shell obtained

Luckily, I was able to read and retrieve the user flag:

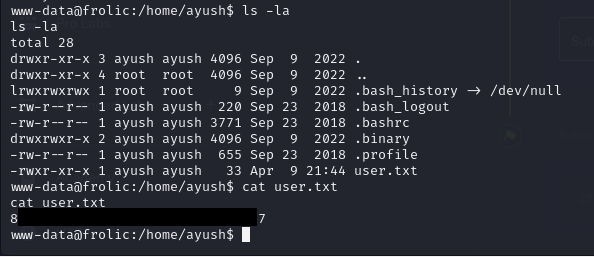


Figure 16 - User flag

# **Privilege escalation**

Finally, I was able to work on the privilege escalation. In the Ayush home folder, I found a non-standard folder named . This folder contains an executable file named rop. This file was owned by the user and has the suid bit set. Also, it required a message that is returned in the output. Therefore, I investigated it using the command:

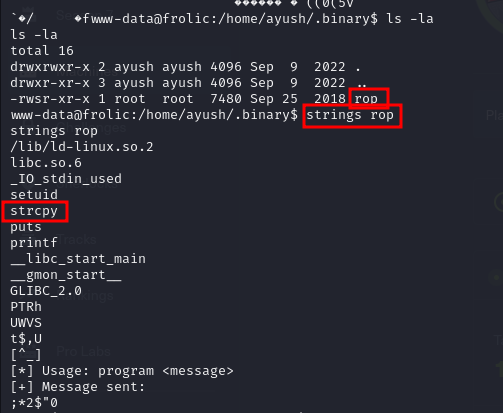


Figure 17 - Interesting custom executable file

Since it used the strcpy unsafe function, it could be vulnerable to the buffer overflow issue. Therefore, I performed some checks, as shown in the following pictures:



Figure - ASLR check



Figure 19 - Other checks related to buffer overflow

Since these outputs provided that PIE and ASLR were not enabled, it was possible to leverage the buffer overflow issue. Also, since the NX flag is enabled, I was not able to execute code from the stack. At this point, I calculated the offset to the return address:

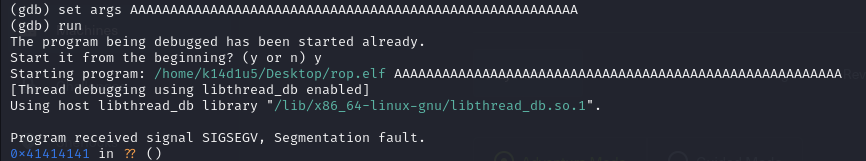


Figure 20 - Offset to the return address

I did it manually, but I was able to use the tool. The offset value was 52 bytes. At this point, I needed to find a function to call to leverage the buffer overflow. I just listed the libraries used by the rop executable file:

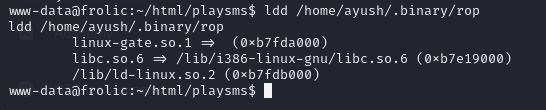


Figure 21 - List of libraries used by rop executable

Therefore, I checked the libraries implementation:

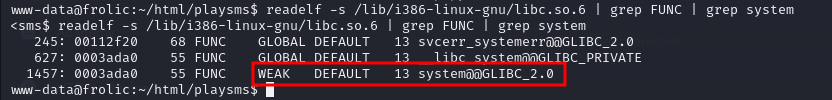


Figure 22 - Function to use to exploit the buffer overflow

Luckily, I found a function in the library. Finally, I have all the information I needed to run the exploit. I wrote a Python script to generate the full payload and I run the rop executable file using it. In this way, I obtained the root shell and I retrieved the root flag, as shown in the following picture:

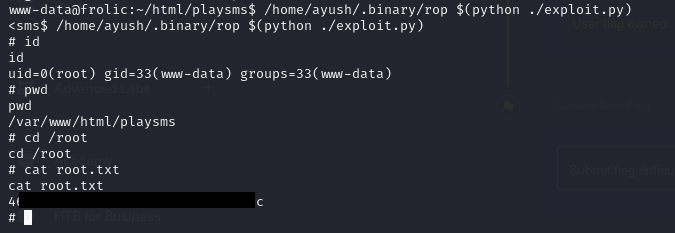


Figure 23 - Privilege escalation and root flag

# **Personal comments**

I experienced some strange behaviors when I did this box. I don’t know why FFUF worked only using the IP and not the URL and Burp Suite didn’t properly decoded hexadecimal code. It was a little bit strange and annoying. Even if the user flag required a lot of steps, it was very direct and easy to understand how to obtain it. The root flag was a little bit hard because it was a buffer overflow. Also, I needed to note that I was not able to execute code by the stack. So, it was not a “standard” kind of buffer overflow, but a particular one. Due to this, I spent more time to complete the box. At the end, I evaluate it as a Medium difficulty box.

# **References**

* Site used to decode codes: <https://www.geocachingtoolbox.com/index.php?lang=en&page=brainfuckOok>;
* SMSPlay exploit: <https://www.exploit-db.com/exploits/42044>;
* Special permission in Linux: <https://www.scaler.com/topics/special-permissions-in-linux/>;
* Difference by ASLR and PIE: <https://stackoverflow.com/questions/54747917/difference-between-aslr-and-pie>;
* Find a buffer overflow using GDB: <https://cseweb.ucsd.edu/~dstefan/cse127-fall20/notes/bufferoverflow.html>;
* How to exploit a ROP buffer overflow (Italian source): <https://medium.com/@steccami/buffer-overflow-rop-chain-un-esempio-pratico-34ce06096783>;
* ROP Buffer overlow: <https://github.com/ahkecha/ROP>.