

Report: Installation and Usage of DVWA for SQL Injection Testing

AMJAD AMEEN PP

CONTENTS

Indroduction.....	2
Installation of DVWA using Docker	3
Performing SQL Injection on DVWA	3
(Low Security Level).....	5
(Medium Security Level).....	6
(High Security Level)	8
Conclusion	11

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](https://github.com/eystsen/pentestlab.git) using the following command:

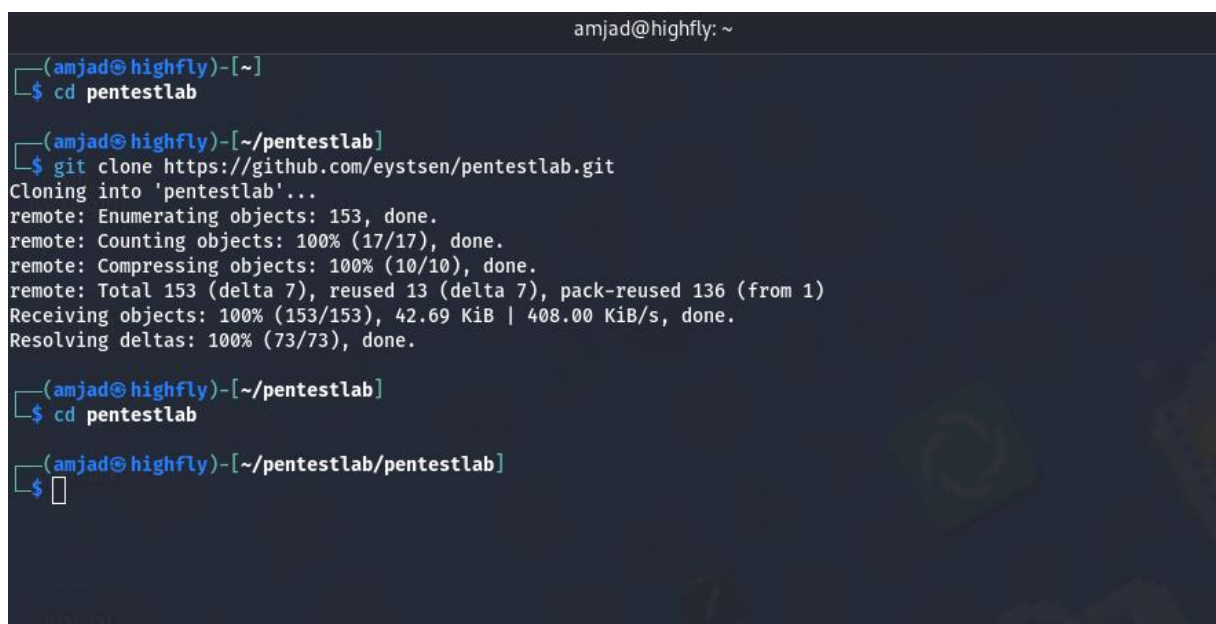
```
git clone https://github.com/eystsen/pentestlab.git
```

- 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:

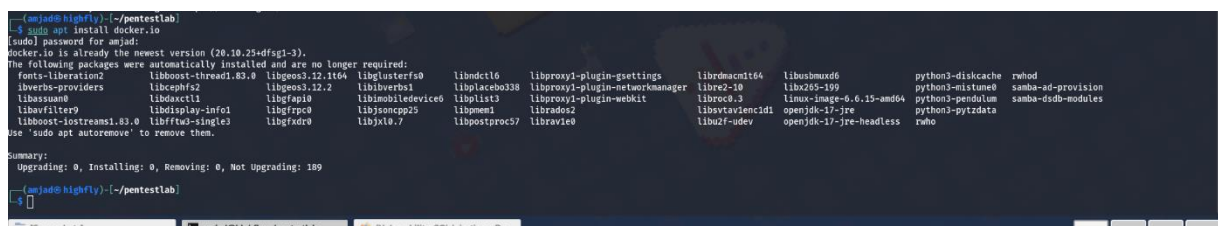
1. Opened the terminal and navigated to the cloned pentestlab folder.
2. Ran the following command to install Docker container:

```
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 (20.10.22+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 librdmacm164 libusbmuxd6 python3-diskcache rwho  
  libverbs-providers libcephfs2 libgeos3.12.2 liblberb51 libplacebo338 libproxy1-plugin-networkmanager libre2-10 libx265-199 python3-mistune6 samba-ad-provision  
  libassuan libdnet11 libgmp10 libmobiledevicec libplist3 libproxy1-plugin-webkit libroc3 libx265-199 python3-pendulum samba-csdb-modules  
  libavfilter9 libdisplay-info1 libgfrpc0 libjsoncpp25 libmnl librados2 librtv1 libstavienc1d1 openjdk-17-jre libx265-199 python3-pytdat  
  libboost-iostreams1.83.0 libbftw3-single3 libgfrpc0 libx10.7 libpostproc57 libravie0 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]  
$
```

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`

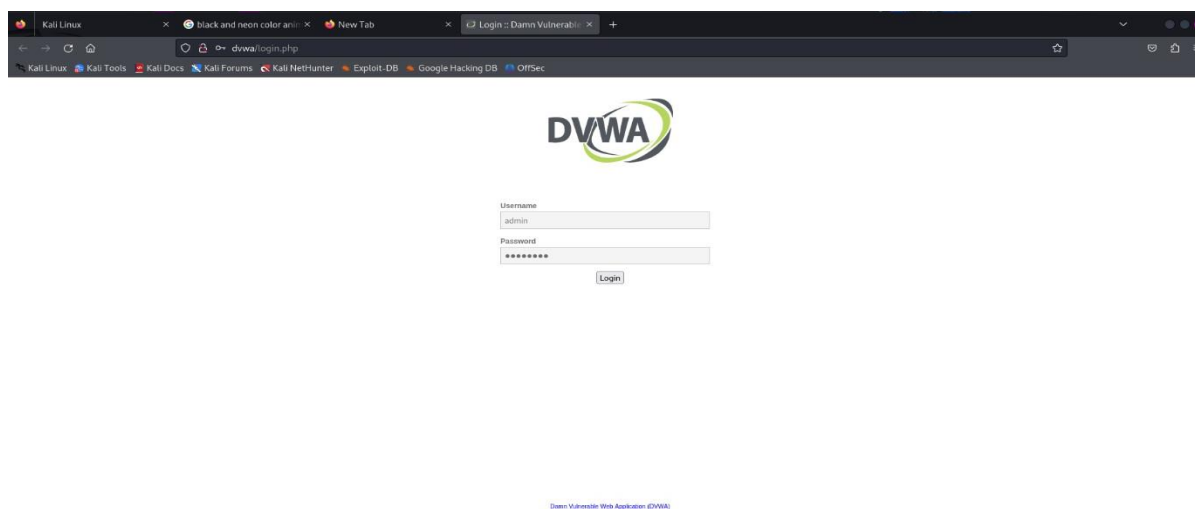
```
amjad@highfly:~/pentestlab
└─$ ./pentestlab.sh start dvwa
Starting Damn Vulnerable Web Application
Adding dvwa to your /etc/hosts
127.8.0.1    dvwa was added successfully to /etc/hosts
Not set
Running command: docker run --name dvwa -d -p 127.8.0.1:80:80 vulnerables/web-dvwa
Unable to find image 'vulnerables/web-dvwa:latest' locally
latest: Pulling from vulnerables/web-dvwa
1e17c0e6d6c: Pull complete
0c37d616dbf: Pull complete
eb05d18be401: Pull complete
e998e6e981d2: Pull complete
2c0720ba8257: Pull complete
6cfff33147f: Pull complete
098cfd43468: Pull complete
b3d64a32a2d4: Pull complete
Digest: sha256:d0e203fe11046a86937bf04db0879ade795f426da6a52b4e3b181f337daa7
Status: Downloaded newer image for vulnerables/web-dvwa:latest
66ff94d2f1709375d43f904789abacc6c1992f4007f2078ad094d173e273
docker: Error response from daemon: driver failed programming external connectivity on endpoint dvwa (1eb4023a07feca7d0abced62f5008a1f8c532cce8425fb095c285f238f071dab): Error starting userland proxy: listen tcp4 127.8.0.1:80: bind: address 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**



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

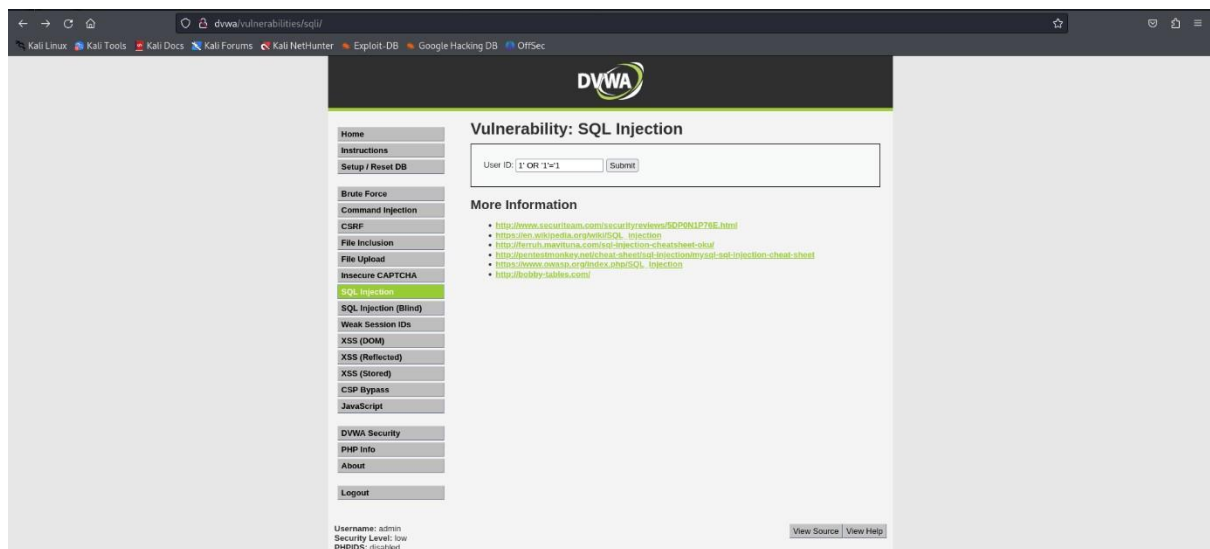
I started by testing SQL injection at the Low security level.

- 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`

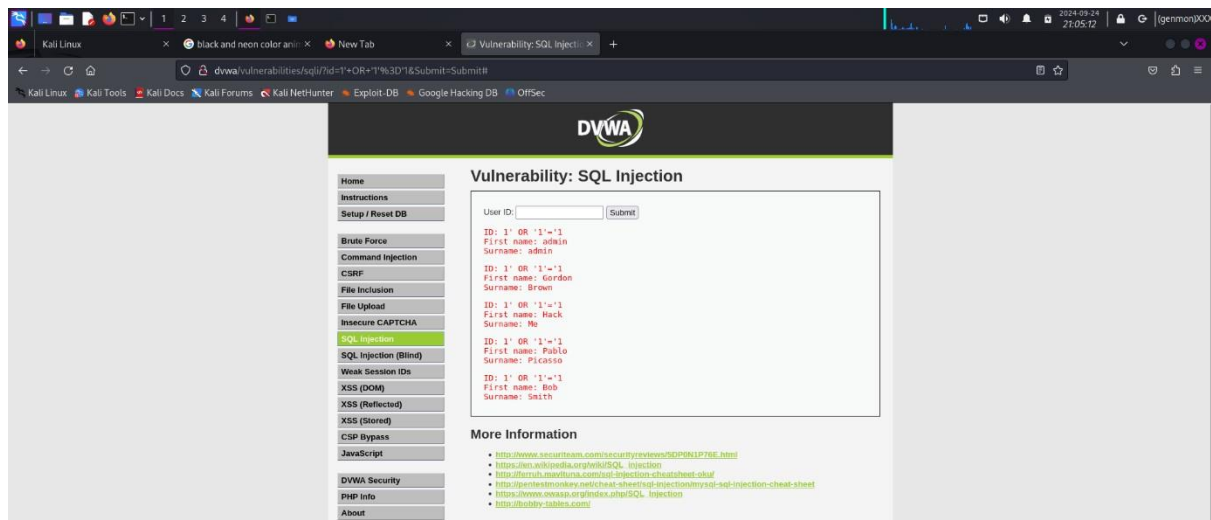


Screenshot 5

Find all users in the database:

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

This will return all rows from the users table.



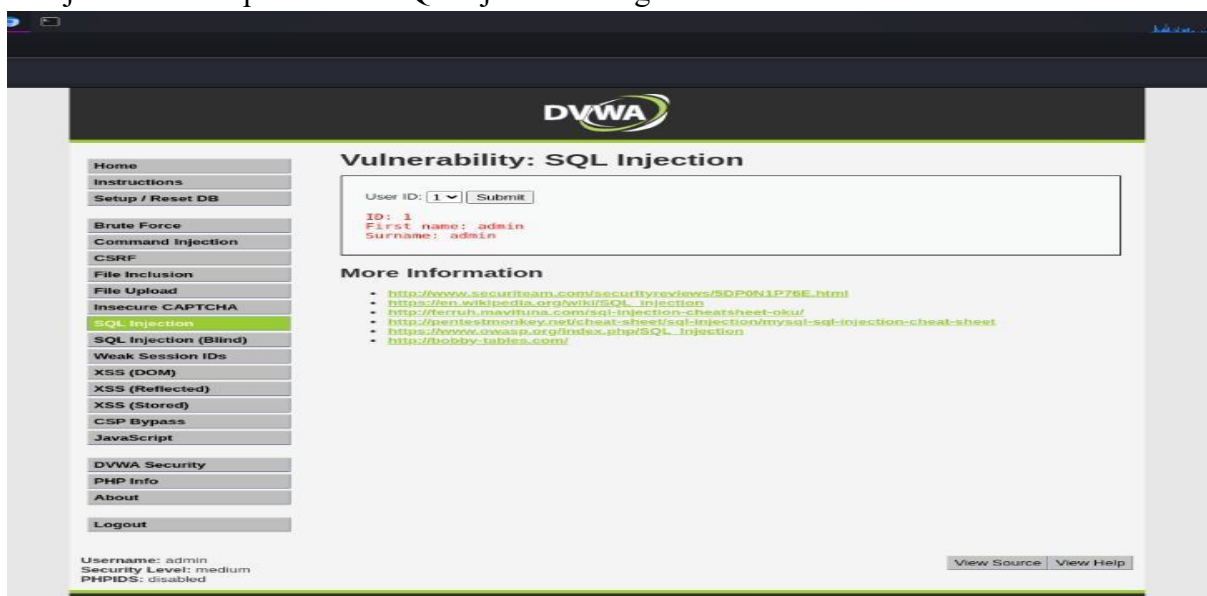
Screenshot 6

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.



Screenshot 7

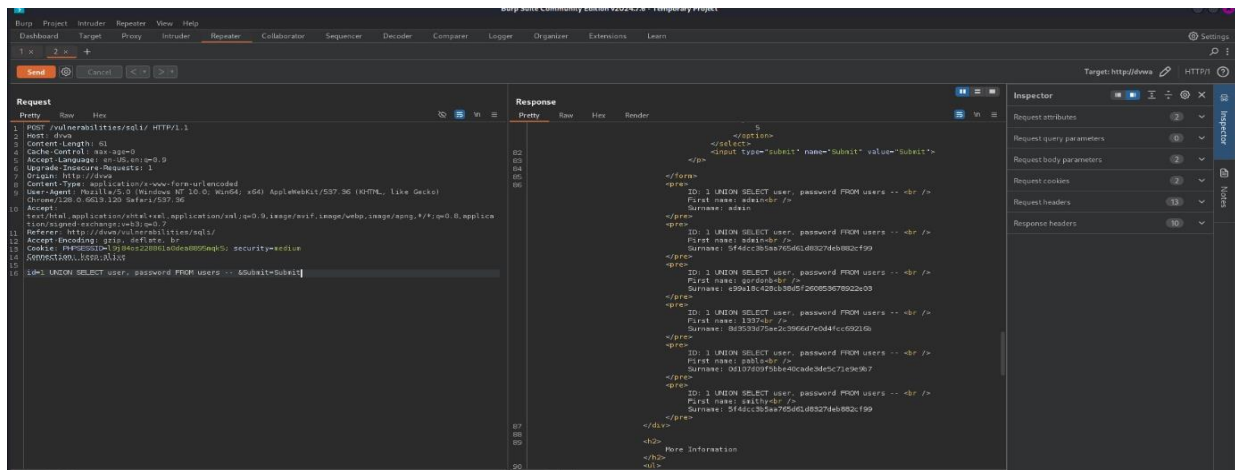
SQL Injection String

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.



Screenshot 8

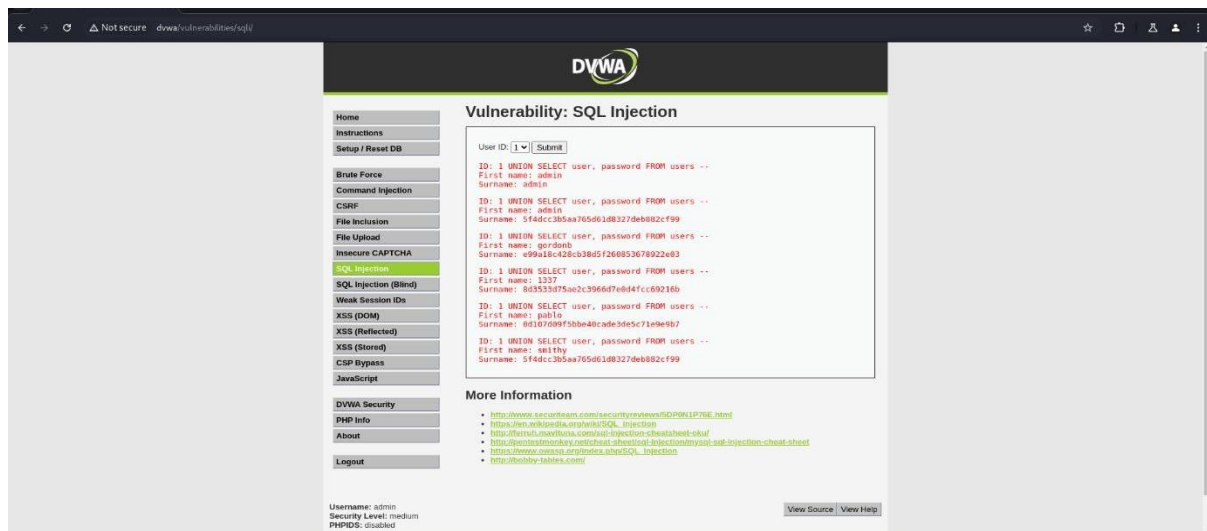
Execution :

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.



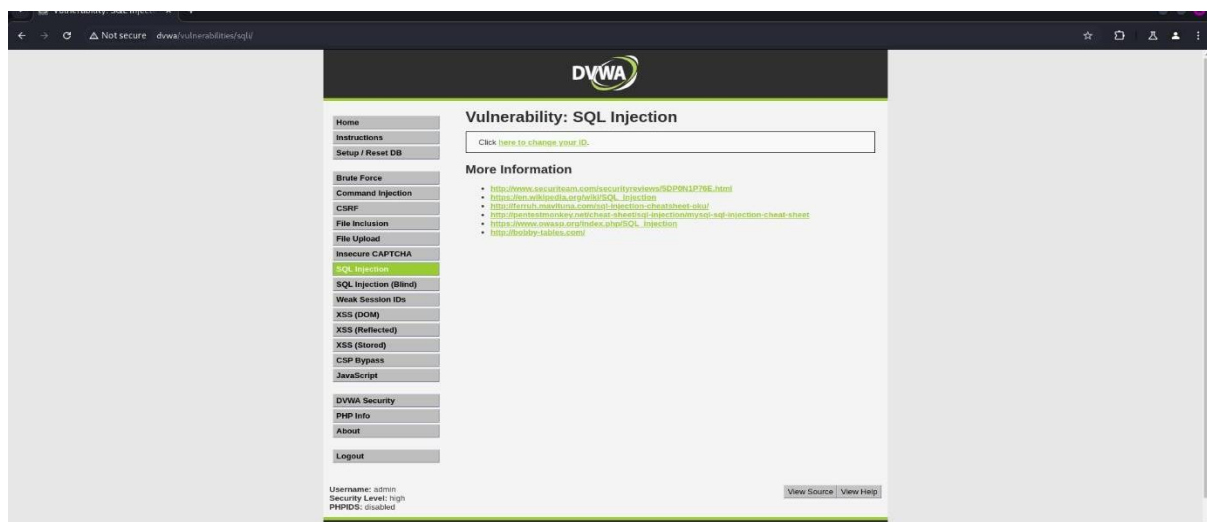
Screenshot 9

High Security Level

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

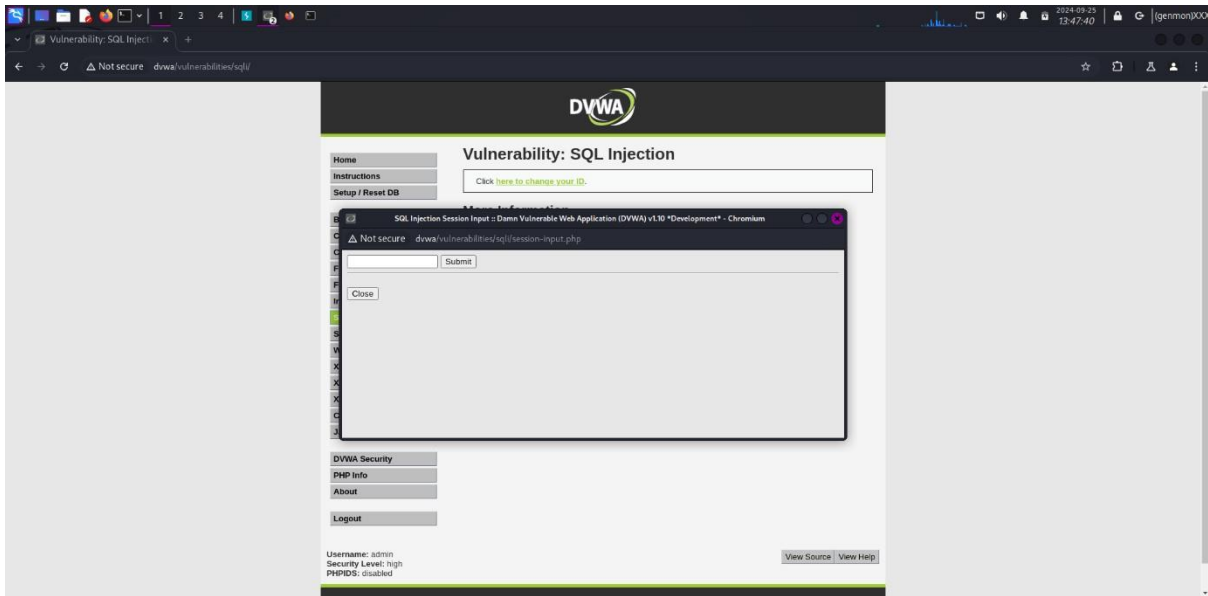
- Identifying the Injection Point

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



Screenshot 10

a new window appeared where I could input SQL command.



Screenshot 11

- Injection Payload

I inserted the following SQL injection string:

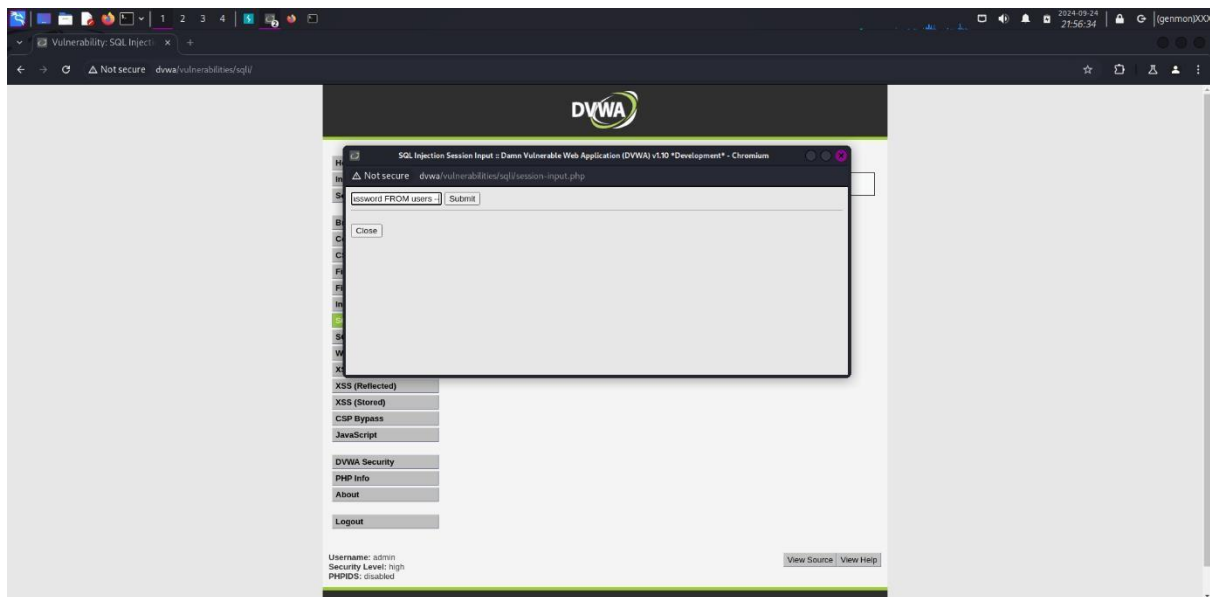
```
' UNION SELECT user, password FROM users --
```

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:

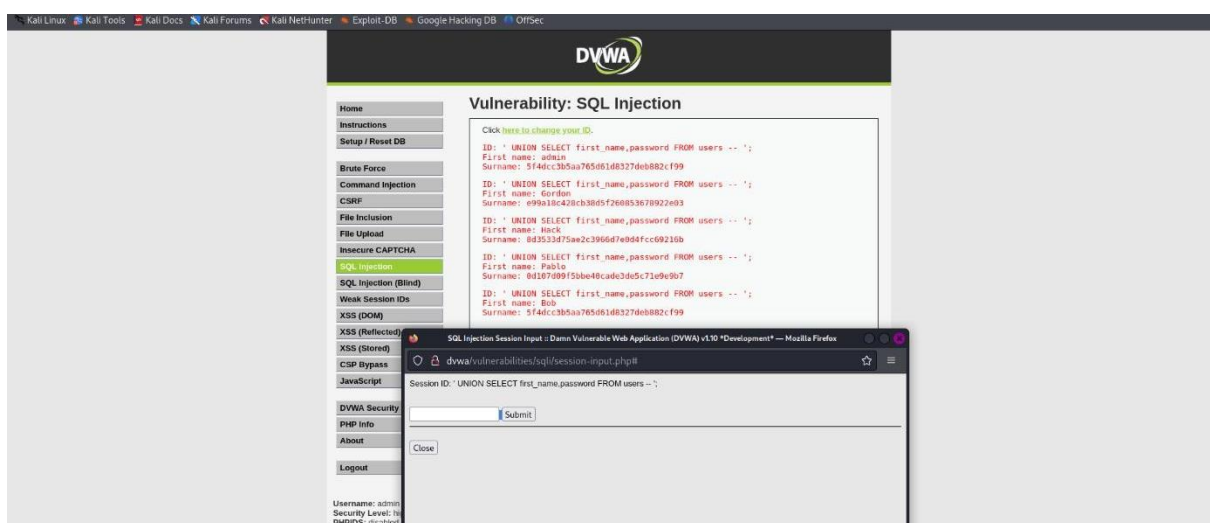
- The application will now display the results of both the original query (even though it returns no results due to the empty string ") and the injected query.
- The output on the webpage will show a list of usernames and their corresponding passwords, exposing sensitive information.



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.



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.