

# Report: Installation and Usage of DVWA for SQL Injection Testing

ABHIRAM T

# CONTENTS

<b>Installation of DVWA using Docker .....</b>	<b>2</b>
<b>Performing SQL Injection on DVWA .....</b>	<b>2</b>
SQL Injection (Low Security Level) .....	4
SQL Injection (Medium Security Level) .....	5
SQL Injection (High Security Level) .....	6
<b>Conclusion .....</b>	<b>9</b>

# 1. 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:

## 1.1 Cloning the Repository

I started by cloning the DVWA repository from [pentestlab.github.io](https://github.com/eystsen/pentestlab.git) using the following command:

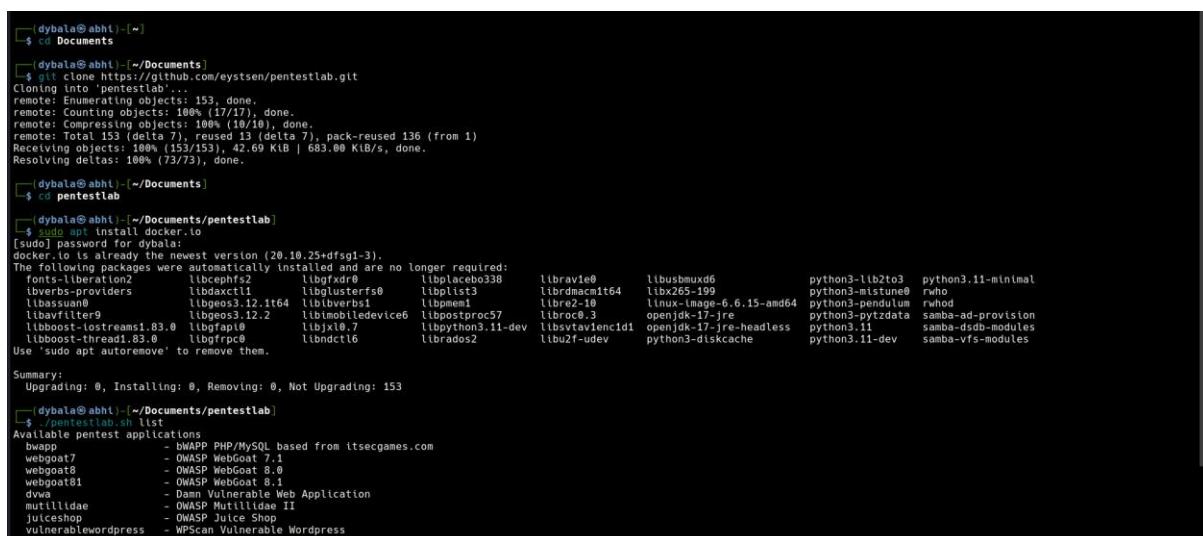
```
git clone https://github.com/eystsen/pentestlab.git
```

## 1.2 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
```



```
(dybala@abht:~)
$ cd Documents
(dybala@abht:~/Documents)
$ 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 | 683.00 KiB/s, done.
Resolving deltas: 100% (73/73), done.
(dybala@abht:~/Documents)
$ cd pentestlab
(dybala@abht:~/Documents/pentestlab)
$ sudo apt install docker.io
[sudo] password for dybala:
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 libcephfs2 libgfs2 libplacebo338 libbravie0 libbusmuxd6 python3-lib2to3 python3.11-minimal
 libverbs-providers libdaxctl1 libglusterfs0 libplist3 librdmacm164 libx265-199 python3-mistune0 rwho
 libassuan0 libgeo3.12.1t64 liblverbs1 libpmem1 libre2-10 linux-image-6.6.15-amd64 python3-pendulum rwho
 libavfilter9 libgeo3.12.2 liblmdb1 libpostproc57 libroc0.3 openjdk-17-jre python3-pytzdata samba-ad-provision
 libboost-test1.83.0 libgflags libjx10.7 libpython3.11-dev libsvtavienc1d1 openjdk-17-jre-headless python3.11 samba-dsdb-modules
 libboost-thread1.83.0 libgflags libndctl6 librados2 libu2f-udev python3-diskcache python3.11-dev samba-vfs-modules
Use 'sudo apt autoremove' to remove them.
Summary:
Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 153
(dybala@abht:~/Documents/pentestlab)
$ ./pentestlab.sh list
Available pentest applications
dwapp - DWAPP PHP/MySQL based from itsecgames.com
webgoat7 - OWASP WebGoat 7.1
webgoat8 - OWASP WebGoat 8.0
webgoat81 - OWASP WebGoat 8.1
dvwa - Damn Vulnerable Web Application
mutillidae - OWASP Mutillidae II
juiceshop - OWASP Juice Shop
vulnerablewordpress - WPScan Vulnerable Wordpress
```

Screenshot 1

## 1.3 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`

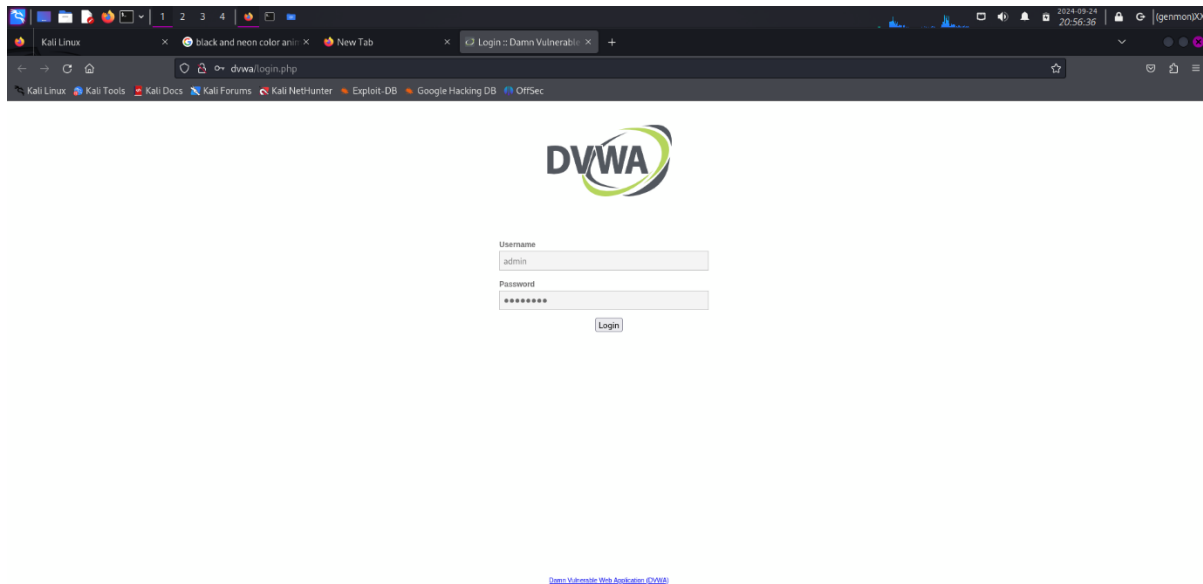
```
dybala@abhl: [~/Documents/pentestlab]
$ ./pentestlab.sh start dvwa
Starting Damn Vulnerable Web Application
Adding dvwa to your /etc/hosts
127.0.0.1 dvwa was added successfully to /etc/hosts
not set
Running command: docker run --name dvwa -d -p 127.0.0.1:80:80 vulnerables/web-dvwa
Unable to find image 'vulnerables/web-dvwa:latest' locally
latest: Pulling from vulnerables/web-dvwa
3e17c6aa66c: Pull complete
0c57df616dbf: Pull complete
eb65d18be401: Pull complete
e9969e5981d2: Pull complete
2cd72dba8257: Pull complete
6cfff5f35147f: Pull complete
098cfff43466: Pull complete
b3d64a3242d: Pull complete
Digest: sha256:dae203fe11646a86937bf04db0079adef295f426da68a92b40e3b181f337daa7
Status: Downloaded newer image for vulnerables/web-dvwa:latest
d98721dfe5d04d51258029791a23b29ac47c61e85dd73bcfe2e5e1f258f2c
DONE!

Docker mapped to http://dvwa or http://127.0.0.1
Default username/password: admin/password
Remember to click on the CREATE DATABASE Button before you start
```

## 1.4 Logging In

At the login page, I used the default credentials:

- Username: admin
- Password: password



Screenshot 3

## 1.5 Resetting the Database

After logging in for the first time, I was prompted to reset the database. I clicked the "Reset Database" button (I missed capturing a screenshot of this step). Once the reset was completed, the system redirected me back to the login page.

## 1.6 Logging In Again

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

## 1.7 Completion

At this point, the DVWA setup was complete, and the environment was ready for vulnerability testing.

## 2. Performing SQL Injection on DVWA

### 2.1 SQL Injection (Low Security Level)

I began by testing SQL injection on the Low security level.

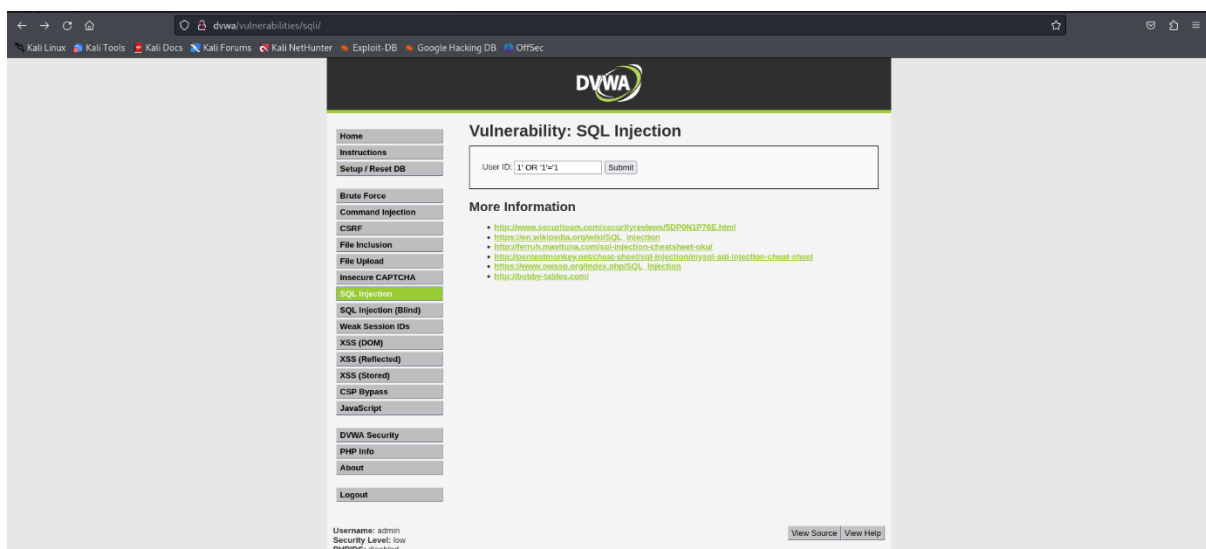
#### 2.1.1 Initial Injection

After accessing the SQL injection page, I quickly identified the input field for injecting SQL code.

#### 2.1.2 SQL Payload

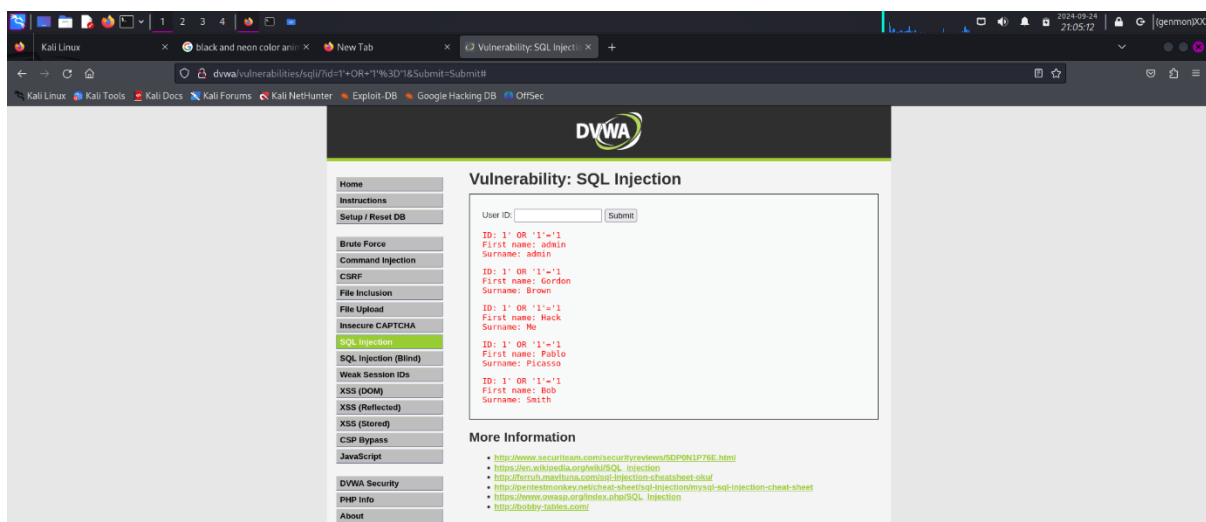
I used the following basic SQL injection string:

**1' OR '1'='1**



Screenshot 4

This payload bypassed the need for valid input and displayed the first name and surname of all users.



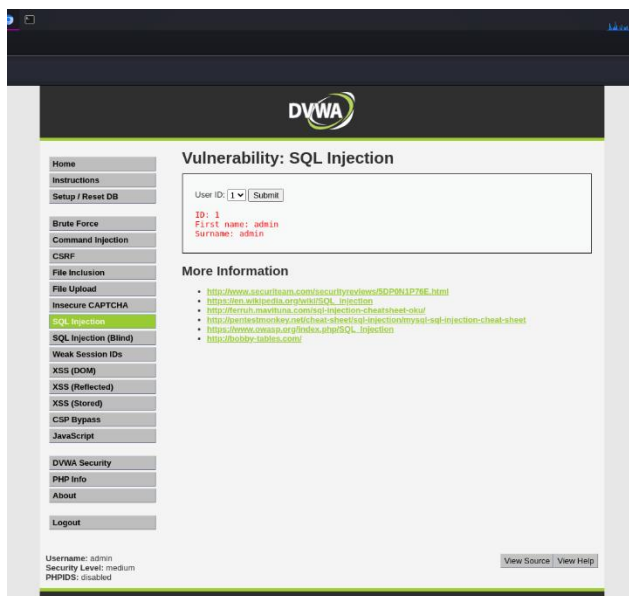
Screenshot 5

## 2.2 SQL Injection (Medium Security Level)

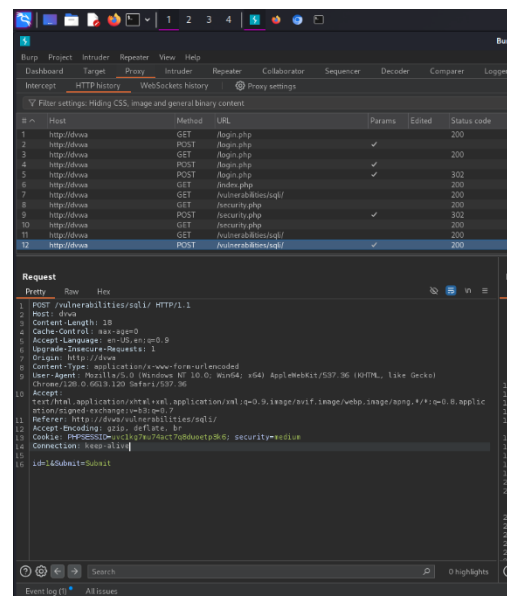
Next, I changed the DVWA security setting to Medium and conducted the test with an enhanced payload.

### 2.2.1 Using Burp Suite

I used Burp Suite to intercept the HTTP request. I modified the `id` parameter in the request to insert a more advanced SQL injection string.



Screenshot 6

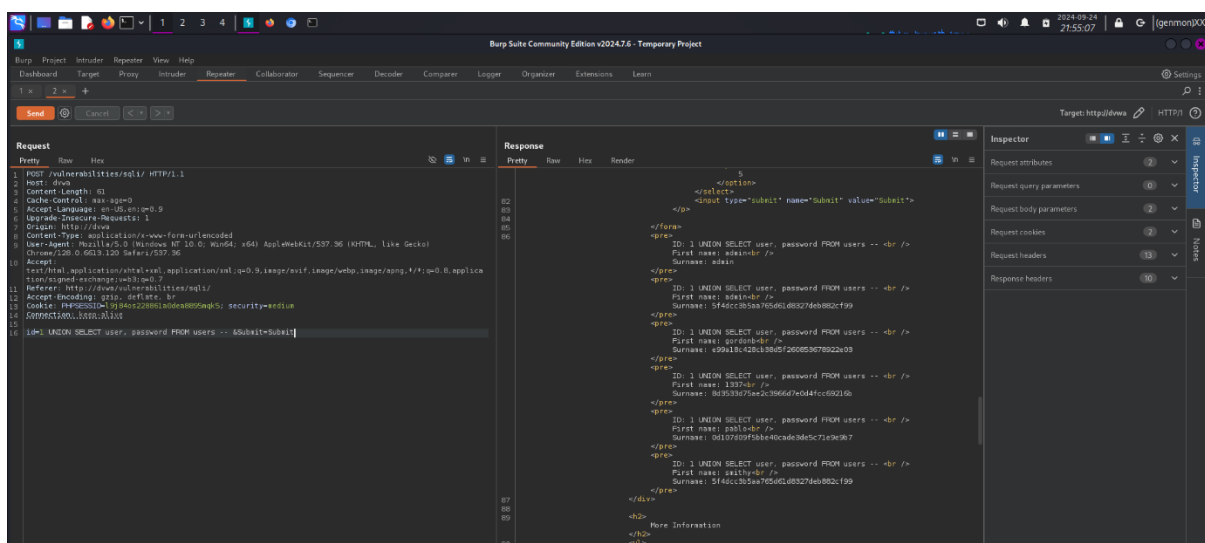


Screenshot 7

### 2.2.2 SQL Injection String

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

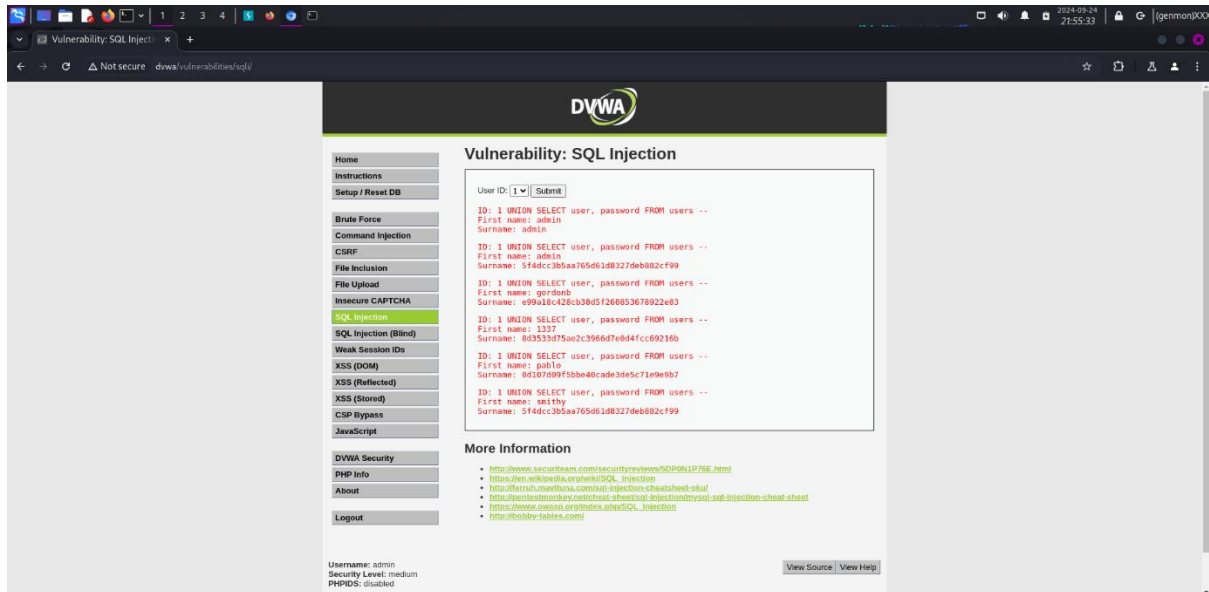
**1 UNION SELECT user, password FROM users --**



Screenshot 6

### 2.2.3 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 (refer screenshot 6 and screenshot 7).



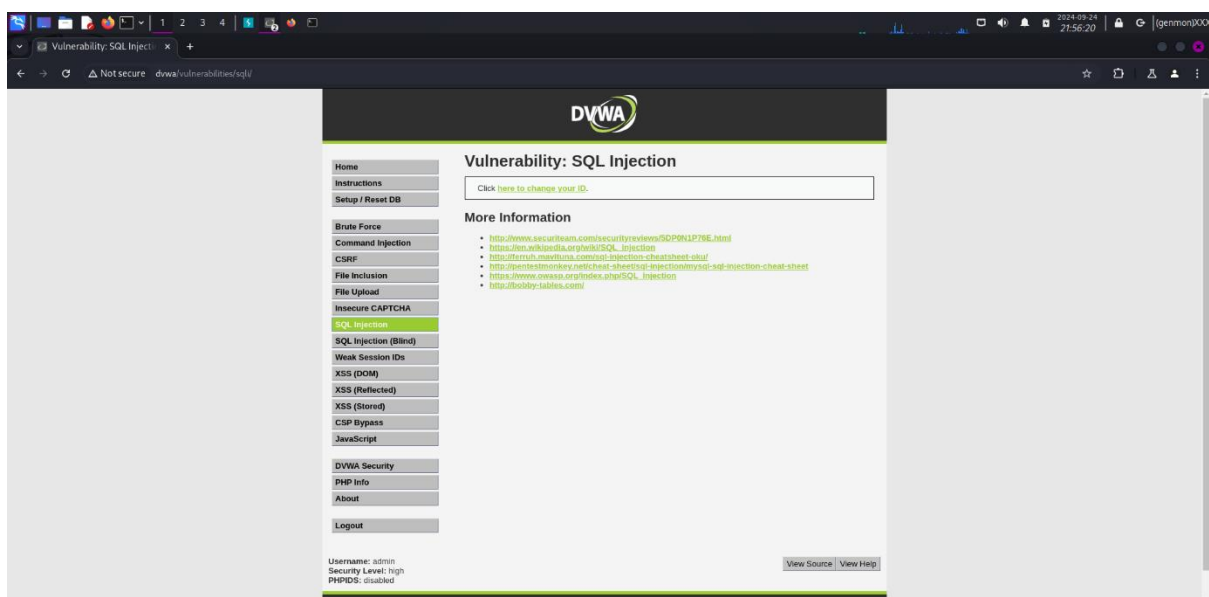
Screenshot 7

## 2.3 SQL Injection (High Security Level)

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

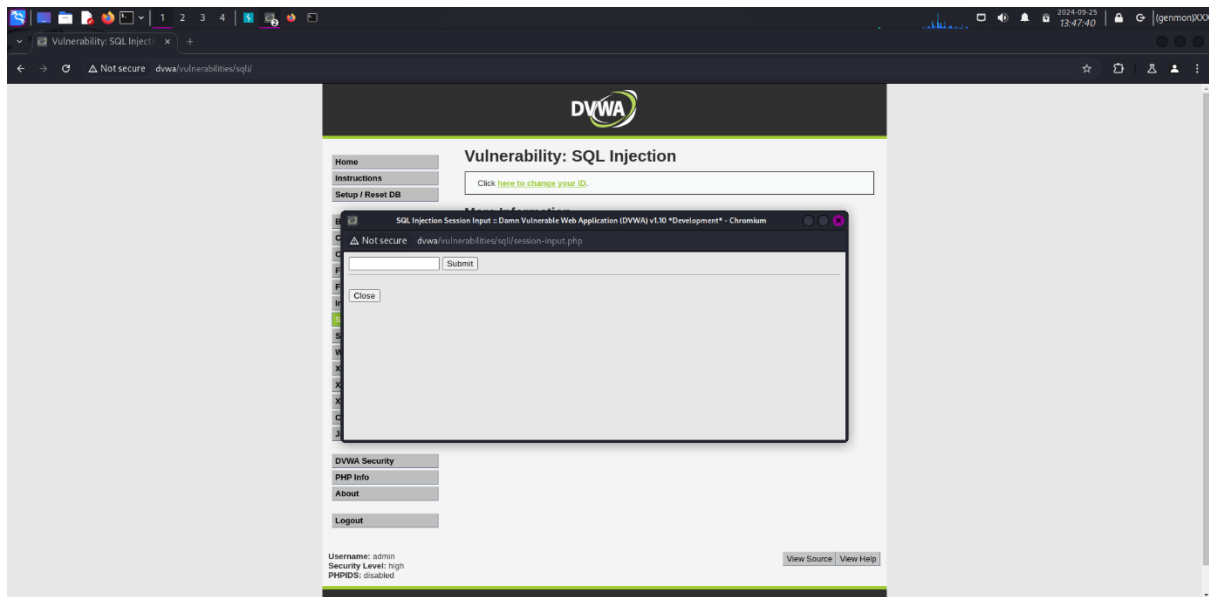
### 2.3.1 Identifying the Injection Point

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



Screenshot 8

a new window appeared where I could input SQL command.

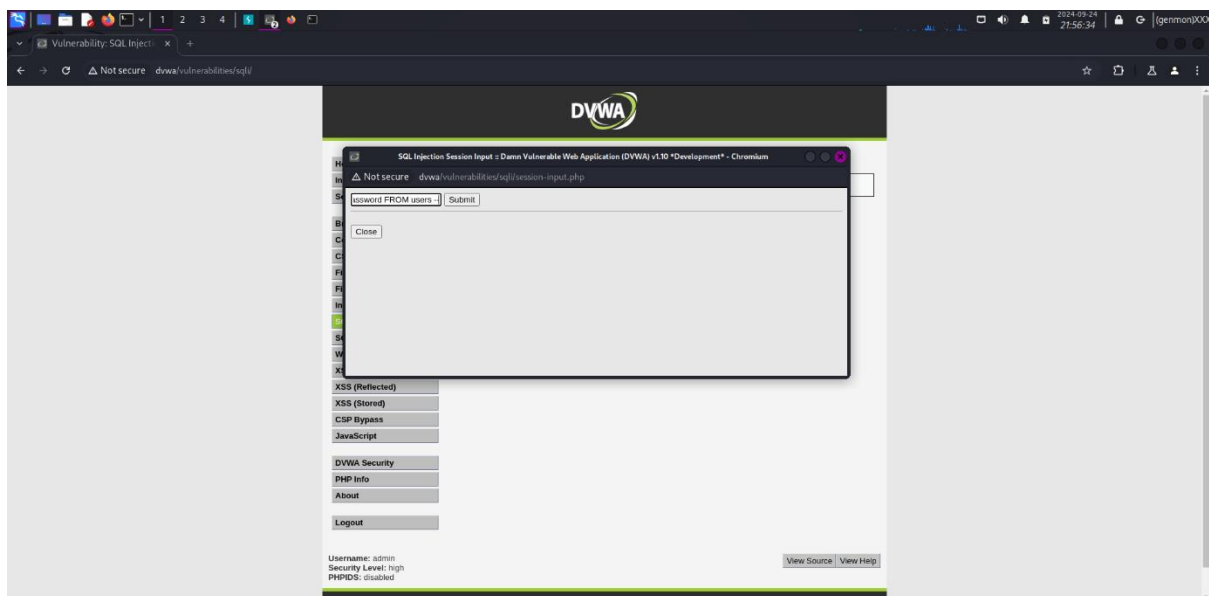


Screenshot 9

### 2.3.2 Injection Payload

I inserted the following SQL injection string:

' UNION SELECT user, password FROM users - -

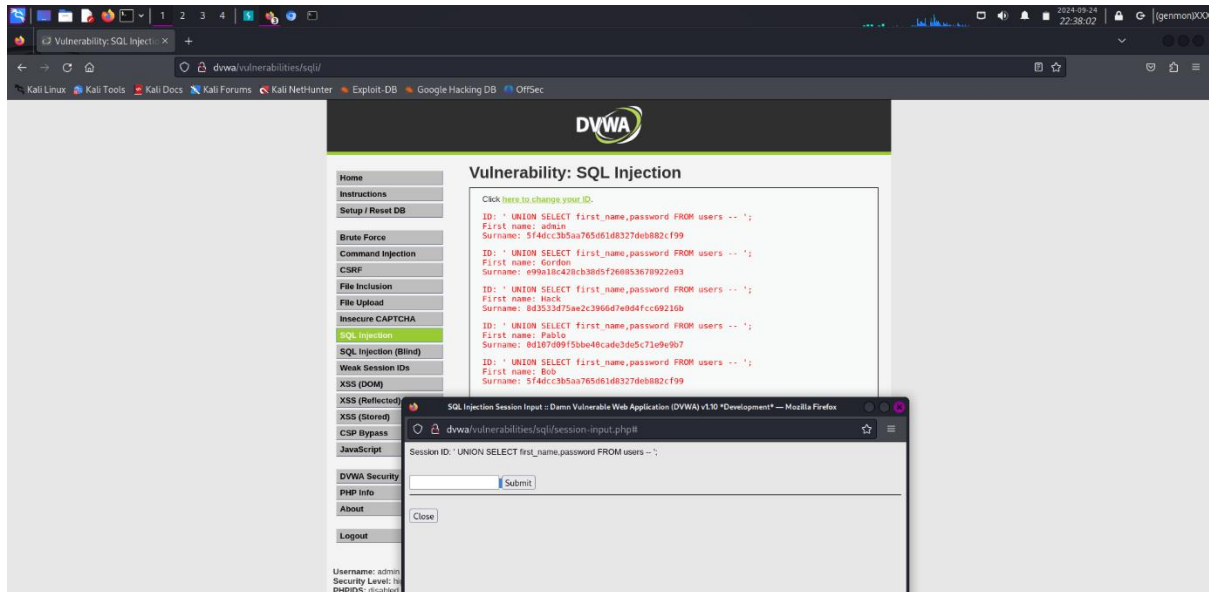


Screenshot 10



### 2.3.3 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 11

## **Conclusion**

I successfully installed DVWA using Docker and tested SQL injection vulnerabilities at different security levels. Using simple and advanced SQL injection payloads, along with Burp Suite for request interception, I was able to extract sensitive information from the database across all security settings, demonstrating the effectiveness of these attacks.