SQL Injection Vulnerability Assessment Using DVWA

SUBMITTED BY:

MUBEENA D

CONTENTS

1.Setting up DVWA with Docker

For installing Damn Vulnerable Web Application (DVWA), I opted for Docker to simplify the process. Here's an overview of the steps I took to complete the setup:

1.1 Repository Cloning

I began by pulling the DVWA repository from pentestlab.github.io using the following:

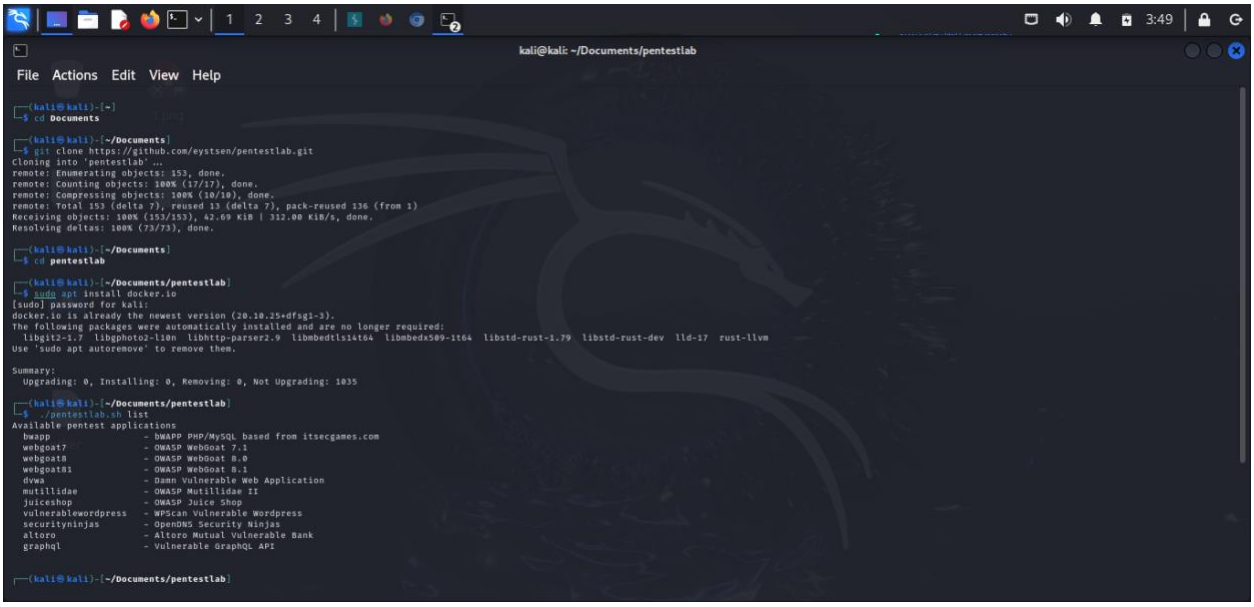
git clone https://github.com/eystsen/pentestlab.git

1.2 Starting the Docker Container:

Once I had cloned the repository, I accessed the DVWA directory and executed Docker commands to start the web application. The detailed steps I took are as follows:

1. Opened a terminal and moved into the cloned pentestlab folder.
2. Used the following command to set up the Docker container:

sudo apt install docker.io



Screenshot 1

1.3Accessing the DVWA Web Interface:

After successfully starting the Docker container, I executed this command to open the DVWA web page.

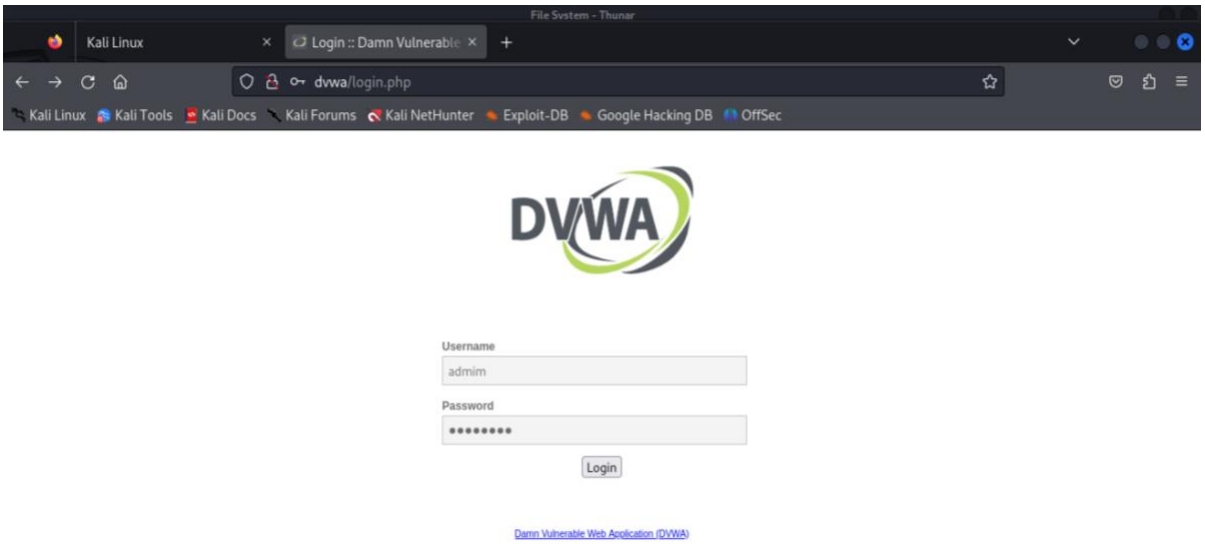
Command: ./pentestlab.sh start dvwa

Screenshot 2

1.4 Logging In

On the login screen, I entered the default credentials:

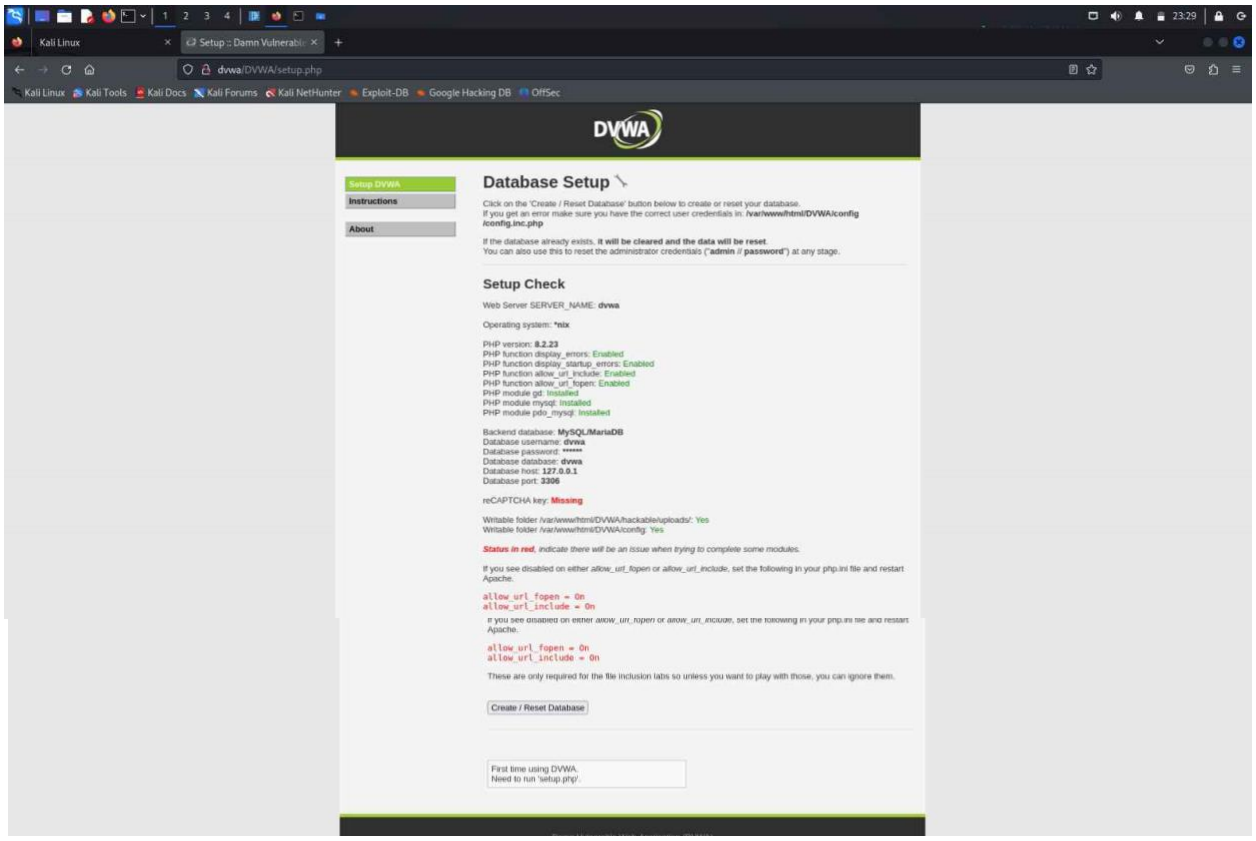
- Username: admin
- Password: password



Screenshot 3

1.5 Resetting the Database:

I was prompted to reset the database,After logging in for the first time.Then i clicked the "Reset Database" button.the system redirected me back to the login page,Once the reset was completed.



Screenshot 4

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 testin

2. Performing SQL Injection on DVWA

2.1 SQL Injection (Low Security Level)

I started by attempting SQL injection on the Low security level.

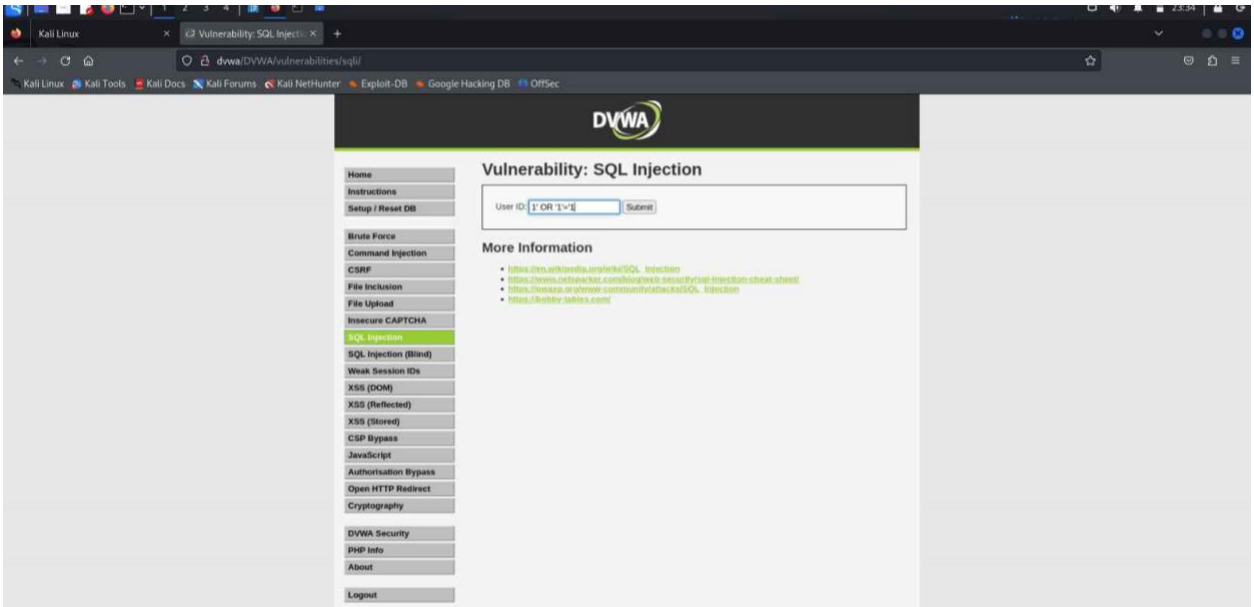
2.1.1 Initial Injection

Upon navigating to the SQL injection page, I promptly located the inputfield for inserting SQL code.

2.1.2 SQL Payload

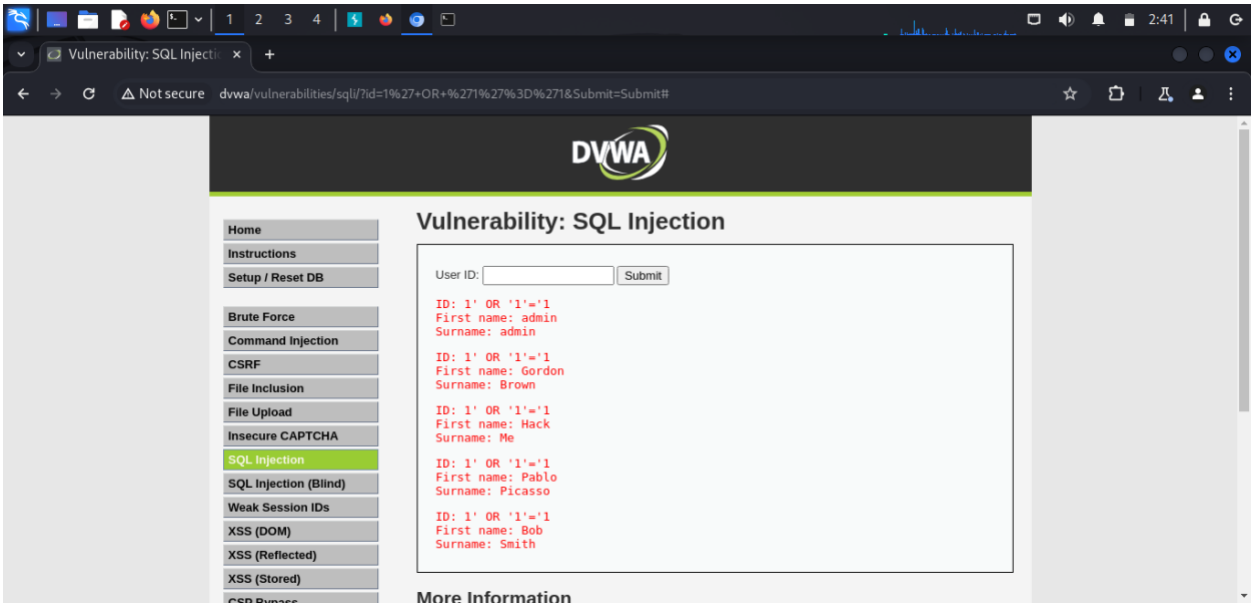
I employed the following simple SQL injection string.

1' OR '1'='1



Screenshot 5

This payload successfully bypassed the requirement for valid input, revealing the first and last names of all users.



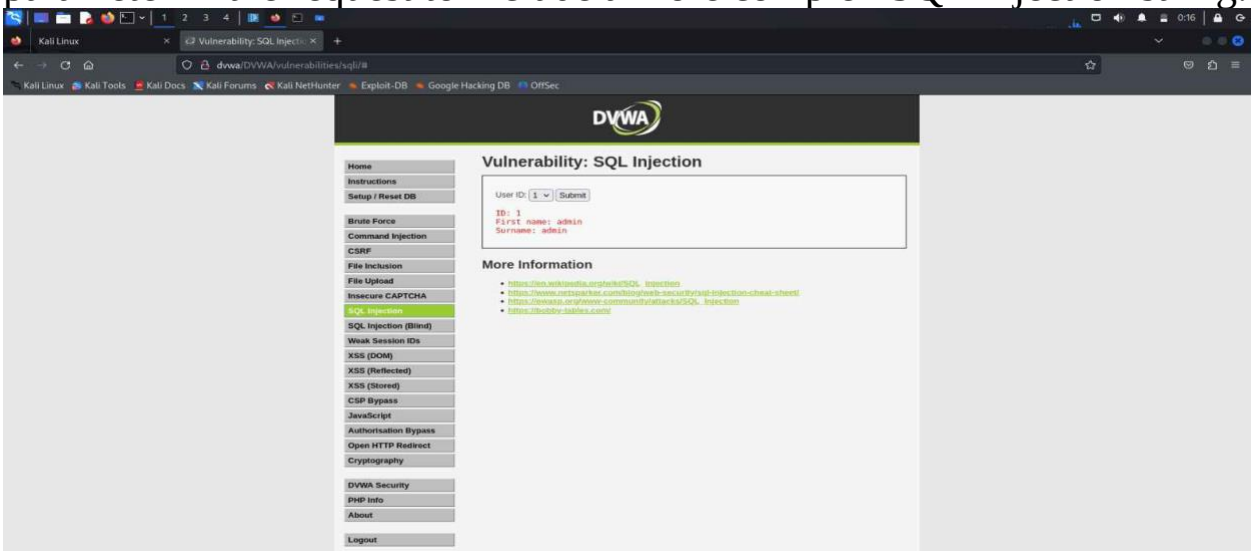
Screenshot 6

2.2 SQL Injection (Medium Security Level)

Next I changed the DVWA security setting to Medium and performed the test using a more sophisticated payload.

2.2.1 Using Burp Suite

I leveraged Burp Suite to intercept the HTTP request and altered the id parameter in the request to include a more complex SQL injection string.

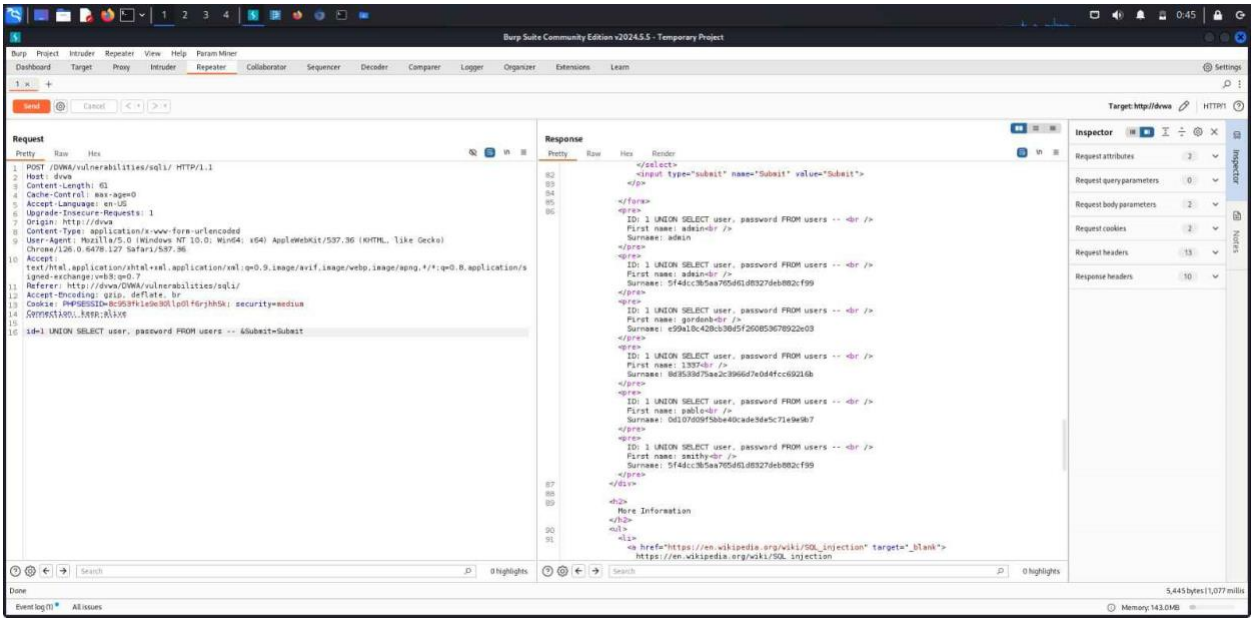


Screenshot 7

2.2.2 SQL Injection String

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

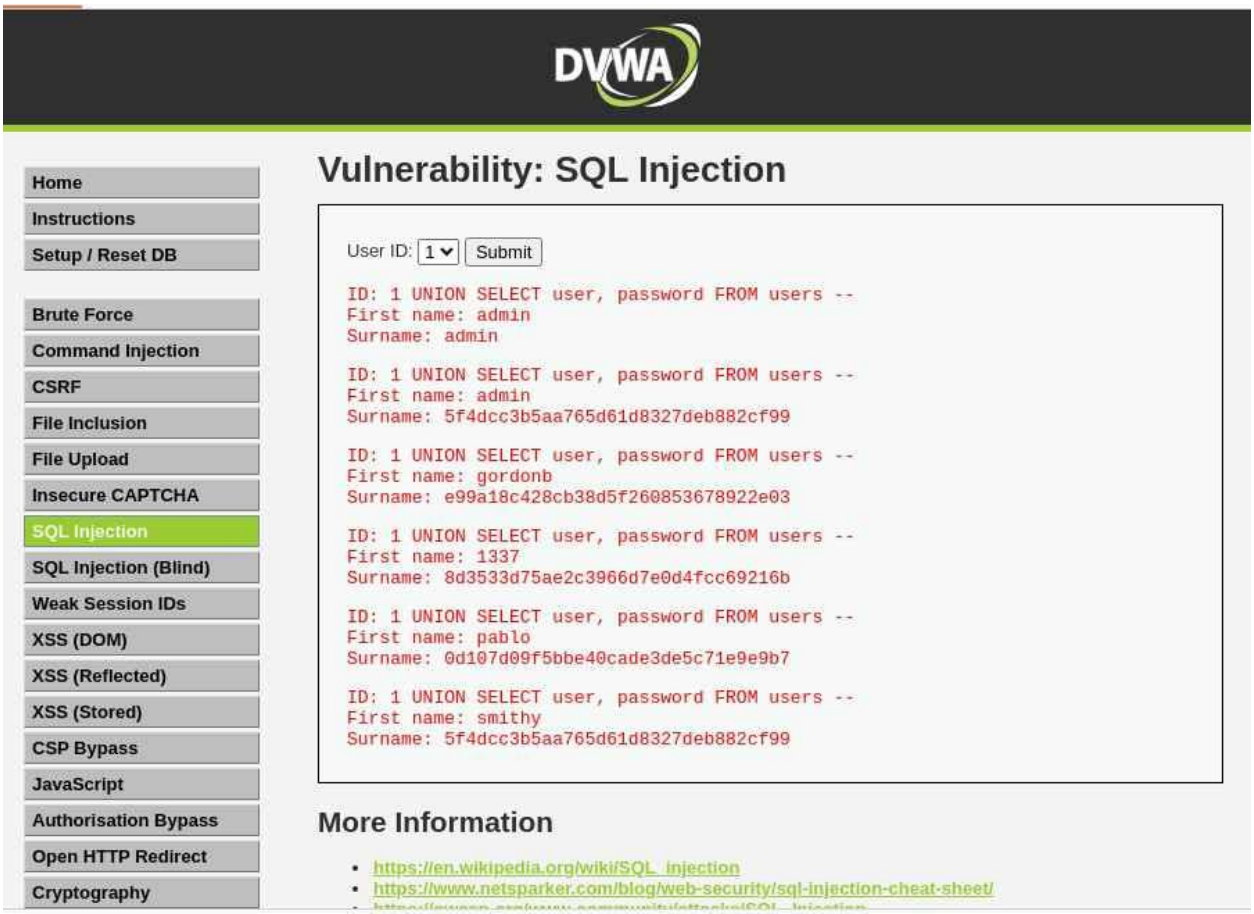
1 UNION SELECT user, password FROM users - -



Screenshot 8

2.2.2 Execution

After modifying the request in Burp Suite.I sent it to the server.As a result,I was able to extract usernames and passwords from the system's response.



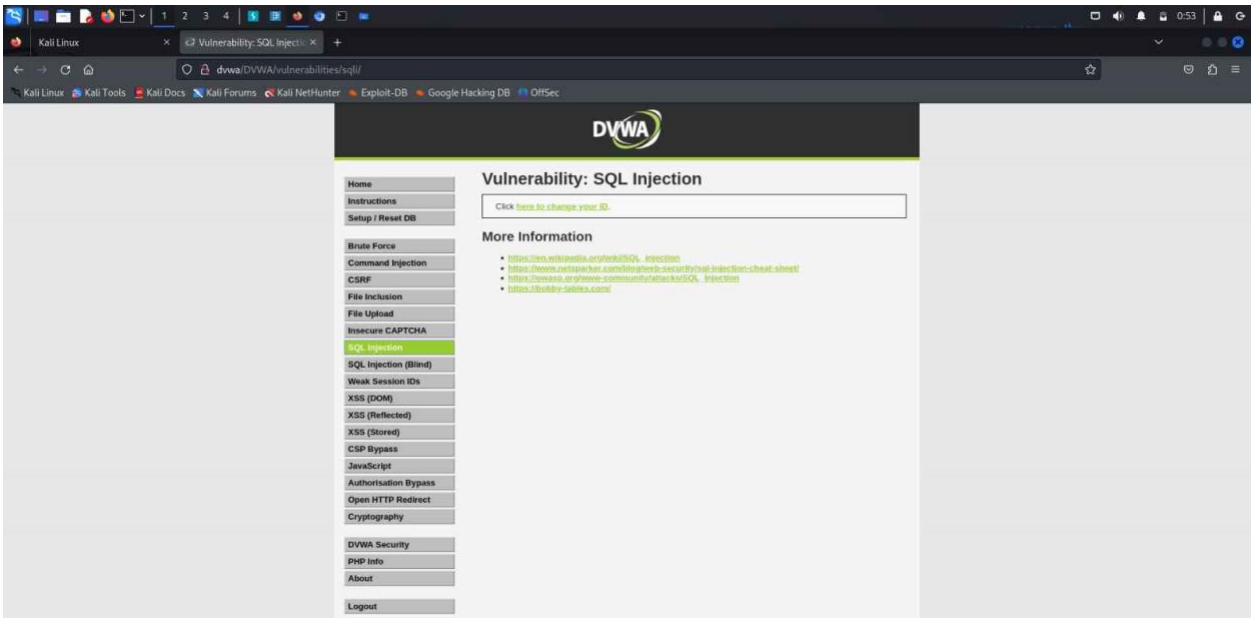
Screenshot 9

2.3 SQL Injection (High Security Level)

Finally, I conducted the SQL injection test on the High security level.

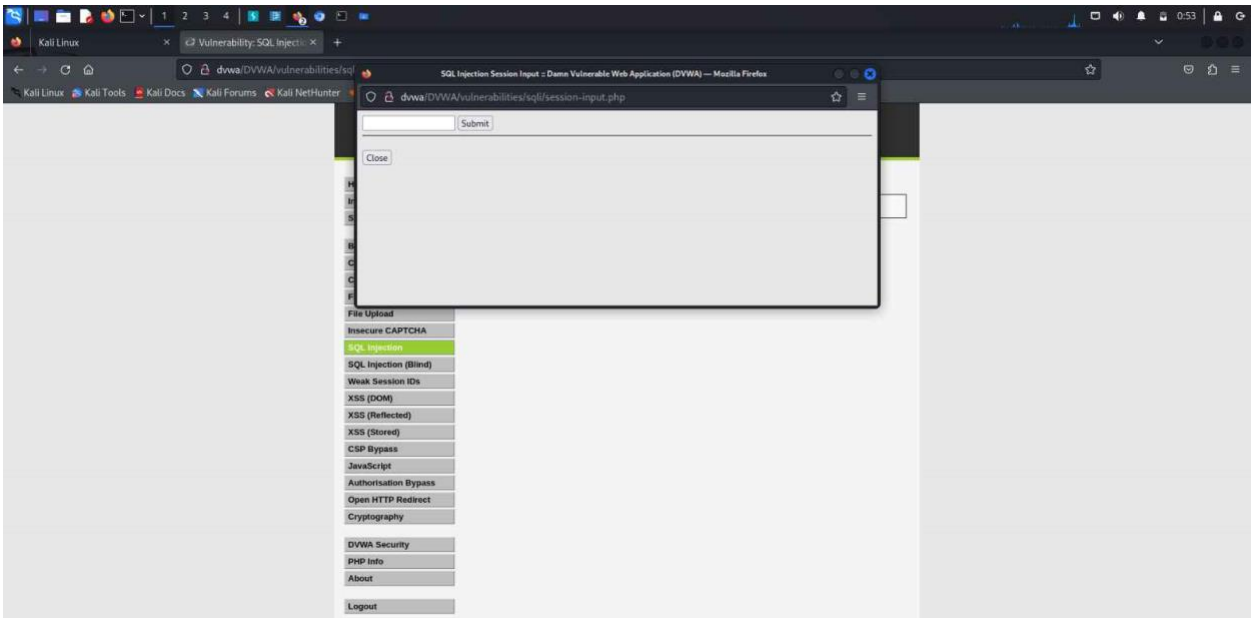
2.3.1 Identifying the Injection Point

Upon clicking the ‘Here to Change your ID’ button, a slight difference in the interface becomes noticeable at the high security level.



Screenshot 10

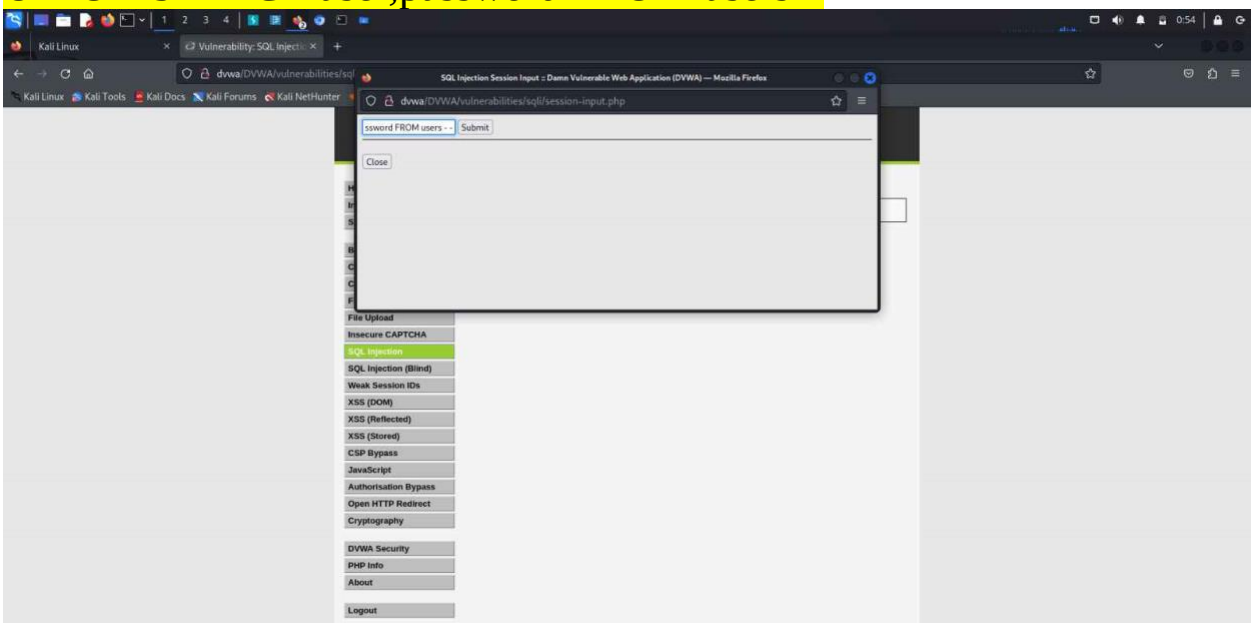
a new window appeared where I could input SQL command



Screenshot 11

2.3.2 Injection Payload I inserted the following SQL injection string:

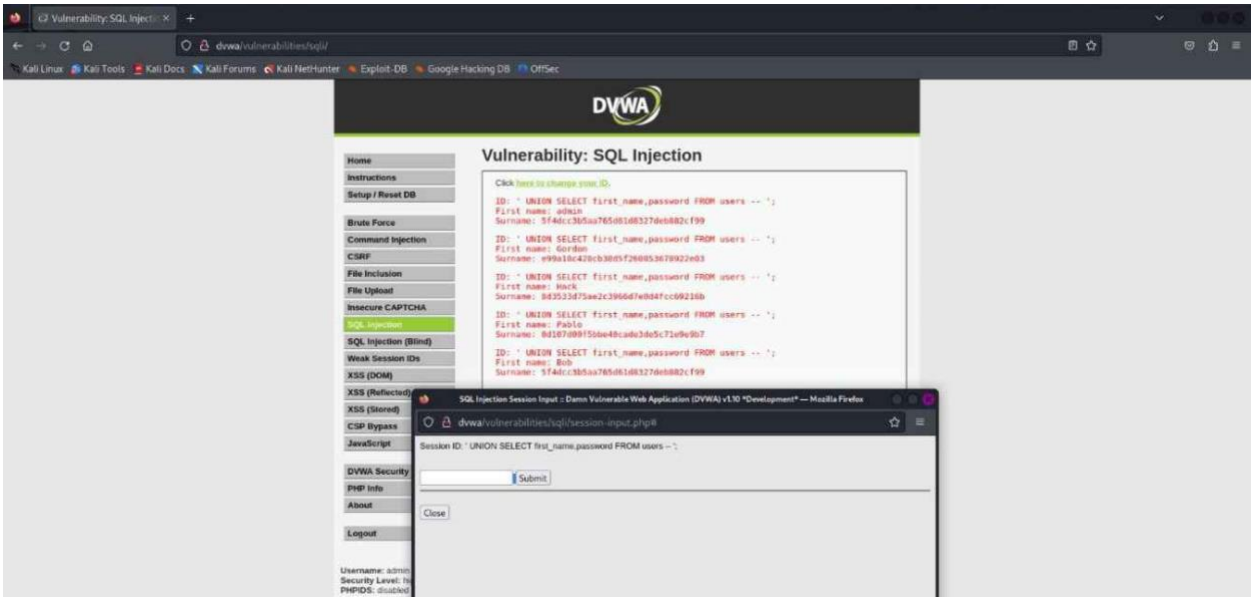
UNION SELECT user,password FROM users –



Screenshot 12

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 13

CONCLUSION

I successfully set up DVWA using Docker and tested SQL injection vulnerabilities at various security levels. By using both basic and complex SQL injection payloads and leveraging Burp Suite to intercept requests, I managed to retrieve sensitive data from the database on all security settings, showcasing the effectiveness of these attacks.