SURE TRUST

MINI PROJECT-4

SQL Injection (low/med/high) - Damn Vulnerable Web Application (DVWA)

GUIDED BY PRESENTED BY

MR HIMANEESH SIR K.SAI JAYANTH

SURE TRUIST ORGANISATION

CYBER SECURITY

Introduction

SQL injection is a common web application vulnerability that occurs when user-supplied data is not properly sanitized before being incorporated into an SQL query. This can allow attackers to inject malicious SQL code into the query, potentially enabling them to read sensitive data, modify data, or even execute commands on the database server.

The Damn Vulnerable Web Application (DVWA) is a web application designed to be intentionally vulnerable to a variety of attacks, including SQL injection. It is a valuable tool for security professionals to practice identifying, exploiting, and mitigating SQL injection vulnerabilities.

Process of DVWA Using XAMPP

Step 1: Install XAMPP

1. Download the XAMPP installation package from the official website (<https://www.apachefriends.org/download.html>)
2. Run the installer and follow the on-screen instructions.
3. Once XAMPP is installed, start the Apache and MySQL services.

Step 2: Download and Install DVWA

1. Download the DVWA package from the official website (<https://sourceforge.net/projects/dvwa.mirror/>)
2. Extract the DVWA package to your XAMPP installation directory.
3. Open the DVWA configuration file (config.inc.php) in a text editor.
4. Update the database connection settings to match your MySQL database.
5. Save the changes to the configuration file.

Step 3: Access DVWA

1. Open a web browser and go to <https://www.ionos.com/digitalguide/server/know-how/localhost/>
2. You should see the DVWA login page.
3. Use the default credentials (username: admin, password: password) to log in.

Example SQL Injection Scenarios

Scenario 1: Low-level SQL Injection

Vulnerability: A web application displays a user-submitted product name on a search results page.

Attack: Submit a search term containing a single quote (') to see if it is properly escaped.

' OR 1=1--

Expected Result: The search results page should display all products, regardless of the search term.

Scenario 2: Medium-level SQL Injection

Vulnerability: A web application allows users to register with a username and password.

Attack: Submit a registration request with a username containing a single quote (') to determine if user input is properly sanitized.

admin' OR 1=1--

Expected Result: The login page should display an error message indicating that the username is invalid.

Scenario 3: High-level SQL Injection

Vulnerability: A web application displays user-submitted comments on a blog post.

Attack: Submit a comment containing JavaScript code to see if it is executed.

<script>alert('XSS');</script>

Expected Result: When the blog post is viewed, an alert box should appear with the text "XSS."

Scenario 4: Blind SQL Injection

Vulnerability: A web application displays a list of users based on a user-supplied ID.

Attack: Submit a user ID with a single quote (') and observe the error message. The error message may reveal information about the database structure or the contents of specific tables.

' OR username LIKE '%a%'--

Expected Result: The application should display an error message indicating that the user ID is invalid. The error message may also reveal information about the database structure or the contents of specific tables.

Scenario 5: Out-of-band SQL Injection

Vulnerability: A web application sends an email notification when a new user registers.

Attack: Submit a registration request with an email address containing a single quote (') and observe if the email contains the injected code. The injected code could be used to send a malicious email to a different recipient.

admin' OR 1=1-- AND email='attacker@example.com'

Expected Result: If the application is vulnerable to out-of-band SQL injection, the attacker will receive an email containing the injected code. The attacker can then modify the code to send a malicious email to a different recipient.

``````Scenario 6: Union-based SQL Injection

Vulnerability: A web application displays a list of products based on a user-supplied search query.

Attack: Submit a search query containing a UNION statement to retrieve additional data from the database.

SELECT \* FROM products WHERE product\_name LIKE '%{}%';

Expected Result: The search results page should display all products, as well as any additional data retrieved using the UNION statement.

Scenario 6: Union-based SQL Injection

Vulnerability: A web application displays a list of products based on a user-supplied search query.

Attack: Submit a search query containing a UNION statement to retrieve additional data from the database.

SELECT \* FROM products WHERE product\_name LIKE '%{}%';

Expected Result: The search results page should display all products, as well as any additional data retrieved using the UNION statement.