Lab Manual 4 : SQL Injection Simulation

Objective

To understand how **SQL Injection** works in a vulnerable application and how attackers can bypass login authentication by injecting malicious SQL code.

Background

- SQL Injection is one of the **most common web application vulnerabilities**.
- It occurs when user input is directly inserted into a SQL query without proper validation or sanitization.
- Attackers can manipulate the query to bypass authentication or extract sensitive data.

Lab Setup

- A **mock database** with a users table will be created (using SQLite in Colab).
- The table will have sample users (e.g., admin, john, alice).
- A **vulnerable login function** will be provided, where user input is directly concatenated into the SQL query.

Lab Tasks

Task 1: Normal Login

1. Run the login function with correct credentials (e.g., username john, password john123).
2. Observe that login is successful.
3. Try with a wrong password and note that login fails.

Task 2: SQL Injection Attempt
Try logging in with the following payload in the **username field**:

' OR '1'='1' --

1. (Password can be anything.)

Observe that the query becomes:

SELECT * FROM users WHERE username = '' OR '1'='1' --' AND password='anything'

2.
3. Notice how the -- comments out the password condition.
   Since '1'='1' is always **true**, login is bypassed.

Task 3: Password Field Injection

1. Use a valid username (e.g., john).

In the password field, try:

' OR '1'='1' --

2.
3. Observe how this also bypasses authentication.

Task 4: Compare with Secure Version

1. Run the **safe login function**, which uses parameterized queries.

2. Try the same injection payloads.

3. Observe that the attack **fails** because input is treated as data, not code.

Observations

- **Without protection:** Injection succeeds and bypasses authentication.
- **With parameterized queries:** Injection fails, proving the system is safe.

Learning Outcomes

By completing this lab, students will:

1. Understand how SQL Injection exploits poorly written queries.
2. Learn why certain payloads (' OR '1'='1') work only with comments (--).
3. Recognize the importance of **prepared statements / parameterized queries** in preventing SQL Injection.