Day 19

SQL Injection - Day 1 Report

Introduction

SQL Injection (SQLi) is a critical web security vulnerability that allows an attacker to interfere with the queries an application makes to its database. It is one of the most common and dangerous types of attacks against web applications, primarily because it can lead to unauthorized access to sensitive data, database manipulation, and even the complete compromise of the affected system.

SQL Injection attacks exploit vulnerabilities in an application's software by injecting malicious SQL statements into an entry field for execution. This type of attack can occur in any application that processes user-supplied data without proper validation and sanitization, regardless of the programming language or database management system (DBMS) in use.

How SQL Injection Works

The fundamental concept of SQL Injection involves the manipulation of SQL queries by injecting SQL code into input fields that are included in database queries. Here's a simplified example to illustrate this:

Consider a web application with a login form that accepts a username and password. The application processes these inputs to authenticate users by querying the database as follows:

```
<SELECT * FROM users WHERE username = 'input_username' AND password =
'input_password';>
```

An attacker can input malicious data to alter the intended query. For example, if the attacker enters ' OR '1'='1' -- in the username field and leaves the password field empty, the query becomes:

```
<SELECT * FROM users WHERE username = '' OR '1'='1' --' AND password = '';>
```

The -- symbol indicates a comment in SQL, causing the rest of the query to be ignored. Thus, the query effectively becomes:

```
<SELECT * FROM users WHERE username = '' OR '1'='1';>
```

Since 1=1 is always true, the query returns all rows in the users table, potentially allowing the attacker to bypass authentication.

Types of SQL Injection

1. **In-Band SQLi (Classic SQLi)**: This type of SQL injection occurs when the attacker uses the same communication channel to both launch the attack and gather results. It includes:
    - **Error-Based SQLi**: The attacker intentionally causes SQL errors to retrieve information from the database.
    - **Union-Based SQLi**: The attacker uses the UNION SQL operator to combine the results of two or more SELECT queries to retrieve data.
2. **Inferential SQLi (Blind SQLi)**: In this method, the attacker sends queries to the database and observes the behavior of the application to infer information, without seeing the actual database output.

    - **Boolean-Based Blind SQLi**: The attacker sends queries that return different results depending on whether the query is true or false.
    - **Time-Based Blind SQLi**: The attacker uses queries that cause a time delay if the condition is true, allowing inference based on response time.
3. **Out-of-Band SQLi**: This type of SQL injection involves sending the data through different channels, such as an email or a DNS query, often used when in-band methods are not feasible.

Impact of SQL Injection

The consequences of a successful SQL injection attack can be devastating:

1. **Unauthorized Access to Data**: Attackers can retrieve sensitive information such as usernames, passwords, personal data, and financial records.
2. **Data Manipulation**: Attackers can insert, update, or delete database records, leading to data loss or corruption.
3. **Authentication Bypass**: Attackers can gain unauthorized access to systems by bypassing authentication mechanisms.
4. **Compromise of the Entire System**: In severe cases, attackers can execute arbitrary commands on the database server, leading to the full compromise of the underlying operating system.
5. **Loss of Confidentiality and Integrity**: The confidentiality and integrity of the data can be severely compromised, leading to legal and regulatory consequences.

Prevention and Mitigation Strategies

1. **Input Validation**: Ensure that all user inputs are validated and sanitized. Use a whitelist approach to allow only expected input formats and reject any input that does not conform to these expectations.
2. **Parameterized Queries**: Use prepared statements and parameterized queries to ensure that SQL code is separated from data. This prevents attackers from altering the intended SQL commands.

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