PROJECT

SQL injection attack on today's technology using mysql database queries

Prepred by

Nikita Bacha

Guided by

Zakir hussain

INDEX

SQL Injection

Introduction

The World Wide Web has experienced remarkable growth in recent years. Businesses, individuals, and governments have found that web applications can offer effective, efficient and reliable solutions to the challenges of communicating and conducting commerce in the Twenty-first century.

However, the security of Web applications has become increasingly important in the last decade. With more and more Web-based applications deal with sensitive financial and medical data, it is crucial to protect these applications from hacker attacks. A security assessment by the Application Defence Centre, which included more than 250 Web applications from e-commerce, online banking, enterprise collaboration, and supply chain management sites, concluded that at least 92% of Web applications are vulnerable to some form of attack.

Much vulnerability in web applications is caused by permitting unchecked input to take control of the application, which an attacker will turn to unexpected purposes. SQL Injection is the most common type of technique used.

Beside SQL Injection the other type of attacks are:·

- Shell injection.
- Scripting language injection.
- File inclusion.
- XML injection.
- SQL Injection·
- XPath injection.
- LDAP injection.
- SMTP injection

What is SQL Injection ?

- **SQLi** or SQL **Injection** is a web page vulnerability that lets an attacker make queries with the database.
- Attackers take advantage of web application vulnerability and inject an SQL command via the input from users to the application.
- Attackers can SQL queries like SELECT to retrieve confidential information which otherwise wouldn't be visible.
- SQL injection also lets the attacker to perform a **denial-of-service (DoS)** attacks by overloading the server requests.

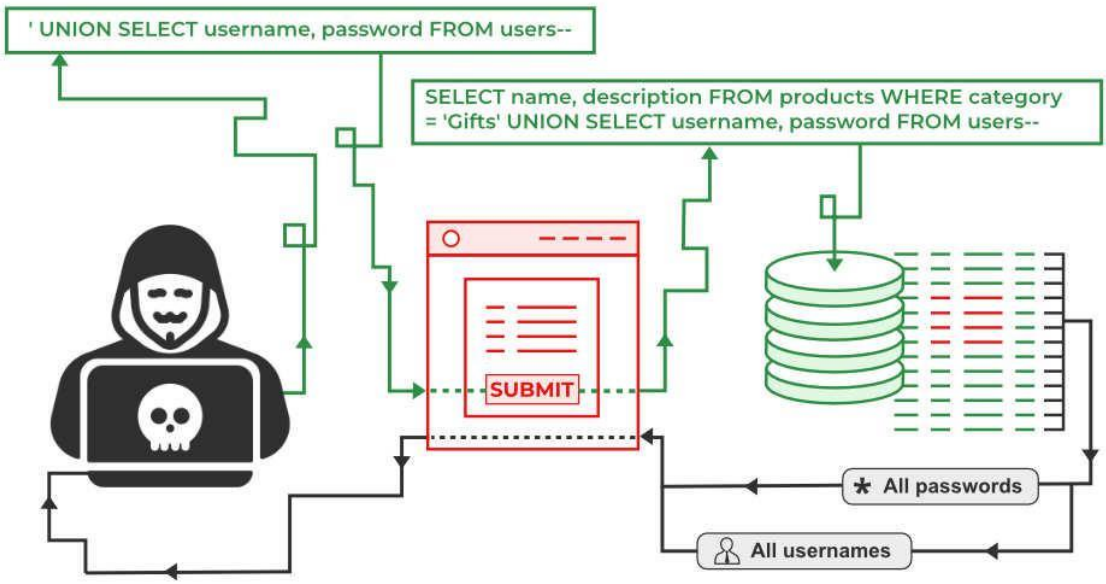


Figure 1 : SQL Injection

What is the impact of a successful SQL injection attack?

- A successful SQL injection attack can have severe consequences, including unauthorized access to sensitive data, such as personal information and financial records.
- Attackers may manipulate or delete critical data, compromising its integrity and causing operational disruptions.
- They can also bypass authentication mechanisms, gaining unauthorized access to user accounts, including administrative privileges.
- This can lead to the exposure of confidential information, identity theft, and significant financial losses.
- Additionally, SQL injection attacks can result in service downtime and damage to the organization's reputation.

How to Detect SQL injection Vulnerabilities?

- To detect SQL injection vulnerabilities, you can start by performing input validation testing, where special characters like ' or " are inserted into inputs to see if they cause errors.
- Automated tools like SQLMap or Burp Suite can scan for vulnerabilities by simulating attacks.
- Reviewing the source code helps identify insecure practices, such as using dynamic SQL queries without proper parameterization.
- Monitoring for unexpected database error messages can reveal potential issues.
- Finally, conducting thorough penetration testing, including both black-box and white-box methods, provides a comprehensive assessment of security weaknesses.

WORKING

Working of SQL Injection:-

The principles behind a SQL injection are simple and these types of attacks are easy to execute and master. To exploit a SQL injection flaw, the attacker must find a parameter that the web application passes through to a database. By carefully embedding malicious SQL commands into the content of the parameter, the attacker can trick the web application into forwarding a malicious query to the database.

For example, consider the login form which accepts the username and password from the user

Figure 2 : Injecting SQL

The values supplied in the field “Username” and “Password” are directly used to build the SQL Query like:

SELECT * FROM customers
WHERE name = ‘ & name & ’ AND password = ‘ & password’

Now, Suppose the user supplied the Username =”Admin” and Password = ‘Magic’

The query will become:

SELECT * FROM customers
WHERE name = 'Admin' AND password = 'magic'

This will work with no problem. But suppose the user supplied some poorly devised string of code then that will allow the attacker to by-pass the authentication and access the information from the database. i.e., if user supplied username=' OR 1=1— then the query will be passed as:

SELECT * FROM customers
WHERE name = '' OR 1=1 --' AND password = ' ';

It Works as follows:
': Closes the user input field.
OR : Continues the SQL query so that the process should equal to what comes before OR what comes after.
1=1: A statement which is always true.
--: Comments outs the rest of the lines so that it won't be processed.

The data we're filling is used by the WHERE clause. And because the application is not really thinking about the query - merely constructing a string -our use of OR has turned a single-component WHERE clause into a two-component one and the 1=1 clause are guaranteed to be true no matter what the first clause is. The query means that:

"Select everything from the table customers if the name equals "nothing" Or 1=1. Ignore anything that follows on this line.

Seeing as 1 will always equal 1, the server has received a true statement and is fooled into allowing an attacker more access than they should have. The code that refers to the password input field is never run by the server and therefore does not apply.

SQL Injection Types

There are different types of SQL injection attacks:

1. In-band SQL Injection

- It involves sending malicious SQL queries directly through the web application's interface.
- It allows attackers to extract sensitive information or modify the database itself.

2. Error-based SQL Injection

- Attackers exploit error messages generated by the web application by analyzing error messages to gain access to confidential data or modify the database.

3. Blind SQL Injection

- Attackers send malicious SQL queries and observe the application's response.
- By analyzing the application's behavior, attackers can determine the success of the query.

4. Out-of-band SQL Injection

- Uses a different channel to communicate with the database.
- Allows attackers to exfiltrate sensitive data from the database.

5. Inference-based SQL Injection

- Uses statistical inference to gain access to confidential data.
- Attackers create queries that return the same result regardless of input values.

Real-Time SQL Injection Attack Example Using MySQL

- To demonstrate the risk and impact of SQL Injection, let's go through an example
  scenario using MySQL. Assume we are working on a web application with a login form that allows users to enter their username and password to access their accounts. The SQL query used in the backend might look something like this:

- **SELECT * FROM users WHERE username = '$username' AND password = '$password';**

- If the web application doesn't properly sanitize the input, an attacker can exploit this by inputting the following into the username field:

- admin' –

- The query now becomes:

- **SELECT * FROM users WHERE username = 'admin' -- ' AND password = '';**

- This modified query will bypass the password check, granting the attacker access as the "admin" user.

- Next, we will explore a real-time scenario where we create a database structure and insert data to perform SQL Injection in a controlled environment. Using MySQL Workbench, we will test these queries to demonstrate the effectiveness of the attack and potential solutions for mitigating it.

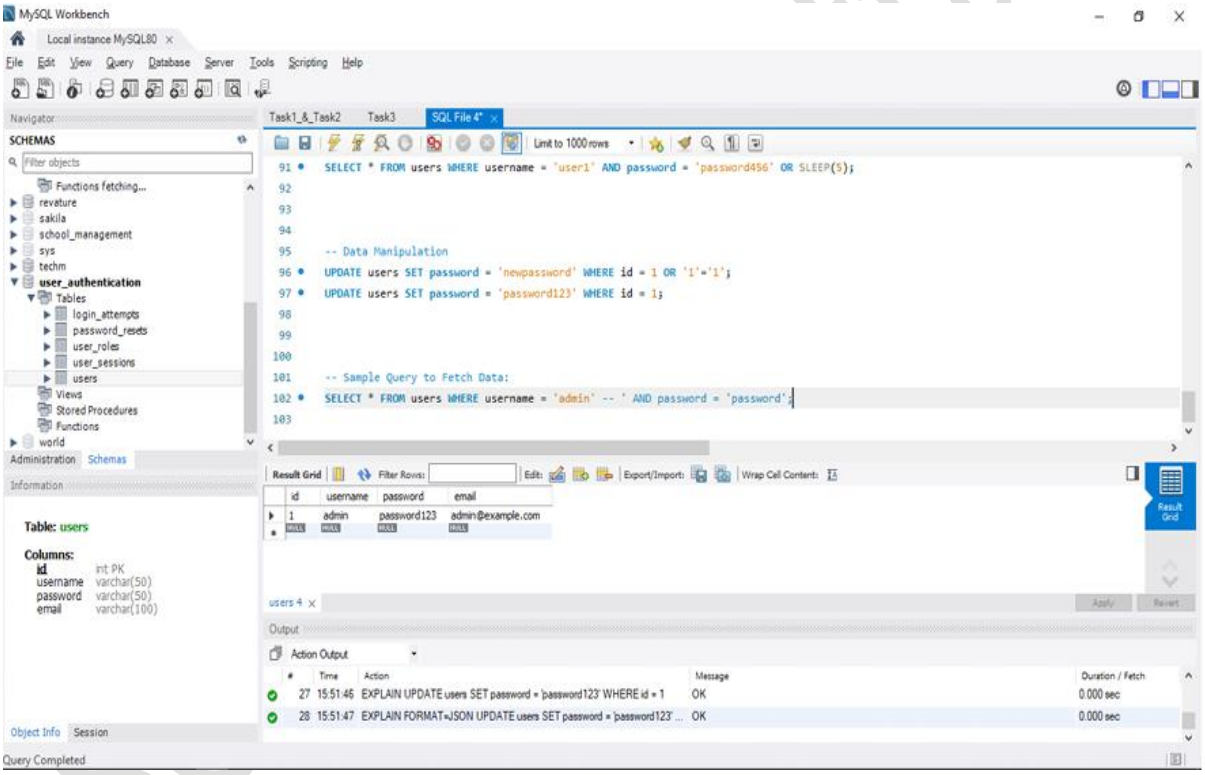
- You can create databases, tables, and insert the following data:

➢ Database: User_Authentication

➢ Tables: users, user_roles, login_attempts, user_sessions, password_resets.

- Sample Query to Fetch Data:

- SELECT * FROM users WHERE username = 'admin' -- ' AND password = 'password';



CONCLUSION

In conclusion, SQL injection remains a prevalent and dangerous vulnerability in modern web applications, particularly those utilizing MySQL databases. This project has demonstrated how easily attackers can exploit poorly secured systems to access sensitive data or manipulate database records.

Through various scenarios, we observed that even minor lapses in input validation can lead to significant breaches, underscoring the critical need for robust security measures. Employing best practices such as prepared statements, input sanitization, and comprehensive security audits is essential for safeguarding applications against such attacks.

As technology evolves, so do the techniques used by attackers, making it imperative for developers and organizations to stay informed and proactive in their approach to database security. Continuous education and implementation of advanced security protocols will be vital in defending against SQL injection threats in the future.