SQL INJECTION ATTACK LAB

By

HEMACHANDU NAIDU B V

RA2011030010200

ABSTRACT

In modern days, cyber threats and attacks are triggered to corrupt or steal the information of a person in huge volume of data from different lines of businesses. Across the globe, nowadays it became mandatory to protect the database from security related attacks. SQL injection is a familiar and most vulnerable threat which may exploit the entire database of any organization irrespective whether it is a private organization or a government sector, where code is injected in a web page. This code injection technique is used to attack data-driven web applications or applications. A SQL statement will be altered in such a manner, which goes with ALWAYS TRUE as constraint. This study paper is prepared to give a comprehensive coverage about topics like basics of SQL Injection, types, recent attacks as a case study. This survey will not be complete, if we miss out to learn the algorithms, being used as a base to trigger vulnerability in this internet connected world; which in turn exploits the database and exposes top secrets. Tautology SQL injection – one of the code injection techniques is widely used as a data – driven attack as per the security related literatures and causes severe damage to the organizational data banks.

KEYWORDS –

- SQL Injection
- Tautology
- Security
- Detection
- Prevention

INTRODUCTION

Web applications are everywhere on the Internet. Almost everything you do online is done through a web application whether you know it or not. They come in the form of web-based email, forums, bulletin boards, bill payment, recruitment systems, health benefit and payroll systems. It is important to understand that these types of websites are all databases driven. Databases are an essential element of web applications because they are able to store user preferences, personal identifiable information, and other sensitive user information Web applications interact with databases to dynamically build customized content for each user. The web application communicates with the database using Structured Query Language (SQL). SQL is a programming language for managing databases that allows you to read and manipulate data in MySQL, SQL Server, Access, Oracle, DB2, and other database systems. The relationship between the web application and the database is commonly abused by attackers through SQL injection. SQL injection is a type of injection attack in which SQL commands are supplied in user-input variables, such as a web form entry field, in an attempt to trick the web application into executing the attacker's code on the database. SQL injection was one of the primary attack vectors responsible for many of 2011’s high profile compromises including Sony Pictures, and PBS. It was also responsible for the more recent Adobe data breach in which names, email addresses, and password hashes were stolen from one of their customer databases. SQL injection is a dangerous vulnerability that is easily detected and inexpensive to fix. This method of attack has been employed by hackers for over ten years, yet it is still the most common attack vector in data breaches today.

OVERVIEW

SQL injection is a code injection technique that exploits the vulnerabilities in the interface between web applications and database servers. The vulnerability is present when user's inputs are not correctly checked within the web applications before being sent to the back-end database servers. Many web applications take inputs from users, and then use these inputs to construct SQL queries, so the web applications can get information from the database. Web applications also use SQL queries to store information in the database. These are common practices in the development of web applications. When SQL queries are not carefully constructed, SQL injection vulnerabilities can occur. The SQL injection attack is one of the most common attacks on web applications.

In this lab, we have created a web application that is vulnerable to the SQL injection attack. Our web application includes the common mistakes made by many web developers. Students' goal is to find ways to exploit the SQL injection vulnerabilities, demonstrate the damage that can be achieved by the attack, and master the techniques that can help defend against such type of attacks.

TYPES OF SQL INJECTION ATTACKS

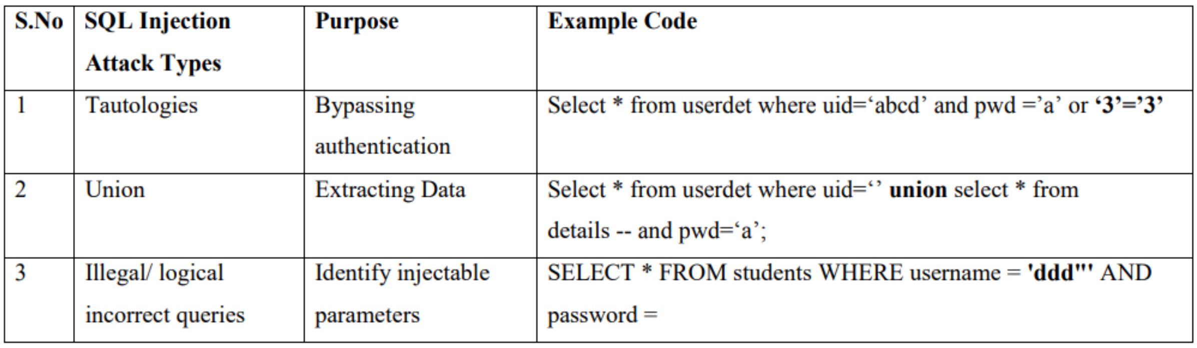
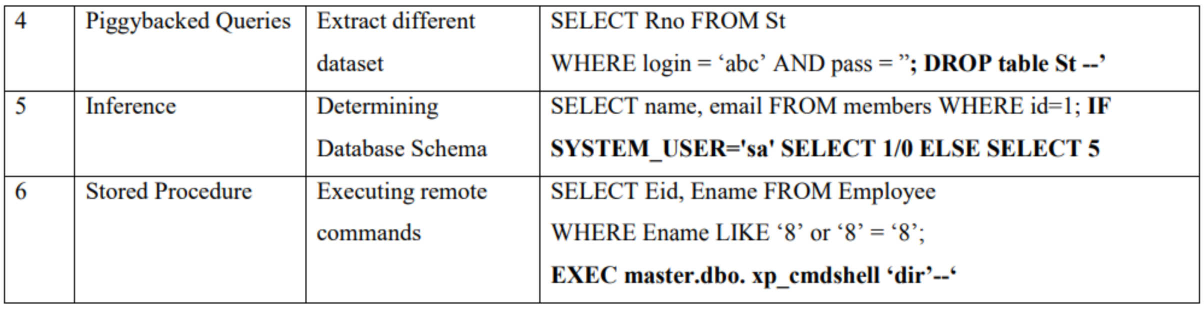
There are numerous SQL Injection attacks and it is performed sequentially or in combinatorial. Table-1 shows examples for each types of attack and its illustration is given below.

- **Tautological attack:**
  Result - authentication page is bypassed for data extracting
  Tricks applied – 'where' clause in SQL tokens is injected to make the conditional query remains true
- **Union Query:**
  Result – Different dataset is returned from the Database
  Tricks applied – SQL Injected query remains safe by joining the keyword 'union'
- **Illegal/Logically Incorrect Queries:**
  Result – Error message with useful debugging information

Tricks applied – By cause injects query with type mismatch, syntax error, logical errors

- **Piggybacked Queries:**
  Result – multiple queries are executed without the knowledge of the user which may lead to Database exploitation
  Tricks applied – injected queries are added to the normal executable query
- **Inference:**
  Result – different responses from database is cross checked by changing its behavior.
  Tricks applied – True/False questions using SQL statements is asked in serious (Blind attack). Based on time delay injected SQL queries are executed using if/then statement (Timing attack)
- **Stored procedure:**
  Result – remote commands, denial of service is performed.
  Tricks applied – Injection is done to the stored procedure present in the Database.

Table 1 Types of SQL Injection with Example.

S.No	SQL Injection Attack Types	Purpose	Example Code
1	Tautologies	Bypassing authentication	Select * from userdet where uid=‘abcd’ and pwd =’a’ or **‘3’=’3’**
2	Union	Extracting Data	Select * from userdet where uid=‘’ **union** select * from details -- and pwd=‘a’;
3	Illegal/ logical incorrect queries	Identify injectable parameters	SELECT * FROM students WHERE username = **'ddd"'** AND password =
4	Piggybacked Queries	Extract different dataset	SELECT Rno FROM St WHERE login = ‘abc’ AND pass = ”**; DROP table St --’**
5	Inference	Determining Database Schema	SELECT name, email FROM members WHERE id=1; **IF SYSTEM_USER='sa' SELECT 1/0 ELSE SELECT 5**
6	Stored Procedure	Executing remote commands	SELECT Eid, Ename FROM Employee WHERE Ename LIKE ‘8’ or ‘8’ = ‘8’; **EXEC master.dbo. xp_cmdshell ‘dir’--‘**

ATTACK SCENARIO

Consider a simple SQL injection vulnerability. The following code builds a SQL query by concatenating a string entered by the user with hard coded strings:

String query = "SELECT * FROM items WHERE owner = '" + userName + "' AND itemName = '" + ItemName.Text + "'";

The intent of this query is to search for all items that match an item name entered by a user. In the example above, userName is the currently authenticated user and ItemName.Text is the input supplied by the user. Suppose a normal user with the username smith enters benefit in the web form. That value is extracted from the form and appended to the query as part of the SELECT condition. The executed query will then look similar to the following:

SELECT * FROM items WHERE owner = 'smith' AND itemName = 'benefit' However, because the query is constructed dynamically by concatenating a constant base query string and a usersupplied string, the query only behaves correctly if itemName does not contain a single quote (') character. If an attacker with the username smith enters the string:

anything' OR '1'='1

The resulting query will be: SELECT * FROM items WHERE owner = 'smith' AND itemName = 'anything' OR '1' = '1'

The addition of the OR 'a'='a' condition causes the WHERE clause to always evaluate to true. The query then becomes logically equivalent to the less selective query:

SELECT * FROM items

The simplified query allows the attacker to see all entries stored in the items table, eliminating the constraint that the query only returns items owned by the authenticated user. In this case the attacker has the ability to read information he should not be able to access.

Now assume that the attacker enters the following:

anything'; drop table items—

In this case, the following query is built by the script:

SELECT * FROM items WHERE owner = 'smith' AND itemName = 'anything'; drop table items—’

The semicolon (;) denotes the end of one query and the start of another. Many database servers allow multiple SQL statements separated by semicolons to be executed together. This allows an attacker to execute arbitrary commands against databases that permit multiple statements to be executed with one call. The double hyphen (--) indicates that the rest of the current line is a comment and should be ignored. If the modified code is syntactically correct, it will be executed by the server. When the database server processes these two queries, it will first select all records in items that match the value anything belonging to the user smith. Then the database server will drop, or remove, the entire items table.

CONCLUSION

It is important to know how to identify and remediate SQL injection vulnerabilities because the vast majority of data breaches are due to poorly coded web applications. Any code that constructs SQL statements should be reviewed for SQL injection vulnerabilities since a database server will execute all queries that are syntactically valid. Also, keep in mind that even data that has been parameterized can be manipulated by a skillful and persistent attacker. Therefore, web applications should be built with security in mind and regularly tested for SQL injection vulnerabilities. More information about SQL injection and how to prevent it, including specific examples for a variety of scripting languages, as well as guidelines to review code and test for SQL injection vulnerabilities can be found at the Open Web Application Security Project (OWASP). See references below.