DETECTION AND PREVENTION OF SQL INJECTION ATTACKS USING NOVEL METHOD IN WEB APPLICATIONS

Tejinderdeep Singh Kalsi, Navjot Kaur

Address for Correspondence
Department of Computer Science, Global Institute of Management and Emerging Technologies, Amritsar, Punjab, India

ABSTRACT
In today’s era, Web applications play a very great role in individual life as well as in the development of the any country. The use of web application has become raisingly popular in our daily life as reading newspaper, making online payments for shopping etc. A SQL injection attack imposes a serious threat to the insurance of web applications because they may give attackers unrestricted access to databases that contain sensitive information. This paper gives an overview to the SQL Injection attacks (SQLIA) and methods how to prevent them. We will discuss all the proposed models to stop SQL Injections. In this paper we present a detail on numerous types of SQL injection attacks and prevention technique for web application. We also describes the technique to prevent injections attacks occurring due to dynamic SQL statements in database stored procedures, which are often used in e-commerce applications. Along with presenting our findings from the study, we also note down future expectations and possible development of countermeasures against SQL Injection attacks.

KEYWORDS— SQL Injection, database security, stored procedures.

INTRODUCTION
Web applications are often vulnerable for attackers that can be easily accessed to the application’s underlying database. SQL injection attacks occurs only when a malicious user causes a web application to generate and send a query that functions differently than the programmer intended. A SQL injection takes place when the application fails to properly sanitizing the user supplied input used in SQL queries. An attacker can manipulate the SQL statement which is passed at the backend of database management system. This statement runs with the same permissions as the application that executing the query. From now on it will be referred as session user. Modern database management systems are the most powerful applications. They usually provide built-in instruments to interact with the operating system. However, when they are absent, a motivated attacker will still access the system and execute arbitrary commands on the underlying system, this research will walk through how it can be achieved via SQL injection vulnerability, focusing on web-based applications.

WHAT IS SQL INJECTION ATTACK?
SQL Injection is a kind of web application security vulnerability in which an attacker is able to submit a database SQL command, which can be executed by a web application, uncovering the back-end database. SQL Injection attacks takes place when a web application employ user-supplied data without perfect validation or encoding as part of a command or query. The specially crafted user data schemes the application into executing unexpected commands or changing data. SQL Injection allows an attacker to create, read, update, alter, or delete data stored in the back-end database. In its most common form, SQL Injection allows attackers to have access of sensitive information such as social security numbers, credit card number or other financial data.

Main Concepts of SQL Injection
- SQL injection is a software vulnerability that takes place when data which is entered by the users is sent to the SQL interpreter as a part of an SQL query.
- An Attacker provides specially designed input data to the SQL interpreter and traps the interpreter to execute unintended commands.
- Attackers utilize this vulnerability by providing specially designed input data to the SQL interpreter in such a manner that the interpreter is not able to distinguish between the intended commands and the attacker’s specially crafted data. The interpreter is tricked into executing unexpected commands.
- SQL injection exploits security vulnerabilities at the database layer. By exploiting the SQL injection fault, attackers can create, read, modify, or delete sensitive data.

PROBLEM DEFINITION
There are different kinds of SQL Injection attacks and each attack is performed for some specific purposes. These purposes are as follows:

Identifying Injectable Parameters
The attacker wants to know what kind of parameters and user input fields are vulnerable to SQLIA in a web application.

Performing Database Finger Printing
The attacker wants to know that what type and version of database is being used by the Web application. Different types of databases will be responding differently to different queries and attacks, and this information can be used to “finger print the database. If the attacker knows the type and version of the database used by a Web application then it allows the attacker to design database specific attacks.

Determining Database Schema
The attacker wants to discover database schema information such as table names, column names, and column data types in order to correctly clipping data from a database.

Extracting Data
These types of attacks employ techniques that will clip valuable data values from the database.
Adding or Modifying Data
The main motive of these attacks is to add or modify information in a database.

Performing Denial of Service
These attacks are used to shut down the database of a Web application, thus denying service to other users even to logical ones.

Evading Detection
This type of attacks refers to those which are employed to avoid verification and detection by system protection mechanisms.

Bypassing Authentication
The motive of these types of attacks is to let the attacker to bypass authentication Mechanisms of application and database. By passing such mechanisms could let the attacker to assume the rights and privileges associated with another application.

Executing Remote Commands
These types of attacks try to execute arbitrary commands on the database. These commands can be stored procedures or functions available to database users.

OBJECTIVES
1. To design technique for effective dynamic detection and prevention of SQLIAs without access to the application source code.
2. To implement a heuristic approach to searching a valid query structure in our database to minimize the response time.
3. To compare computational overhead of string matching algorithm for matching tokens by converting the token to some integer values and checking those integer values instead of the tokens.

RESULTS AND DISCUSSION
This research investigates what are SQL injection attacks in Real Time web Application with example and how that can be prevented. SQL injection means injecting some SQL commands in SQL statements to hack the data or delete data or change contents in tables via web page input.

When we run code we will get throughout like as shown below:

How we can inject SQL in our queries from our webpage input:
First in our textbox enter text like “10 or 1=1” as shown above and check the result. It will return all the rows from table because our textbox input value converts query as like as shown below:
In above query it will check for value =10 as well as it will check for 1=1 means always true that’s the reason it will be returning all the values from table this way they can inject values to change our queries and access all the values from table.
In another case if user enters value like “10; Drop TABLE country details” in it will drop table from our database because our query will changed like this:
Select Name,Total=value from countrydetails where value =10; Drop TABLE country details from the above query can inject SQL and get all the details or delete data or drop tables.

select Name, Total=value from countrydetails where value =10 or 1=1

![Fig.2 - Injecting SQL attack](image)

![Fig.3 - Injecting SQL attack (on front end)](image)

![Fig.4 - Injecting another SQL attack (on front end)](image)

![Fig.5 - Query to SELECT & DROP customer table](image)

SQL INJECTION PREVENTION
To avoid these SQL injection attacks always we need to use parameterized queries like as shown below:
select Name, Total=value from countrydetails where value =@value

![Fig.6 - Query to SELECT & DROP customer table](image)
Kevin et al., International Journal of Advanced Engineering Technology  

$$\text{SqlCommand cmd = new SqlCommand("select Name,Total=value from countrydetails where value =@value", con);}$$
$$\text{cmd.Parameters.AddWithValue("@value", txtSearch.Text);}$$
$$\text{SqlDataAdapter da = new SqlDataAdapter(cmd);}$$

Fig. 6 - Wrong query will display error

There are many architectures which can manage and organize any data-driven systems, but the most common architecture which is used is the three-tier architecture that depends on dividing the system into three tiers as follows:

1. Presentation Tier (a Web browser or rendering engine).
2. Logic Tier (a server code, such as C#, ASP, .NET, PHP, JSP, etc ...).
3. Storage Tier (a database such as Microsoft SQL Server, MySQL, Oracle, etc.).

Fig. 7 - Summarizes the steps of exchanging data among the three-tier system architecture

The suggested approach is based on different steps to reject any malicious query from being passed through the database engine before its execution course, and those steps could be listed as follows:

Replicate system databases

For each database for which we want to be secured from SQLIAs, there should be a new replication database and it must contain a small amount of sample data.

Creating “database Behaviors” database

The suggested approach must have a separate database called “database Behaviors” that will be containing all system database queries and their familiar behaviors that have resulted from SQL queries execution in normal cases. This database is located in the replicated instances.

Redirect SQL queries

Any SQL query appointed to be executed in the target database will be initially delayed and replicated by the database engine then this replicated query is sent to the virtual database (Schema Replicated database). Hence, the original SQL query will be not executed yet in this stage and it will be delayed to a later stage.

Simple SQL syntax checking

All SQL queries which are passing through the replicated database must also pass through numerous check processes before they change to the next step namely, “The execution process”. The following list will presents the checks processes that the SQL queries should pass through.

Encoding analysis

Before enduring to any next step the received SQL queries must be tested to determine the character encoding which will be used to write these queries. There are many techniques that can be used to do this analysis process such as “Automatic Identification of Language and Encoding”.

Simple White-Box validation

The query must go through simple syntax validation and refining for specific SQL reserved words especially those that use (EXECUTE, SHELL commands).

Parameters replacement

Any parameter that was found in the SQL query should be restored by an indexing parameter names. Such as (@par_1, @par_2 ... @par_n).

Virtual execution

After the SQL order checking process, the SQL query will be processed on the replicated database “Virtual Database” in which it is a process that is running simultaneously with the execution process, it monitors and traces the behaviors of the SQL query.
**SQLIA Detection**

This stage is the most important stage in the suggested technique, its purpose is to detect whether the received SQL query is valid or not. The idea here is to catch the object that was affected by the current SQL query whatever the type of such objects and create a list of these objects to use them in the upcoming step of this stage. The resulted list of affected objects will be matched with the “database Behaviors”. If there is a query which is handling the entire listed objects with the previous step then this behavior query will be added to a new list (Expected Queries). Any resulted behavior that is detected as a suspicious must be rejected and deleted from the actual database instance execution queue; otherwise the query will be transferred to the actual database instance for being executed.

**4.3 Algorithm for detecting and preventing SQL Injection Attack**

An efficient algorithm for detecting and preventing SQL Injection Attack is made known that is based on Pattern matching. The planned architecture is given in fig. 4.9. The expected scheme has the following two modules, 1) Static Phase and 2) Dynamic Phase. In the Static Pattern list, a list of known Anomaly Pattern is preserved. In Static Phase, the user generated SQL Queries are been checked by applying Static Pattern Matching Algorithm. In Dynamic Phase, if any form of new anomaly will occur then alarm will be indicating and new Anomaly Pattern will be generated. The new anomaly pattern will be updated to the Static Pattern List. The following stages are performed during Static and Dynamic Phase:

**Static Phase**

**Step 1:** User which is generating SQL Query is send to the proposed Static Pattern Matching Algorithm.

**Step 2:** The Static Pattern Matching Algorithm that is given in Pseudo Code is given below.

**Step 3:** The Anomaly patterns are maintained in Static Pattern List, during the pattern matching process each pattern is matched with the stored Anomaly Pattern in the list.

**Step 4:** If the pattern is exactly matching with one of the stored pattern in the Anomaly Pattern List then the SQL Query is affected with SQL Injection Attack

**Dynamic Phase**

Otherwise, Anomaly Score value was calculated for the user generated SQL Query, and If the Anomaly Score amount is greater than the Threshold amount, then a Pattern Matching Algorithm is followed. In the planned architecture, Static Pattern Matching Algorithm is the main part and the pseudo code for the Algorithm is given below:

![SQL Pattern matching Architecture](image)

**Fig. 10- SQL Pattern matching Architecture**

**I. ALGORITHMS**

**Static Pattern Matching Algorithm**

1: Procedure SPMA(Query, SPL[])

**INPUT:** Query User Generated Query

**SPL[]** Static Pattern List with m

**Anomaly Pattern**

2: For j = 1 to m do

3: Compare all values query length and pattern values if both are same then

4: Calculate anomaly value

5: If (anomaly) Score Value Anomaly ≥ Threshold

6: Then

7: Return Alarm Administrator

8: Else

9: Return Query Accepted

10: End If

11: Else

12: Return Query Rejected

13: End If

14: End For

15: End Procedure

**Multiple Keywords Matching Algorithm**

1: N[] integer representing the text length (SQL Query Length) q0 [] initial state (first character in pattern)

2: State [ ] q0

3: For i = 1 to n do

4: While g (State, y[i] = fail) do

5: State ← f(State)

6: End While

7: State ← g(State,y[i])

8: If o(State) then

9: Output i

10: Else

11: Output

12: End If

13: End For

14: End Procedure

**CONCLUSION**

The suggested compound technique will be done in two main phases: runtime analysis, and static analysis. The first phase is a dynamic/runtime analysis method which depends upon applying tracking methods to process and monitor the execution processes of all received queries. The resultant of affected objects of this monitoring will be matched with a planned set of expected changes that the developer had created before, and the output of this comparison process will decide if there is an existence of any type of SQLIA and if so that will be forwarded to the next phase. The next phase is a static analysis phase that is performing a string matching between the received SQL queries and previous expected SQL queries to stop any query that is described as a cautious query.

**Fig 11- Pattern Matching**
REFERENCE


