Efficient Method for Preventing SQL Injection Attacks on Web Applications Using Encryption and Tokenization

P. Saravana Kumar  
Assistant Professor  
Senthamarai college of Arts and Science, Madurai, TamilNadu, India

M. Parvathi  
Head/Assistant Professor  
Senthamarai college of Arts and Science, Madurai, TamilNadu, India

M. Kanmani  
Assistant Professor  
Senthamarai college of Arts and Science, Madurai, TamilNadu, India

Abstract—Web applications are increasingly used in recent years to provide online services such as banking, shopping, social networking, etc. These applications operate with sensitive user information and hence there is a high need for assuring their confidentiality, integrity, and availability. The proposed system focuses on how to detect and prevent SQL injection attacks on web applications using encryption and tokenization technique. The tokenization process is applied on the input query by detecting spaces, single quotes and double dashes etc. This process converts the input query into fruitful tokens on both client and server side and that are stored in a dynamic tables. Both dynamic tables are compared and if both are equal, it seems that there is no injection attacked in the given query, hence the query is proceed further to main database for retrieving result. If they are different, query is rejected and not forwarded to the database server. The Customized error message notification is given to the client. It has better performance and provides increased security in comparison to the existing solutions. The goal of this project is to provide increased security by developing a method which prevents illegal access to the database.

Keywords—Web Application, SQL, Encryption, Tokenization, SQL Injection Attacks.

I. INTRODUCTION

The paper deals the main intent to use SQL injection attack includes illegal access to a database, extracting information from the database, modifying the existing database, escalation of privileges of the user or to malfunction an application. Ultimately SQLIA involves unauthorized access to a database exploiting the vulnerable parameters of a web application.

II. WHAT IS SQL INJECTION?

In order to locate the hotspots where SQLIA vulnerability occurs, we first discuss about the 3-tier logical view architecture of web application.

1) User interface tier: This layer forms the front end of the web application. It interacts with the other layers based on the inputs provided by the user.

2) Business logic tier: The user request and its processing are done here. It involves the server side programming logic. Forms the intermediate layer between the user interface tier and the database tier.

3) Database tier: It involves the database server. It is useful in storage and retrieval of data.
There are two types of SQLIA detection

i. Static Approach: This approach is also known as pre-generating approach. Programmers follow some guidelines for SQLIA detection during web application development. An effective validity checking mechanism for the input variable data is also requires for the pre-generated method of detecting SQLIA.

ii. Dynamic Approach: This approach is also known as post-generated approach. Post-generated technique are useful for analysis of dynamic or runtime SQL query, generated with user input data by a web application. Detection techniques under this post-generated category executes before posting a query to the database server.

A. Classification of SQLIA –

SQLIA can be classified into five categories:

1) Bypass Authentication - using tautology
2) Unauthorized Knowledge of Database - using illegal/incorrect Queries
3) Unauthorized Remote Execution of Procedure
4) Injected Additional Query - Using Piggy-Backed Queries.
5) Injected Union Query

1) Bypass Authentication: Researchers have proved that query injection can’t be applied without using space, single quotes or double dashes (--). In bypass authentication, intruder passes the query in such a way which is syntactically true and access the unauthorized data.

2) Unauthorized Knowledge of Database: In this type of attack, intruder injects a query which causes a syntax, or logical error in to the database. The result of incorrect query is shown in the form of error message generated by the database and in many database error messages, it contains some information regarding database and intruder can use these details

3) Unauthorized Remote Execution of Procedure: SQLIA of this type performs a task and executes the procedures for which they are not authorized. The intruder can access the system and perform remote execution of procedure by injecting queries.

4) Injected Additional Query: When an additional query is injected with main query and if main query generates Null value, even though the second query will take place and the additional query will harm the database.

5) Injected Union Query: In this type of attack, the intruder injects a query which contains set operators. In these queries, the main query generates Null value as a result but attached set operators data from database.
To secure data from SQL Injection attack, Prevention using Tokenization Model is proposed which block the malicious input query at entry point. The proposed solution aims to make this model secure and effective which emit out the fruitful results. The main purpose of this work is to prevent database from the most serious web vulnerabilities. This solution completely based on query tokenization technique to prevent SQLIA. To prevent the SQL injection attack, the overall working of this model takes place into the sequence of essential steps which are given below:

1. User enters the input query.
2. Divide the query into number of useful tokens based on delimiter such as detecting spaces, single quotes and double dashes etc.
3. These tokens will be stored in a dynamic table which contains three columns such as name of the token, type of the token and the number of occurrence of each token.
4. Before forwarded to the server side, the attributes and data in the input query are encrypted using AES (Advanced Encryption Standard) algorithm which is fast, and requires little memory.
5. Both Dynamic table and encrypted query are forwarded to the server side.
6. At the server side, input query is decrypted and in turn converts into various token which are stored in to another dynamic table.
7. Both the dynamic tables are compared and if both are equal, it seems that there is no injection attacked in the given query
8. Hence the query is proceed further to main database for retrieving records.
9. If both are different, query is rejected and not forwarded to the database server. The Customized error message is sent back to the client.
III. Scope of AES Encryption and Decryption

Scope includes implementing the type – The attributes and data in the input query are encrypted using AES (Advanced Encryption Standard) algorithm which is fast, and requires little memory. Once the query is arrived at server side, which is decrypted by using the same key and in turn converts into various token which are stored in to another dynamic table. The performance comparison of cipher text over normal text shows that, cipher text is very difficult and time consuming to crack.

IV. Query Tokenization and Forwarding

Query tokenization technique converts the input query into the useful tokens. These tokens are generated by detecting single quote, double dashes and space in an input query. All string before a single quote, before double dashes and before a space constitutes a token. Tokenization process executes in following four essential steps and then forwarded to the server side. The overall tokenization process is depicted on the figure 3.

Step 1: Process the input query by replacing all the unnecessary characters From the query
Step 2: Detect Single Quote, Double Quote, Double slashes and space in the input query.
The figure 3 shows how tokens are formed by detecting spaces single quote and double dashes in input query for the given below input query.

Input Query: “SELECT eid, ename FROM Employee WHERE salary > 2000 ”

Comparison of Dynamic Tables

In this module, the length of both the dynamic tables such as D1 and D2 are compared. If lengths are same, then the each and every occurrence of both D1 and D2 are compared. After comparison, if both are same, there is no injection presented in the query and the query is steps forward further to main database for accessing the table. But if the length of D1 and D2 are different or else even any one token is different, then injection has attacked and query does not forwarded to the database. The attacked query has rejected and send customized error message back to the client.
Step 3: Break the input query into useful tokens.
Step 4: Store the tokens in a Dynamic table

Step 5: Query Forwarding – After tokenization, the attribute, table name and data in the query are encrypted using AES encryption algorithm and then the query along with token table are forwarded to server.
Comparison of Dynamic Tables

In this module, the length of both the dynamic tables such as D1 and D2 are compared. If lengths are same, then the each and every occurrence of both D1 and D2 are compared. After comparison, if both are same, there is no injection presented in the query and the query is steps forward further to main database for accessing the table. But if the length of D1 and D2 are different or else even any one token is different, then injection has attacked and query does not forwarded to the database. The attacked query has rejected and send customized error message back to the client.
V. ROLES AND RESPONSIBILITIES OF TOKENIZATION TECHNIQUES

This form has received any type of input query from the user/client and converted into cipher text. The on click of Generate client side token button generate the client side tokens and on click of Send to server forward the input query into the server side.

This page displayed the client side tokens in a dynamic table in which the TOKEN column has the name of all the tokens, TYPE column indicates the type of the column such as keyword, Attribute, Literals and Data. The third column such as count hold the number of occurrence of each token.
This page is received the query which has sent by the client and then translate the received query into number of token which has shown in the next figure. Both the dynamic tables are displayed in this page and then the server performed comparison of both tables.

This figure shows the dynamic tables that are occurred after successful comparison. In this figure both the tables are same, hence no attack, query is steps forward further to main database for accessing the table.
This figure shows the dynamic tables that are occurred after successful comparison. In this figure both the tables are not same. Hence there is an attack, query does not forwarded to the database. The attacked query has rejected and send customized error message back to the client.

This page retrieve the results from the table after sending the successful query to the main database for accessing the records.

**VII. CONCLUSION**

This paper has presented a lightweight method to prevent SQL injection attacks by applying query tokenization technique to convert SQL queries into number of useful tokens and then encrypting the table name, fields, literals and data on the query using AES Encryption algorithm. This approach avoids memory requirements to store the legitimate query in repository and facilitates fast and efficient accessing mechanism with database. Our experimental results show that this approach can effectively prevent all types of SQL injection attempts. This approach does not
require major changes to application code and has negligible effect on performance even at higher load conditions due to its low processing overhead. It can also be easily applied to any other language & database platform without major changes. Further explore on the query transformation scheme is needed to make use of own new encryption algorithm for preventing SQL injection attacks.

REFERENCES


