A Generalized Way to Prevent SQL Injection Attacks (SQLIAs) Based on Encryption Algorithms

Shakti Bangare¹, G. L. Prajapati²

¹Assistant Professor Department of Information Technology, Sushila Devi Bansal College of Engineering, Indore
²Reader, Institute of Engineering and Technology, Devi Ahilya Vishwavidyalaya, Indore

Abstract—As most of the web applications are Dynamic and all Dynamic web applications have databases as backend. Dynamic web applications usually interact with databases using some input value provided by user in input box of Dynamic web pages. This input value is checked at server to provide access to databases for which intention it is designed by the programmer. But some SQL code, for which application is not intentionally designed, is provided by malicious users and gain access to the databases. These SQL code to gain unintentionally access are SQL Injection Attacks (SQLIAs). There are various solutions for prevention and detection of SQLIAs [2]. Some of them are based on Encryption which is best suited to detect and prevent SQLIAs. So in this paper we have given one generalized solution to prevent SQLIAs based on Encryption algorithms.

Keywords— Generalized Encryption for SQLIAs, Remove SQLIAs, Security of Web Applications, SQL Injection Attacks (SQLIAs), SQL Vulnerability.

I. INTRODUCTION

Today designing of web applications is not a major issue. Security of these applications is a major issue. According to White Hat report of 2013 on web Security SQL Vulnerability is at the 14th position and 7% of overall vulnerability [1]. Using SQL Injection Attacks (SQLIAs) any malicious user can perform some tasks which are not intentionally programmed (to steal personal information, to do some unintentionally financial operation and to destroy information etc.) by the programmer for that application. Much research work has been done and more to be done in this area to prevent web applications from SQLIAs [1].

Dynamic web applications are intentionally designed to perform some task. For better understanding of dynamic web applications we can refer three-tier architecture of web applications [6].

When user wants to accomplish their tasks using these Dynamic web applications for which they are designed, they give input to input box of these dynamic web applications at client end (user interface tier). These values pass to server end (business logic tier). At server end application interact with the database tier. Database works as backend in the application.

Server which lies in middle of application interacts with database using Standard Query Language (SQL). At server end a query is formed using input received from user and fired in database, database responses to server based on query fired. Responses are collected by server and some specific behaviour of application is performed as results to user.

Some malicious users enter values in input box of dynamic web pages which is syntactically correct SQL code but intentionally not desired by the programmer. This SQL code is SQLIAs. Using SQLIAs a malicious user can perform some different functions which are not intentionally programmed by programmer. A malicious user can steal, modify or delete important data from database. He can also alter and drop the tables.

Figure 1: Login Form

For example there is one web application. Front view of web page is given in figure 1. There is one makecall table in database which is accessing through query provided by server (business logic tier). Query is as follows:

```
‘‘select * from makecall where username=‘’
+request.getParameter(“uname”) +‘’ and password=‘’
+request.getParameter(“pwd”) +‘’;’’
```

Where request.getParameter(“uname”) acquires value entered in input box of password field of figure 1. Above code is intentionally written to login a user by validating username and password.
However a malicious user can enter “kavitakashyap” into the username field and “” OR ‘1’=’1” into the password field, the query string becomes:

```
"Select * from makecall where username = 'kavitakashyap' and password='" OR '1'='1" ;
```

So this query will always execute whether password is right or wrong. This is a way to unintentionally access to database. There are some other ways of SQLIAs [2][3]. There are various solutions to detect and prevent SQLIAs. Various comparisons have been done among them [3]. Some solutions are Encryption based solutions. In all Encryption based solutions emphasized on the specialized Encryption algorithms and in this paper we emphasized on generalized way used in the Encryption based solutions. Advantage of our algorithm over other Encryption based solutions is that we are not bounded on any particular Encryption algorithm. We can use any conventional Encryption algorithm to adapt our solution.

II. EXISTING ENCRYPTION BASED SOLUTIONS

There has been various research works going on in the field of prevention of SQLIAs based on Encryption. Following are some Encryption based solutions to prevent SQLIAs.

The very first Encryption based solution is SQLrand [4]. In SQLrand developer creates queries based on randomization of keywords. This query is de-randomized by a proxy.

Indrani Balasundaram and E.Ramaraj presented one paper named “An Authentication Mechanism to Prevent SQL Injection Attacks” [5]. In this mechanism database have one login table which has three fields username, password and unique secret key which are initialized during the registration phase. When user wants to login username and password, are encrypted using users secret key at client end. Query is generated at server. Generated query have four values username, password and its encrypted values. Database matches username and password during verification if matches then encrypted username, password are decrypted using secret key stored in database. If decrypted username and password matches with username and password provided by user then user correctly login otherwise malicious user.

Next solution is Random4 [6]. The basic idea behind it is that, they proposed one encryption algorithm which is not any standard algorithm but sufficient to encrypt any data. Input data provided by user is first encrypted and checked with data stored in database. Data stored in database is also encrypted to prevent blind injection.

Sometimes table name and table columns are also encrypted so piggy backing queries were also prevented.

Mittal and Jena [7] proposed one solution to prevent SQLIAs. In this solution bit slice method is used. In their paper they proposed an encryption method which is not any conventional method. Key is chosen on the basis of username and password provided by the user during the registration phase. All the field which are probable to inject are encrypted during the registration phase and stored in database in encrypted form. When user tries to login that time key is again formed using username and password and fields are encrypted after that query is fired in database.

All the solutions provided above are based on some encryption method but idea behind it is same. So we can use this idea in a generalized way. We should not bound on any specific encryption algorithm to encrypt data and we can have many options to encrypt our data using any standard encryption algorithm. One generalized way to encrypt data is used in this paper.

III. PROPOSED ALGORITHM

A Dynamic web application works in three phases. Registration phase, inject able phase and verification phase. Our algorithm may be understood using these phases, described follows:-

A. Registration Phase

When user visit any web site first time and want to become a registered user to access functionality of that web application then he has to fill registration form. A registration form may consist of user’s personal information. For implementation of our algorithm, first step is to find out the input values in registration form which are probable to inject. Values which are inserted as input in input box of web form in future to access functionality of web application from client end are probable to inject. When server (middleware) receives these values, all injectable data are encrypted using any standard encryption algorithm depends on application and non- injectable data are kept identical and put together with intended query. Query is fired in the database. Usually insert query is fired during registration phase.

B. Inject able Phase

After successfully registration, user uses that application in future. That time user may send inject able code by entering data as input to input box of web form. For example user tries to login after successfully registration. Scenario is given in figure 1. A user tries to login using username and password fields.
At middleware, these received username and password might be injected SQL code. So according to our algorithm all injectable values received from user are encrypted using the identical encryption algorithm as used in registration phase and put together with intended query.

C. Verification Phase

Query is fired in database. All encrypted values received from user are verified with values in database. If values are identical then we conclude that query is fired by registered user otherwise query is fired by malicious user. Inject able values had been always stored in encrypted form in database during registration phase.

IV. Formally Defined Algorithm

Now our algorithm can be defined formally using one theoretical model. Let there is one machine \( S \) representing to any Encryption/Decryption tier in response to input provided. \( I = \{ I_1, I_2, I_3, ..., I_n \} \).

\( O = \{ O_1, O_2, O_3, ..., O_m \} \).

\( M = \) It is a machine working at middleware (business logic tier) representing to any Encryption/Decryption algorithm.

Machine \( M \) provides one level of abstraction to machine \( S \). It can be defined as four tuple values:

\[ S = \{ I, O, M \} \]

Where \( I \) set of input provided by user to web application at client end. \( I = \{ I_1, I_2, I_3, ..., I_n \} \).

\( O \) = Set of output received by user from business logic tier in response to input provided. \( O = \{ O_1, O_2, O_3, ..., O_m \} \).

\( M \) = It is a machine working at middleware (business logic tier) representing to any Encryption/Decryption algorithm.

Machine \( M \) provides one level of abstraction to machine \( S \).

It can be defined as four tuple values:

\[ M = \{ P, C, E, D, R \} \]

Where, \( P \) set of plain text. \( P = \{ P_1, P_2, P_3, ..., P_l \} \)

\( C \) = It is set of cipher text. \( C = \{ C_1, C_2, C_3, ..., C_n \} \), \( 1 \leq n \).

\( E \) = It is any traditional/standard Encryption algorithm

Which encrypts set of plain text \( P \) into equivalent set of cipher text \( C \).

\( D \) = It is decryption algorithm which decrypts set of cipher text into equivalent set of plain text.

\( R \) = It is a set of received output values when query executes. \( R = \{ R_1, R_2, R_3, ..., R_s \} \), \( m \leq s \).

\( l, m, n \) and \( s \) are any arbitrary integers.

Formally Defined Algorithm:

1) When Input \( I \) are received at middleware then \( P \) is initialized where \( P \subseteq I \). Set of plaintext \( P \) contains only those values which are probable to inject.

2) \( E \) encrypts all elements of \( P \) into Cipher text \( C \).

3) Put these elements of Cipher text \( C \) and elements \( (I - P) \) to intended query.

4) If Query will execute successfully in database then \( R \) is some result with success and go to step 6, else \( R \) is some result with not success then initialize \( R = \emptyset \), \( P = \emptyset \) and \( C = \emptyset \).

5) Give some results \( O \) to user with query not successful and go to 12.

6) There are two cases for query with success. First, if query is select operation then it will receive some output \( R \) which have to provide to user, so go to step 8. Second if query is insert or update operation, then initialize \( R = P = C = \emptyset \).

7) Give some results \( O \) to user as message successfully inserted/updated and go to 12.

8) In \( R \) some or whole data may be in cipher text form, Now initialize set \( C \) with some elements which are in encrypted form and have to provide to user in decrypted form. \( C \subseteq R \).

9) Give Cipher text \( C \) as input to Decryption algorithm \( D \).

10) \( D \) decrypts set of Cipher text \( C \) into set of Plain texts \( P \).

11) Now initialize set of Output \( O \) which forms some message to user. It has elements from \( P \) and \( R - C \). \( O \subseteq (P \text{ and } R - C) \).

12) Exit.

V. Implementation And Results

We may use any traditional Encryption and decryption method to implement proposed algorithm. For example we implemented our algorithm using Advance Encryption Standard (AES). Let there is one web application i.e., any mobile user can make free call to any other mobile user.

See figure 2, figure 1 and figure 3 at front end as registration, login and make a call forms respectively.

During registration phase a user fills registration form (figure 2). In injectable phase username, password and mobile number fields are used to inject query. These fields are injectable because these are taken as input in input box of web form (”username” and “password” is used in figure 1 and mobile number would be used as “Enter Your Number” field in figure 3). So during registration phase these fields are stored in encrypted form in database and other remain identical. Query working at middleware for registration form is given below. See Table 1 for snapshot of values stored in database. Query encrypting code is as follows:-
Insert into makecall (username, password, name, mobile, address)
values(""+c.encrypt(request.getParameter("uname"))+","+
+c.encrypt(request.getParameter("pwd"))+","+request.getParameter("name")+","+c.encrypt(request.getParameter("mobile"))+","+request.getParameter("address")+")+");

During verification phase injectable values are encrypted and matched with the values stored in database. If these encrypted values are required to show to user then first decrypted and are shown at client end. Following query is working for figure 1 as verification.

"select * from makecall where username=""+c.encrypt(request.getParameter("uname"))+"" and password=
""+c.encrypt(request.getParameter("pwd"))+"";

If any malicious user enter any injectable code in the injectable fields of dynamic web pages then after encrypting these values at server end do not match with the values stored in the database.

All the advantages of encryption based solutions can be achieved by the use of our algorithm. Comparison of results with other solutions is same as all encryption based solutions (see table III [6]).

<table>
<thead>
<tr>
<th>username</th>
<th>password</th>
<th>name</th>
<th>mobile</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kavita</td>
<td>Kashyap</td>
<td>Kavita</td>
<td>9981470226</td>
<td>551/17, Meghdoot Nagar Indore</td>
</tr>
<tr>
<td>Shakti</td>
<td>Bangare</td>
<td>Shakti</td>
<td>9981461189</td>
<td>Meghdoot Nagar Indore</td>
</tr>
</tbody>
</table>

VI. CONCLUSION

We have presented a generalized way to prevent SQLIAs based on encryption algorithms. We are not emphasized on any encryption algorithm although we have used idea of encryption to remove SQLIAs. In this paper we presented formal definition of our algorithm. We presented one possible implemented for our algorithm i.e. based on AES encryption algorithm. Results are also compared.

REFERENCES


