ISSN:0975-9646

Deevi Radha Rani et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 3 (2) , 2012,3689-3692

# Web Security by Preventing SQL Injection Using Encryption in Stored Procedures

Deevi Radha Rani, B.Siva Kumar, L.Taraka Rama Rao, V.T.Sai Jagadish, M.Pradeep

*Department of Computer Science and Engineering*
*Koneru Lakshmaiah College of Engineering*
*Green Fields, Vaddeswaram, Guntur District, A.P., INDIA.*

***Abstract*— SQL Injection attacks target databases that are accessible through a web front-end, and take advantage of flaws in the input validation logic of Web components such as CGI scripts. SQL Injection attacks can be easily prevented by applying more secure authentication schemes in login phase itself. In this paper we are going to prevent SQLIA (SQL Injection Attacks) by using encryption in Stored Procedures. Advance Encryption Standard (AES) Encrypted user name and password are used to improve the authentication process with minimum overhead. The server has to maintain encrypted parameters of every user's username and password.**

***Keywords- sql injection; encryption; stored procedures; parameterized queries; bind variables; sanitization; authentication***

## 1. INTRODUCTION

SQL injection is a basic attack used either to gain unauthorized access to a database or to retrieve information directly from the database. They are used most often to attack databases and for extracting any confidential information such as Credit card information, Social Security numbers etc. Web applications are at highest risk to attack since often an attacker can exploit SQL injection vulnerabilities remotely without any proper database or application authentication. An application is vulnerable to SQL injection for only one reason – end user input string is not properly validated and is passed to a dynamic SQL statement without any such validation. If we are sanitizing the user input, then indirectly we are restricting them to not entering single quotes and double quotes in the input.

SQL injection is too much vulnerable that it can bypass many traditional security layers like Firewall, encryption, and traditional intrusion detection systems. SQL injection can not only be used for violating the security by seeing the private data of the people but also can be used for bypassing the authentication of user which is a big flaw in the web applications. Normally, web applications is a three tier architecture, the Application tier at the user side, Middle tier which converts the user queries into the SQL format, and the backend database server which stores the user data as well as the user's authentication table

Till now we have so many solutions to prevent SQLIA. Those are using bind variables, proper input validation, customized error messages, limiting database permissions. Any program or application may be vulnerable to SQL injection including stored procedures executed with a direct database connection. Write the stored procedure in one way, you can prevent SQL injection. Write it in another way, and you are still vulnerable to SQL injection.

Stored procedures are an important part of modern-day web applications. It is an operation set that is stored in the database. Since stored procedures are stored on the server side, they are available to all clients. They add an extra layer of abstraction in to the design of a software system. This layer of abstraction also helps put up an extra barrier to potential attackers. The benefits of stored procedures are encapsulation of business logic in a single entity, faster execution, exception handling, create it once and store it in database and can be called any no. of times. SQL statements are built at run time according to the different user inputs. For example, in SQL server, EXEC (varchar(n)@SQL) could execute arbitrary SQL statements. This feature offers flexibility to construct SQL statements according to different requirements, but faces a potential threat from SQL Injection Attacks.

In this paper we are proposing a technique to prevent the SQL injection especially in stored procedures by encrypting user input fields. For this purpose we are using AES encryption algorithm. We are going to encrypt user data in stored procedures so that SQL injection can be eliminated in stored procedure. If a user wants to access the database form remote place then he has to logon to the system through web site using the user name and password. In the middle tire, SQL query is generated and the web server verifies user name and password, if it is matches then the user can access the database. We use one mechanism like SQL-Injection in order to bypass the login phase without entering proper user name and password.

The original query after entering correct user name and password will be like this:

$stmt ="SELECT * FROM users where username='*username*' and password= '*password*'

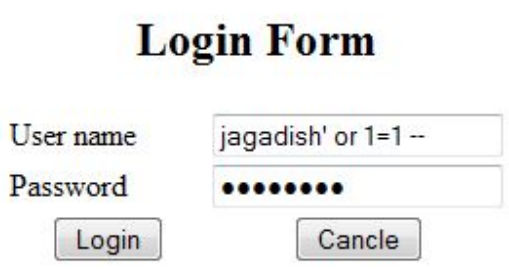



**Figure. 1**

For example if user enters "jagadish'*or 1=1*--" and "*any password*" in username and password fields instead of correct user name and password, the resulting query will be:

$stmt="SELECT * FROM users where username='jagadish' *or 1=1* --' *and* password='*any password*'

After --, the rest of the sentence will be treated as comment and because of '1=1' is always true.

This paper deals with prevention of SQL-Injection in login-phase and in UPDATE query. Both of these are implementing in stored procedures with proper encryption so that SQLIA can be prevented.

## 2. COMPARATIVE ANALYSIS

**Do Stored Procedures Protect Against SQL Injection?** Stored procedures *do not*, by themselves, necessarily protect against SQL-Injection. Write a stored procedure one way, and you can prevent SQL Injection. Write it another way, and you are still vulnerable. let's suppose we have the following (admittedly contrived) login script:

```
<form method="post" action="test1.php">
    Username:<input type="text" name=
"Username" id="Username"/></br>
    Password:<input type="text" name=
"Password" id="Password"/></br>
    <input type="submit" name="submit"
value="Submit" />
</form>
```

```
<?php
if(isset($_POST['Username']))
{
    $username = $_POST['Username'];
    $password = $_POST['Password'];

$con = mysql_connect("localhost","root","");

    $params = array($username, $password);
    $stmt = sqlsrv_query($conn, "{call
VerifyUser( ?, ? )}", $params);

    if(sqlsrv_has_rows($stmt))
    {
        echo "Welcome.";
    }
    else
    {
        echo "Invalid password.";
    }
}
?>
```

Let's take a look at two ways to write that stored procedure…

**The Wrong way**

Suppose the VerifyUser stored procedure was created by dynamically building a SQL string within the stored procedure, like this:

```
CREATE PROCEDURE VerifyUser
    @username varchar(50),
    @password varchar(50)
AS
BEGIN
    DECLARE @sql nvarchar(500);
    SET @sql = 'SELECT * FROM UserTable
       WHERE UserName = ''' + @username + '''
AND Password = ''' + @password + ''' ';
    EXEC(@sql);
END
GO
```

As we discussed in figure.1: login form, the dynamically generated SQL Query will be look like this:

```
SELECT * FROM UserTable WHERE UserName =
'jagadish' --' AND Password = 'any password'
```

The last half of the query is commented out!

**The Right way**

Now suppose the VerifyUser stored procedure was created like this:

```
CREATE PROCEDURE VerifyUser
    @username varchar(50),
    @password varchar(50)
AS
BEGIN
    SELECT * FROM UserTable
    WHERE UserName = @username
    AND Password = @password;
END
GO
```

Now, an execution plan for the SELECT query exists on the server before the query is executed. The plan only allows our original query to be executed. Parameter values (even if they are injected SQL) won't be executed because they are not part of the plan.

## 3. RELATED WORK

Many techniques have been proposed to prevent SQL injection Attacks for example, dynamic monitoring tools. Various SQLIA detection techniques for the application layer have been proposed in literature, but none of them pay enough attention to SQLIA in stored procedures by providing enough security. Many existing techniques, such as filtering, information-flow analysis, penetration testing, and defensive coding, can detect and prevent subset of vulnerabilities that lead to SQLIAs. A number of techniques are in use for securing the web applications. The most common way is the authentication process through the username and password. One of the major problems in the authentication process is the input validation checking. Most of the papers are restricting the user by not entering single quotes and double quotes in the user fields. Here in this paper we will concentrate on encrypt user entered data and passed to the stored procedure so that SQL Injection will be prevented.

Various techniques have been proposed for controlling SQL injection attacks, for example, Ke Wei [1] is using static analysis and runtime analysis in order to detect and prevent the SQL-Injections in stored procedures with the help of $SQLIA_{CHECKER()}$ function that identify the user input by the current session id and build a finite state automata.

In AMNESIA (Analysis and Monitoring for Neutralizing SQL-Injection attacks) (Halfond and Orso 2005) [4], the authors are using runtime checking of the query and declare it valid or malicious. AMNESIA checks query in different steps. In the first step it

identifies the "hotspot". Hotspots are application code which issues SQL query to database. Second, it forms a model for legitimate query in the form of NDFA (Non-Deterministic Finite Automata). Finally, as a request comes it checks the query with NDFA and declares it legitimate or malicious.

MeiJunjin [2] is using an approach for the detection of SQL injection vulnerabilities. The above mentioned author has used static, dynamic and automatic testing method for the detection of SQL Injection vulnerabilities. The approach traces user queries to vulnerable location. Although these techniques are effective, they cannot capture more general forms of SQLIAs that generate syntactically and type correct queries. Wassermann and Su combine static analysis with automated reasoning in [30] to detect tautologies in the dynamically generated SQL queries, but the other forms of SQLIAs would still succeed rendering the system vulnerable.

In [3], Indrani Balasundaram and E.Ramaraj have proposed a technique to prevent SQL-Injection attacks by using encryption but not in stored procedures. The use of stored procedures alone does not protect one against SQLIAs as is commonly assumed by most developers, but appropriate use of parameters along with stored procedures is necessary to achieve a minimal defense against such attacks [8] [11].

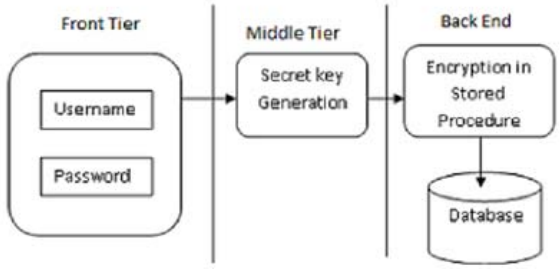
## 4. PROPOSED SYSTEM

We propose an SQL-Injection Attack prevention technique that addresses all types of SQLIAs. This technique works by combining encryption of user entered data within the stored procedure. The basis of such a technique is that effect of the malicious code can be avoided by using encryption algorithm.

After submitting the user registration form, a unique secret key will be generated corresponding to the given username and password. A stored procedure will be called by passing the following parameters:

- Username
- Password and
- Secret key

**Basic System Model**




Using AES_ENCRYPT() function username and password are encrypted with the secret key generated earlier. The encrypted values of username and password along with username are inserted into user table. The following query is used to insert these values.

***Insert into user_table values ('Username', 'Enc_Username', 'Enc_Password');***

Because of encryption in registration phase different attacks like "; drop table user" will be prevented.

During Login phase, up on submitting the user credentials, secret key will be generated using the same procedure used in registration phase. Stored procedure will be called by passing the following parameters:

- Username
- Password and
- Secret key

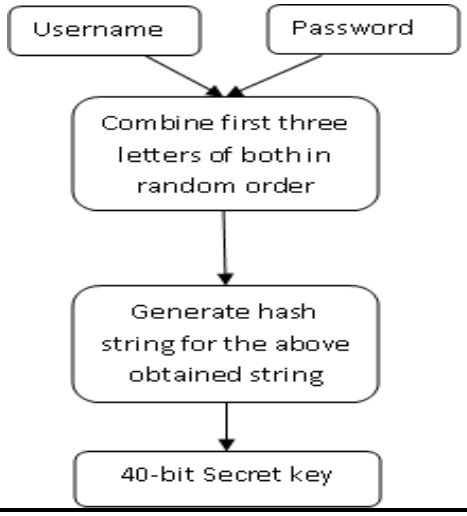
Again these values are encrypted as earlier and then used in the select query to check whether the given credentials are valid or not. The following is a query used to verify the user credentials.

***Select * from user_table where username=@Enc_Username and password=@Enc_Password;***

If user has entered malicious code like " ' or 1=1 - -", because of Encryption, they will be converted into some hash code.

Secret key will be generated by using the combination of the two fields, username and password. First three letters of username & password will combined in random order and then encoded to get a sequence of bits which is the secret key. This will be done dynamically without storing the secret key every time user enters username and password.

**Generation of Secret key**




**Typical Stored Procedure**

```
CREATE PROCEDURE VerifyUser
        @username varchar(50),
        @password varchar(50) ,
        @secret_key varchar(50)
AS
BEGIN
    DECLARE @sql nvarchar(500);

    @username=AES_ENCRYPT( @username,
    @secret_key );
    @password=AES_ENCRYPT ( @password,
    @secret_key );

    SET @sql = 'SELECT * FROM UserTable WHERE
    UserName =@username AND Password = @password;

EXEC (@sql);
END
GO
```

Vulnerability in Update query will be avoided by the use of Encryption. Whenever user enters " password' - - ", whatever he entered will be converted in to some other form, so that " - -" will not be executed.

***Update user_table set password=enc('password' - -) where username='user1';***

**Algorithms for the proposed system**

**Algorithm 1: Registration phase.**

***Inputs:*** *username, password entered by the user*
***Output:*** *encrypted username and password will be stored in database.*

1. *User enters his/her username and password and other registration information.*
2. *Pass this information to stored procedure[a] which will perform the following actions.*
3. *Generating unique secret key [b] corresponding to username and password.*
4. *Encrypt username and password using the generated secret key and AES encryption algorithm.*
5. *Store encrypted username and password along with username and password in the database.*

[a]: Implemented Stored procedure is not vulnerable to SQLIA. For this, instead of using dynamic SQL statement we will use prepared statement in JSP.
[b]: Secret key will be generated by using the combination of the two fields, username and password. First 3 letters of username & password will combined in random order and then encoded to get a sequence of bits which is the secret key.

**Algorithm2: Login phase**

***Inputs:*** *username, password entered by the user*
***Output:*** *encrypted username and password will be stored in database.*

1. *User enters his/her username and password.*
2. *Generate secret key [b] corresponding to username and password.*
3. *Encrypt username and password using the generated secret key and AES encryption algorithm.*
4. *Stored procedure will be called with parameters encrypted username and password.*

## 5. CONCLUSION

SQL Injection is one of serious security threat issues for the organizations and businesses operating on the web. Security of data is very important for every organization. SQL injection is a common technique hackers employ to attack underlying databases. The attack alters the SQL queries and behavior of the system for the benefits of hacker. In our proposed system the Encrypted username and password are used to improve the authentication process with minimum overhead. We propose a SQL Injection attack prevention technique that addresses almost all types of SQLIA's. The technique monitors all dynamically generated SQL queries associated with user input and captures the original structure of the SQL statement. Execution of malicious code can be avoided by using encryption algorithm. The advantage with the proposed system is that the vulnerabilities in stored procedures can be avoided. We need not to sanitize the user input and therefore not restricting the user from entering special characters. This is highly secured due to encryption algorithm.

## REFERENCES

[1] Ke Wei, Preventing SQL Injection attacks in Stored procedures. *Australian Software Engineering Conference,(2006).*
[2] MeiJunji, An approach for SQL injection vulnerability detection. *Sixth International Conference on Information Technology, (2009)*
[3] Indrani Balasundaram An Authentication Mechanism to prevent SQL Injection Attacks, *International Journal of Computer Applications Volume 19– No.1, April 2011.*
[4] Halfond, W. G. J. and A. Orso (2005). AMNESIA: analysis and monitoring for Neutralizing SQL-injection attacks. . *ASE'05.* Long Beach, California, USA.
[5] W. G. J. Halfond and A. Orso. Combining static analysis and runtime monitoring to counter sql-injection attacks. *WODA*, 2005.
[6] G.Wassermann and Z. Su. An analysis framework for security in web applications. *SAVCBS*, 2004.
[7] Shaukat Ali, Azhar Rauf, and Huma Javed, 2009. "SQLIPA: An Authentication Mechanism Against SQL Injection," European Journal of Scientific Research, ISSN 1450-216X Vol.38 No.4, pp 604-611.
[8] H. to: Protect from SQL Injection in ASP.NET. *http://msdn.microsoft.com/library/default.asp?url=/library/enus/dnpag2/html/PAGHT000002.asp.*
[9] Sruthi Bandhakavi and Prithvi Bisht Preventing SQL Injection Attacks using Dynamic Candidate Evaluations, Alexandria, Virginia, USA, 2007
[10] Etienne Janot, Pavol Zavarsky, Preventing SQL Injections in Online Applications: Study, Recommendations and Java Solution Prototype Based on the SQL DOM.
[11] C. Cerrudo. Manipulating Microsoft sql server using sql injection. *http://www.appsecinc.com/presentations/Manipulating SQL Server Using SQL Injection.pdf*, White Paper.
[12] Ntagwabira Lambert and Kang Song Lin, Use of Query Tokenization to detect and prevent SQL-Injection attacks, IEEE 2010.
[13]http://www.andhrahackers.com/forum/sql-injection/sql-injection-tutorial/, Basic Sql injection types
[14]http://palisade.plynt.com/issues/2006Jun/injection-stored Procedu res/ Stored Procedures
[15] https://www.owasp.org/index.php/Avoiding_SQL_Injection Avoiding SQL_injection Attacks.
[16]http://www.darkreading.com/databaseSecurity/167901020/security/application-security/227300073/five-ways-to-stop-mass-sql-injectio n-attacks.html; basic concepts of SQL-Injection and Avoiding SQL-Injection attacks.
[17]http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14258/d_crypto.htm; Encryption in stored procedures
[18]http://dev.mysql.com/doc/refman/5.5/en/encryption-functions.ht ml; Encryption functions