Asian Journal of Information Technology 15 (22): 4445-4449, 2016
ISSN: 1682-3915

Automatic Prevention of Union Query Type SQL Injection Attack Using Private Synonym and Error Message Controller

[1]N. Gunaseeli and [2]D. Jeya Mala
[1]Department of MCA, K.L.N College of Engineering, Pottapalayam, Tamil Nadu, India
[2]Department of MCA, Thiagarajar College of Engineering, Madurai, Tamil Nadu, India

Abstract: Web applications are software applications which allow the end users to access the most valuable services like credit card services, purchase orders, online booking services and so on. The developers of the web applications pay more concentration on developing the features and functionality of the applications. They spend only little amount of time to secure web applications. Unfortunately, the web applications are vulnerable to various threats like SQLIA, cross site scripting, buffer overflow, etc. Despite, the web applications are vulnerable to many kinds of threats and attacks, SQLIA (SQL injection attack) is the most vulnerable to web applications. It is a kind of attack where malicious users try to access the database layer of an application through crafted input query strings. Ignoring the existence of these kinds of attacks leads to various kinds of SQLIA. One among them is union queries SQL injection attack. Through this attack, an attacker gets the result set of original query along with the result set of injected query. This study analyzes the weaknesses of union query SQL injection attack and proposes a novel approach to prevent the union query at run time.

Key words: Database security, SQLIA, web application, security threats, run time monitoring

INTRODUCTION

Web applications are prevalent due to the increased sharing of information through web browsers. Now a days, it becomes an essential technology to business people for delivering services to their clients. Despite, web applications provide flexible services to end users, it is often attacked by hackers directly. There are various kinds of attacks are faced by web applications. They are SQLIA, buffer overflow, cross site scripting, denial of service attack, memory corruption, data breach etc. Despite the web applications are vulnerable to many kinds of threats and attacks, SQLIA (SQL Injection Attack) has been described as one of the most vulnerable threats to web applications (Halfond *et al*., 2008).

SQL injection attack: SQL injection is a code injection technique that exploits a security vulnerability occurring in the database layer of an application (Halfond *et al*., 2008). The reason for SQLIA is improper validation of user input. If user input is not properly validated, attackers may be able to change the developer's intended SQL command by inserting new SQL keywords or operators through specially crafted input strings. The results of these attacks are often disastrous and can range from leaking of sensitive data like customer data to the destruction of database contents (Halfond *et al*., 2008).

Classification of SQLIA: There are many forms of SQLIA. Depends upon the type of SQLIA attack, the attacker can get different access like collecting data about the tables, fields of database, destroying the database, converting channels etc. SQLIA has been classified into five basic classes with respect to the attacker's target and the vulner abilities in web applications (Sun *et al*., 2007) (Patel *et al*., 2011) A classification of SQLIA and the attacker's achievements are illustrated in Table 1.

Union query: This is a type of SQLIA attack in which the attacker gets additional data from crafted query. To get the additional data, the attacker uses the key word "union" in order to join the malicious query with original query. The crafted query brings the results of the original query along with the results of malicious query (Halfond *et al*., 2008; Dharam and Shiva, 2012; Kermalis and Tzouramanis, 2008; Buehrer *et al*., 2005). For example, let's consider the following query.

```
SELECT acct FROM account WHERE login =' ' UNION
    SELECT debitcardNo FROM Cardlist WHERE
    acctno=1001 – AND pin =
```

Corresponding Author: N. Gunaseeli, Department of MCA, K.L.N College of Engineering, Pottapalayam, Tamil Nadu, India

Table 1: A classification of SQLIA and the attacker's achievements

Types of SQLIA	Achievement of attakcer
Union and union all query	The actual runtime query returns null data. However, the injected query generates data from the database
Piggy backed queries	The database server executes the second injected query. Thus, a harmful operation may also be performed on the database with such injected query(s)
Alternate encodings	The attackers easily access the data from database by using alternate codlings like ASCII, hexadecimal etc
Malformed queries	Perform SQLIA, the attackers need some prior knowledge of database schema, which is often unknown. Malformed queries allow for overcoming this problem by taking advantage of overly descriptive error messages that are generated by the database
Bypass authentication	Two dashes, comment the remaining text. Expression 1=1 is always true. User will be logged in with privileges of the first user stored in the database

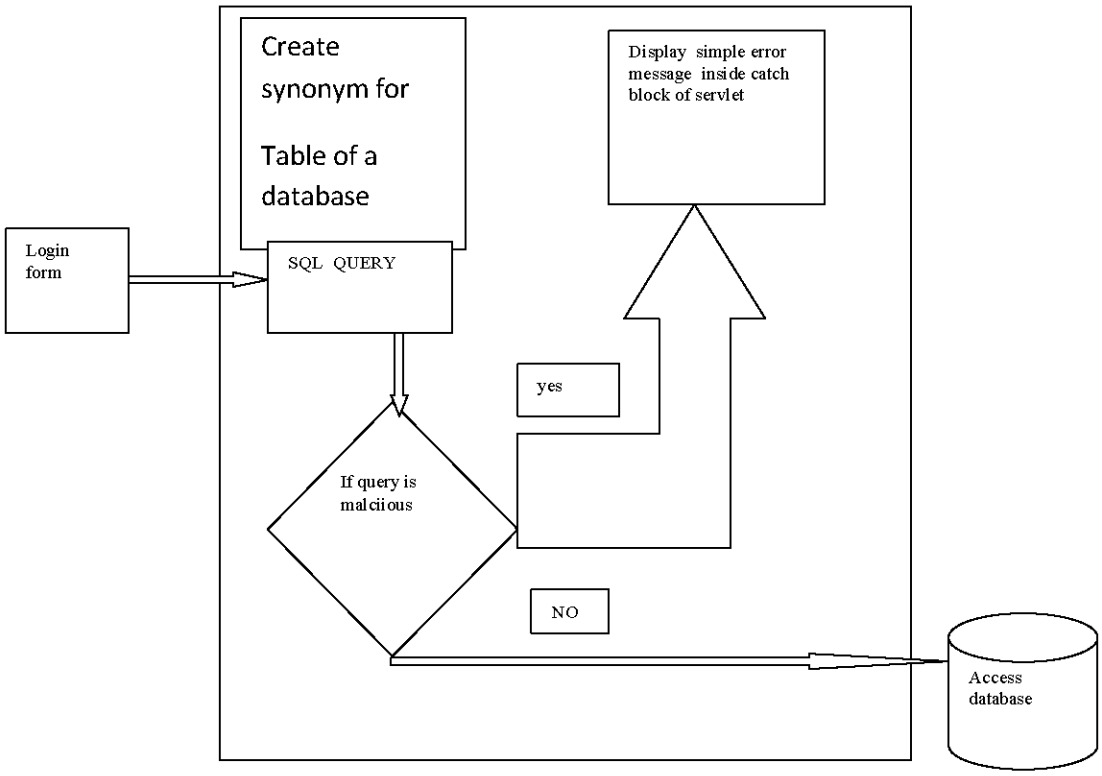

Fig. 1: Shows the flowchart of proposed system

The above query returns the null set along with the data from cardlist. Therefore, the database displays the results of these two queries and returns to the application.

Proposed approach: After reviewing the SQLIA and its classification we proposed a new technique for preventing Union query type attack. The backbone of the proposed technique is private synonym and error controller. In this section, we discuss the strength of private synonym and error controller (Fig. 1).

MATERIALS AND METHODS

Private synonym approach and error message controller: A private synonym helps to create an alternate name for user defined table. This strength helps to hide the table identity. Therefore, it is harder for a malicious user to target the underlying tables. One more advantage of synonym is that it is an alternate name for a table. Therefore it does not require any storage space other than it definition.

We apply the private synonym technique because the major drawback of union query is that it can be performed after knowing the table names of a database. Since, we apply the synonym of the tables in the query of the web applications, the malicious user can't reach the database, even though they know the name of the tables of a database (Fig. 1).

The detailed error message of an application leads to know the details of tables, data types of database etc. Therefore, we control detailed error message through catch block of servlet program. We implemented the

approach in such a way that it should display only simple and general messages so that the attackers should not know the tables, fields of a database.

Algorithm of Private synonym approach

Step 1: To create a private synonym for a table, we must activate the CREATE ANY SYNONYM system privilege.
Step 2: connect to the database.
Step 3: create a table using data definition Language.
Step 4: create an alias or synonym for the table created using data definition language.
Step 5: Use the synonym in the query utilized in the web application.
Step 6: If error occurs, display error messages without revealing table names, fields of the database

RESULTS AND DISCUSSION

Performance evaluation: To evaluate the proposed system, a web based college management application was developed. The application had been deployed on glassfish server with oracle as database. The application is executed in the Netbeans IDE. Then, the performance of the proposed system is analyzed for union query based SQL injection attack. The proposed system is tested with the following steps creation of synonym for a table of a database creation of web application using JSP and servlet testing the web application using malicious query and legitimate query .

Creation of synonym for a table of a database: To activate private synonym in a schema, we must provide create any synonym system privilege to administrator or developer. After the activation of create synonym, the developer must connect to the database. In the next step the developer has to create a new table. For Example, create table debitcard (cardnumber varchar, cardholdername varchar; After creating the table, the developer has to change the name of the table using private synonym. For example create synonym list for debit card. Therefore, table name has been changed due to synonym.

Creation of web application using jsp and servlet: To evaluate the proposed system, we developed a web based college management system. In this web application, we created a client page which consists of login form. The login form gets input from the user. When we click the submit button, the inputs are transfer to servlet. The servlet consists of database connectivity code. When we write the query string, we used synonym instead of tables. This protects the tables from malicious user.

Creation of client page using JSP:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title> PREVENTION OF UNION QUERY TYPE SQL INJECTION ATTACK USING PRIVATE SYNONYM and Error controller </title>
 </head>
 <body>
<h1>PREVENTION OF UNION QUERY TYPE SQL INJECTION ATTACK USING PRIVATE SYNONYM and error  controller</h1>
   <form action="NewServlet">
 <input type="text" name="username"/>
          <input type="text"  name="password"/>
      <input type="submit" value="enter"/>
   </form>
</body>
</html>
```

Creation of servlet page using JSP:

```
mport java.io.IOException;
 import java.io.PrintWriter;
 import javax.servlet.ServletException;
 import javax.servlet.annotation.WebServlet;
 import javax.servlet.http.HttpServlet;
 import javax.servlet.http.HttpServletRequest;
 import javax.servlet.http.HttpServletResponse;
 import java.sql.Connection;
 import java.sql.DriverManager;
 import java.sql.ResultSet;
 import java.sql.Statement;
 /** * *@author gunaseeli */
 @WebServlet(name = "NewServlet", urlPatterns = {"/NewServlet"})
 public class NewServlet extends HttpServlet {
          protected void processRequest(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();try{
        String u= request.getParameter("username");
          String p= request.getParameter("password");
  //step1 load the driver class
 Class.forName("oracle.jdbc.driver.OracleDriver");
  //step2 create  the connection object
 Connection con=DriverManager.getConnection(
 "jdbc:oracle:thin:@localhost:1521:card","system","oracle");
  //step3 create the statement object
 Statement stmt=con.createStatement();
  //step4 execute query
String s= SELECT * FROM login WHERE username='u' AND password='p';
 ResultSet rs=stmt.executeQuery(s);
 If (rs!=NULL)
  <jsp:forware page="authenticate.jsp">
 //step5 close the connection object
 con.close();
}catch(Exceptione)  { System.out.println("page could not displayed");}  }
}
```

Testing the web application: To test the web application for malicious union query, we give the following inputs to login form. The outcome of the application is displayed in the following Table 2.

Since the malicious query contains original table name of a database, the execution of query inside web application leads to SQL error. Therefore, execution is move from try block to catch block. To control the error message, we do not encourage to display the detailed error message. This detailed error message leads to know

Table 2: Shows the response output when we give both legitimate and malicious inputs are given

Username	Password	Query string	Expected output	Actual output
Apple	Orange	Select * from login where username='Apple' and password='orange'	Since it is a legitimate query, it should go to next authenticated page	It goes to authenticated page
Apple	Orange union select * from debitcard	select * from login where username = 'apple' and password='orange' union select * from debitcard';	Since the original name of table "debitcard" is changed to "list" execution of the query leads to error. Therefore, the output should displayed as "The page could not displayed"	The page could not displayed

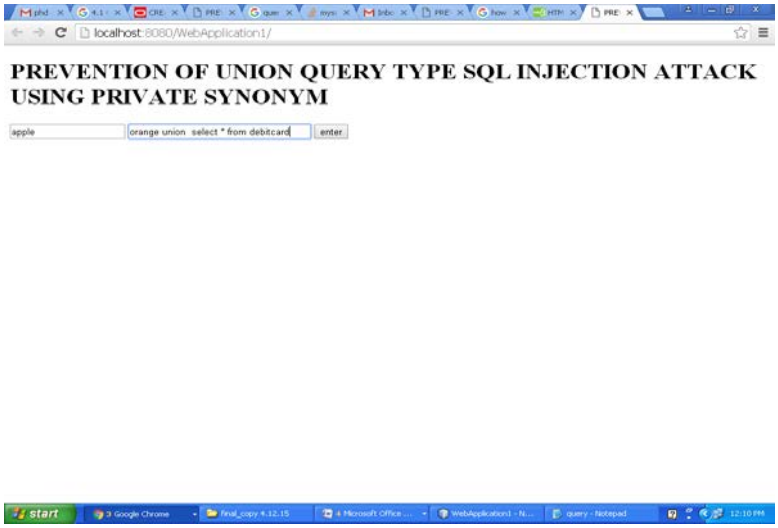

Fig. 2: The malicious input from a user

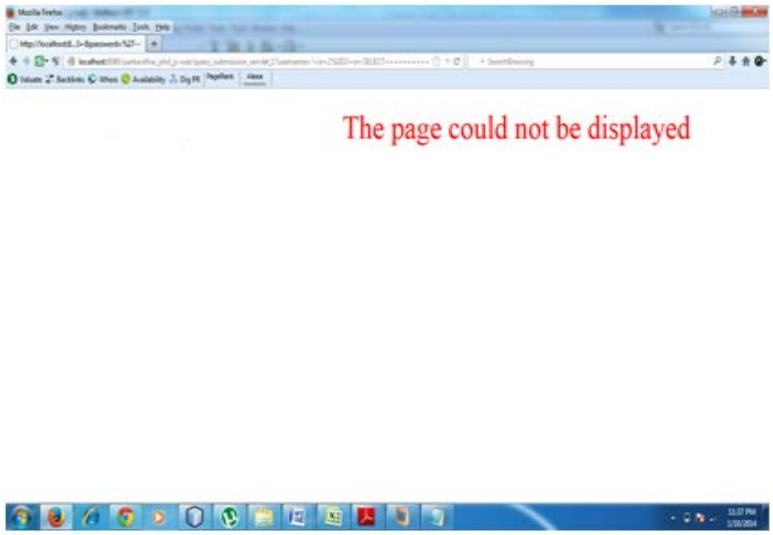

Fig. 3: The response when a malicious input is given

the details of tables, data types of database. Therefore, we prefer simple and general messages like. "The page could not displayed" (Fig. 2 and 3).

CONCLUSION

This study presented a new technique for preventing Union query type SQLIA attack based on the novel

concept of private synonym and controlling error messages. The proposed system contributes the following: there is no need to store the query structure before deployment. No need to change source code of the web application. An evaluation of the technique shows the effectiveness and efficiency. We also study the limitations of our approach. We have tested only oracle database and planned to test more number of databases in future. We implemented the tool using java which is platform independent. In future, we planned to implement it using dotNet framework.

REFERENCES

Buehrer, G., B.W. Weide and P.A.G. Sivilotti, 2005. Using parse tree validation to prevent SQL injection attacks. Proceedings of the 5th International Workshop on Software Engineering and Middleware, September 5-6, 2005, Lisbon, Portugal, pp: 106-113.

Dharam, R. and S.G. Shiva, 2012. Runtime monitors for tautology based SQL injection attacks. Proceedings of the 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), June 26-28, 2012, IEEE, Tennessee, USA, ISBN: 978-1-4673-1425-1, pp: 253-258.

Halfond, W.G.J., A. Orso and P. Manolios, 2008. WASP: Protecting web applications using positive tainting and syntax-aware evaluation. IEEE Trans. Software Eng., 34: 65-81.

Kemalis, K. and T. Tzouramanis, 2008. SQL-IDS: A specification-based approach for SQL-injection detection. Proceedings of the 2008 ACM Symposium on Applied Computing, March 16-20, 2008, Fortaleza, Ceara, Brazil, pp: 2153-2158.

Patel, N., F. Mohammed and S. Soni, 2011. SQL injection attacks: Techniques and protection mechanisms. Int. J. Comput. Sci. Eng., 3: 199-203.

Sun, S.T., T.H. Wei, S. Liu and S. Lau, 2007. Classification of SQL Injection Attacks. University of British Columbia, Vancouver, British Columbia,.