A Detection and Prevention Technique on SQL Injection Attacks

Zar Chi Su Su Hlaing
Faculty of Information Science
University of Computer Studies (Magway)
Magway, Myanmar
zarchissh@gmail.com

Myo Khaing
Faculty of Computer Science
University of Computer Studies (Maubin)
Maubin, Myanmar
myokhaingucsm@gmail.com

Abstract

With the web advancements are rapidly developing, the greater part of individuals makes their transactions on web, for example, searching through data, banking, shopping, managing, overseeing and controlling dam and business exchanges, etc. Web applications have gotten fit to numerous individuals' day by day lives activities. Dangers pertinent to web applications have expanded to huge development. Presently a day, the more the quantity of vulnerabilities will be diminished, the more the quantity of threats become to increment. Structured Query Language Injection Attack (SQLIA) is one of the incredible dangers of web applications threats. Lack of input validation vulnerabilities where cause to SQL injection attack on web. SQLIA is a malicious activity that takes negated SQL statement to misuse data-driven applications. This vulnerability admits an attacker to comply crafted input to disclosure with the application's interaction with back-end databases. Therefore, the attacker can gain access to the database by inserting, modifying or deleting critical information without legitimate approval. The paper presents an approach which detects a query token with reserved words-based lexicon to detect SQLIA. The approach consists of two highlights: the first one creates lexicon and the second step tokenizes the input query statement and each string token was detected to predefined words lexicon to prevent SQLIA. In this paper, detection and prevention technologies of SQL injection attacks are experimented and the result are satisfactory.

Keywords—*SQL Injection Attack, Web applications, Malicious activity, Vulnerabilities, Input validation*

I. INTRODUCTION

Web application is one of the most mainstream communication streams with the rapid development of web advances. Information is a significant job in data frameworks. Many associations run their transactions on database appended web applications to get information from clients. Web application is a significant wellspring of data for any organization to get business process basic information and broadly utilized in different applications. With the ubiquity of web applications, there are numerous security issues in the web world and furthermore increment in web application vulnerabilities. SQL injection is software vulnerabilities in web applications which is brought about by absence of information approval. The information approval weakness is where user input is utilized in the product without affirming its legitimacy. SQL represents Structured Query Language, which is the standard programming language for creating social databases. It an order and control language utilized in the making, altering, erasing, and recovering the information and structures that involve the social database framework. SQL injection is one of the most serious dangers to the security of backend database from driven applications.

SQL injection is an assault method with negated SQL articulations used to abuse how site pages speak with back-end databases. It can take a shot at defenseless website pages that adventure a backend database like MySQL, Oracle and MSSQL. Attackers can give directions (made SQL explanations) to a database utilizing input fields on a site. Figure 1 shows the procedure of SQLIA. These directions control a database server behind a web application to get self-assertive information from the application, meddle with its rationale, or execute directions on the database server itself. Along these lines, the impacts of SQLIA are side step verification, extricating information, loss of privacy and respectability.

SQL injection is strikingly like a XSS. The essential contrast being that a XSS attack is executed at the web front end, though the SQL assault is executed at the server. The issue in the two cases is that client input was never approved appropriately.

There is assortment of methods are accessible to identify SQLIA. The most favored are Web Framework, Static Analysis, Dynamic Analysis, consolidated Static and Dynamic Analysis and Machine Learning Technique. Web Framework gives separating way to deal with channel exceptional characters however different assaults are not recognized. Static Analysis checks the information parameter type, yet it neglects to identify assaults with right info type. Dynamic Analysis method is equipped for

checking vulnerabilities of web application however can't distinguish a wide range of SQLIA. Joined Static and Dynamic Analysis incorporates the advantage of both, yet this technique is unpredictable so as to continue. AI strategy can recognize a wide range of assaults however results in number of bogus positives and negatives.

SQLIA can be presented with the following segment of vulnerable Java code:

```
String uname= request.getParameter("uname");
String pword= request.getParameter("password");
String sql_query= "SELECT name FROM member
WHERE username=' "+uname+" ' AND password= '
"+pword+" ' ";
Statement stmt=connection.createStatement( );
ResultSet rset=stmt. executeQuery(sql_query);
```

In this code, string variable sql_query is used for keeping the cunning SQL query statement that is being executed in the database.

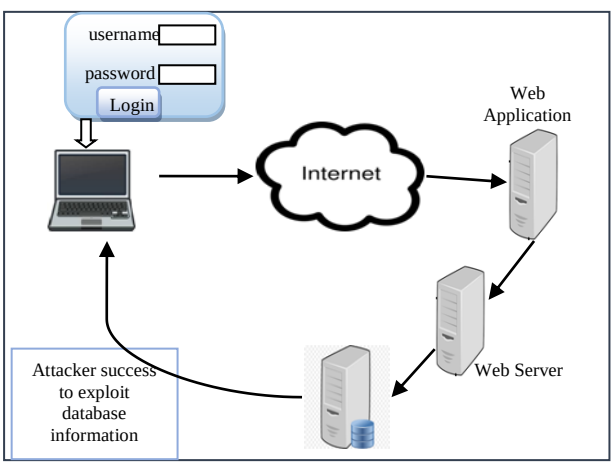

Figure 1. How SQL Injection Attack Works

II. RELATED WORKS

N. Lambert et al. [1][5] proposed a model that uses a tokenization technique to detect SQL injection attacks, so query containing Alias, Instances and Set operations can also be blocked at the entry point. It checks whether the generated query based on user's input its intended result, and compare the results by applying tokenization technique on an original query and input query. If the results are same, there is no injection, otherwise it is present. I. Balasundaram et al. [2] proposed a technique for SQLIA using ASCII based string matching. This technique is used to check the user input field with static and dynamic analysis to detect and prevent SQLIA.

M. Kumar et al. [4] and William G.J. Halfond et al, G.Yiğt, M. Arnavutoglu [14], discussed the detection and prevention techniques of SQL injection attacks and analyze existing techniques against such attacks. B.J.S. Kumar & P.P. Anaswara [15] experimented on detection and prevention of SQL injection attack. D. Kilaru [12] observed how SQL injection occurs and how to update a web app with SQL injection vulnerability. R. M. Nadeem et al. [13] proposed a system which is based on dynamic analyzer model. This model received the user request and analyzed to check that request is for pages without vulnerabilities (P') and with vulnerabilities (P), with help of knowledge base. J.O.Atoum and A.J.Qaralleh [16] described static and runtime SQL queries analysis called hybrid techniques which is to create a defense strategy that can detect and prevent various types of SQLIA.

III. TYPES OF SQLIA

In web-based applications, most of work is to access data from databases. If the user input data is not properly performed or validated, users can access information they were not supposed to get access to. The following techniques are types of SQLIA.

A. Tautologies

In tautology based attack, the general goal of attacker is to input crafted SQL tokens that cause the conditional query statement to be evaluated always true. An attacker undertakes an input field with vulnerability that is utilized in the query's WHERE predicated and transform the condition into a tautology which is always true. This predicate logic is assessed as the database examines each tuple in the table. If the predicate logic at WHERE clause is evaluated as a tautology, the database match and returns all tuples in the table rather than associating only one tuple, as it would normally do in the sense of injections. This type of attack proceeds to bypass authentication and extract data [4, 7, and 14].

Smith ' OR 'a'='a	SELECT name FROM member WHERE username='Smith' AND password= ' ' OR 'a'= 'a'
' OR 'a'='a ' OR 'a'='a	SELECT name FROM member WHERE username =' ' OR 'a'='a' AND password=' ' OR 'a'='a'
'OR 'a'='a' - -	SELECT name FROM member WHERE username =' ' OR 'a'='a' - - ' AND password=' '

This query statement is always true because it have been added by the tautology statement ('a'='a'). Double dash "--" instructs the SQL parser that the rest of statement is a comment and should not be executed

B. Malformed Queries

In this approach, when the attacker abuses an illegitimate or deficient SQL token, the rejected error message is come back from the database including helpful debugging information. The error message causes assailants to precisely distinguish which parameters are vulnerable in an application and the total outline of the underlying database. This situation was misused by assailant's crafted SQL tokens or garbage input that causes syntax mistake, type jumble or logical error into the database. To recognize injectable parameters, syntax errors can be utilized. Type errors can be applied to conclude the information kinds of specific attribute or to remove information. Logical errors can be out the table names and attribute names that cause the mistake or error [9].

C. Union Query

In this technique, attackers injected invalid statement is joined with the valid query by the utilizing UNION keyword. Attackers misuse a query statement of the structure "UNION <injected query>" as far as possible of the end of legitimate statement. It makes the application return information from the results of original query and furthermore information from another table. Then, the statement written the double dash "- -" will be comment out [11].

```
SELECT name FROM member WHERE username="UNION SELECT password FROM member WHERE username='admin' -- AND password="
```

In this query, the original query returns null set whereas the exploited query statement returns data from the same table.

D. Piggy-backed Queries

In the piggy-backed query-based attack, an attacker attempts to add extra queries into the first inquiry string. It abuses database by the query delimiter, for example, ";" to add additional query to the first query. In the event that the attack is fruitful, the database gets and executes a query statement that contains numerous particular inquiries. The first query is the original legitimate query, which is to execute the database whereas the second query, malicious query is to misuse the database [4].

```
SELECT name FROM member WHERE username='Smith' AND password="; DROP table users - -
```

The two queries were separated by the delimiter, ";", and both were executed. The second query makes the database fails to client table information. Different sorts of queries can be executed with this technique, for example, addition of new clients into the database or execution of stored procedures. It is worth nothing that numerous databases don't require a special character to isolate distinct queries, so basically examining for an exceptional or special character isn't an effective way to prevent this type of attack.

E. Inference

In inference-based technique, attackers make queries that cause an application or database to act contrastingly based on the consequences of the query. There are two well-known assault strategies that depend on inference: blind injection and timing attacks.

In blind injection, developers omit detail information of error messages. These messages assist attackers to exploit to the databases. In this case, Attackers are trying to exploit the database with the vulnerable query statement that has a boolean result. Then they analyze differences based on the applications responses.

In timing attacks, attackers collect data from database by detecting timing delays in the database's responses. It depend on the database pausing for a specified time limit, then returning the information that is indicating successful query executing [8].

```
SELECT name FROM member WHERE username= IF(((SELECT UPPER(MID(password,1,1)) FROM member WHERE username='admin') ='A'), SLEEP(5),1)
```

F. Stored Procedure

In this approach, stored procedures are victims for attackers to exploit database. Stored procedures are codes that are stored in the database and execute directly by the database engine. To activate SQLIA, attackers can create injected text to exploit this stored procedure as

```
SELECT name FROM member WHERE username='; SHUTDOWN; - - password="
```

IV. PREVENTION TECHNIQUES

A. Prepared Statement

One of the best ways to prevent from SQL injection is to use prepared statement instead of statement. The problem of SQL injection is that user's input is used as part of the SQL statement. By using prepared statement, the SQL statement uses a parameter to insert a value into the database. Instead of inserting the values directly into the statement, thus prevent the backend database from running invalidated SQL queries that are unsafe and harmful to the database.

B. Using Stored Procedures

Stored Procedures adds an additional security layer to the database other than utilizing Prepared Statements. It performs the getting away from required so that the application takes input as data to be worked on instead of SQL code to be executed. The distinction between prepared statement and stored procedure is that the SQL code for a stored procedure is composed and stored in the database server, and then called from the web application.

If user access to the database is just at any point allowed by means of stored procedure strategies, permitted access for user to legitimately get to data doesn't not need to be explicitly granted on any database table. Along these lines, the database is still safe.

C. Validating User Input

In validating, user supplied input should be used after confirming its validity. So, input validation is first to ensure the user supplied input value is of the accepted type, length, format, etc. Only the input which passed the validation process prevent data from information sources, database.

D. Limitting Privileges

Limiting privileges is the concept of restricting resources from user's accesses. When you do not need to access the important part of the database, don't connect to your database using an administrator account because the attackers might have access to the entire source of system. Therefore, to identify the authenticated user there should be used an account with limited privileges to limit the extent of harms in the occurrence of SQL Injection.

E. Encrypting Data

Unencrypted data stored in database can be stolen if missing authorization or invalidated input allows users to read the data. If attackers try to gain access to database and its table, the encrypted data value will prevent attacker to read sensitive data and any further changes to databases would have no effect.

V. PROPOSED APPROACH

There are many different techniques for detecting the SQL injection attacks. The proposed technique is based on sanitizing the query statement. This approach consists of two steps: the first one is tokenizing the user inputted query. The tokenization process is made by detecting a white space, double dashes (--), sharp sign (#) and all strings before each symbol are tokens. In second step, after the query is tokenizing, each string token was detected with the contents of predefined lexicon. The contents of lexicon are most of reserved words (commands) and some logical operators. The contents of lexicon are collected most of injected commands or words in SQL injection attacks. The following table, Table I describes some words of lexicon's contents. There are 20 words in it. When the input query statement enters, whether to detect injection or not. During execution, the inputted data are matched with the corresponding lexicon's contents to check for validity. If there are matched to any other words, there is an attempt to SQL injection attack. If no, there is not an injection.

TABLE I. SAMPLE CONTENS OF LEXICON

No	Injected command
1	alter
2	concat
3	drop
4	delete
5	execute
6	sleep
7	shutdown
8	union
9	or
10	if

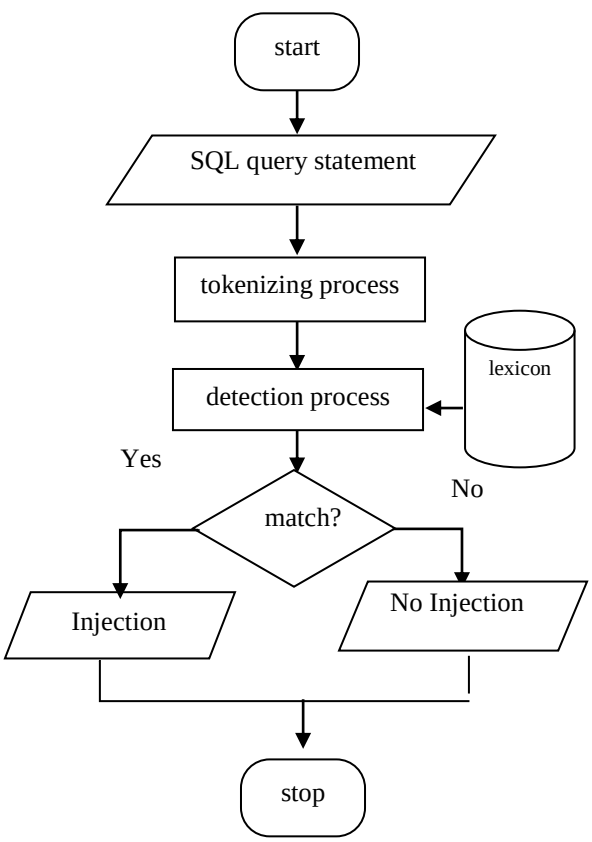

Figure 1.Flow of Proposed Approach

TABLE II. ALGORITHM FOR PROPSED SYSTEM

Algorithm: Detection

```
begin
Input:  SQL query statement
file= read contents in lexicon
N=Tokenize the query statement
flag=false
while (!eof(file))
{
        flag= false;
         for (int i=0; i< N; i++)
            if (token[i]== word in lexicon)
                flag=true;
}
if (flag) print “Injection detected”;
else print “No injection”;
end
```

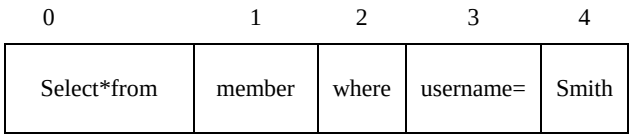

Figure 2. Tokening result without injection

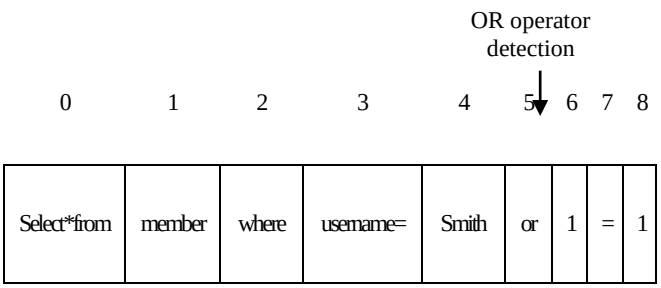

Figure 3.Tokening result with injection

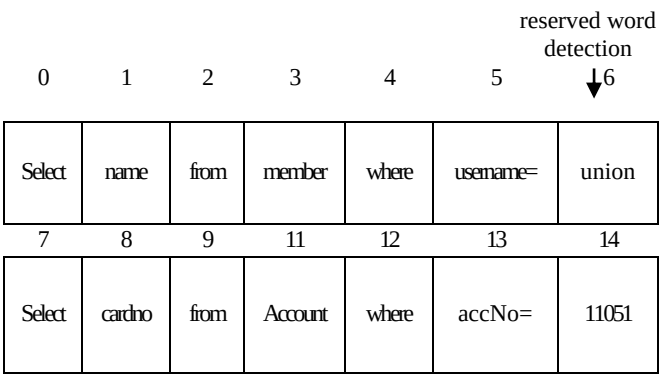

Figure 4. Tokening result with injection

The main concept of this approach is to detection to SQLIA by searching each token in predefined word of lexicon which causes to WHERE condition is always true.

VI. EXPERIMENT

In this section, the system performed some SQL injection queries on vulnerable query statement. The system tested 10 SQL query statements. These are 3 normal statements and 7 injection statements. The result outcomes describe in Table IV. The false positive and true negative scores are 0 and 7 respectively and accuracy is very good. The following table, TABLE V presents the outcomes of analysis which have evaluated. Therefore, the proposed system is done for successful prevention from various malicious query for injections.

```
String uname= “alice”;
String pass= “alice123”;
String query= "SELECT *  FROM member WHERE username=' "+uname+' " AND password= ' "+pass+" ' ";

uname= “alice”;
pass= “ ' or '1=1 ”;
String query_bad= "SELECT *  FROM member WHERE username=' "+uname+' " AND password= ' "+pass+ " ' ";
```

Display Query Statement

```
Normal:
select * from member where username= 'alice' and password= 'alice123'

Injection:
select * from member where username= 'alice' and password= '' or '1=1'
```

The normal query statement is no problem, as the proposed system will get data from this *member* table that satisfy the predicate. However, the approach detects the injection attack with crafted SQL statement.

```
String uname= ''; DELETE FROM member WHERE 1 or username = '';
String query= "SELECT *  FROM member WHERE username=' "+uname+" ';
```

Display Query Statement

```
Injection: SELECT *  FROM member WHERE username=''; DELETE FROM member WHERE 1 or username = ' '
```

When the system run this query, the injected delete statement would completely empty the *member* table. This system also detects the statement before to execute.

TABLE III. SQL QUERY STATEMENT

input	SQL statement
smith 123	SELECT * FROM member WHERE username ='smith' AND password ='123'
' or '1=1 ' or '1=1	SELECT * FROM member WHERE username =' ' or '1=1' AND password =' ' or '1=1'
smith ' or 'a'='a	Select * from member where username ='smith' and password =' ' or 'a '=' a'
' or ''=' ' or ''='	SELECT * FROM member WHERE username =' ' or ' '=' ' AND password =' ' or ' '=' '
smith ' or ''='	SELECT * FROM member WHERE username ='smith' AND password =' ' or ' '=' '
' or '1=1'-- 123	SELECT * FROM member WHERE username ='' or '1=1' --' AND password ='123'
'"; DELETE FROM member WHERE 1 or username = '";	SELECT * FROM member WHERE username=' '; DELETE FROM member WHERE 1 or username = ' '
"; SHUTDOWN; - -	SELECT name FROM member WHERE username="; SHUTDOWN; - - password="
Smith "; DROP table users - -	SELECT name FROM member WHERE username='Smith' AND password="; DROP table users - -
john john123	SELECT * FROM member WHERE username = 'john' AND password ='john123'
blake blake123	SELECT * FROM member WHERE username = 'blake' AND password ='blake123'

TABLE IV. OUTCOMES OF QUERY STATEMENTS

		Prediction		Total
		normal	*injection*	
Actual	*normal*	3	0	3
	injection	0	7	7
		3	7	10

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

TABLE V. EXPERIMENT OUTCOMES

SQLIA Techniques	Proposed approach's outcomes
Tautologies	Successful prevention
Malformed queries	Successful prevention
Union queries	Successful prevention
Piggy-back queries	Successful prevention
Inference	Successful prevention
Stored procedure	Successful prevention

VII. CONCULSION

SQLIA is one of the dominant threats to web application. Web applications need to protect their database from varying number of threats in order to provide confidentiality and integrity. In SQLIA, intruders allow attacking with a crafted query statement through a web input form into the system and then theft identity, access to sensitive information, and tamper with existing data, which can cause many disastrous effects. This paper is presented on the techniques of SQLIA and prevention approaches. The proposed approach is used for the detection and prevention of SQL injection and also suitable the outcomes.

ACKNOWLEDGMENT

I would like to express my deepest thanks to all my teachers for their valuable advice, helpful comments, and precious time for this research. Most

importantly, none of this would have been possible without the love and patience of my family throughout the process. My heartfelt thanks also extend to all my colleagues and friends for their help, interest and valuable hints for discussions about this.

REFERENCES

[1] N. Lambert, K.S. Lin; "Use of Query tokenization to detect and prevent SQLinjection attacks", *Proceedings of the 3rd International Conference on Computer Science and Information Technology (ICCSIT)*, Chengdu, China:IEEE (2010). pp: 438-440, 2010.

[2] I. Balasundaram, E. Ramaraj, "An Efficient Technique for Detection and Prevention of SQL Injection Attack using ASCII Based String Matching", International Conference on Communication Technology and System Design, Prodedia Engineering, pp. 183-190, 2012.

[3] Dr. R. Shettar, A. Ghosh, A. Mohan, A. Pramod, C. Raikar, " SQL Injection Attacks and Defensive Techniques", International Journal of Computer Technology & Applications, vol. 5, no. 2, pp. 699-703, March-April 2014.

[4] M. Kumar, L. Indu, "Detection and Prevention of SQL Injection Attack", International Journal of Computer Science and Information Technologies, vol. 5, no. 1, pp. 374-377, 2014.

[5] A. Kumar, S. Bhatt, "Use of Query Tokenization to Detect and Prevent SQL Injection Attacks", International Journal of Science Technology & Engineering, vol. 2, issue. 01, pp. 97-103, July 2015.

[6] RubidhaDevi.D, R. Venkatesan, Raghuraman.K, "A Study on SQL Injection Techniques", International Journal of Pharmacy & Technology, vol. 8, issue. 4, pp. 22405-22415, December 2016.

[7] A. Gupta, Dr. S. K. Yadav, "An Approach for Preventing SQL Injection Attack on Web Application", International Journal of Computer Science and Mobile Computing , vol.5, issue. 6, pp. 01-10, June 2016.

[8] M. Ŝtamper, "Inferential SQL Injection Attacks", International Journal of Network Security, vol 18, no. 2, pp. 316-325, Mar 2016.

[9] Sonakshi, R, Kumar, G. Gopal, "Case Study of SQL Injection Attacks", International Journal of Engineering Science & Research Technology, pp. 176-189, July 2016.

[10] Z. S. Alwan, M. F. Younis, "Detection and Prevention of SQL Injection Attack: A Survey", International Journal of Computer Science and Mobile Computing, vol. 6, issue. 8, pp. 5-17, August 2017.

[11] G. Yigit, M. Amavutoglu, "SQL Injection Attacks Detection & Prevention Techniques", International Journal of Computer Theory and Engineering, vol. 9, no. 5, pp. 351-356, October 2017.

[12] D. Kilaru, "Improving Techniques for SQL Injection Defenses", University of Colorado Colorado Springs, 2017.

[13] R. M. Nadeem, R.M. Saleem, R. Bashir, S. Habib, "Detection and Prevention of SQL Injection Attack by Dynamic Analyzer and Testing Model", International Journal of Advanced Computer Science and Applications, vol. 8, no. 8, 2017.

[14] William G.J. Halfond, A. Orso, "Detection and Prevention of SQL Injection Attack", Georgia Institute of Technology.

[15] B.J. S. Kumar, P.P. Anaswara, "Vulnerability Detection and Prevention of SQL Injection", International Journal of Engineering & Technology, pp. 16-18, 2018.

[16] J.O.Atoum, A.J.Qaralleh, "A Hybrid Technique for SQL Injection Attacks Detection and Prevention", International Journal of Database Management System, vol. 6, no. 1, February, 2014.