# An Approach to Attain Reliable Query Processing by Obstructing SQL Injection Attack

**[1]Pradeep Natani, [2]Dolly Chandani, [3]Kishor Kumar Mali, [4]Meenakshi Kothari, [5]Shweta Agrawal**

[1]Asst. Professor, Department of Computer Sc., Poornima Institute of Engineering and Technology, Jaipur
[2]Asst. Professor, Department of Information Tech., Poornima Institute of Engineering and Technology, Jaipur
[3,4,5]Student-B.Tech., Department of Computer Sc., Poornima Institute of Engineering and Technology, Jaipur

Abstract— **In this paper we present an overview of SQL Injection Attacks that is a major concern in maintaining the security of the web applications. There are so many techniques to prevent intrusions that can exploit the sensitive data stored in the database but cannot be implemented everywhere. We are presenting a technique using Hash Function to solve the SQLIA problem.**

Keywords— Authentication Bypass, Intrusion, Database, Hash Function, SQLIA

## I. INTRODUCTION

SQLIA –Structured Query Language + Injection+ Attack SQL is a language used for creating the database and injection means to inject something which can make changes and attack means try to destroy or remove. In overall, SQLIA is a technique through which attacker inserts an unauthorized SQL statement through a SQL data channel. SQLIA is generally imposed on web applications by the attackers or third party to identify the confidential information or to make changes in the database or to gain the control on the application and run according to their wish. SQLIA can cause great impact on web applications and also affect the organization to which that web application is belong. In order to run the web application smoothly over internet or in any other network like LAN, WAN, there is basic need to prevent them with SQLIA. There are so many methods to prevent the web application from SQLIA such as Validation for input values, checking of input data by generating parse tree, using prepared statement, using hash value but all have some limitations either in terms of time required to execute or some have implementation constraint and none of them provide 100% efficient way to prevent web application from SQLIA.

## II. GENERAL METHODOLOGIES FOR SQLIA

There are different ways of intrusion that are basically used by attackers to get into the database and make the undesired changes in it. For a successful SQLIA the attacker used to append a command that is syntactically right with the original SQL query. The classification of
SQLIAs in accordance to [1] [3] be presented.

**Tautologies:** In this type of attack, SQL tokens are injected in the conditional query statement which is always evaluated as true. This type of attack used to bypass authentication control and exploits the vulnerable input field which use WHERE clause.

**"SELECT * FROM login WHERE uname = 'pradeep' and password ='natani' OR '1 '='1'** as the tautology
statement (1=1) has been added to the query statement so it is always true.

**Illegal/Logically Incorrect Queries:** When an incorrect query is fired, the resultant SQL error message is helpful to attack on vulnerable areas of database. Attacker injects junk input or SQL tokens in query to produce syntactic error, type mismatches, or logical errors. For example-

**Injected query : String query = "insert into mysql.login values(\'"+id+"\')";**
Error message showed: Error code 1366, SQL state S1000: General error message from server: "Incorrect integer value: 'xx' for column **'uid'** at row 1"From the message error we can find out name of table and fields: name; Employee; id. By the gained information attacker
can organize more strict attacks.

**Union Query:** By this technique, attackers add injected query to the original query by the word UNION and then can get data about other tables from the application. For example-
**select uname from login where id='usfaujdar' and password='' union all select password from login--'**
This will join the result of the original query with all

the passwords.

**Piggy-backed Queries:** By this attack, intruders exploit database by using the query delimiter, like ";", to append extra query to the original query. On succeeding database receives and execute a multiple distinct queries. Usually the initial query is legitimate query, while the other queries could be illegitimate. Hence the attackers can affect the SQL commands easily. In the given example, intruder injects " 0; drop table user" into the pin input field instead of logical value. Then the application would generate the query:

**SELECT * FROM login; DROP TABLE emp;**

Due to ";" terminator, database accepts both queries and executes them. The later query is not the desired one and

can drop emp table from the database. Moreover, some databases do not require special separation characters in multiple distinct queries. Hence to detect this attack, scanning of special characters is not an efficient solution.

## III. PROPOSED TECHNIQUE

In previous topic we have seen that in SQLIA hacker changes the structure of database query [2] [4].

The database query is fix for application means what &

how many queries are required for particular application. We only required valid input from user not any database query which changes our database structure.

Solution is that we storing the valid database query & then compare with dynamic generated database query.

If dynamically generated database query is match with already stored query then its ok otherwise there is a SQLIA.

Method proposed is using linked list representation.

**1) Using linked list representation:**

We store all valid queries in list representation form. Each query is store in singly linked list to store the addresses of singly linked list we use doubly linked list. The structure of a node of singly linked list is shown in the figure 1 as shown-

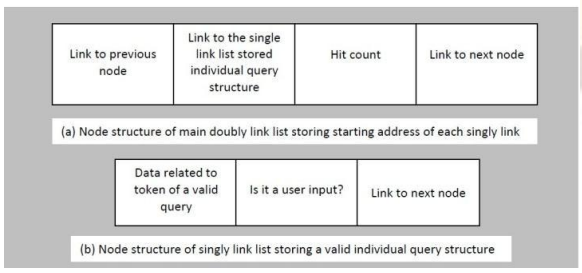


Figure 1- Node Structure of Link List

Now to store valid queries & dynamically generated queries in list form we follow some steps:

STEP 1: Convert queries into tokens.

STEP 2: Separation of tokens & Maintain Sequence of token.

STEP 3: Transform tokens into corresponding integer

value.

Multiply ASCII value to corresponding its position.

Example:

Select * from login where username='xyz' and password='mypassword'

Step 1: Figure 2(a)

tokens

'Select' '*' 'from' 'login' 'where'
'username' '=' ''xyz'' 'and'
'password' '=' 'mypassword'

Step 2: Figure 2(b)

| Select | * | From | login | where | username | = | 'xyz' | and | password | = | 'mypassword' |
|---|---|---|---|---|---|---|---|---|---|---|---|

Step 3: Figure 2(c)

Transform each string into corresponding integer value. For SELECT S=83 E=69 L=76 E=69 C=67 T=84; POSTION S=1 E=2 L=3 E=4 C=5 T=6;

83*1+69*2+76*3+69*4+67*5+84*6=1564

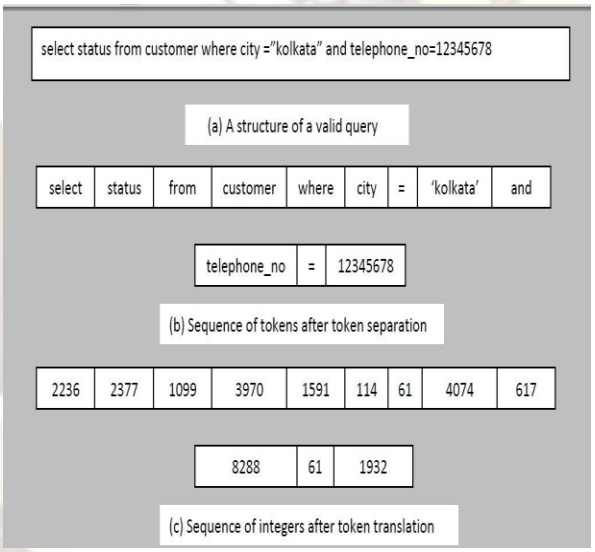


Figure 2- Query after token separation

The corresponding integer values have been arranged in the form of link list as shown in figure 3

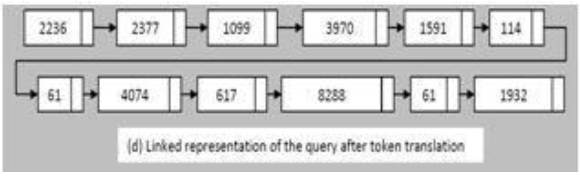


Figure 3-Query Translation to its integer sequence

We already know the no of tokens of incoming query so we combine all queries of same no of tokens as in Figure 4:

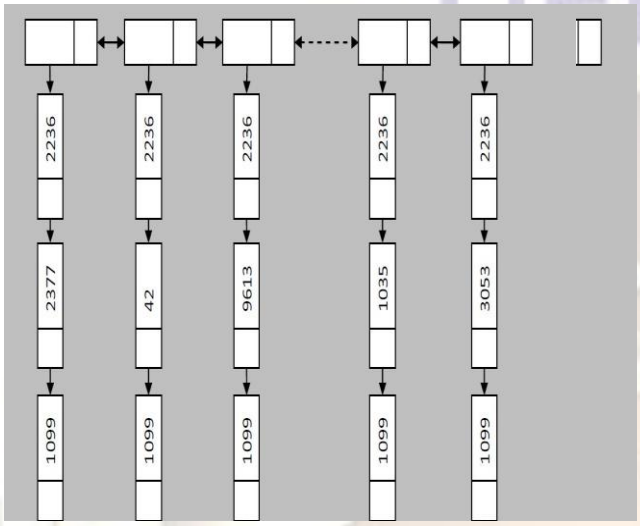


Figure 4: Grouping of query having 3 tokens [2] However,

there is a disadvantage of this method. By using this method, if the number of tokens is same as present in the stored database, then it will not identify the SQLIA.

**2) Using hash table method:**

When user inserts values into database table we calculate the hash value for that.

Hash Value can be calculated by transforming the string

into corresponding integer value.

We add two extra columns with the database table one for username hash value & other for password shown in . Whenever user enters the username & password we calculate their hash value dynamically. Now these hash value is compare with the already calculated hash value. If both are equal then ok otherwise its SQLIA as shown in figure 5.

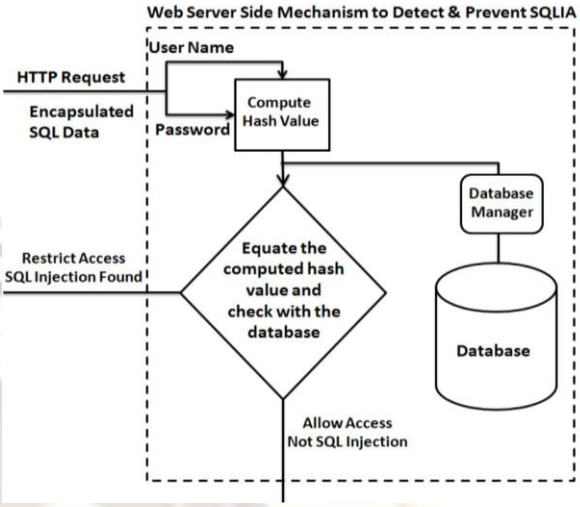


Figure 5: Proposed Hash Scheme for Detecting SQLIA & Prevent Them[1]

Example : CREATE TABLE login(User_name varchar(20), Password varchar(20), H_Username integer, H_Password integer);

**LOGIN TABLE: (Table 1)**

| User_name | Password | H_Username | H_Password |
|---|---|---|---|
| Kishor | k@k.com | 56 | 70 |
| Pradeep | P12345_n | 29 | 47 |
| Xyz | Welcome99 | 74 | 48 |
| | | | |

Table 1: User Login table

In Table 1, there are 4 field in database instead of 2. when database grows it is not better way to check all hash values of username & password .

So to increase the efficiency we EX-OR the two field values & then store only one field into database table.

Modified Query- CREATE TABLE login(User_name varchar(20), Password varchar(20), H_User_Password integer);

H_User_Password = H_Username **EX-OR** H_Password

**EFFICIENT LOGIN TABLE: (Table 2)**

| User_name | Password | H_User_Password |
|---|---|---|
| Kishor | k@k.com | 83 |
| Pradeep | P12345_n | 34 |
| Xyz | Welcome99 | 55 |
| | | |

Table 2: Modified User Login Table

This is efficient way to reduces one field of table as shown in table 2.

## IV. CONCLUSIONS

The linked list representation is not much efficient & it is also time consuming it follows some steps to check SQLIA so new approach that is based on the hash method of using the SQL queries, which is much secure and provide the prevention from the attackers SQL

## REFERENCES

[1] Mayank Namdev, Fehren Hasan, Gaurav Shrivastav. "Review of SQL Injection Attack and Proposed Method for Detection and Prevention of SQLIA", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2, Issue 7, July 2012.

[2] Shaukat Ali, Azhar Rauf and Huma Javed , SQLIPA: An Authentication Mechanism Against SQL Injection. European Journal of Scientific Research ISSN 1450-216X Vol.38 No.4 (2009), pp 604-611.

[3] William G.J.Halfond and Alessandro Orso "AMNESIA:Analysis and Monitoring for Neutralizing SQL-Injection Attacks".

[4] Dibyendu Aich " Secure Query Processing by Blocking SQL

[5] injection", NIT, Rourkela May 2009.