International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 05 | May 2020 www.irjet.net p-ISSN: 2395-0072

SQLIA Detection and Prevention using Dynamic Parse Tree with Query Tokenization

Manikandan[1]

[1]*Manikandan, Department of MCA, ASM IMCOST, Maharashtra, India*

Abstract- *A survey held by OSWAP in 2013 and 2017 shows web application vulnerabilities and SQL Injection attack remains number consecutively on top 10 Web Application Vulnerability list. SQLIA mostly is carried out on Dynamic Web applications as they interact with databases for various operations. SQL Injection allows an attacker to add malicious SQL code to a web form inputs or in URL to obtain access or to make changes to the database that can affect business-sensitive information, which can lead to exposure of valuable data, for example, usernames, passwords, etc This paper proposed a technique to prevent SQL injection based on building a dynamic parse tree by applying tokenization with Query Parser and is very effective in preventing SQL Injection Attacks.*

***Key Words*:** SQL Injection, Dynamic Parse Tree, Query Tokenization, Burp Suite, SQLIA Classical and Modern Attacks

1. Introduction

The main purpose of the paper is to use SQL injection attack to carry out altering the existing database, unauthorised access to a database, escalating the privileges of the user, obtaining information from the database or to breakdown a web Application.

1.1 What is SQLIA

SQL Injection Attacks refers to the deployment of malicious code in SQL statements, via Query String or through web page form inputs.

It is one of the most commonly followed web hacking techniques. SQLIA occurs when attackers modify the syntax, logic or semantics of an SQL query. The main aim of the Attacker for using SQLIA is to access database which he is not authorized to so. To access the database without authorization attackers applies SQLIA in the form of systematically correct queries.

SQL injection is a hacking method that was found over fifteen years back is despite everything ending up being devastatingly viable today, staying atop database security superiority.

1.2 Outcomes of SQLIA

With SQL injections, Full remote control of the database can be taken by penetration hacker, and some of the consequences are: [1]

Authentication bypass: A hacker can log into application if a loophole or vulnerability found in the web application without providing proper credentials.

Gaining access to unauthorized data: Through SQL injection, a hacker may access information which he isn't qualified for. Insert a command to get access to all record details in the database, including usernames and passwords.

Unauthorized Data Manipulation: SQL infusion may likewise permit an application hacker to insert, change or erase data which he isn't allowed to. This results into the data integrity exploitation.

Gain administrative privileges: SQL injection could permit a hacker or an anonymous user to obtain authoritative rights on the database or the database server and at last, could perform activities like closing down the database. This affects the accessibility of the database and thus, inaccessibility of the web application.

1.3 Main Cause of SQL Injection

The main reason for any type of SQL Injection attack is because of Web Application Weaknesses. In this part, vulnerabilities that may exist normally in web applications and can be misused by SQL injection attacks will be introduced: [2]

Invalidated input: This is nearly the most widely recognized weakness in playing out an SQLIA. There are a few parameters in the web application which are applied in SQL queries. If they are not validated properly, SQL injection can be carried out easily.

Improper variable size: All variables length should be precise in the SQL statement to avoid specific attacks to take place like the buffer overflow.

Display of Error message: If an attacker inserts malicious code in the web form, the database generates some error messages which are presented in the web browser. Normally these errors contain data about

database or content structure that helps attacker to organize a stricter attack.

Generous privileges: For the most part in a database, the privileges are characterized as rules to state which database subject approaches which object and what activity are related with the user to be permitted to perform on the items like INSERT, UPDATE, DELETE, SELECT, DROP. Such actions should be restricted to certain objects. An attacker or interceptor, who bypasses authentication gets the benefits equivalent to the legitimate profiles when more benefits are given to the profile, the number of accessible attack strategies increases.

Variable Orphism: The variable which stores the malicious data should not acknowledge any data type since an attacker can misuse this element and store malicious data in it.
Dynamic SQL: at the point when the program runs by merging user input, such as name and password with SQL query such WHERE conditions dynamic SQL is created. It is recognizable that another query could be made by an attacker in runtime.

Client-side only control: if there is no validation on the server-side, the client slide Input validation can be evaded with XSS (cross-site scripting).

Stored procedures: In this kind of assault, the attacker is attempting to execute a stored procedure present in the database. Most of the database comes with a standard set of stored procedure which expands the usefulness of the database and permits to associate with the working framework. Subsequently, in the wake of deciding the back-end database being used, SQLIA's can be controlled by that particular database

2. Related Work

According to [3], SQL Injection attack arises when user's input contains SQL keywords so that the SQL Query is dynamically generated which changes the SQL query intended function. Indeed, for a wide range of SQL injection, it's absolutely impossible somebody can perform injection without embeddings a single quote, space, double dash or hash in a query. The manner in which a user can perform injection without single quote is the point at which the user input is of type number; for example, the **Figure 1**: shows SQL injection query should be possible without utilizing single quote in numeric type:

The nonappearance of space between '101' and 'or' does not allow the query to retrieve data about users, though the space between 'or' and the '1' is compulsory else it will result in a syntax error.

```
#Sample Query
SELECT * FROM `users` WHERE user_id=101
#without using single quotes
SELECT * FROM `users` WHERE user_id=101or 1=1
```

Figure 2: Numeric data

Figure 2: show the necessary to user for single quotes in type character or else syntax error occurs:

```
#Sample Query
SELECT * FROM `users` WHERE user='admin';
#without using single quotes
SELECT * FROM `users` WHERE user='admin' OR 1=1;
```

Figure 1: Character data

It is necessary to use single quotes between 'admin' and 'or'. The injection can also be carried out using double dashes or hash in query which are used to comment. The double dash is used to inject the query with comment to avoid further queries to run and execute the queries before the comment.

This is highlighted by **Ke Wei, M. Muthuprasanna and Suraj Kothari** saying that the characters '- -' mark the beginning of a comment in SQL, and everything after that is ignored.

3. Types of SQLIA

For past 2 decades SQL injection remains a serious threat to Web Application. A survey held by OSWAP in 2013 and 2017 shows web application vulnerabilities and SQL Injection attack remains number consecutively on top 10 Web Application Vulnerability list. SQL Injection attacks keeps on changes with advancements in web application, based on this SQL injection is classified into two types namely Classical and Modern types of SQLIA [4]

3.1 Classical Types of SQLIA

Here I have listed some of the Classical Types of SQL Injection Attacks

1.Tautologies: One technique to acquire unapproved access to information is to insert a true or false condition into the query. In SQL, if the WHERE clause of a SELECT or UPDATE query is combined with a true or false condition, at that point each row in the database table is stored for the result set. Tautology-based attacks carried out by injecting one or more conditional code in SQL queries so as to make the SQL query assess as a genuine condition, for example, (1=1) or (- -). The most widely recognized utilization of this procedure is to bypass authentication on web applications.
Fig 3 shows the Tautology attack.

```
SELECT * FROM `users` WHERE user='admin' AND password='pass' or 1=1 -- '
```

Figure 3: Tautology attack

2. Piggy-backed Query: By means of a query delimiter, this is a sort of attack that exploits a database, for example, ";", to inject additional queries to the original query. In this technique, the primary query is original

though the subsequent queries are injected. This attack is extremely dangerous; an attacker can utilize it to inject to any kind of SQL queries. Below fig 4 shows the piggy backed attack.

```
SELECT * FROM `users` WHERE user='admin' AND password='pass';DROP users-- ;
```

Figure 4: Piggy-Backed attack

3. Logically Incorrect: This kind of attack exploits the mistake messages that are returned by the database for an incorrect SQL query. These database mistake messages frequently contain helpful data that permit attackers to discover the defenceless parameter in a web application and the database design. Fig 5 shows Logically Incorrect attack.

```
SELECT * FROM `users` WHERE user='admin' AND password='pass' AND CONVERT(CHARACTER,NO)
```

Figure 5: Logically Incorrect attack

4. Union query: Union or Association query is a type of SQL Injection attack where an attacker embeds malicious queries into the original SQL query. This assault should be possible by embeddings either a UNION query or injecting into vulnerable parameters. The aftereffect of this attack is that the database returns a dataset that is the association of the consequences of the original query with the result of the injected query. Fig 6 shows the Union attack.

```
SELECT * FROM `users` WHERE user='admin' AND password='pass' UNION SELECT
null,null,column_name,table_name,null,null,null,null FROM information_schema.columns#'
```

Figure 6: Union attack

5.Stored Procedure: In this attack, attacker concentrates on stored procedure which is available in the database framework. Most of the database comes with a standard set of stored procedure which expands the usefulness of the database and permits to associate with the working framework. Stored procedures run legitimately by the database engines. it is a bit of code which is exploitable.

```
SELECT * FROM `users` WHERE user='admin' AND password='pass' ; SHUTDOWN#'
```

Figure7: Stored Procedure attack

the stored procedure gives Boolean values for the approved or unapproved users. For SQLIA, an attacker will compose "; SHUTDOWN; - "with login or secret key. Fig 7 shows stored procedure attack

6.Inference: This attack enables the attacker to change the behaviour of a database. Furthermore, this kind of attack can classify into two types i.e. Blind Injection and Timing attack.

a) Blind Injection: This kind of SQLIA happens when developers neglect to conceal an error message which causes the database application uncertain, this error message help SQLIA to exploit the database through querying a series of logical questions through SQL queries.

b) Timing Attacks: This kind of attacks allows an attacker to gather data from a database by watching timing delays in the database's replies. This sort of attack utilizes if condition queries to accomplish a time delay purpose. To delay database response by a specified time, WAITFOR keyword is used in the injected query.

7.Alternate Encodings: This sort of attack happens when an attacker changes the injected query via utilizing alternate encoding, for example, ASCII, hexadecimal and Unicode. By this technique, the attacker can escape from the whitelist filter, which examines input inquiries for special known "bad character". At the point when this sort of attack joins with other attack strategies, it could be strong, since it can target various layers in the application so developers should be recognizable to every one of them to give an effective defensive coding to avoid the alternate encoding attacks.
Below table 1 list out of the purpose of SQL Injection Attacks

Table 1: List out the purpose of the attack

	SQLIA Types	Purpose	Example Code
1	Tautologies	Bypassing authentication	SELECT * FROM users WHERE uid='abcd' and pwd ='a' **or '3'='3'**
2	Union	Extracting Data	www.targetwebsite.com/index.php?id=-8 -union select 1,2,3,4— -union select 1, **version ()**,3,4— -union select 1, **database ()**,3,4— -union select 1, **user ()**,3,4--
3	Logical incorrect Queries	Identify injectable parameters	www.targetwebsite.com/index.php?id=**1'**
4	Piggybacked Queries	exploit database	SELECT * FROM users WHERE login='doe' AND pin=0; **drop table users**
5	Stored Procedure	Executing remote commands	SELECT accounts FROM users WHERE login="doe" AND pass=" "; **SHUTDOWN--**;
6	Alternate Encodings	Escape Filter Scan	SELECT accounts FROM users WHERE login='' AND pin=**0 (char(0x73687574646f776e)) –** '

3.2 Modern Types of SQLIA

Here some of the modern types of SQLIA has been listed out which are as follows [5]:

1.Fast Flux SQL Injection Attack:
By utilizing this sort of SQLIA, attacker aims to extract information from the database and phishing. a phishing attack is a social engineering attack wherein an attacker

falsely gains sensitive data by incorporating as a third party from the user. Traditional phishing host can be recognized effectively just by tracing the open Domain Name Server (DNS) or the IP address. This traceback method could prompt the shutdown of the facilitating sites. The attacker apprehended that directing like that attack could have a huge impact on load adjusting of a server; thus, to ensure its resources, the administrator of phishing sites began utilizing Fast Flux strategy. Fast Flux is a Domain Name Server strategy which is utilized to cover up phishing and malware spreading sites behind an ever-changing compromised host network.

2.Compounded SQL Injection Attack:
This kind of SQLIA is a mix gotten from a combination of SQLIA and other Web Application Attacks. The attacker aims to exploit the database and cause serious genuine impact than other traditional sorts of SQLIA (which are recently examined). The fast advancement of detection and prevention strategies against different SQLA, implemented the anonymous attacker to build up a procedure called compounded SQL Injection. compounded SQLIA is described in the following points.

- **SQL Injection + DDoS (Distributed Denial of Service) Attacks:**
  This type of attack is carried out to exhaust the resources, hang a server with the goal that the client will not able to access it. The malicious SQL commands keeps track with DDOS Attack is to join, compress, encode etc.

- **SQL Injection + XSS (Cross-Site Scripting):**
  one of the client-side injection attacks is XSS where an attacker can infuse malicious code into the web form inputs. After embedding the XSS script, it will execute and attempt to associate with the database of an application. With the help of iframe command, the malicious code can extract data from the database

- **SQLIA using Cross-Domain Policies of Rich Internet application (RIA):**
  Cross-Domain policies is an XML file which offers authorization to the web client application, for example, Adobe Flash, Adobe Reader, Java, and so on., to get to the information in multiple domains. Cross-domain policies characterize the rundown of RIP facilitating areas that are permitted to recover content from the substance suppliers' space.

- **SQL Injection + Insufficient Authentication:**
  this kind of attack happens when the client or an administrator is a novice. The security parameters have not been introduced and the attacker can access the sensitive content without confirming the personality of the client. consequently, the attacker exploits this vulnerability to inject SQL injection code. Thus, this sort of attack is not as complicated as contrasted with different kinds of attacks. The initial step is to discover whether the application has lacking invalidation and the event that it has, at that point the SQL Injection attack can take place.

4. Existing System

In the sections, one of the popular detections and prevention technique against both Classical and Modern types of attack is discussed.

In [6], the author proposed a technique, called **AMNESIA** (Analysis and Monitoring for Neutralizing SQLIA). For detecting and preventing web application exposures it combines both dynamic and static analysis. It generates a different type of query statements by static analysis. In the dynamic phase, and interprets all queries before sending them to the database and validates each query in the dynamic stage.

In [11], **SQL Map** has been discussed by authors as a protection of the (SQLI + DNS) Attack. It has the feature of the DNS Exfiltration and other specialized commands for DNS prevention and detection. SQL Map is compatible with most of the SQL Database versions.

5. Proposed Technique

To prevent SQL Injection attacks, this paper purpose a technique which utilizes Tokenization model for blocking the malicious code in web form input at the entry point and a dynamic parsing tree to picture the structure of SQL command. This technique detects a single quote, hash, space or double dashes and forms a dynamic tree structure. The main purpose of this technique is to prevent the database from most web application vulnerabilities. This technique is completely based on Tokenization and dynamic parse tree to prevent SQLIA.

To prevent SQL Injection Attack, the following sequence takes place for essential steps which are as follows:

1. Input data received from web user input or through a query string.
2. Dividing the query based on delimiters such as hash, double dashes or single quote, etc.

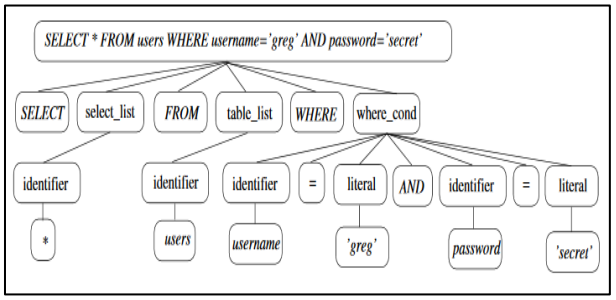

Figure 9: Select query with user input values

3. The programmer applied SQL commands is the head of the parse tree and the user-provided portion is a leaf node of the parse tree.

4. The Tokenization process is applied for both the original query and query with the malicious code and will be stored in a dynamic table.
5. Both the Dynamic table and the dynamic query will be sent to the server.
6. At the server-side again tokenization process is applied and stored in another dynamic table.
7. The server will compare both the table, if both are same, then the injection is presented in the query.
8. The server will then process the query further to the database for retrieving records.
9. If both are different server will reject the query and will forward a custom error message back to the user.

Dynamic Parse Tree

A dynamic parse tree is a kind of data structure for parsed SQL commands. Parsing the SQL queries requires the grammar of the SQL language. By parsing two queries and comparing their parse tree, we can identify if both queries are the same.

When an attacker injects malicious code in the query, the parse tree of the SQL statement will not match the original queries parse tree.

Example of select query show in Figure 8

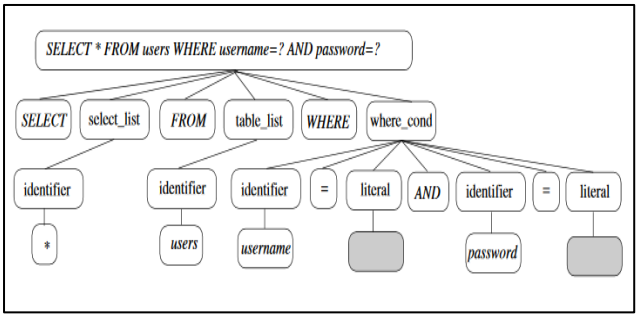

Figure 8: Select query with two user input

By this dynamic tree, when a programmer writes code to query database will follow a formulation to structure the query.

The Programmers applied portion is the hard-coded part of the parse tree and the user-supplied is the parse tree's leaf portion. The user inputs values are assigned to the leaf portion of the parse tree.

An example of the parse tree SQL query is:

SELECT * FROM 'users' WHERE username =? and password =?

The question is the values to be provided by the user to the leaf nodes.

Another example of the parse tree with Injection SQL

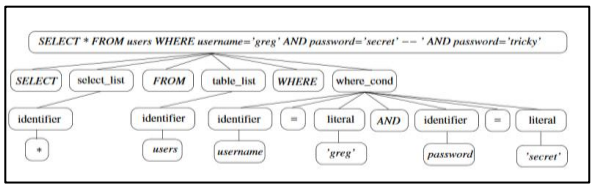

Figure 3: Comment injected query

query shown in the following figure 10

The figure 11 represents the potential vulnerability is that the query is parsed properly will be assumed by the program and retrieve the values from the database. Fortunately, Parse tree catches this vulnerability by parsing the token to comment. The figure is the same query with the comment token included.

Query Tokenization

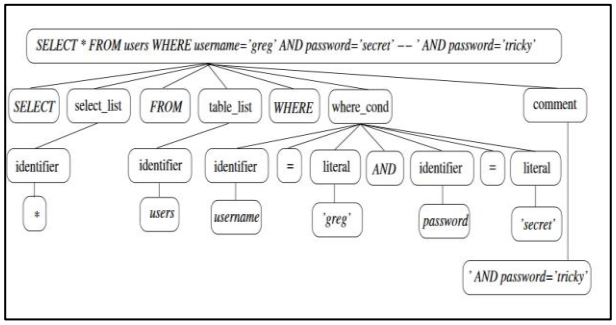

Figure 4: Injected Query with another leaf node

Query Tokenization is a technique which converts the SQL query into tokens. This token is generated by identifying the quotes, hashes, double dashes or space in the SQL query. All the keywords in the query which are before delimiters like space, single quotes, a double dash will be assigned a token.

An Example of Token Formation show in the Figure 12

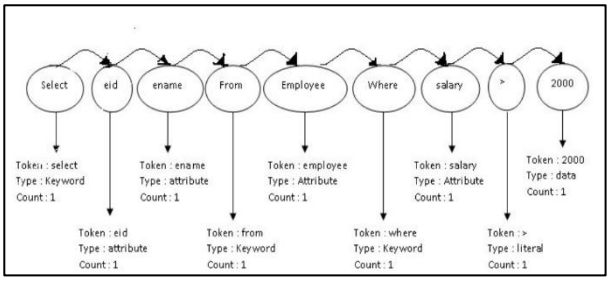

Figure 5: Token Formation

Tokenization process takes place in four steps which are as follows

1. By replacing all unwanted character from the user input query
2. Identify all the hash, single quotes, double dash in the query
3. Break the queries into tokens
4. Store them into Dynamic Table
5. After Tokenization, the queries are forwarded to the server

Query Tokenization stores the tokens in a dynamic table with its token names, token types and the occurrences of the token. The following figure 13 shows the working of the Tokenization process.

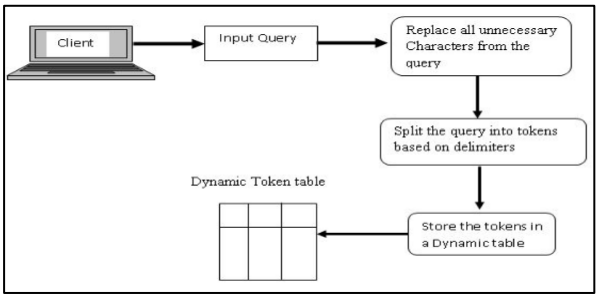

Figure 6: Tokenization Process

In the server, the received SQL statement will again follow the same Tokenization process and stores it in another dynamic table. The server will compare both the table if both are same, then the injection is presented in the query. The server will then process the query further to the database for retrieving records. If both are different server will reject the query and will forward a custom error message back to the user.

Table2: SQLIA Counter measure technique

SQLIA	SQL Prevention Techniques	
	Existing	**Proposed**
1.Bypass Authentication	Prevented	Prevented
2.Unauthorized access to database	Prevented	Prevented
3.Injected Addition Query	Prevented	Prevented
4.Unauthorized Remote execution procedure	Prevented	Prevented
5.Injected Union Query	Not Prevented	Prevented
6.Injected Alias Query	Not Prevented	Prevented
7.Injected Instance Query	Not Prevented	Prevented

Conclusion

SQL injection attacks are one of the serious threats to the web application. In SQLIA, the attacker injects malicious code in user input form or in the query string and access unauthorized data. To conduct SQL Injection attacks, an attacker must use SQL keywords such as single quotes, double dash or double quotes in user input form or in the query string. This paper proposed techniques which use Dynamic Parse Tree with Query Tokenization to eliminate the drawbacks of other existing techniques. The proposed method consists of Tokenizing the original query and injected query to find the injected commands in the user input or in the query string by applying tokens.

References

[1] A. N, M. V. K. and V. G. , “Preventing SQL Injection Attacks,” International Journal of Computer Applications, vol. 52, no. 13, p. 8887, August 2012.

[2] A. Tajpour, S. Ibrahim and M. Z. Heydari, “Detection of SQL Injection by Honeypot,” PARIS2012, December 2012.

[3] M. M. a. S. K. Ke Wei, “Preventing SQL Injection Attacks in Stored Procedures,” IEEE, 2006.

[4] R. ,. R. Devi.D and R. , “A STUDY ON SQL INJECTION TECHNIQUES,” International Journal of Pharmacy & Technology, vol. 8, no. 4, Dec-2016.

[5] M. F. Y. Zainab S. Alwan, "Detection and Prevention of SQL Injection Attack: A Survey," International Journal of Computer Science and Mobile Computing, vol. 6, no. 8, pp. 5-17, August 2017.

[6] H. W. G. J and A. O. , "Preventing SQL injection attacks using AMNESIA," International conference on Software engineering(ICSE), pp. 795-798, 2006.

[7] P. S. Kumar, M. P. and M. K. , "Efficient Method for Preventing SQL Injection Attacks on Web Applications Using Encryption and Tokenization," International Journal of Latest Trends in Engineering and Technology (IJLTET), vol. 4, no. 4, November 2014.

[8] N. A. Lambert and S. K. Lin, "Use of Query Tokenization to detect and prevent SQL Injection Attacks," IEEE, 2010.

[9] G. Shrivastava and K. Pathak, SQL Injection Attacks: Techniqueand Prevention Mechanism, Ujjan: International Journal of Computer Applications, May 2013.

[10] J. Clarke, SQL Injection Attacks and Defense, Syngress, July 2012.

[11] M. Stampar, "DataRetrieval over DNS in SQL Injection Attacks," [Online]. Available: http://arxiv.org/abs/1303.3047, 2013..

[12] J. P. Singh, "Analysis of SQL Injection Detection Techniques," Theoretical and Applied Informatics, vol. 28, no. 1&2, 2016.

[13] XuePing-Chen, "SQL injection attack and guard technical research," Elsevier Ltd, 2011.

[14] N. Basit, A. Hendawi, J. Chen and A. Sun, "A Learning Platform for SQL Injection," Association for Computing Machinery, 2019.

[15] Voitovych O.P., Yuvkovetskyi O.S. and K. L. , "SQL Injection Prevention System," Radio Electronics & InfoCommunications, 2016.

[16] A. B. M. A. A. Y. I. S. M. S. A. and J. A. , "SQL-injection vulnerability scanning tool for automatic creation of SQL-injection attacks," Elsevier Ltd, 2010.

[17] P. Y. and M. , "SQLIA:Detection And PreventionTechniques: ASurvey," IOSR Journal of Computer Engineering (IOSR-JCE).