SQL Injection Prevention using Query Dictionary Based Mechanism

Adwan F. Yasin[1]
Department of Computer Science,
Arab American University
Jenin , Palestine.

Nael Zidan[2]
Department of Computer Science
Arab American University
Jenin , Palestine.

Abstract— SQL Injection Attack (SQLIA) is a technique of code injection, used to attack data driven applications especially front end web applications, in which heinous SQL statements are inserted (injected) into an entry field, web URL, or web request for execution. "Query Dictionary Based Mechanism" which help detection of malicious SQL statements by storing a small pattern of each application query in an application on a unique document, file, or table with a small size, secure manner, and high performance. This mechanism plays an effective manner for detecting and preventing of SQL Injection Attack (SQLIA), without impact of application functions and performance on executing and retrieving data. In this paper we proposed a solution for detecting and preventing SQLIAs by using Query Dictionary Based Mechanism.

Index Terms—SQL Injection Attack, SQL Injection Attack Detection, SQL Injection Attack Prevention, Query Dictionary.

1 Introduction

Structured Query Language (SQL) [1, 2] is a standard, comprehensive language, based on the relational model, SQL includes capabilities of many functions. DDL statements for creating schemes and specifying data types and constraints. DML statements for specifying data retrieves, and data modifications. SQL Language is a textual language that used on all relational database management systems (RDBMS), the most known and used are Oracle, Microsoft SQL Server, MySQL, PostgreSQL, DB2 and SQLite

SQL Injection Attack (SQLIA) [3] is a code injection technique, used to attack data driven applications especially front end web applications, in which heinous SQL statements are inserted (injected) into an entry field, web URL, or web request for execution, used to gain unauthorized data, or to retrieve information from SQL relational database. SQLIA used most often to attack databases for retrieving and extracting secret information such as credit card information, private information, user information's, and financial records. The highest risk application for attack is web applications, since web applications accessed through internet and available for all internet users and devices, also mobile applications now are at highest risk for SQLIA. An application is vulnerable for SQLIA since the injection is legal for SQL standards, and DB engine execute it. The vulnerability exists at user inputs, in which they bypass validation or no validation at all and passed to dynamic SQL statement without validation and checking. If we are validating the user input, then with another way we are forbidden them to entering single and double quotes, multiple dashes, and SQL Language keywords in the input.

Hackers have ability to input directly malicious queries via a web form or by directly insert it to the end of the URL or to URL variables or through HTTP headers. For example, if the query accepts username and passwords like this:

"SELECT User_Name, User_FullName FROM TABLE_USERS WHERE User_Name=' ' AND User_Password=' '; "

The above query will select "UserName" from the table "TBL_USERS" by filtering using query search condition "User_Name" and " User_Password". Now we can manipulate it by various SQL code snippets by just input them in User_Name and User_Password fields at web form or URL variables like Ahmad' or '1 "= '1

When web form front end processes web form and generates SQL statement to send it to DBMS, generated SQL query with above inputs will be:

```
SELECT User_Name, User_FullName FROM TBL_USERS
WHERE User_Name='Ahmad' AND
        User_Password='' or '1'= '1';
```

Because of malicious input the query search condition is always true condition as the query is asking to retrieve User_Name and User_FullName with condition that User_Name is *Ahmad* and USER_PASSWORD equal to *'' or 1 = 1*. We can also use SQL comments operator "--", so SQL engine ignore the portion after comment operator, if User_Name field input is Ahmad'--, this will manipulate query search to just check condition on User_Name only, UNION can also lead to a successful SQL injection attack.

Open Web Application Security Project [4] published that SQL Injection Attack (SQLIA) is the top one and most vulnerable among the top ten web application vulnerabilities.

SQL Injection attack not limited for web applications, it could be on desktop applications, mobile applications. According to OWASP [5], according to reports on 2008 for SQL injection vulnerabilities, 25% of all vulnerabilities reported for web applications.

In this paper we are proposing a solution for detection and prevention of SQL Injection Attack (SQLIA) using

International Journal of Computer Science and Information Security (IJCSIS),
Vol. 14, No. 6, June 2016

Query Dictionary Based Mechanism, in which we will store all queries search portion patterns, then we compare query generated from web forms back end and compare with stored one, the result will show if form query is injected, based on result action taken. In section 2 we are talking about Web Application and SQL Injection attacks, Section 3 about Types of SQL Injection Attacks. Section 4 about SQL Injection Attack Detection. Section 5 is a summary of related work on SQL Injection detection and Prevention. Section 6 we are talking about our Proposed solution. In last section the conclusion.

2 Web App and SQL Injection Attacks

Web application is a computer application that located on a server and users request it using web browsers through World Wide Web abbreviated (WWW). Web applications requested using HTTP or HTTPS protocols. In early web application started to be static, with web technology development most of web applications now dynamic content, this means its contents from a database. Client using browser by entering web application URL request a web application document by using HTTP methods "Get, Post, Put, Delete". Web application N-tier architecture consists of Presentation, Business/Logic, and Data tiers. The most architectures used is 3-tier in which each layer can potentially run on a different machine and the three layers are disconnected as shown on Fig.1.

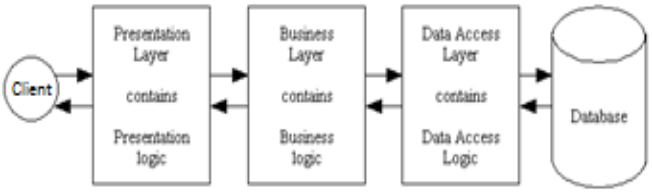

Fig. 1. 3-teir Web Application Architecture

This architecture in which presentation layer exists on client machine which is displayed using browsers like Google Chrome, Mozilla Firefox, Microsoft Internet Explorer. In addition, the ability of user for changing URL variables also input fields and weakness of client validation and easy of validation bypassing allow hackers to use vulnerabilities of dynamic SQL queries generated at web application backend programming code.

SQL Injections [6] are attacks by which an attacker makes changes on the structure of the original SQL query by inserting (injecting) additional SQL code in the input fields of the web form or desktop app form or on URL in order to acquire unauthorized access to the database. Despite that vulnerabilities that drive to SQLIAs are well known and understood, they persist and continued to be available because of lack of effective solutions and techniques for detecting and preventing them. SQLIA is a hacking technique in which attacker makes modifications on SQL statements through web form or application form inputs or web form URL variables or hidden fields to access unauthorized resources. Weakness of input field and URL variables validation help hacker to success. Web application vulnerabilities is the main cause of SQL injection, the most of these vulnerabilities are:

A. **Weakness in input validation:** this the common vulnerability in which no input validation for web form input fields or URL variables, so this allow hacker to add SQL code easily.
B. **Generous privileges:** when web application access a database need a user with specific privileges, for example privileges for reading data, modification of data includes insertion, updating and deleting, privileges for DDL like creating tables, dropping tables. So the weakness here to use a general user that have all privileges, so any SQL statement this DB user can execute. So here if attacker bypass authentication he gains access to all DB user privileges, for example he can drop any table.
C. **Uncontrolled variable size:** variable sizes that uncontrolled and generic specially the biggest domain of them like String, lead to an easy way for attacker to alter SQL query with many characters the variable contains.
D. **Error message***:* the generated error messages by backend server code may return to client, these messages may contain database name, tables name and attributes, etc., this information help hacker to know the structure of database. So error messages should not be shown to client and should the web application send it to webmaster by email or audit it in a log file.
E. **Dynamic SQL:** SQL queries that dynamically generated on backend code, these queries generated by concatenating SQL where condition attributes with variable values from input field or hidden fields or URL variables. In dynamic SQL the most research focus since no way to prevent using it, and it should not infect with SQLIAs**.**
F. **Client-side only control:** if web application web forms validation depends on client side only, this is vulnerable, since hacker can bypass validation and validation scripts at client can be altered by using cross-site scripting.
G. **Stored procedures (SP):** SP is an assigned name for a set of SQL statements and logic of procedures that compiled, verified and stored in database server, and it controlled through database server security. SP is more secure than web form dynamic generated query. The vulnerability to use dynamic generated SQL statements and use database function like EXEC to execute generated query, in this case it is vulnerable same with web form dynamic generated query.
H. **Input Output file support:** if database user has privilege to execute input form file or output file, then it will allow hacker to execute any statement that output to text file or excel file, for example MariaDB and MySQL "SELECT INTO OUTFILE…".
I. **Multiple statements:** database user privilege for executing multiple statements allow hacker to use UNION and retrieve additional information, or he can add additional insert statement or delete statement or drop table statement.
J. **Sub-selects**: supporting of sub-selects or sub-queries lead to vulnerability, so additional SQL query can be

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

added inside WHRE condition.

There are code practices [7] should be followed to reduce SQLIA, the most important of these practices are:

A. **Manual Coding Practices Defense:** here developer learn SLQIA techniques and how to prevent them on coding stage, these practices divides to four categories. **Using Parameterized Queries or Stored Procedures**, this will reduce vulnerabilities on dynamic query generation by concatenating, and replace values with placeholders (parameters) with values. Also stored procedures can check of parameters data types and hide query structure from attacker. And developers should avoid using dynamic generation of queries in Stored procedures. The second category is **Escaping**, which is a technique for elimination SQL keywords. Each Programing language or script language has suitable connector to DBMS and it has its own escaping functions embedded in their libraries, as an example MySQL connector for PHP has mysql_real_escape_string() function. Third category *Data Type Validation*, here developer should use suitable data types and he should check and validate inputs with data types. Last group is *White List Filtering* [8], by filtering allowed and legitimate key words, then check for list to accept and execute.

B. **SQL DOM:** [9] the solution is an executable "sqldomgen", which executed with connection to database and generate a compiled Dynamic Link Library (DLL) file. This file used by developer to execute against database. DLL file contains classes refer to them with SQL Domain Object Model (SQL DOM).

C. **Parameterized Query Insertion:** by using this technique, SQL queries vulnerabilities is detected inside source code and replaced with secure parameterized Structured Query Language (SQL) queries.

3 Types of SQL Injection Attacks

There are different methods performed together or sequentially depending on attacker goals. For an effective and succeeded SQLIA, attacker should add a command with right syntax to the original SQL query. SQLIAs [6,10] classified to:

A. Tautology.
B. Illegal/Logically Incorrect Queries.
C. End of Line Comment.
D. Timing Attack.
E. Union Queries.
F. Blind SQL Injection Attacks.
G. Piggy-Backed Queries.

For clarifying these types of SQLIAs I will use an example of a web form that contains two input fields Username & Password and a login button as shown in Fig. 2

In this example we use below URL HTTP://www.anydomian.com?page=login to request login page. We use Username "Ahmad" and Password "P@ssw0rd", after Ahmad click on Login button, at backend web form code that connects to database to verify that Ahmad account is available and correct. If SQL query return "True' Ahmad will be redirected to his account main page, if "False" a message will appear from him telling him a wrong username or password. For a more reading of code read it from [10]. Now we will discuss the seven types "methods" of SQLIAs and show how an attacker access Ahmad account main page without knowing the correct full Account information, in our example, the username and password of "Ahmad" account.

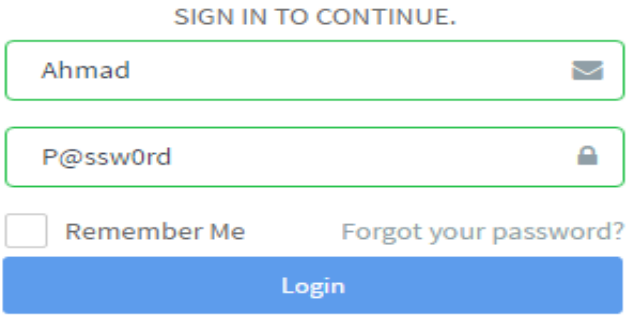

Fig. 2. Login Page

A. **Tautology**
This SQLIA attack injects to SQL query so query evaluated to "True" always.
Injected Query:
SELECT User_Name, User_FullName
FROM TABLE_USERS
WHERE User_Name='Ahmad' AND User_Password='' or '1'= '1';

B. **Illegal/Logically Incorrect Queries**
This type of SQLIA collect database information from making page return error messages from backend code. Attacker inject junk input to URL or input fields or SQL query tokens to produce syntax or logical errors. In our example attacker inject to URL variables a single quote.
HTTP://www.anydomian.com?page=login'
Injected Query:
SELECT PAGE_LOC FROM TBL_PAGES WHERE Page_ID=login'
This injection will fire a syntax error when generating dynamic query that return location of login page form database and the error will show:
Error: Invalid Query "SELECT PAGE_LOC FROM TBL_PAGES WHERE Page_ID=login'"

C. **End of Line Comment**
In this type of SQLIA attacker use SQL comment operator "- - "to ignore part from SQL query search.
In our example attacker insert for Username input field "Ahmad'--'"and Password "12345"
Injected Query:
SELECT User_Name, User_FullName
FROM TABLE_USERS
WHERE User_Name='Ahmad'-- AND User_Password='12345';

D. **Timing Attack**
An inference attack. In this type attacker make timing between web page responses. This technique used "IF-Then" conditional statement for queries injection and "WAITFOR" to make database delay query response by a specific time.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

E. Union Queries

This type attacker appends a new query to original one using SQL UNION keyword, so he can access to unauthorized data. In our example attacker can inject a union query to URL:
HTTP://www.anydomian.com?page=login' union all select UserName from TBL_USERS'
Injected Query:
SELECT PAGE_LOC FROM TBL_PAGES WHERE Page_ID='login' UNION ALL SELECT USERNAME FROM TBL_USERS
This injected query will return all user names stored in table TBL_USERS which is not authorized to page navigator to access to this information.

F. Blind SQL Injection Attacks

An inference attack, as we talked one of best code practices to hide error messages from shown to client. So in this case attacker does not have any error messages since developer make error to a generic web page error. It difficult for attacker now to make SQLIA but it does not impossible. Attacker can request True/False requests from SQL queries and he could success and steal information.

G. Piggy-Backed Queries

In this SQLIA type, attacker use SQL statements delimiter ";". Attacker append additional statement so he can execute more that query. In our example attacker could add another query to URL:
HTTP://www.anydomian.com?page=login';DROP TBALE TABLE_USERS'
Injected Query:
SELECT PAGE_LOC FROM TBL_PAGES WHERE Page_ID='login'; DROB TABLE TABLE_USERS
So here in this case first SQL query is legal, but the second is illegal and will fire database to drop table TABLE_USERS.

4 SQL Injection Attack Detection

There are many techniques used for SQLIA detection [2, 7], we will present them:

A. SQLUnitGen

Abbreviation for "SQL Injection Testing Using Static and Dynamic Analysis. This technique proposed by Shin and fellow workers. It uses static analysis to track flow of user inputs for testing attacks. Most tools and techniques utilize "JCrasher" which is a tool used to obtain test cases upon generated attack inputs.

B. MUSIC

Abbreviation for "Mutation-based SQL Injection vulnerabilities checking". This technique proposed by Zulkemine. He used mutation method based on error checking and catching by injecting syntax errors to check if any misshapen occurred. Then by comparing output it can conclude if a query contains misshapen and vulnerabilities.

C. SUSHI

It is an abbreviation which stands for "string constraint solver". It proposed by Fu and Li. It is a recursive algorithm that found it very help in finding complex SQLIAs. It Solves SLSE (Simple Linear String Equation) constraint in an effective approach.

D. Ardilla

A technique and a tool for creating SQLIA. It proposed by Kiezun and fellow workers. This tool generates attacks as inputs and run the application for each attack input. So it can check and detect the SQLIA from generated attack inputs.

E. String Analyzer

Wassermann and Su proposed this technique. Their solution depends on a based grammar algorithm, it strategizes string values as context free grammar (CFGs) and operations based on transducers of language following minimization. This technique then labels user input strings and summarize them and find contexts. Then by regular languages and context free languages usage, it checks the security of each labeled string in aspect of syntax.

F. PHP Miner

It is a solution rather than a tool, it proposed by Khin Shar and Kuan Tan. This solution statically looks for attributes in source code, then produces models and flowcharts of vulnerabilities prediction.

G. Vulnerability and Attack Injection

A method proposed by Fonseca and fellow workers, the solution upon attack application by pragmatic SQL injection vulnerabilities. For getting more pragmatic results the solution used predefined collected data from actual attacks. The technique composed of two parts that work together, a tool for injection attack and another for injection of vulnerability.

5 Related Work

Deevi Radha Rani, B.Siva Kumar, L.Taraka Rama Rao, V.T.Sai Jagadish, M.Pradeep [3]. They proposed a technique that handles all SQLIAs types. The technique upon encryption of user information and using of stored procedures. They apply that on users' authentication information (Username, Password). They encrypt user data with AES algorithm using 40-bit secret key. On user registration, his info encrypted and stored as a chipper text in database. On user authentication, on back end code at login form called stored procedure with parameters "Username, Password, Secret Key". Stored procedure encrypt Username & Password using secret key, after that it compares the generated encrypted username and password with encrypted username & password saved at users table. This technique is not suitable for dynamic queries from various tables since encryption of big data will consume time and size. But it is very valuable for injection attacks on user authentication.

Biji.K.P [11] proposed data dictionary based mechanism against SQLIA. The method for detecting ant prevention SQLIA using a combination of DDL & DML Mapping. She creates a new database image as a mirror from principal database. Mirror database contains schema structure

and data contents of SQL queries implemented in web application forms, which will be stored in parallel. She generates a formula which it is a combination of DDL & DML Mapping along with Vectorization of SQL Queries. The Vectorization of SQL queries stored in a new created tables in mirror database, for including different syntax. She resolves the parse tree of different generated queries. She monitors the detection of abnormalities among the queries within production database from the result of the output of the different generated queries. For SQLIA detection shed used two methods. Static method which is known as pre-generating approach. In static method developers follow some guidelines and validation checking. The second method is Dynamic approach which is known as post-generated approach, a technique used in run time. It analysis dynamic or runtime generated SQL query from web form after user inputs or web form request.

Inyong Lee a, Soonki Jeong b, Sangsoo Yeoc, Jongsub Moond [12]. They proposed a simple, easy and effective technique for detecting SQLIAs based on static and dynamic analysis and by taking of attribute values at runtime (Dynamic Analysis) and compare it with original one in which also removed attribute values (Static Analysis). The technique used for numeric attributes and string attributes. They create an algorithm for attribute values removal from query. Also they create a generalized SQLIA detection algorithm to check if the query at web forms is normal or abnormal in advance.

Debabrata Kar, Suvasini Panigrahi [13] proposed a technique for SQLIA detection using query transformation and hashing. Their technique to transform the original query parameter values "where condition parameter" with question mark symbol "?", and SQL keyword to uppercase keywords, system objects like table names and column names with keywords they proposed. So with this transformation they reduce number of different queries structure, also this will reflect on performance of search. They used hashing function for generating unique hash key, so the search will be efficient during runtime. The advantages of using hashing is the size of hash key will be smaller than the transformed query, so size needed in storage reduced. Also the same hash will be primary index, as they are unique, to facilitate fast and efficient searching at runtime.

R.Latha, Dr.E. Ramaraj [14] proposed a technique for detection of SQLIA by replacement of query search condition attributes string of original query used in web form with symbols they proposing like "PQ, GQ, STR, NUM, etc.". At runtime they are making a replacement of query search condition attributes for both the original query and dynamic generated query from web form after user inputs. So they have now a two generated restructured queries. They compare the two restructured queries for SQLIA detection by measuring the distance between the two restructured queries using levenstein method. This technique satisfies both static and dynamic analysis.

Swapnil Kharche1, Jagdish patil, Kanchan Gohad, Bharti Ambetkar [15]. They proposed an efficient technique and algorithm for detection and prevention of SQLIAs using Aho–Corasick pattern matching algorithm. Their proposed technique has two phases, static phase and dynamic phase. In static phase they create a list of known anomaly pattern, and SQL queries that checked by enforcing static pattern matching algorithm by comparing of known anomaly pattern list created. During runtime and using dynamic phase if new anomaly is occurring, then new anomaly will be generated and added to static anomaly pattern list. On new anomaly generation score calculated for the query, if the score greater than a determined threshold then the query passed to an administrator to analyses the query manually, if the query infected a new anomaly generated and added to static anomaly list.

6 Proposed Solution

We propose an effective solution for SQL Injection detection and prevention without any impact on application functions and performance. This solution based on a Query Dictionary Mechanism. Our solution general view focus on:

A. SQL query statements numbers.
B. SQL query has UNION
C. SQL Query where suffix pattern.

To save this information about each query, many approaches can be used. It could be generating a memory allocation at application start, so this information can be collected for all queries exists on the application start one time, or it could be collected on first query calling and appended to memory allocation. For memory allocation we propose to create application variable that contains a list of objects to save query information on, the allocation created below using C# language and ASP. NET web application.

```
class SQLIA_DP
{
  public int Id { get; set; }
  public string Query_Caption { get; set; }
  public byte Query_Statemnts_Count {get;set;}
  public bool Query_Has_Union { get; set; }
  public string Query_Pattern { get; set; }
 }
List<SQLIA_DP> ls = new List<SQLIA_DP>;
```

At Global class, in Application_Start method we create an application variable that holds the ls instance of query information, the statement for creating is:

```
Application["SQLIA_DET_PREV"] = ls;
```

Another approach to save query information is in a JSON file or in NoSQL Database for example MongoDB, the format as following:

```
[  "Id": 1,
   "Query_Caption":"loginfrm",
  "Query_Statements_Count": 1,
  "Query_Has_Union": "FALSE",
   "Query_Pattern":
            " WHEREUser_Name=ANDUser_Password="
]
```

Another approach to save query information on any relational database table, it could be on same application database or in a different database, table structure will be:

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

```
CREATE TABLE [dbo].[TBL_SQLIA_DET_PREV](
[Id] [BIGINT] IDENTITY(1,1) PRIMARY KEY,
[Query_Caption] [VARCHAR](15),
[Query_Statements_Count] [TINYINT] NOT NULL,
[Query_Has_Union] [BIT] NOT NULL,
[Query_Pattern] [VARCHAR](1000) NOT NULL );
```

Another approach to save query information on XML file as shown in Fig. 3.

```
<?xml version="1.0" encoding="UTF-8" ?>
<SQLIA_DET_PREV>
    <query>
        <id>1</id>
        <Query_Caption>loginfrm</Query_Caption>
        <Query_Statements_Count>1</Query_Statemnts_Count>
        <Query_Has_Union>false</Query_Has_Union>
        <Query_Pattern>WHEREUser_Name=ANDUser_Password=</Query_Pattern>
    </query>
</SQLIA_DET_PREV>
```

Fig. 3 XML Format for SQLIA_DET_PREV

For Query Pattern extraction, it could be generated and inserted manually by developers or database designers, or developers can use below proposed algorithm that automatically extract query pattern and insert it to SQLIA_DET_PREV list, or developers could use this algorithm at runtime. The algorithm for Query Pattern extraction as following and as shown on Fig. 4:

STEP 1: Take Dynamic generated SQL Query without values from source code.

STEP 2: Check if UNION key word exists and

STEP 3: Count semicolon times which represents number of SQL statements in Query.

STEP 4: Split SQL Query by "WHERE" key word.

STEP 5: If Splitted SQL Query Output Array has more than one item then next steps for second items in Array, if it has one item then next steps for First Item (One Item means Query does not have where statement)

STEP 6: Remove single quote and values between from chosen array item.

STEP 7: Remove each value after SQL Equal Operator "=" and before first Space.

STEP 8: Remove all Spaces.

STEP 9: Save Needed information on SQLIA_DET_PREV list, if Semicolon times is zero then save it one.

So our example query "SELECT User_Name, User_FullName FROM TABLE_USERS WHERE User_Name='Ahmad' AND User_Password='12345'" Does not has UNION, zero semicolon, after splitting and execute steps from 5 to 9, Query Pattern will be "WHERE User_Name=ANDUser_Password=", values saved as shown in Fig. 3, since semicolon times is zero, this mean the query consist of one statement. For "Query_Caption", this field can be used for query retrieve to increase search performance, so developer can use it the same for example "loginfrm" for queries in login form as shown in Fig. 3 so I linked it with web form class which can extracted dynamically.

Above extraction algorithm could be used static or dynamic, depends on application and developer needs. On application run and after user enter the inputs send his request and we assume here user is an attacker and he injected SQL query. Query after its dynamic generation and before sending to database engine for execution should send to SQLIA_CHECK algorithm which described as following:

STEP 1: Use Query Pattern Extraction Algorithm above to extract new dynamic generated query with parameter values.

STEP 2: Create SQLIA_DP object (SQLIA_DP_CURRENT).

STEP 3: Get Query Pattern object saved at SQLIA_DET_PREV List, if not available it should be generating using Query Pattern Extraction Algorithm and save it to (SQLIA_DP_ORIGIN).

STEP 4: Compare Query_Statemnts_Count on SQLIA_DP_CURRENT and SQLIA_DP_ORIGIN, if result is equal GO TO STEP 4, if not Return 1 and Exit.

STEP 5: Compare Query_Has_Union on SQLIA_DP_CURRENT and SQLIA_DP_ORIGIN, if equal GO TO STEP 5, if not Return 1 and Exit.

STEP 6: Compare Query_Pattern on SQLIA_DP_CURRENT and SQLIA_DP_ORIGIN, if equal Return 0 and Exit, if not Return 1 and Exit.

Above Algorithm return value 1 means there is an SQLIA, so query execution should be canceled. If return value 0 then query is clean and it should be send to database engine for execution.

In our example if attacker inject a query "SELECT User_Name, User_FullName FROM TABLE_USERS
WHERE User_Name='Ahmad' AND
User_Password='' or '1'= '1';
The generated query will be send to SQLIA_CHECK ALGORITHM, the result explanation will be as following:

STEP 1: Query Pattern will be "WHEREUser_Name=ANDUser_Password=or=" and no UNION key word and 1 statement, this info saved to (SQLIA_DP_CURRENT) object.

STEP 3: Get Saved Query Pattern from List, this will return, 1 statement, no UNION, "WHEREUser_Name=ANDUser_Password=" and saved to (SQLIA_DP_ORIGIN) object.

STEP 4: Compare result is equal Go to Step 5

STEP 5: Compare result is equal Go to Step 6

STEP 6: Compare Query Pattern is not Equal, Algorithm return 1 so there is an SQLIA and Query does not forward to database engine.

7 Conclusion

In this paper we have presented an effective SQL Injection Attack detection and prevention without any impact in application functions and performance. Our proposed solution used static and dynamic approaches. Easy to implement by developers and database designers or developers. Our solution detects all types of SQLIAs. Upon application needs or and developer experience or and application sensitive degree it could be implemented for part of queries or for all queries, it could be implemented static or dynamic. Query information extracted could be stored in encrypted manner to make the solution more secure. As a future work we could implement our solu-

tion and calculate performance issues and compare it with other solutions.

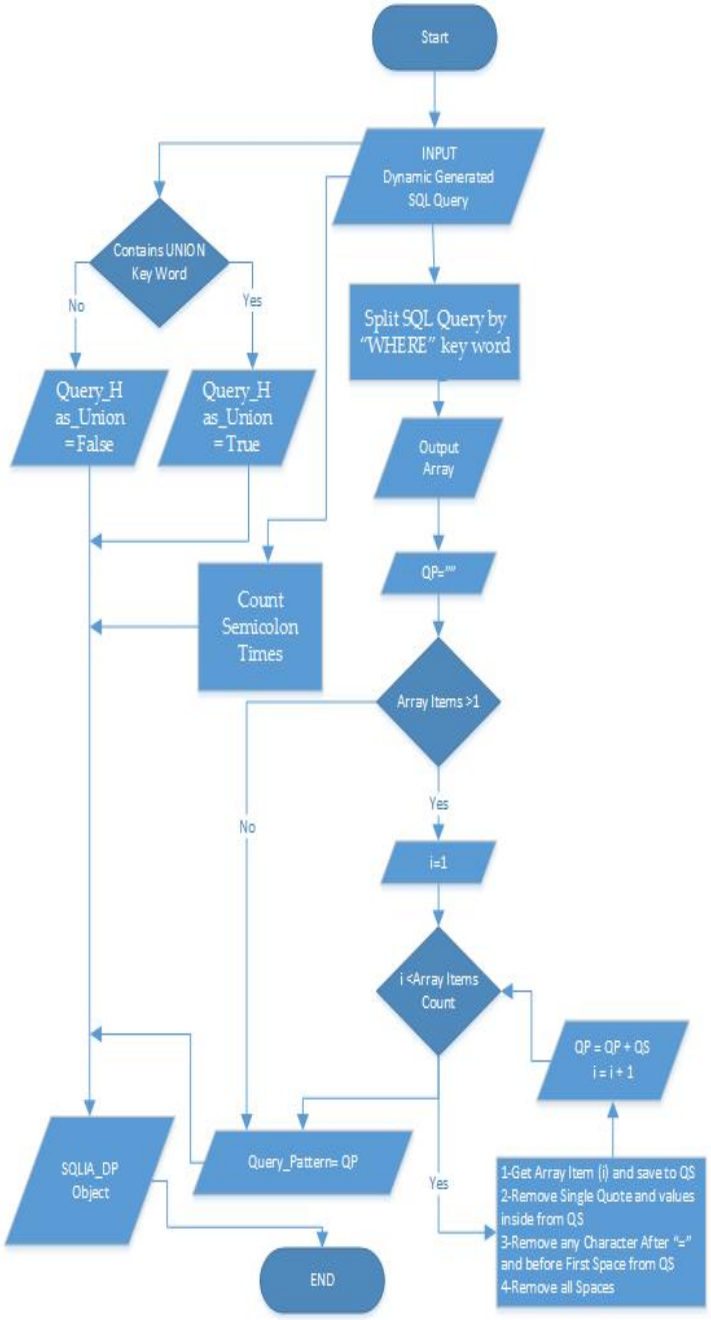

Fig. 4 Query Pattern Extraction Algorithm

REFERENCES

[1] Ramez Elmasri, Shamkant B. Navathe, "Fundamentals of Database Systems, Sixth Edition", 2011.

[2] Shubham Mukherjee, Sudeshna Bora, "SQL Injection: A Sample Review", 2015.

[3] Deevi Radha Rani, B.Siva Kumar, L.Taraka Rama Rao, V.T.Sai Jagadish, M.Pradeep, "Web Security by Preventing SQL Injection Using Encryption in Stored Procedures", 2012.

[4] OWASP (Open Web Application Security Project) https://www.owasp.org/index.php/Top_10_2013-Top_10, visited on May 2015.

[5] B.Hanmanthu, B.Raghu Ram, Dr.P.Niranjan," SQL Injection Attack Prevention Based on Decision Tree Classification", 2015.

[6] Atefeh Tajpour , Suhaimi Ibrahim, Mohammad Sharifi, "Web Application Security by SQL Injection DetectionTools", 2012.

[7] Amirmohammad Sadeghian, Mazdak Zamani, Azizah Abd. Manaf, "A Taxonomy of SQL Injection Detection and Prevention Techniques", 2013.

[8] Janot, E. and P. Zavarsky. "Preventing SQL Injections in OnlineApplications: Study, Recommendations and Java Solution Prototype Based on the SQL DOM." in OWASP App. Sec. Conference. 2008.

[9] McClure, R.A. and I.H. Kruger. "SQL DOM: compile time checking of dynamic SQL statements. in Software Engineering, 2005." ICSE 2005. Proceedings. 27th International Conference on. 2005.

[10] Mahima Srivastava, "Algorithm to Prevent Back End Database against SQL Injection Attacks", 2014.

[11] Biji.K.P, "Data Dictionary Based Mechanism against SQL Injection Attacks", 2015.

[12] Inyong Lee a, Soonki Jeong b, Sangsoo Yeoc, Jongsub Moond, "A novel method for SQL injection attack detection based on removing SQL query attribute values", 2011.

[13] Debabrata Kar, Suvasini Panigrahi, "Prevention of SQL Injection Attack Using Query Transformation and Hashing", 2013.

[14] R.Latha, Dr.E. Ramaraj, "SQL Injection Detection Based On Replacing The SQL Query Parameter Values", 2015.

[15] Swapnil Kharche1, Jagdish patil, Kanchan Gohad, Bharti Ambetkar, "Preventing Sql Injection Attack Using Pattern Matching Algorithym", 2015.

Adwan Yasin is an associate Professor, Former dean of Faculty of Engineering and Information Technology of the Arab American University of Jenin, Palestine. Previously he worked at Philadelphia and Zarka Private University, Jordan. He received his PhD degree from the National Technical University of Ukraine in 1996. His research interests include Computer Networks, Computer Architecture, Cryptography and Networks Security.

Nae'l A. Zidan received the B.S. in Computer Information Technology in 2005 from Arab American University Jenin (AAUJ), Palestine. He is a Master candidate of Computer Science at AAUJ, Palestine. He has 10+ years' experience of programming and development, networking, data-bases, and virtualization. His research interests include Computer Networks and Information Security.