An Analysis and Detection of SQL Injection Queries Using Feature Based Approach

Shivraj Sharma
PG Scholar, Department of Computer Science & Engineering, Shekhawati Institute of Engineering & Technology, Sikar, Rajasthan, India

***Abstract*- SQL Injection has been one of the most critical security threats for web based applications. As per Open Web Application Security Project (OWASP) top ten most critical threat list for web applications SQL Injection stands first in the list published in 2013 and 2017. Researchers and practitioners have been broaching various schemes to hammer away at the SQL injection problem. However, prevailing approaches either fall short to cope with the full scope of the problem or have bottlenecks that prevent their use and adoption. The basis behind SQL injection attack is fairly straightforward. When a web application receives user data as input, at that juncture, there is a chance for malicious user to enter carefully concocted data that cause the input to be construed as part of a SQL query instead of data. A successful SQL injection attack divulges critical confidential information to the hacker. In this paper a comprehensive review of various types of SQL injection attacks has been carried out. For the readers to understand better, a real time scenario of an vulnerable application has been designed that does not detect SQL injection attack query and this application lets that attack reveal the information stored in the underlying database to the malicious user. This paper proposes an enhanced approach of defensive coding to mitigate SQL injection attack. In the proposed work, features of various SQL injection queries have been closely examined to identify them. This technique has been named as feature based methodology to identify SQL injection queries. In this paper the analysis of the feature based SQL injection identification methodology has been presented.**

Index Terms Attack, Injection, SQL, Query, Vulnerability, Tautology, Hacker, Database, Web, Application, Threat, OWASP, Feature Based Approach.

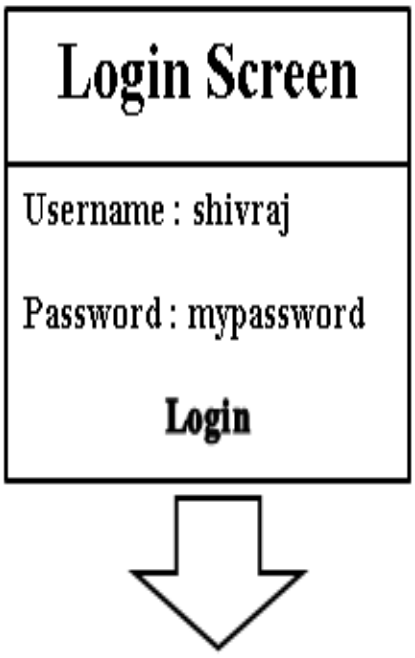
INTRODUCTION

These days individuals are more intrigued by setting up their business on the web or exchanging their regular business to online mode. These online organizations are completed by a web based application. These web applications stretch out offices of the organizations to their particular clients. The acknowledgment of these web applications is on upsurge inferable from simplicity of their utilization, their 24×7 accessibility and efficient qualities. Relatively every business has begun its web form with the goal that it never again gets behind in th period of data innovation. Presently, how these web applications play out their business to actualize their lumbering business. These web application are intelligent in nature do that they are anything but difficult to use by the clients. Clients who need the administrations of the web application can open the application on their web empowered gadgets, for example, cell phones, PCs, work areas and so forth.

Database is the most vital resource of a web application. It stores the helpful data of an association. Inferable from the significance of the data databases are inclined to assaults to increase unauthorized access to the data. SQL remains for Structured Query Language. SQL is the main language that is utilized by the application designers to build up the correspondence between the front end UIs with the backend databases. The backend database is associated with the web application. The data can be recovered at the client's request or included, altered or erased. SQL is the main language that is utilized to set the correspondence with the underneath database. A defenseless web application offers approach to unlawful access to the underneath database by the application of SQLIA. SQLIAs are propelled by the clients through exceptionally oversaw inputs that are provided to the web application. In this way it is constantly required to recognize the inquiries that can dispatch SQLIA. Work displayed here goes for the same.

SQL injection alludes to a class of code injection assaults in which information given by the client is incorporated into a SQL query so that piece of the client's info is dealt with as SQL code. By utilizing these vulnerabilities, an aggressor can submit SQL orders specifically to the database. These assaults are a genuine risk to any web application that gets contribution from clients and joins it into SQL inquiries to a hidden database. A large portion of the web applications operational on the web work along these lines and could subsequently be powerless against the SQL injection [1].

An unauthorized access to the database by a noxious client can risk its privacy, trustworthiness and specialist. Thus, the framework could bear overwhelming misfortune in giving appropriate administrations to its clients or it might confront finish demolition [2]. As indicated by the reviews of performed by Open Web Application Security Project (OWASP) in the years 2013 and 2017 the SQL Injection has been set at the primary spot of the rundown of 10 most basic web based application security dangers [3].



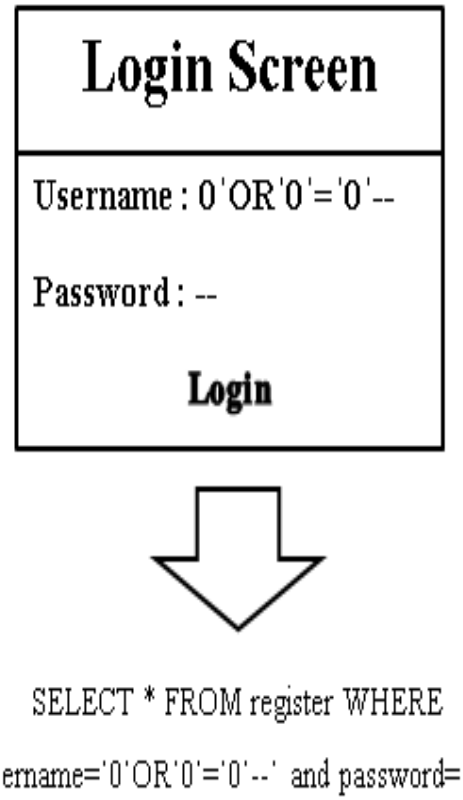


Fig. 1 Demonstration of SQL Injection

In the figure 1 demonstration of a SQL injection attack has been presented. At the login screen of a web application a user provides his username and password so that he can securely log in at the application. However, a malicious user provides a specially crafted username and or password. The application, in response of these inputs, generates a dynamic SQL query that is transmitted to the underneath database. Following query is generated in response of malicious inputs by the application:

SELECT * FROM register WHERE username= '0' OR'0'='0'--' and password= ' '

The above query when executed by the DBMS of the application allows the malicious user to login in the application. The hacker has used inline comment to bypass the authentication so password cannot be verified in this particular case.

To mitigate these vulnerabilities, many prevention techniques have been suggested such as manual approach, automated approach, secure coding practices, static analysis and so on. Though proposed approaches have achieved their goals to some extent, SQL Injection Vulnerabilities in Web applications

remain as a major concern among application developers [4].

The rest of the paper is organized as follows; Section II defines SQL injection types that are pertinent to our present discussion. Section III discussed the related work in the detection and prevention of SQLIA. Section IV presents the proposed work to detect SQLIA. In section V various results have been analyzed. Section VI concludes the discussion with a future scope.

II.SQL INJECTION TYPES

2.1 Tautologies

The main objective of tautology based attacks is to inject code in the conditional based statements so that they are always evaluated as true [5]. It is done, by simply making the where clause always true for every query, which results in bypassing the condition inside the SQL statement, for instance

SELECT * FROM User_info WHERE Username='RAHUL' and Password='12500'

In this query attacker can inject OR'1'='1' The resulting query will be:

SELECT * FROM User_Info WHERE Username=''OR'1'='1'-- and Password='Idontcare'

This enables the attacker to get all the records from the table. So in this way username and password of all the stored users in the database can be extracted.

Tautology- based SQL injection techniques are used by maximum hackers to bypass the authentication phase by just adding "--" inline comment, which makes the rest of the SQL command as comment.

2.2 Logically Incorrect Queries

The aim of the attacker is to gather all possible information about the structure and the schema of the tables and their fields inside the database. This belongs to the SQL manipulation attack where the error message generated by the database provides the attacker with an advantage. The working is quite simple. Here, injected wrong or incorrect SQL statement will generate some error message from databases that will provide the attacker the necessary information. Due to incongruous error handling, some internal database error message, specific to that particular database version, can be shown to the attacker because of which vital information about the database structure is revealed to the attackers which help him to conduct more exact attack that will have more impact on its target website [6].

We can limit the output errors or use customized error messages to avoid information leakage. The error that is shown will be devoid of any specific useful information.

Some specific examples that falls under this category are:-

SELECT * FROM User_unit1 WHERE Username=abc HAVING '1'='1'—and Password='123456'

Error generated:

"Column 'User_unit1.UserID' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause"

This error message displays the table with column name 'UserInfo.Username'

In this way all column names of the table can be extracted. We can have two types of error returns logical and syntactical.

The name of the columns are revealed by the logical errors which fetches the columns or table names while syntactical errors informs which parameters are vulnerable for an injection attack.

2.3 Union Query

In this type of SQLIA, which is injected based query which once joined with the safe query using the keyword UNION in order to get information one which related to the other tables from the concepts or the application [4] And also this type of attack is surely used in order to bypass the authentication process and also to fetch or extract data by inserting the UNION operator to the normal query. In the following example the second query is malicious because the text "--" is disregarded as it becomes comment for the SQL parser. However, if the query is executed the attacker receives the credit card information [11].

SELECT * FROM accounts WHERE id='212' UNION SELECT * FROM credit_card WHERE user='admin'—and pass='pass' [11].

2.4 Piggy- Backed Queries

This attack also inserts the malicious SQL queries into the normal SQL query. It is possible because many SQL queries can be processed if the operator ";" is added after each query. Following query is an

example. Note that the operator ";" is inserted at the end of query.

SELECT * FROM user WHERE id='admin' AND password='1234';DROP TABLE user;--;

The result of query 3 is the deletion of the user table [10].

Beside these attacks there are a few more attacks: Stored Procedures, Inference, Alternate Encodings; Blind Injection and Timing Attacks. However, these additional attacks are not pertinent to our current discussion.

III.RELATED WORK

A. Detection and Prevention of SQL Injection Attacks Using Novel Method in Web Applications[7]

Tejinderdeep Singh Kalsi and Navjot Kaur propose a new approach. This technique has two main phases: runtime analysis and static analysis. The first phase is dynamic analysis method which depends upon applying tracking methods to process and monitor the execution processes of all received queries. The next phase is a static analysis that is performing a string matching between the received SQL queries and previous expected SQL queries to stop any query that is described as a cautious query [7].

B. Study on SQL Injection Attacks: Mode, Detection and Prevention[8]

Subhranil Som, Sapna Sinha and Ritu Kataria suggest a strategy to change SQL query into number of helpful tokens by applying tokenization and after that encoding all literals, fields, table and information on the query by AES algorithm to avoid SQLIA [8].

C. Enhanced Approach to Detection of SQL Injection Attack[9]

Raja Prasad Karuparthi and Bing Zhou propose an enhanced approach of DUD by using a SQL injection sanitizer in the flow which enables a detection of attack at the initial level by minimizing the utilization of time in processing [9].

D. Machine Learning for SQL Injection Prevention on Server - Side Scripting[10]

Krit Kamtuo and Chitsutha Soomlek propose a framework of SQL injection prevention using compiler platform and machine learning. In this framework SQL injection command datasets are extracted. The input attribute will be sent to the machine learning models as well as prediction of SQL injection is reported [10].

IV. PROPOSED WORK

Proposed Solution

This section describes the algorithms which are involved in the proposed dissertation. In the proposed solution three algorithms have been proposed to design an application: These are,

1. Algorithm for Feature Based SQL Injection Query Detection
2. Algorithm for Detection of SQLIA Related to Tautologies containing Relational Operators
3. Algorithm for Detection of SQLIA Related to UNION and INTERSECT Queries

4.1 Algorithm for Feature Based SQL Injection Query Detection

In our proposed concept we have proposed an algorithm, which will be used for performing a check that the query fired by the user is an SQL Injection or not.

The algorithm contains the following steps:

Step 1: First the Query is provided as input in the form which we created for the Query Analysis

Step 2: Handling of the Nested Queries , firstly the components of the queries are separated by searching for the () parenthesis , in order to find the presence of any of the inner query and once the query presence is identified then the two or more than two queries are processed separately and have to undergo all the remaining steps of the algorithm.

Step 3: In the First Check the Query is check for the DROP keyword as , to avoid SQL Injection which can delete the table structure

Step 4: In the Second check we check for the validity of the SQL statement, in order to check whether it is proper SQL statement i.e. begin with SELECT,INSERT etc..

Step 5: In the third check we will avoid the SQL Injection for the value '1'='1' , this type of injection can be given in various ways , so we implemented this in two sub section , firstly containing OR statement , where we split the query on the basis of OR keyword and then checked the parameters for similarity and if same then it SQL Injection Attack and second a simple

Query which contains only statements like '1'='1' is handled after checking presence of = and checking parameters for equality.

Step 6: Apply the constraints for checking the relational operators based equality 8>7 ,5<6 etc.

Step 7: Also compound queries like the UNION and INTERSECT.

Step 8: Then we have check for the queries with intension of knowing the tables in the databases.

Step 9: Finally we have checked the queries with have no results, just fired in order to know the table structure.

Following flowchart shows how various steps of this algorithm are performed in order to get the desired results.

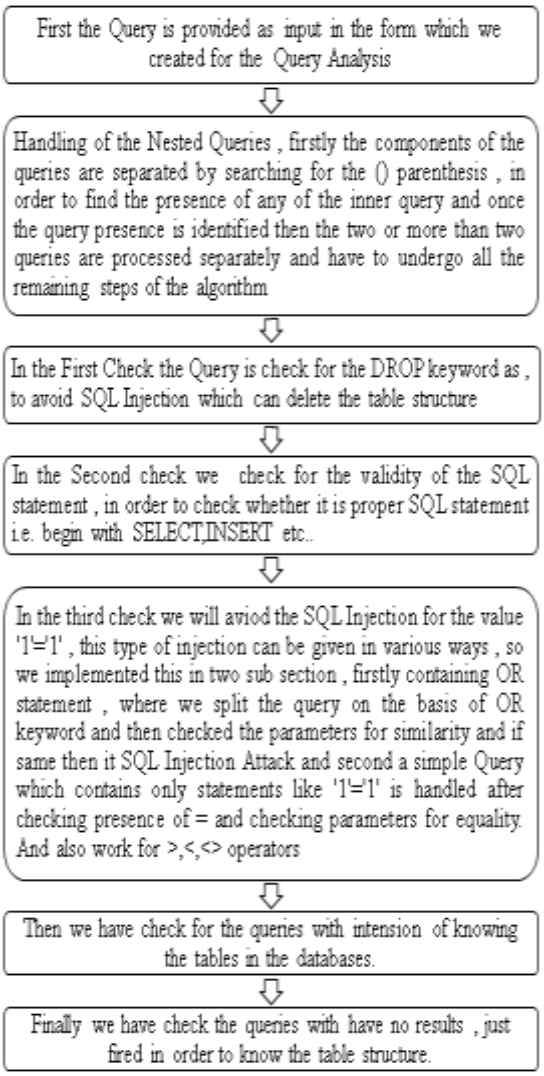


Fig. 2 Flowchart for Feature Based SQL Injection Query Detection

4.2 Algorithm for Detection of SQLIA Related to Tautologies containing Relational Operator

The algorithm of the login process is described in the following steps:

Step 1: First the Query is provided as input in the form which we created for the Query Analysis

Step 2: Search for presentation of Relational Operator like >,<,<>,=.

Step 3: Split into two part and extract operands on both sides.

Step 4: If the operands are numeric and condition of operator holds true , then stop else goto step 5.

Step 5: Execute the query 1 and display its results.

Step 6: Stop Application

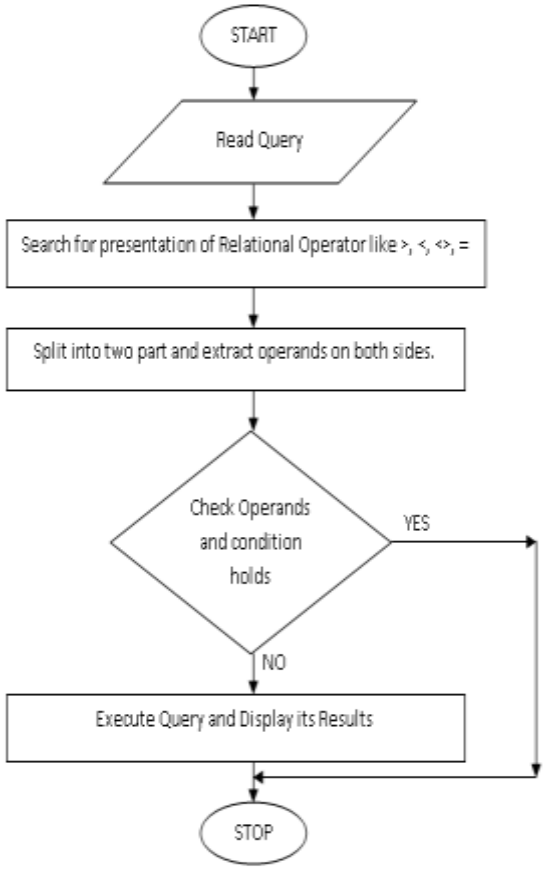


Fig. 3 Flowchart for Detection of SQLIA Related to Tautologies containing Relational Operator

4.3 Algorithm for Detection of SQLIA Related to UNION and INTERSECT Queries

Step 1: First the Query is provided as input in the form which we created for the Query Analysis

Step 2: Search for the keyword like UNION and INTERECTION and then split the queries into

two sub-queries using the function related to split.

Step 3: Examine the sub-query1 for the SQL Injection using the algorithm 3.2.3.

Step 4: If SQL Injection is found then stop the execution of query otherwise goto step 5.

Step 5: Execute the sub-query 1 and display its results.

Step 6: Examine the sub-query2 for the SQL Injection using the algorithm 3.2.3.

Step 7: If SQL Injection is found then stop the execution of query otherwise goto step 8.

Step 8: Execute the sub-query 2 and display its results.

Step 9: If no SQL Injection found in sub-query 1 and sub-query 2 then execute complete query and display its results

Step 10: Stop Application

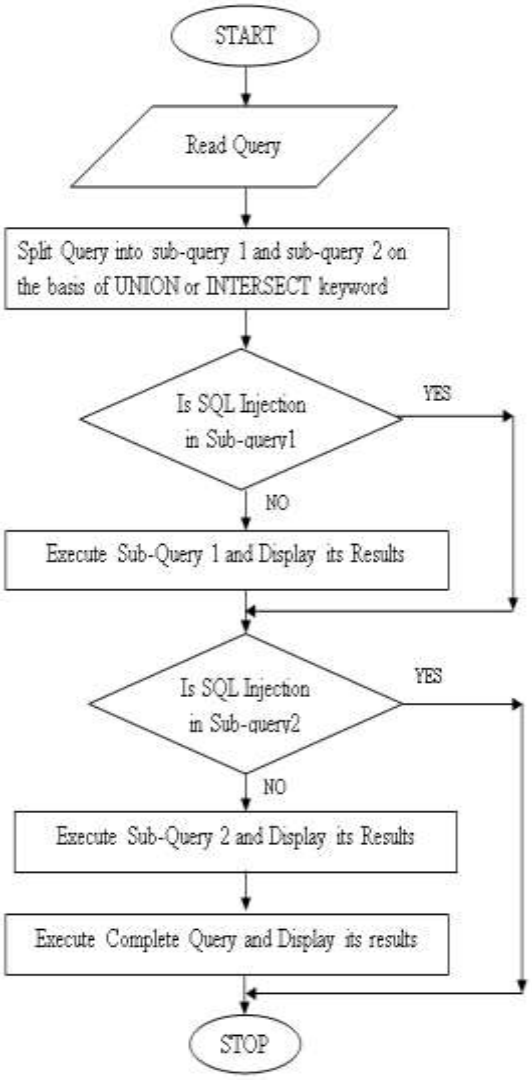


Fig. 4 Flowchart for Detection of SQLIA Related to UNION and INTERSECT Queries

V. RESULT ANALYSIS

Case I: Execution Result of Query 'X'='X'

SELECT * FROM employee WHERE emp_id='emp_001' OR 'x'='x'

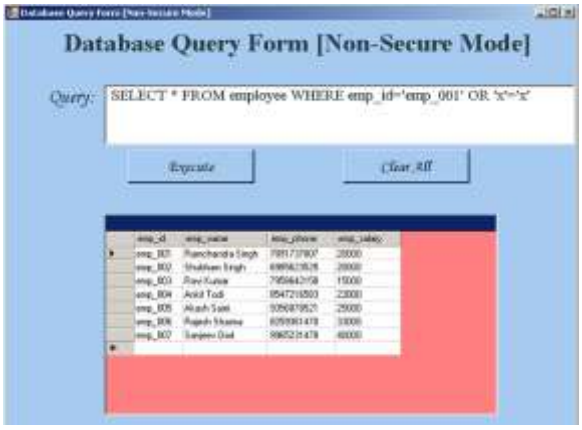


Fig. 5 Non-Secure Mode: Demonstration for Case I

Fig. 6 Secure Mode: Demonstration for Case I

Case II: SQL Injection Attack to get the column names

SELECT * FROM employee WHERE emp_id='emp_001' AND emp_name IS NULL

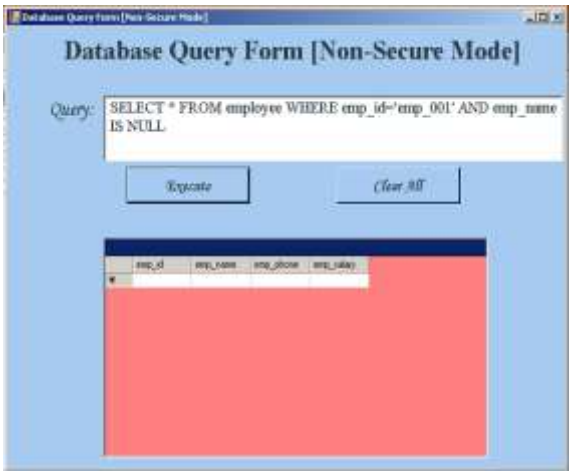


Fig. 7 Non-Secure Mode: Demonstration for Case II

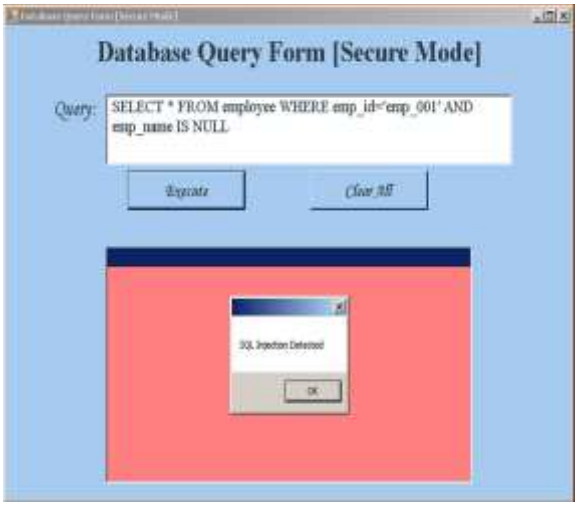


Fig. 8 Secure Mode: Demonstration for Case II

Case III: SQL Injection Attack to delete the table with its schema

SELECT * FROM employee; DROP TABLE employee

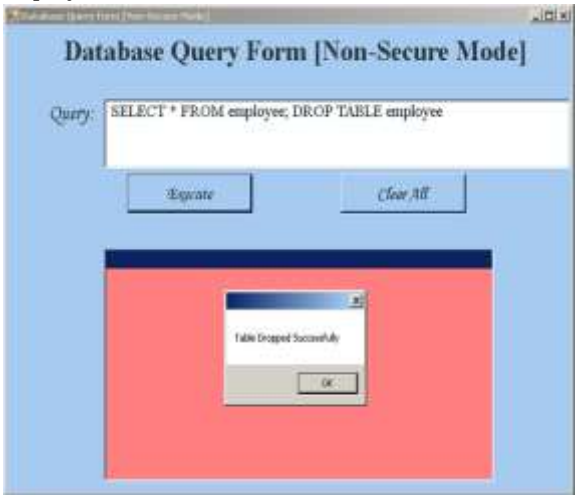


Fig. 9 Non-Secure Mode: Demonstration for Case III

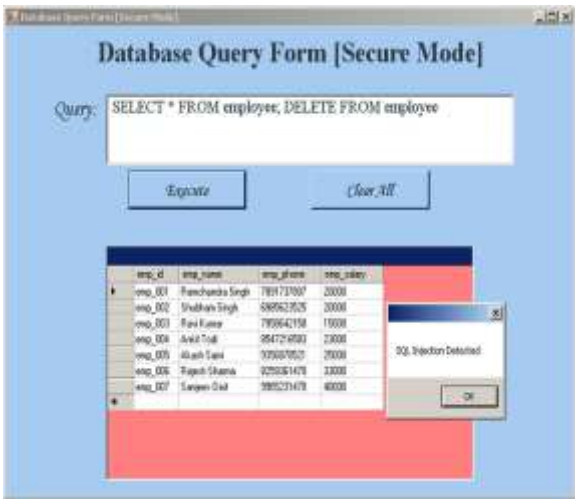


Fig. 10 Secure Mode: Demonstration for Case III

Case IV: SQL Injection Attack to get the table names from the database

SELECT table_name FROM information_schema.tables

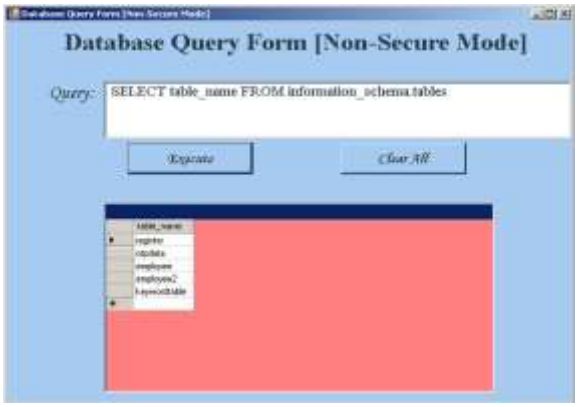


Fig. 11 Non-Secure Mode: Demonstration for Case IV

Fig. 12 Secure Mode: Demonstration for Case IV

Case V: SQL Injection Attack pertaining to relational operator based tautology

SELECT * FROM employee WHERE emp_id='emp_001' AND 5>2

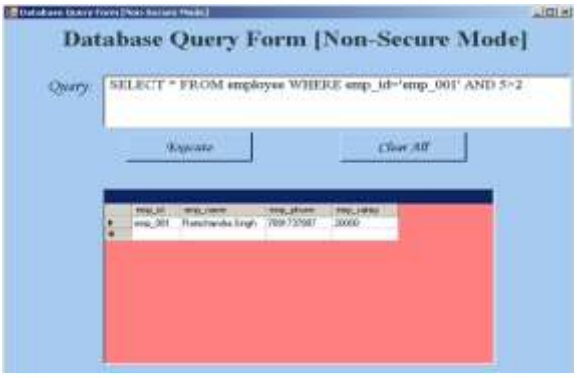


Fig. 13 Non-Secure Mode: Demonstration for Case V

Fig. 14 Secure Mode: Demonstration for Case V

Case VI: SQL Injection Attack using UNION of SQL Queries

SELECT * FROM employee UNION SELECT * FROM employee2 WHERE 5>2

Fig. 15 Non-Secure Mode: Demonstration for Case VI

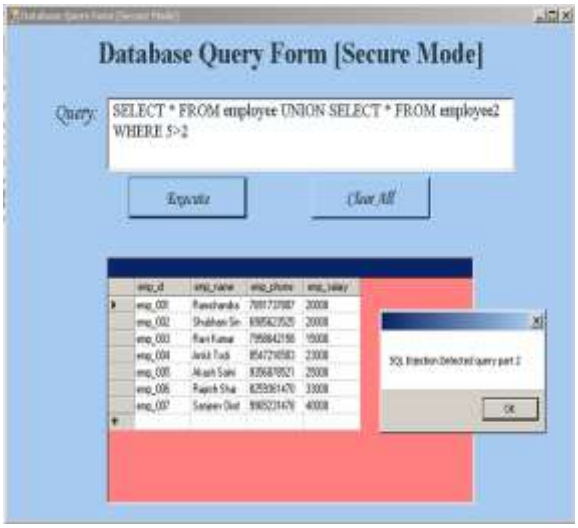


Fig. 16 Secure Mode: Demonstration for Case VI

Table I. Comparison of the vulnerable and proposed approach

Sr. No.	SQL Injection Query	Result Analysis	
		Non-Secure Mode	Secure Mode
1	Tautology based SQL Injection Attack		
	SELECT * FROM employee WHERE emp_id='emp_001' OR 'x'='x'	All the records retrieved from table employee	SQL Injection Attack Query Detected
2	SQL Injection Attack to get the column names		
	SELECT * FROM employee WHERE emp_id='emp_001' AND emp_name IS NULL	All Column Names from table employee retrieved	SQL Injection Attack Query Detected
3	SQL Injection Attack to Delete the Table with its schema		
	SELECT * FROM employee; DROP TABLE employee	Table employee deleted with its schema	SQL Injection Attack Query Detected
4	SQL Injection Attack to get the table names in the database		
	SELECT table_name FROM information_schema.tables	All the table names retrieved from the database	SQL Injection Attack Query Detected
5	SQL Injection Attack pertaining to Relational Operator based Tautology		
	SELECT * FROM employee WHERE emp_id='emp_001' AND 5>2	All the records retrieved from table employee	SQL Injection Attack Query Detected
6	SQL Injection Attack to Delete all the records of a table		

	SELECT * FROM employee ;DELETE FROM employee	All the records from table employee deleted	SQL Injection Attack Query Detected
7	SQL Injection Attack using UNION of SQL Queries		
	SELECT * FROM employee UNION SELECT * FROM employee2 WHERE 5>2	All the records retrieved which are present in the result sets of either of the sub-queries	SQL Injection Attack Query Detected
8	SQL Injection Attack using INTERSECT of SQL Queries		
	SELECT * FROM employee INTERSECT SELECT * FROM employee2 WHERE 5>2	All the records retrieved which are present in the result sets of both sub-queries	SQL Injection Attack Query Detected
9	SQL Injection Attack using EXCEPT of SQL Queries		
	SELECT * FROM employee EXCEPT SELECT * FROM employee2 WHERE 5>2	All the records retrieved which are present in the result set of first query but second query	SQL Injection Attack Query Detected

VI.CONCLUSION

Any web application is prone to the hacker's attacks. Thus, it is always the first and the foremost requirement to safeguard the web applications from such attacks. In this paper, we have presented the approach based on the algorithm which examines the query segment for the SQL Injection based attack before preceding the query. This code when clubbed with the website or any other online application helps to filter out the data which is entered by the end-user, and will precede which only that data which is free for the SQL Injection related queries.

Future scope of the work, still lies on dealing with the high end queries, as our dissertation relies on the segmentation based approach, in which we segment the query and then analyze for the SQL Injection , so in future work we will try to extend this to deal with much more complicated queries. Nested queries can also be incorporated in the future work. Handling nested queries can be achieved using the same segmentation approach. First of all presence of any parenthesis can be identified. Now presence of all the inner queries can be found out. These inner queries can be checked for the SQLIA and once verified the complete query can be checked for the SQLIA. In case the complete query is not launching any SQLIA to the underneath database then the whole query can be granted permission for the execution otherwise the execution of the query can be halted.

Following query would be a stimulus for the interested researchers:

SELECT * FROM employee WHERE emp_id IN (SELECT emp_id FROM employee2 WHERE emp_salary=20000 UNION SELECT emp_id FROM employee WHERE emp_salary =33000 OR 2=2)

REFERENCES

[1] William G.J. Halfond, Jeremy Viegas and Alessandro Orso, "A Classification of SQL Injection Attacks and Countermeasures", IEEE, 2006

[2] Diallo Abdoulaye Kindy, Al-Sakib Khan Pathan, "A Survey on SQL Injection: Vulnerabilities, Attacks and Prevention Techniques", IEEE 15th International Symposium on Consumer Electronics (ISCE), vol. 11, pp. 468-471, 14-17 June 2011

[3] OWASP Top 10 - 2017, "OWASP Top 10 Most Critical Web Application Security Risks", pdf of the document is available at https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf [last accessed on 21 July 2018]

[4] Diallo Abdoulaye Kindy, Al-Sakib Khan Pathan, "A Detailed Survey on various aspects of SQL Injection in Web Applications: Vulnerabilities, Innovative Attacks and Remedies", International Journal of Communication Networks and Information Security (IJCNIS), vol. 5, no. 2, pp. 80-92, August 2013

[5] Ashish John, "SQL Injection Prevention by adaptive algorithm", IOSR Journal of Computer Engineering, vol. 17, pp. 19-24, January 2015.

[6] Pankajdeep Kaur, Kanwal Preet Kour, "SQL Injection: Study and Augmentation", IEEE International Conference on Signal Processing,

Computing and Control (ISPCC), pp. 102-107, 24-26 September 2015

[7] Tejinderdeep Singh Kasli, Navjot Kaur,"Detection and Prevention of SQL Injection Attacks using Novel Method in Web Applications", International Journal of Advances in Engineering and Technology (IJAET), vol. 6, Issue 4, pp. 11-15, December 2015

[8] Subranil Som, Sapna Sinha and Ritu Kataria,"Study on SQL Injection Attacks: Mode, Detection and Prevention", International Journal of Engineering Applied Sciences and Technology (IJEAST), vol. 1, Issue 8, pp. 23-29, July 2016

[9] Raja Prasad Karuparthi and Bing Zhou, "Enhanced Approach to Detection of SQL Injection attack", IEEE International Conference on Machine Learning and Applications, pp. 466-469, 18-20 December 2016

[10] Krit Kamtuo, Chitsutha Soomlek, "Machine Learning for SQL Injection Prevention on Server-Side Scripting", IEEE International Computer Science and Engineering Conference(ICSEC), pp. 1-6, 14-17 December 2016

[11] Dr. Ahmad Ghafarian, "A Hybrid Method for Detection and Prevention of SQL Injection Attacks", IEEE Computing Conference, pp. 833-838, 18-20 July 2017