Modified Algorithm for SQL Injection Detection in Single and Nested Queries

Khushboo Gupta[1], Sayar Singh Shekhawat[2]
[1]*M. Tech. Scholar, Department of Computer Science ,Arya Institute of Engineering & Technology, Jaipur, Rajasthan*
[2] *Head of Department, Department of Computer Science, Arya Institute of Engineering & Technology, Jaipur, Rajasthan*

***Abstract*- Today is the world of the information sharing and the information is shared using the information based websites, E-commerce websites. But every category of website contains the information which is protected by the login credentials but intruders try to access such information using SQL Injection. Here in this paper , we propose the algorithm , which is simulated in the VS 2010 and identifies the various types of the SQL Injection like schema related queries, tautologies, etc**

***Index Terms*- Data Security, Hacking , SQL Injection.**

1. INTRODUCTION

SQL injection attacks are one of the topmost threats for applications written for the Web. These attacks are launched through specially crafted user input on web applications that use low level string operations to construct SQL queries.SQL injection vulnerability allows an attacker to flow commands directly to a web application's underlying database and destroy functionality or confidentiality. SQL injection vulnerabilities have been described as one of the most serious threats for Web applications [3, 11]. Web applications that are vulnerable to SQL injection may allow an attacker to gain complete access to their underlying databases. Because these databases often contain sensitive consumer or user information, the resulting security violations can include identity theft, loss of confidential information, and fraud. In some cases, attackers can even use an SQL injection vulnerability to take control of and corrupt the system that hosts the Web application. Web applications that are vulnerable to SQL Injection Attacks (SQLIAs) are widespread—a study by Gartner Group on over 300 Internet Web sites has shown that most of them could be vulnerable to SQLIAs. In fact, SQLIAs have successfully targeted high-profile victims such as Travelocity, FTD.com, and Guess Inc.

SQL injection refers to a class of code-injection attacks in which data provided by the user is included in an SQL query in such a way that part of the user's input is treated as SQL code. By leveraging these vulnerabilities, an attacker can submit SQL commands directly to the database. These attacks are a serious threat to any Web application that receives input from users and incorporates it into SQL queries to an underlying database. Most Web applications used on the Internet or within enterprise systems work this way and could therefore be vulnerable to SQL injection.

The cause of SQL injection vulnerabilities is relatively simple and well understood: insufficient validation of user input. To ad-dress this problem, developers have proposed a range of coding guidelines (e.g., [18]) that promote defensive coding practices, such as encoding user input and validation. A rigorous and systematic application of these techniques is an effective solution for prevent-ing SQL injection vulnerabilities. However, in practice, the application of such techniques is human-based and, thus, prone to errors. Furthermore, fixing legacy code-bases that might contain SQL injection vulnerabilities can be an extremely labour-intensive task.

Main cause of SQL injection:

Web application vulnerabilities are the main causes of any kind of attack. In this section, vulnerabilities that might exist naturally in web applications and can be exploited by SQL injection attacks will be presented:

Invalidated input: This is almost the most common vulnerability on performing a SQLIA. There are some parameters in web application, are used in SQL queries. If there is no any checking for them so can be abused in SQL injection attacks. These parameters may contain SQL keywords, e.g. INSERT, UPDATE or SQL control characters such as quotation marks and semicolons.

Generous privileges: Normally in database the privileges are defined as the rules to state which database subject has access to which object and what operation are associated with user to be allowed to perform on the objects. Typical privileges include allowing execution of actions, e.g. SELECT, INSERT, UPDATE, DELETE, DROP, on certain objects.

Web applications open database connections using the specific account for accessing the database. An attacker who bypasses authentication gains privileges equal to the accounts. The number of available attack methods and affected objects increases when more privileges are given to the account. the worst case happen If an account can connect to system that is associated with the system administrator because normally has all privileges.

Uncontrolled Variable Size: If variables allow storage of data be larger than expected consequently allow attackers to enter modified or faked SQL statements. Scripts that do not control variable length may even open the way for attacks, such as buffer overflow.

Error message: Error messages that are generated by the back-end database or other server-side programs may be returned to the client-side and presented in the web browser. These messages are not only useful during development for debugging purposes but also increase the risks to the application. Attackers can analyze these messages to gather information about database or script structure in order to construct their attack.

Variable Orphism: The variable should not accept any data type because attacker can exploit this feature and store malicious data inside that variable rather than is suppose to be. Such variables are either of weak type, e.g. variables in PHP, or are automatically converted from one type to another by the remote database.

Dynamic SQL: SQL queries dynamically built by scripts or programs into a query string. Typically, one or more scripts and programs contribute and finally by combining user input such as name and password, make the WHERE clauses of the query statement. The problem is that query building components can also receive SQL keywords and control characters. It means attacker can make a completely different query than what was intended.

Client-side only control: If input validation is implemented in client-side scripts only, then security functions of those scripts can be overridden using cross-site scripting. Therefore, attackers can bypass input validation and send invalidated input to the server-side.

Into Out file support: Some of RDBMS benefit from the INTO OUTFILE clause. In this condition an attacker can manipulate SQL queries then they produce a text file containing query results. If attackers can later gain access to this file, they can abuse the same information, for example, bypass authentication.

Multiple statements: If the database supports UNION so, attacker has more chance because there are more attack methods for SQL injection. For instance, an additional INSERT statement could be added after a SELECT statement, causing two different queries to be executed. If this is performed in a login form, the attacker may add him or herself to the table of users.

Sub-selects: Supporting sub-selects is weakness for RDBMS when SQL injection is considered. For example, additional SELECT clauses can be inserted in WHERE clauses of the original SELECT clause. This weakness makes the web application more vulnerable, so they may be penetrated by malicious users easily.

2. SQL INJECTION ATTACKS

There are different methods of attacks that depending on the goal of attacker are performed together or sequentially. For a successful SQLIA the attacker should append a syntactically correct command to the original SQL query. Now the following classification of SQLIAs will be presented.

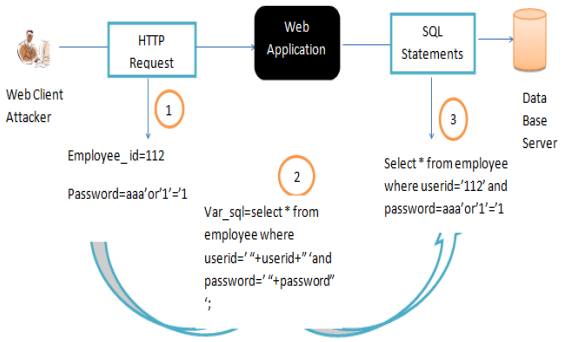

Figure 1. SQL Injection Attacks

Tautologies: This type of attack injects SQL tokens to the conditional query statement to be evaluated always true. This type of attack used to bypass authentication control and access to data by exploiting vulnerable input field which use WHERE clause. "SELECT * FROM employee WHERE userid = '112' and password ='aaa' 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 a query is rejected , an error message is returned from the database including useful debugging information. This error messages help attacker to find vulnerable parameters in the application and consequently database of the application. In fact attacker injects junk input or SQL tokens in query to produce syntax error, type mismatches, or logical errors by purpose. In this example attacker makes a type mismatch error by injecting the following text into the pin input field:

1) OriginalURL: http://www.arch.polimi.it/eventi/?id_nav=8864
2) SQL Injection: http://www.arch.polimi.it/eventi/?id_nav=8864'
3) Error message showed: SELECT name FROM Employee WHERE id =8864\'

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 join injected query to the safe query by the word UNION and then can get data about other tables from the application. Suppose for our examples that the query executed from the server is the following: SELECT Name, Phone FROM Users WHERE Id=$id by injecting the following Id value:

$id=1 UNION ALL SELECT creditCardNumber,1 FROM CreditCarTable

We will have the following query:

SELECT Name, Phone FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber,1 FROM CreditCarTable which will join the result of the original query with all the credit card users.

Piggy-backed Queries: In this type of attack, intruders exploit database by the query delimiter, such as ";", to append extra query to the original query. With a successful attack database receives and execute a multiple distinct queries. Normally the first query is legitimate query, whereas following queries could be illegitimate. So attacker can inject any SQL command to the database. In the following example, attacker inject " 0; drop table user " into the pin input field instead of logical value. Then the application would produce the query: SELECT info FROM users WHERE login='doe' AND pin=0; drop table users Because of ";" character, database accepts both queries and executes them. The second query is illegitimate and can drop users table from the database. It is noticeable that some databases do not need special separation character in multiple distinct queries, so for detecting this type of attack, scanning for a special character is not impressive solution.

3. LITERATURE SURVEY

Ke Weiet al. [1] proposed a novel system to guard against the attacks focused at stored procedures. This strategy joins static application code investigation with runtime approval to kill the event of such attacks. In the static section, a put away technique parser is planned, and for any SQL proclamation which relies on upon client inputs, this parser is utilized to instrument the fundamental articulations keeping in mind the end goal to contrast the first SQL explanation structure with that including client inputs. The sending of this system can be robotized and utilized on a need-just premise.

William G.J. Halfondet al. [2] displayed a broad audit of the diverse sorts of SQL injection attacks known not. For every sort of assault, portrayals and cases of how attacks of that sort could be performed are given. He also presented and broke down existing discovery and aversion systems against SQL injection attacks. For every system, its qualities and shortcomings are talked about in tending to the whole scope of SQL injection attacks.

William G.J. Halfondet al. [3] proposed another exceptionally computerized approach for element discovery and counteractive action of SQLIAs. Instinctively, this methodology works by recognizing "trusted" strings in an application and permitting just these trusted strings to be utilized to make the semantically important parts of a SQL inquiry, for example, watchwords or administrators. The general component that we use to actualize this methodology depends on element polluting, which checks and tracks certain information in a project at runtime.

SruthiBandhakaviet al. [4] proposed a straightforward and novel component, called Candid, for mining developer expected inquiries by powerfully assessing keeps running over benevolent competitor inputs. This component is hypothetically very much established and depends on surmising proposed inquiries by considering the typical inquiry registered on a system run. This methodology has been actualized in an apparatus called Candid that retorts Web applications written in Java to protect them against SQL injection attacks.

Jin-Cherng Linet al. [5] introduced a propelled proposition embracing the idea of utilization level security entryway and more successfully determining the issue than comparable doors or intermediaries. This framework comprises of discovery testing, acceptance capacities and redirection instrument.

Mehdi Kianiet al.[6] portrayed an irregularity based methodology which uses the character conveyance of certain segments of HTTP solicitations to recognize already inconspicuous SQL injection attacks. This methodology requires no client collaboration, and no alteration of, or access to, either the backend database or the source code of the web application itself. Its commonsense results recommend that the model proposed in this paper is better than existing models at distinguishing SQL injection attacks. Specialists additionally assess the adequacy of the model at recognizing distinctive sorts of SQL injection attacks.

Yu Chin Chenget al. [7] proposed a sort of novel Embedded Markov Model (EMM) to recognize diverse web application attacks, screen the on-line client conduct and safeguard the vindictive client immediately. Contrasting with past web application attacks distinguishing approaches, this EMM methodology can identify client's refuted data mistakes as well as discover the nonsensical page move conduct. By identifying outlandish page move, we can quickly safeguard the pernicious or senseless client conduct to maintain a strategic distance from the further web framework disappointments and touchy data exposure.

Proposed Work

A new algorithm is presented to protect Web applications or even the desktop application against SQL injection Attacks. SQL Injection Attacks are a class of attacks that many of these systems are highly vulnerable to, and there is no known foolproof defense against such attacks. Some predefined methods and integrated approach of encryption method with secure hashing can be applied in the database to avoid attack on login phase. This combined method will be applied to a system where user's information is kept and the designing of this system will be done by using .Net Architecture.

Algorithm Proposed

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:

1. First the Query is provided as input in the form which we created for the Query Analysis
2. In the First Check the Query is check for the DROP keyword as , to avoid SQL Injection which can delete the table structure
3. 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..
4. 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.
5. Apply the constraints for checking the relational operators based equality 8>7 ,5<6 etc.
6. Also cover the nested queries based SQL Injection.
7. Then we have check for the queries with intension of knowing the tables in the databases.

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

In our proposed algorithm, we work on the feature analysis of the SQL Attacks. Our algorithm can be well explained with the help of the following flowchart.

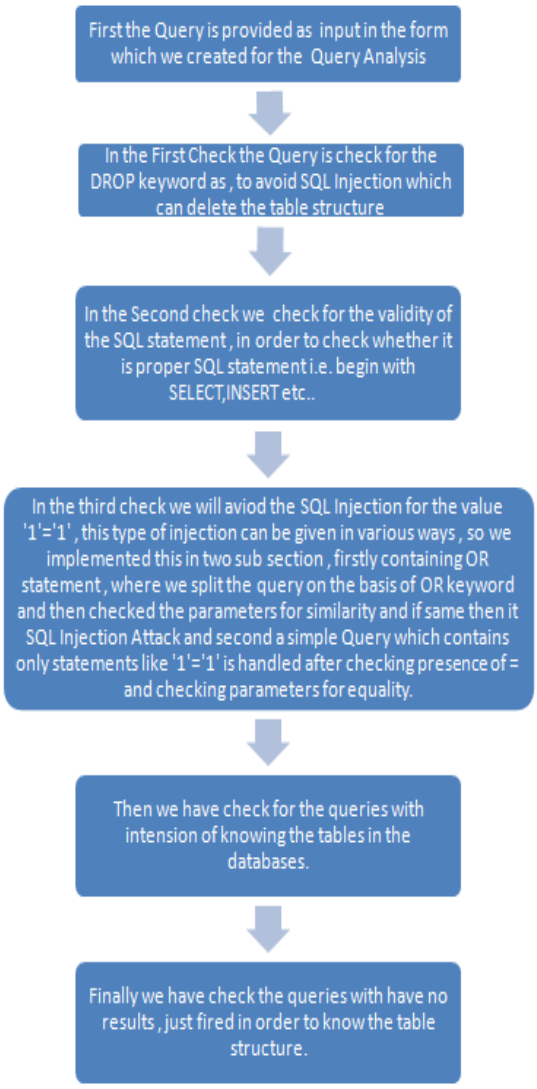

Figure 2. Proposed Work Flow

Consider the following query statement ,

Figure 3. simulated the process of the query SELECT * FROM employee WHERE emp_id= 'e0v' OR 'x'='x';

In the table employee we donot have any emp_id with ‘e00v’ and the result of the query executed is,

As the result of the SQL injection get detected as ‘X’ = ‘X’.

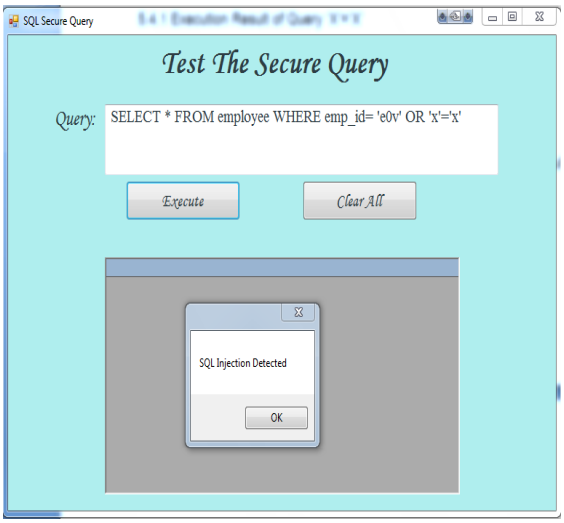

Figure 3. Proposed Implementation

Experiment Results

Pattern#	String Pattern	Expected Result	
		Secure	Insecure
1	‘OR”=’	Login failed	Login Successful
2	0’ or ‘1’=’1	Login failed	Login Successful
3	1’ or ‘1’=’1	Login failed	Login Successful
4	‘ OR ‘1’=’1’	Login failed	Login Successful
5	1’ or ‘a’=’a	Login failed	Login Successful
6	; and 1=1 and 1=2	Login failed	Login Successful
7	“ ’ or 1=1 - -“	Login failed	Login Successful
8	OR '1'='1'"	Login failed	Login Successful
9.	emp_id= 'x' AND emp_name IS NULL	Attack Identified	Columns reterived
10.	SELECT * FROM employee; DROP TABLE employee	Attack Identified	Table Employee Deleted
11.	SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES	Attack Identified	Table Names in the database

Table 1. Experiment Results

The Table 1. Shows the execution of the query types in both the base and proposed implementation and on the comparison it is shown that the proposed implementation is able to identify the various types of attacks.

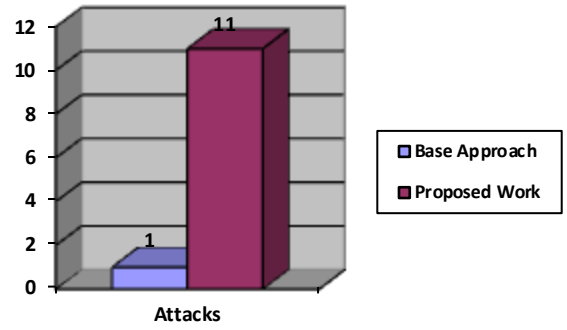

Figure 4. SQL Attacks Comparison Graph

4. CONCLUSION

SQL Injection Attacks are the biggest problem for the entire website whether related to social networking, e-commerce or any other time of applications. The unauthorized access required to be stopped. Implementing the algorithm proposed in the website interfaces will help the websites to stop the SQL Injection Attacks to the greater extents.

REFERENCES

[1] Ke Wei, M. Muthuprasanna, SurajKothari . "Preventing SQL Injection Attacks in Stored Procedures" IEEE (Year 2006)

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

[3] William G.J. Halfond, Alessandro Ors. "WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation" IEEE Transactions On Software Engineering, Vol. 34, No. 1. (Year 2008)

[4] SruthiBandhakavi, PrithviBisht. "CANDID: Preventing SQL Injection Attacks using Dynamic Candidate Evaluations" CCS'07, October 29–November 2 (Year 2008)

[5] Jin-Cherng Lin, Jan-Min Chen. "The Automatic Defense Mechanism for Malicious Injection Attack" IEEE Seventh International Conference on Computer and Information Technology.(Year 2007)

[6] Mehdi Kiani, Andrew Clark and George Mohay. "Evaluation of Anomaly Based Character Distribution Models in the Detection of SQL Injection Attacks" IEEE. The Third International Conference on Availability, Reliability and Security (Year 2008)

[7] 7). Yu-Chin Cheng. "Defending On-Line Web Application Security with User-Behavior Surveillance" IEEE (Year 2008)

[8] HossainShahriar and Mohammad Zulkernine."MUSIC: Mutation-based SQL Injection Vulnerability Checking" IEEE The Eighth International Conference on Quality SoftwareYear(2008)

[9] NunoAntunes, Marco Vieira."Comparing the Effectiveness of Penetration Testing and Static Code Analysis on the Detection of SQL Injection Vulnerabilities in Web Services" IEEE Pacific Rim International Symposium on Dependable ComputingYear(2009)

[10] Dwen-Ren Tsai,Allen Y. Chang,Peichi Liu,Hsuan-Chang Chen."Optimum Tuning of Defense Settings for Common Attacks on the Web Applications" IEEE Year (2009)

[11] Ivano Alessandro Elia. "Comparing SQL Injection Detection Tools Using Attack Injection: An Experimental Study" IEEE.21st International Symposium on Software Reliability Engineering. (Year 2010)

[12] Xin Wang. "Hidden Web Crawling for Sql Injection Detection" IEEE (Year 2010)

[13] TIAN Wei , XU Jing, LIAN Kun-Mei, ZHANG Ying,YANGJu-feng. "Research on mock attack testing for SQL injection vulnerability in multi-defense level web applications" IEEE Year(2010)

[14] Zhang Xin-hua,Wang Zhi-jian."A Static Analysis Tool for Detecting Web Application Injection Vulnerabilities for ASP Program " IEEE Year(2010)

[15] Lijiu Zhang, Qing Gu, ShushenPeng, Xiang Chen, Haigang Zhao, Daoxu Chen."D-WAV: A Web Application Vulnerabilities Detection Tool Using Characteristics of Web Forms" IEEE Fifth International Conference on Software Engineering Advances Year(2010)