ISSN: 2277-9655
(I2OR), Publication Impact Factor: 3.785

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

SQL INJECTION DETECTION AND PREVENTION USING HYBRID APPROACH

Manpreet Kaur*, Anand Kumar Mittal
M.Tech Student GKU, Talwandi Sabo.
Assistant Professor GKU, Talwandi Sabo.

ABSTRACT

Web applications such as blogs, social network, webmail, bank etc have become our way of life. Among the top ten web application vulnerabilities published by Open Web Application Security Project, SQL Injection Attack (SQLIA) is the most vulnerable. An SQLIA occurs when an attacker changes the intended effect of an SQL query by inserting (or injecting) new SQL keywords or operators into the query thereby gaining unauthorized access to a database in order to view or manipulate restricted data. In proposed work, a new hybrid approach is developed to detect and prevent the sql injection attack in sql queries. Proposed system is tested using various queries given by various users and results are evaluated very accurate.

KEYWORDS— SQL Injection Attack, Hybrid Approach, Negative tainting approach.

INTRODUCTION

Web applications are being in a much wider area these days, online shopping, online banking and social networking is some of the key users of these [1]. All these users have the utmost priority for their privacy and security and these are the most vulnerable while being online. Web applications such as blogs, social network, webmail, bank etc have become way of life. The omnipresence of web applications has made them a usual target for nasty minds. Web applications are susceptible to a number of vulnerabilities which can be due to a design flaw or an implementation bug. Among the top ten web application vulnerabilities published by Open Web Application Security Project, SQL Injection Attack (SQLIA) is the most vulnerable. According to OWASP, SQL injection vulnerabilities were reported in 2008, building up 25% of all reported vulnerabilities for web applications [2]. An SQLIA occurs when an attacker changes the intended effect of an SQL query by inserting (or injecting) new SQL keywords or operators into the query thereby gaining unauthorized access to a database in order to view or manipulate restricted data. SQL injection attack allows attackers to gain control of the original query, illegal access to the database and extract or transform the database [3]. The main cause of SQL injection vulnerabilities is: attackers use the input support to attack strings that contains special database commands. An SQLIA occurs when an attacker changes the SQL control by inserting new keywords [4]. A successful SQLI attack hinder privacy integrity and availability of information in the database. In most of cases, SQL injection is used to initiate the denial of service attack on web applications. The strictness of the attacks depends on the role or account on which the SQL statement is executed.

An attacker needs to know loop holes in the application before launch an attack. Attackers use: input format, timing, performance and error message to decide the type of attack suitable for an application. Database is the mind of many web applications, basis for which database more and more coming under great number of attacks. SQLIAs occur when data provided by the user is incorporated directly in the query and is not appropriately validated.

SQL Injection Types

The SQL injection attacks can be performed using a variety of techniques. Some of them are specified as follows:

ISSN: 2277-9655
(I2OR), Publication Impact Factor: 3.785

First Order Attack: Attackers aim the database with strings attached to an input field and receives the answer immediately. Such attacks which exploit the lack of validation in the input field parameter are known as first order attacks [1].

Second Order Attack: An attacker attacks the database with inserting mean queries in a table but implement these queries from other actions [1].

Tautology Attack: Conditional operators are used by the attackers in the SQL queries such that the query always evaluates to TRUE [3,4,6,10].

*For example, SELECT * FROM employee WHERE name = '' OR '1'='1';*

Logically Incorrect Queries: An illegal query used by the attacker to glance at the whole database [3,4,6,10].
*For example, "SELECT * FROM employee WHERE id =" + name + ";"*

Piggy-backed Query: In this attack, attacker tries to add on supplementary queries but terminates the first query by inserting ";" [3,4,7].

*For example, SELECT * FROM employee WHERE id=1;DROP TABLE employee;*

Inference: The main goal of the inference based attack is to change the activities of a database or application. There are two well-known attack techniques that are based on inference: blind injection and timing attacks

Timing attack: In these types of attack an attacker observes the database delays in database response and gathers the information. WAITFOR, IF, ELSE, BENCHMARK [3,4] cause delay in database response.

*For example, SELECT * FROM employee WHERE id=1-SLEEP(15);*

Blind injection: In this situation an attacker performs queries that have a Boolean result [3].

*For example: SELECT * FROM employee WHERE id = '1008' AND 1=1;*

Alternate Encoding: Attacker modifies the injection query by using alternate encoding such as hexadecimal, ASCII and Unicode [1,2] .

*For example: SELECT * FROM employee WHERE id=unhex('05');* .

Union Query: An attacker makes use of vulnerable parameters and attach injected query to the safe query by the word UNION and get data about other tables from the application.

*For example: Select * from company where name="" " union select * from employee –„and Password="anypwd"*

Stored Procedure [10]: A stored procedure is a cluster of Transact-SQL statements compiled into a single execution plan. As stored procedure could be coded by programmer, attacker can execute these built in procedures.

LITERATURE SURVEY

K.Chavda this paper proposed a novel approach to handle SQL injection using framework. This paper has used two level approaches to detect injected parameters [1]. First level, check some symbols like double dashes present in the query. If symbols are present then technique replace it by suitable non injected symbols. Second level, array of SQL keywords are used and contrast array's element with original query. If SQL symbols are present in the input then it is replaced by blank space. This work has been implemented using .net codes.

A. S. Gadgikar *et al* SQL injection attack has become a major threat to web applications, which gives unauthorized access [3]. This paper has used negative tainting approach with linked list structure. This approach is implemented between application program and database server. All the symptoms of SQL injection attack are stored in database. The future goal is to improve the efficiency by reducing false positives. Multithreading can be used to reduce the time requirements. In this paper SQL database is used for testing.

W. G. J. Halfond *et al* An SQLIA occurs when an attacker changes the developers SQL command by inserting new SQL keywords or operators [4]. In this paper their approach works by identifying trusted strings and allowing only those trusted strings to be used to create sensitive part of the SQL query strings. WASP (web application SQL injection prevented) tool implements this technique. It stops all the attacks without generating false positives. In the Future work- the proposed work can be used for binary programs and further improve the efficiency of technique to reduce the amount of information required.

A.John *et al* this paper surveyed existing techniques against SQL injection and analyzed their pros and cons and proposed a new and efficient solution to prevent attacks on login phase [5]. This paper consists of the preeminent features of both parse tree validation technique and code conversion method. The future work will be preventing SQLI attacks that are being performed by any other mean such as cookies or through server variables.

METHODOLOGY

Proposed system works in two phases:

Phase I

In this phase the proposed system does compile time checking. It involves the syntax evaluation of the input query.

Syntax Evaluation: When query entered as an input, the system performs syntax aware evaluation of the query string before it is sent to the database. The technique iterates through the query whether all the tokens identified as keywords or operators were constructed using only trusted data. Syntax analysis of the query is done to the various types of attack syntax. For example tautology attack, piggybacked attack and union attack. In all these type of attacks an attacker attack the system by inserting illegal inputs in the original query. But the system by evaluating syntax analysis declared it as an unauthorized access and declared it a type of attack. Thus various types of attack stopped at the compile time in the proposed system on the basis of syntax evaluation. If all of the keyword and operator are trusted then this phase concludes that query is safe and enters into the next phase.

Phase II

After the phase I the system enters into phase 11. This part has been divided into two parts. One part includes the training phase and second part includes SQL injection detection. These parts have been defined in the following way:

Training phase

This phase include the training phase of the proposed system. System has been trained by storing all the possible attacks in the database table which is also called as primary list. This algorithm works in the following way:

Attack storage: All possible SQL injection attacks are collected and they are stored. Then these attacks are divided into tokens and tokens are changed into integer numbers. Following method has been used for the conversion of tokens into integers:

Tokenization: It is the process of splitting the query into tokens.
*For example: select * from table_name.* In tokenization when this query splits into tokens then the system selects select, *, from as a tokens.

Number Generation: After tokenization system generates the number for tokens. Formula used for this is to multiply each ASCII decimal value of a literal by its position number occurring in tokens and then sum all these values.
For example: consider keyword select, corresponding ASCII value of each literal is s = 115, e = 101, l = 108, c = 99, t = 116. After multiplying ASCII values of literals with their position sum is 2236.

Correspondingly attack indicator is converted into tokens first.

Primary List Formation: This generated number has been stored in the database which used as a primary list for the proposed system.

This saved number helps to compare with the each input query. After the training phase SQL injection detection phase starts.

SQL Injection Detection
This phase check the SQL injection attack at run time and it match the integer number of input query string with the saved number. If the number matches then the input query has been blocked otherwise query processes normally. This algorithm works in following way:
Step I: Tokenize the input query string.
Step II: Scan each token for identifiers and operator symbols.
Step III: Convert each token into integer value.
Step IV: Save this number into the database as a secondary list.
Step V: Compare this integer value with each value of primary list.
Step VI: If the number match then SQL injection will detect otherwise, process query normally.
Step VII: Continue with next query.

As all the systems does work according to a particular flow. Similarly the proposed system does. Firstly, the SQL query goes as an input to the system. First phase check the input query at the compile time. The operations performed at compile time generally include syntax examination. If SQL injection finds in this phase then the system block the particular query, otherwise it goes in the second phase of the design.

Second phase of the system check input query at run time. Run time is the time in which the program is running and generates output. If SQL injection finds in this query then the system block the query and send the alert to database administration. If SQL injection does not find in the input query then query goes to the database engine for normal processing and ends this phase. Figure 4.2 shows the step by step processing of the proposed system.

RESULTS AND COCLUSION

Results
The existing technique and proposed technique have been implemented on web applications for prevention from various SQL injection attacks. The web applications is online shopping, online bidding etc which has been developed using various languages like ASP.NET, JAVA etc and database at back end. The difference between the existing techniques and proposed system is shown on the basis of number of SQL injection attacks handle by the system. Proposed system represents a technique to prevent the SQLIA with the help of Hybrid approach. The result is shown for taking various types of input queries and output is denoted as number of queries handled by the system to check SQL injection attack in the input query. The proposed system can handle various types of attack queries. Some of them are:
Tautology based queries
Union based queries
Logically incorrect queries
Alternate Encoding
Piggybacked Attack
Stored Procedure
Compile time checking
20 queries are taken in each sample so that the total number of queries becomes 140. The queries are also tested for the existing techniques but the proposed system more accuracy than the existing techniques. The system shows an overall accuracy of 94% that means system when checked for 100 SQLIA and system is successful to prevent the 94 attacks in the input queries. Table 5.1 shows the statistics for the proposed system.

ISSN: 2277-9655
(I2OR), Publication Impact Factor: 3.785

Table : Statistics for proposed system

Parameter	Value
No. of Queries Tested	140
Types of Attack Handled	8
Overall Accuracy	94.44%

The proposed system does not tell the accuracy on the basis of 20 attacks for each type of attack only. This system has been checked with 5 attacks for each type of attack then 10 attacks and finally, 20 attacks for each type of attack. Figure 5.1 shows the comparison of existing and proposed system on the basis of 5 attacks. This graph shows the comparison of existing system and proposed system for tautology attack, union attack, logically incorrect query attack, sub query attack and other type of attacks. The result comes when 5 attacks fired for each type of attack and the result for existing system.

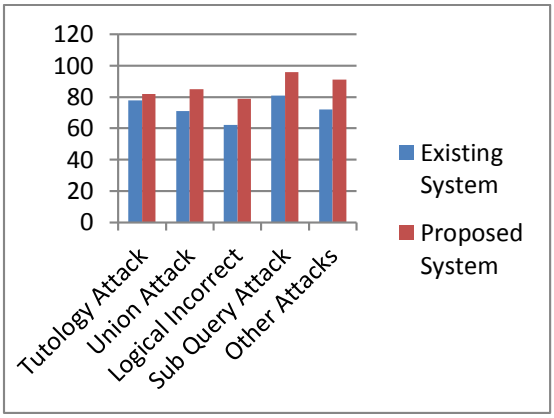

Figure : Comparison on the basis of 5 attacks

This graph provides comparison only for 5 attacks of each type of attack. Now the proposed system comprises of comparison for 10 attacks of each type of attack. Figure 5.2 shows comparison for 10 attacks of each type of attack.

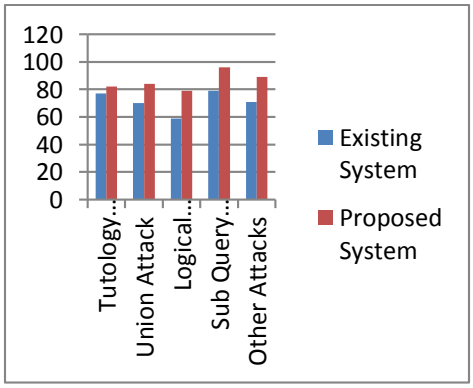

Figure : comparisons of existing system and proposed system for 10 attacks

But the comparison on the basis of 5 and 10 attacks for each type of attacks does not provide the more difference between existing and proposed system. Thus the proposed system provides the result on the basis of 20 attacks for each type of attack. For 20 attacks of each type of attack gives 94.44% accuracy. Figure 5.3 provides comparison of existing and proposed system on the basis of 20 attacks for each type of attack.

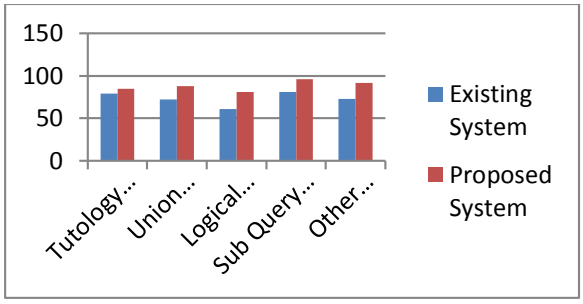

Figure 5.3 Comparison of existing and proposed system

The proposed system is better than existing techniques in three ways. This system done compile time checking, multithreading concept and stored procedure but the existing techniques are not compatible with these. Stored procedure which is basically inbuilt functions. Table 5.2 shows the accuracy comparison with existing techniques on the basis of various types of attacks handle and multithreading technology.

Table 5.2 comparison with the existing techniques

Schemes	Tautology	Logically Incorrect Queries	Union Queries	Stored Procedure	Piggy Backed Queries	Inference	Alternate Encoding	Compile Time Attack Checking	Multi-threading Technology Used	Overall Attack Immunity
AMENSIA	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	66.66%
SQLrand	Yes	No	Yes	No	Yes	Yes	No	No	No	44.44%
CANDID	Yes	No	No	No	No	No	No	No	No	11.11%
SQLguard	Yes	No	No	No	No	No	No	No	No	11.11%
SQLIPA	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	66.66%
Negative Tainting	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	77.78%
Proposed System	Yes	Yes	Yes	Parti-ally	Yes	Yes	Yes	Yes	Yes	94.44%

CONCLUSION AND FUTURE WORK

Conclusion

SQLIA have been described as one of the most significant threats to web application protection. In this dissertation, a new technique has presented to prevent SQLI in web applications. This approach is based on negative tainting and positive tainting. Unlike previous approaches our technique identifies the input attacks at the compile time and run time. In the existing techniques testing has been done on the limited databases. Proposed system has been checked on the three databases MS ACCESS, MY SQL and SQL. This technique uses the concept of multithreading to improve the performance of the system. Evaluation of this approach has been done by putting various SQLI attack queries. The proposed system shows 94.44% accuracy.

Future Work

In future, there are many ways to improve the performance further. In the dissertation, the approach is used for ASP.NET web applications having used three databases (MS ACCESS, MY SQL, SQL) at the backend but in the future it can be used for the rest of web applications (build in java, PHP etc.). Presently the system stops the stored procedure attacks partially and system can be integrated with more databases like ORACLE, db2 etc.

REFRENCES

[1] K. S. Chavda, "International Journal of Advance Engineering and Research," Sci. J. Impact Factor, vol. 1, no. 12, pp. 173–179, 2014.
[2] S. Roy, A. K. Singh, and A. S. Sairam, "Detecting and Defeating SQL Injection Attacks," Int. J. Inf. Electron. Eng., vol. 1, no. 1, pp. 38–46, 2011..
[3] A. S. Gadgikar, "Preventing SQL injection attacks using negative tainting approach," in IEEE International Conference on Computational Intelligence and Computing Research, 2013, pp. 1–5.
[4] W. G. J. Halfond, A. Orso, and P. Manolios, "Using positive tainting and syntax-aware evaluation to counter SQL injection attacks," in Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering - SIGSOFT '06/FSE-14, 2006, pp. 175–185.
[5] A. John, A. Agarwal, and M. Bhardwaj, "An adaptive algorithm to prevent SQL injection," An Am. J. Netw. Commun., vol. 4, pp. 12–15, 2015.
[6] www.google.co.in
[7] B. Shehu and A. Xhuvani, "A Literature Review and Comparative Analyses on SQL Injection : Vulnerabilities , Attacks and their Prevention and Detection Techniques," IJCSI Int. J. Comput. Sci., vol. 11, no. 4, pp. 28–37, 2014.
[8] S. Bangre and A. Jaiswal, "SQL Injection Detection and Prevention Using Input Filter Technique," Int. J. Recent Technol. Eng., vol. 1, no. 2, pp. 145–150, 2012.
[9] A. Sadeghian, M. Zamani, and A. A. Manaf, "A Taxonomy of SQL Injection Detection and Prevention Techniques," in 2013 International Conference on Informatics and Creative Multimedia, 2013, pp. 53–56.
[10] E. Bertino, A. Kamra, and J. P. Early, "Profiling database applications to detect SQL injection attacks," in Conference Proceedings of the IEEE International Performance, Computing, and Communications Conference, 2007, pp. 449–458.
[11] D. Kar and P. Suvasini, "Prevention of SQL Injection Attack Using Query Transformation and Hashing," in Proceedings of the 2013 3rd IEEE International Advance Computing Conference, IACC 2013, 2013, pp. 1317–1323.
[12] P. Kumar and R. Pateriya, "A Survey on SQL injection attacks, detection and prevention techniques," in Computing Communication & Network Technologies, 2012, no. July, pp. 1–5.
[13] R. Dharam and S. G. Shiva, "Runtime Monitors for Tautology based SQL Injection Attacks," IEEE Int. J. Cyber-Security Digit. Forensics, vol. 53, no. 6, pp. 253–258, 2012.
[14] X. Fu, X. Lu, and B. Peltsverger, "A static analysis framework for detecting SQL injection vulnerabilities," in 31st Annual International Computer Software and Application Conference, 2007, no. Compsac, pp. 87–96.