Volume 5, Issue 5, May 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Prevention of SQL Injection Attack Using Hybrid Approach

Navjot Verma, Amardeep Kaur
Punjabi university regional centre for IT & Mgmt
Mohali, Punjab, India

Abstract— 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. All these users have the utmost priority for their privacy and security and these are the most vulnerable while being online. SQL injection (SQLI) is the major susceptible attack in today's era of web application which attacks the database to gain unauthorized and illicit access. This paper presents a new method for prevention of SQL Injection attack. . This hybrid approach is the combination of positive tainting and negative tainting. Positive tainting works for syntax aware evaluation for trusted strings and negative tainting works at run time for untrusted strings. Syntax aware evaluations done at compile time for proposed system. Moreover, multithreading technique uses in the proposed technique and it also works partially on stored procedure.The results of the technique are compared with the result of existing technique and the validation of the proposed system is evaluated on the basis of various types of attacks it can handle.

Keywords— SQL Injection, Positive tainting, Negative tainting, Multithreading, Stored Procedure.

I. INTRODUCTION

Web applications are more and more used in recent years to provide online services such as banking, shopping, social networking, etc. [1]. 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, structuring 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 obstructs 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 severity 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. This paper has focused on prevention of SQLIA using hybrid approach. This system has been implemented with ASP.NET and databases as a backend. Hybrid approach is the combination of two approaches named as positive tainting and negative tainting. Positive tainting works on the trusted strings and negative tainting works on untrusted strings. This proposed system works in two phases. One phase is compile time which comes under positive tainting and second phase is run time that comes under negative tainting.

II. WEB APPLICATION VULNERABILITY TYPES

Vulnerabilities are hole or weakness in application, which can be plan fault or execution bug that allows an attacker to cause harm to stakeholders of an application. Table 1 (Types of vulnerability) present the most common security vulnerabilities found in web programming languages [7].

Table 1 Vulnerability types

Vulnerability Types	Description
Type I	No clear distinction between data types received as input in the programming language used for the web application development
Type II	Delay of operation analysis till the runtime phase where the current variables are considered rather than the source code expressions.

Type III	The validation of the user input is not definite. Attacker taking advantages of insufficient input validation can utilize mean code to conduct attacks.
Type IV	Puny concern of type specification in the design. A number can be used as a string or vice versa.

In SQL injection attack, an attacker uses the SQL statements and these statements are different from originally projected. SQL injection attacks are generally categorized into following types:

III. SQL INJECTION TYPES

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

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: 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 [10].

IV. METHODOLOGY

After a careful analysis of the different models, it is noted that since most of the approaches are focused either on positive tainting which checks the syntax of the query structure with predefined SQL keywords and user inputs to find vulnerabilities or negative tainting which checks all known attacks for every input query, in addition an option to insert any new type of SQLIA if detected later is also provided the query structure with predefined SQL keywords and user inputs to find vulnerabilities. Moreover, all the existing techniques either work at compile time or run time. Hence we decided to go for hybrid approach for comparison. Hybrid approach consists of positive tainting and negative tainting both. For the proposed system positive tainting comes under compile time checking and negative tainting comes under run time checking.

Algorithm consists of two phases to check and prevent the SQL Injection attack in the input queries. These phases are:

Phase I

First phase of the system works at the compile time and done syntax evaluation. At compile time system checks for three types of SQL injection attacks on the basis of syntax. These attacks are tautology attack, union attack and piggybacked attack. If any attack SQLI attack found in the input query, system block the query at the compile time and query neither compiled nor executed. If input query passes the first phase then it enters into the second phase.

Phase II

This phase check the SQL injection attack at the run time. This phase is divided into two parts described below:

Training Phase

In training phase all the possible attacks are stored in the database by database administrator. These attacks are too divided into tokens and then these tokens are converted into integer numbers using their ASCII values along with position. By this method an attack can be represented as an integer number which can be used to compare with the input

query to find SQLI. Formula used for this conversion is to multiply each ASCII decimal value of a literal by its position number occurring in token and then sum all these values. For e.g. *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.* This integer number has been stored in the database as a primary list.

SQLI detection

In this part SQL injection attack is checked at the run time. If attack in query is found then query is blocked by the system otherwise query processed normally. Steps followed for detection are:

Step I: Tokenize the input query string.

for e.g. consider input query q = select * from employee. Sequence of tokens generated for query will be: select, *, from, employee

Step II: Scan each token for identifiers and operator symbols.

Step III: Convert each token into integer value. Tokens will be converted into integer values as explained in training phase. In this case token to integer values will be: 1564 for select, 42 for *, 1099 for from, 3883 for employee

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.

Figure 1 shows the proposed system architecture.

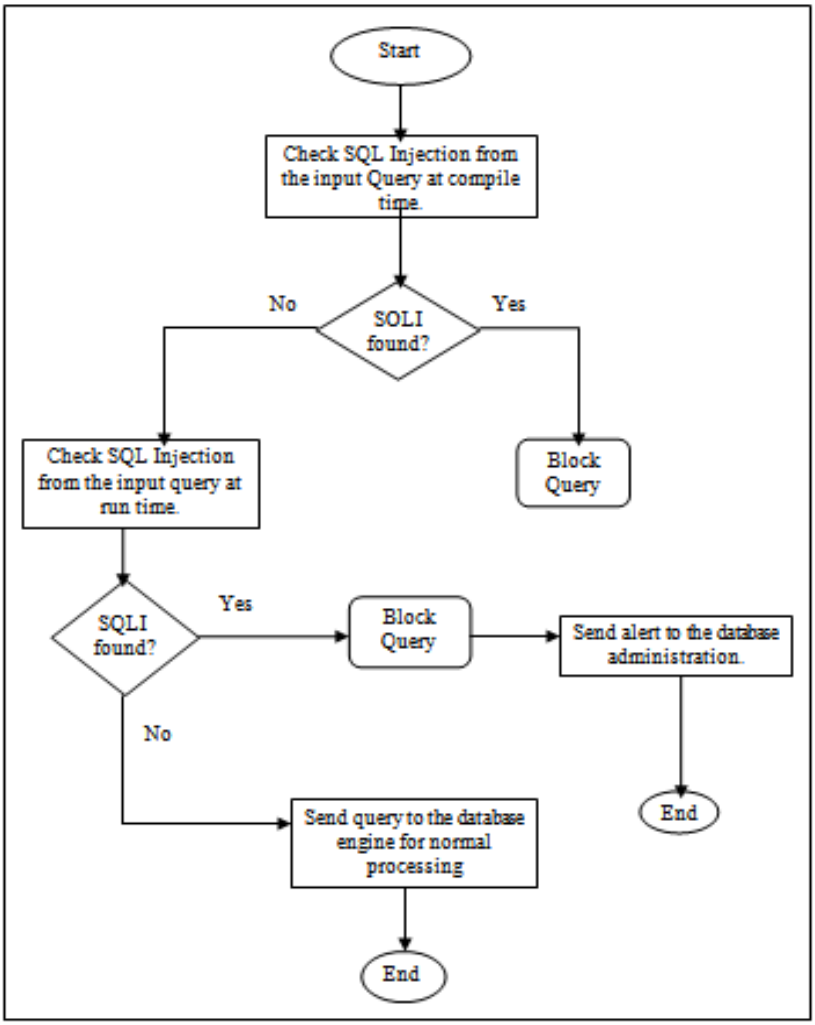

Figure 1 Proposed system architecture

V. RESULTS AND DISCUSSION

The proposed methodology was implemented with ASP.NET visual studio tool 2010. The tests are performed on 170 queries. First 170 queries have been tested with the existing system negative tainting and same queries have been tested for the proposed system. The proposed system is able to prevent from seven types of SQL Injection attacks. Average twenty queries are taken in each sample so that the total number of queries becomes 170 and 50 queries has been taken for run time attacks. The queries are also tested for the existing techniques but the proposed system provides more accuracy than the existing technique. The system shows an overall accuracy of 88.28% that means system when checked for 100 SQLIA and system is successful to prevent the 88 attacks in the input queries. Table 2 shows comparison of existing and proposed system.

Table 2 Comparison table for proposed and existing technique

Sr.No	Types of attack	No. Of attacks fired	Proposed system	Existing system
1.	Tautology Attack	20	20	12
2.	Logically-incorrect Attack	20	18	10
3.	Piggy-backed Attack	20	19	15
4.	Inference Attack	20	17	11
5.	Run Time Attack	50	44	42
6.	Union attack	20	19	16
7.	Stored Procedure Attack	20	14	11

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 and al last for 20 attacks of each type of attack. 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 3 shows the accuracy comparison with existing techniques on the basis of various types of attacks handle and multithreading technology.

Table 3 Various schemes and SQL injection attacks

Scheme	Tautology	Logically Incorrect Queries	Union Queries	Stored Procedure	Piggy Backed Queries	Infer-ence	Alternate Encoding	Compile Time Attack Checking	Multi-threading Technology
AMENSIA	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No
SQLrand	Yes	No	Yes	No	Yes	Yes	No	No	No
CANDID	Yes	No	No	No	No	No	No	No	No
SQLguard	Yes	No	No	No	No	No	No	No	No
SQLIPA	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No
Negative Tainting	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No
Proposed System	Yes	Yes	Yes	Partially	Yes	Yes	Yes	Yes	Yes

Next the comparison has been shown in the graph with existing system. Figure 2 explains the number of attacks fired. Attacks detected by proposed system and the existing one.

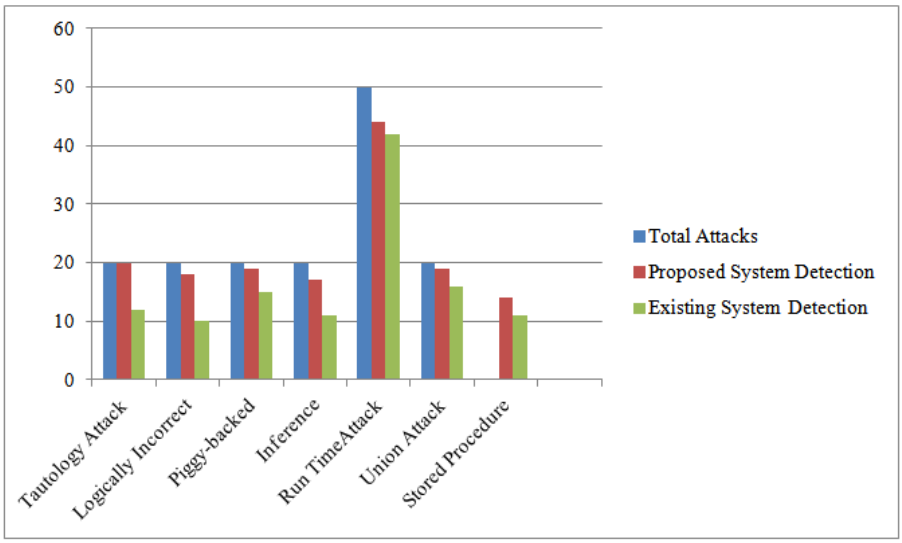

Figure 2 Comparison graph for existing and proposed technique

VI. CONCLUSIONS

This research was conducted to prevent various types of web application attacks. This paper presented general framework for prevention of SQL Injection attack using hybrid approach. SQLIA have been described as one of the most significant threats to web application protection. In this paper, 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, MySQL and SQL Server. 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 result shows that the system identifies the seven types of attacks with an accuracy of 88.28%.

In future work, Preventing SQL injection using Hybrid approach is designed to work with all databases. It is tested with three databases. It could be implemented and tested to work with other databases such as Oracle, Sybase, FoxPro etc.

REFERENCES

[1] K. S. Chavda, "Prevention of SQL Injections From Web Applications," *Int. J. Adv. Eng. Res.*, 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 Proceedings of 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 conference 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] [Online]. Available: blogs.embracadero.com/pawelglowacki/11. [Accessed: 15-Apr-2015].

[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 Proceedings of IEEE 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 Proceedings of the IEEE International Conference on Performance, Computing, and Communications*, 2007, pp. 449–458.

[11] D. Kar and P. Suvasini, "Prevention of SQL Injection Attack Using Query Transformation and Hashing," *in Proceedings of the IEEE 3rd International Conference Advance Computing, IACC*, 2013, pp. 1317–1323.

[12] P. Kumar and R. Pateriya, "A Survey on SQL injection attacks, detection and prevention techniques," *in Proceedings of IEEE 3rd International Conference on Computing Communication & Network Technologies*, July 2012, pp. 1–5.

[13] R. Dharam and S. G. Shiva, "Runtime Monitors for Tautology based SQL Injection Attacks," *Int. J. Cyber-Security Digit. Forensics IEEE*, 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 Proceedings of 31st Annual International Conference on Computer Software and Application*, 2007, pp. 87–96.

[15] K.-X. Zhang, C.-J. Lin, S.-J. Chen, Y. Hwang, H.-L. Huang, and F.-H. Hsu, "TransSQL: A Translation and Validation-Based Solution for SQL-injection Attacks," *in Proceedings of First International Conference on Robot, Vision and Signal Processing*, 2011, pp. 248–251.

[16] K. Kemalis and T. Tzouramanis, "SQL-IDS : A Specification-based Approach for SQL-Injection Detection," *in Proceedings of ACM Conference on Applied Computing*, March 2008, pp. 2153–2158.

[17] P. Bisht, "CANDID : Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks," *ACM Int. J. Comput. Sci.*, vol. V, no. 2, pp. 1–38, 2010.

[18] G. T. Buehrer, B. W. Weide, and P. A. G. Sivilotti, "Using Parse Tree Validation to Prevent SQL Injection Attacks," *in Proceedings of 5th ACM International Conference on Software Engineering and Middleware*, September 2005,, pp. 106–113.

[19] S. W. Boyd and A. D. Keromytis, "SQLrand : Preventing SQL Injection Attacks," *IEEE Appl. Cryptogr. Netw. Secur.*, vol. 2, pp. 292–302, 2004.

[20] W. G. J. Halfond and A. Orso, "Preventing SQL injection attacks using AMNESIA," *in Proceeding of the 28th ACM International Conference on Software Engineering - ICSE*, 2006, p. 795-798.

