WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

SQL injection detection and prevention

Monisha.V*

**First affiliation Chennai,TamilNadu 6000*

A B S T R A C T

Web sites are dynamic, static, and most of the time a mixture of both. websites need to be protected in their databases in order to make sure they are kept safe. An SQL injection attack is a code injection attack that attacks connected applications which provide database services. These applications take user inputs and use them to make an SQL query at run time. An SQL injection attack, allows an attacker to perform unauthorized operations on the database by inserting a malicious SQL query as input. By performing an SQL injection attack, an attacker or a hacker can get unauthorized access to the database content and can retrieve and modify sensitive and confidential data from the database. It is going to put the confidentiality and security of websites at risk that completely relies on databases. This report presents a "code reengineering" that completely protects the applications that are written in PHP from SQL injection attacks. It uses a creative approach that combines static additionally as dynamic analysis. In this paper, an automatic technique for moving out SQL injection vulnerabilities from Java code by converting plain text inputs received from users into prepared statements is discussed.

Keywords:SQL injection attack, code reengineering, PHP

1. Introduction

In recent years, the internet has become widespread and that led to the quick evolution in information technologies. Websites today, try to keep their user's information safe and confidential, and after years of successfully doing business online, these websites have now emerged as experts in information security. A common break-in technique is to do to access confidential information from a database by first generating a question that may cause the database parser to break down, after that by applying this question to the required database. Such an approach to gaining access to non-public information is named SQL injection. The goal of this project is to create an automatic fix generation method to forestall SQL injection vulnerability from plain text SQL statements. In an automated method approach, the information about vulnerabilities that are known, specifically the SQL statement will be collected by a server which then generates a patch and applies the patch. The process can be completed by someone with no security expertise and secure legacy code, which can allow developers to repair the SQL injection vulnerability.

1.1. Outlook for SQL CHECK:

Web-based applications possess SQL injection vulnerabilities since they do not clean the inputs they use to build structured output. In this approach the user's track via the program, the substrings collect input from the user and clean those substrings grammatically. The main target of this program is to stop up the queries in which the substrings of the input swap the syntactic form of the query that is left. to observe the user's input, they apply meta-data shown as '_' and '_,' to spot the ending and beginning of the input string of each user. To construct a parser for increased grammar and strive to parse each query, a parse generator is used. If the generator parses well and the syntactic constraints are met by the query, it is considered legal. Else, it does not meet the syntactic constraint and it is examined as an SQL injection attack.

* *Monisha. V.* Tel.:+91-7708625961.
E-mail address: monishavmuthu@gmail.com

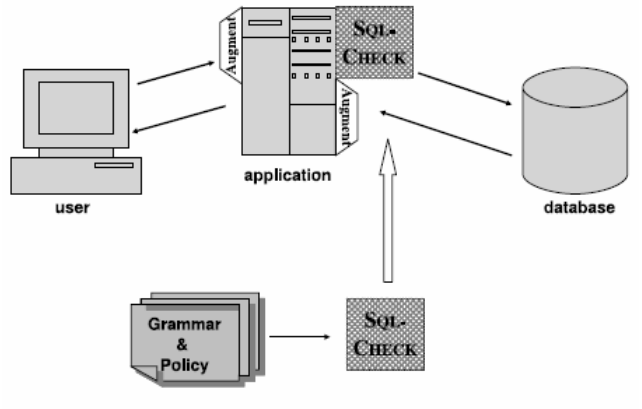


Figure 1: Architecture of SQL checker

2. SQL Statement framework

SQL injection vulnerability is the mixture of dynamic SQL statement collection and a vulnerability in input documentation. This input document drives the input to change the SQL query structure. Combinations like these are commonly seen in java.

2.1 SQL Statements

There are two types of SQL statements that are used to prevent SQL injection attacks.

2.2Prepared Statements

Some statements are pre-compiled with the SQL query. Such statements are called prepared statements. The SQL query is written by the programmer which is nothing but the statement that contains plain text representation. Prepared statements in SQL query permit you to put inputs into following queries by tying up the vulnerabilities. The key aim of the prepared statement is to enlarge security and efficiency. Prepared statements are constructed to implement identical statements repeatedly when compiling the statement. This feature is not present in plain text statements. The performance of the prepared function is identical to the plain text SQL statements. The only difference between them is that the prepared statements possess a more structured way than the plain text SQL statements. The structure of the pre-compiled query if influenced can stop using structure handling of the prepared statement, thus preventing SQL injection vulnerability.

2.3 Run time automated statement

The advantage of an automated statement is that it actively finds out if there are any vulnerabilities present in the state query at run time. This technique not only completely depends on the prepared statement, but it also authenticates the SQL code by placing limitations on the run time environment to stay away from the malicious SQL statements. The suggested solution in this method is to steer clear of an SQL injection attack, by examining the parse tree of the SQL statement, constructing the custom validation code, and packaging the vulnerable statement in the validation code. In run time automated statement parse trees are used in a dynamic way to create the comparison at the run time to check if two queries are practically the same.

3. SQL injection detection technique

An attacker need not mandatorily visit the web pages using a browser to whether SQL injection is possible on the site. Attackers create a web crawler to gather every URL present on each and every web page of the site. Web crawlers can be used to place illegal characters into the query string of a URL and check if there are any possible errors sent by the server. In case the server reports any error message as a result, it strongly indicates that the illegal character will move as a part of the query, and thus the site is exposed to SQL injection attack.

4. SQL injection prevention method

There are two methods used in the prevention of SQL injection attacks

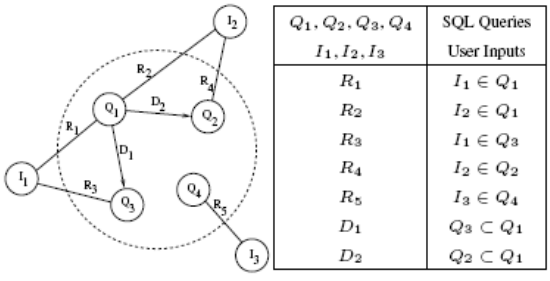
1. Static analysis

2. Run time analysis

In these techniques, stored procedures are used and it has a control flow graph that gives notification about what user inputs to the SQL statement that is built. Control flow graphs are beneficial to reduce the number of SQL statements to check the user input. In run time analysis we get details about stored statements from finite-state automation to limit the verification procedure and to specify the user's input is true or false.

4.1. Static analysis of stored procedure

In static analysis, the parser known as stored procedure parser is provided. It is used to pull out the "control flow graph" from the saved procedures. In the beginning, all the execution statement in the control flow graph is marked and then the backtracking procedure is used to check all statements that took part in the construction of the SQL statement in the control flow graph. In the SQL graph, Statements that rely on the user's input are protected and flags are put on it to keep track of their actions at run time. The statements made by the user's input that attempts to change the original design of the parser will be specified by the flag as a threatening statement and give the corresponding information. There are varieties of procedure statements available, and only those statements that receive users' input are vulnerable to an SQL injection attack.



Q_1, Q_2, Q_3, Q_4	SQL Queries
I_1, I_2, I_3	User Inputs
R_1	$I_1 \in Q_1$
R_2	$I_2 \in Q_1$
R_3	$I_1 \in Q_3$
R_4	$I_2 \in Q_2$
R_5	$I_3 \in Q_4$
D_1	$Q_3 \subset Q_1$
D_2	$Q_2 \subset Q_1$

Fig. 2- Control graph

4.2Benefits of static analysis

1. Representation of SQL graph is useful in minimizing the run time scanning costs of the program by putting a stop to the number of queries that aren't needed to implement within the stored procedure.

2. The query that doesn't draw input from the user isn't taken into count by the SQL control graph.

3. The queries that cover input from users to get the data of the database are added up towards SQL control graph representation.

4.3. Dynamic analysis

In dynamic analysis, the user input is classified by the SQL injection attack checker. In this technique, the "current session" identifier is used to recognize the input drawn from the user. To examine the legality of SQL statements collected from users, the SQL statement together with user inputs is estimated with related SQL statements of limited state automation. If the SQL queries created at a run time uses the user input and does not meet the semantics of the planned SQL queries, these SQL queries are placed as SQL injection attack. Thus, we may simply prevent composing malicious queries and only allow the legitimate queries to obtain databases.

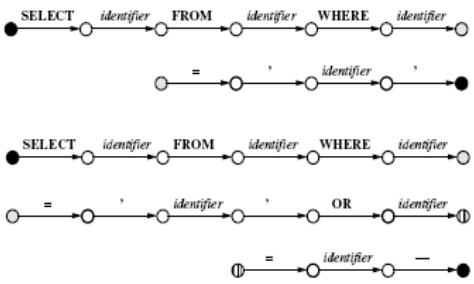


Fig. 3-SQL injection attack recognition

Conclusion

Almost all web applications make use of an intermediary layer to receive a request from the user and recover confidential information from the database. Most of the time they use a scripting language to construct an intermediate layer. Attackers frequently use SQL injection techniques to break the security of databases. In general hackers attempts to trick the middle layer technology by reshaping the SQL queries. Numerous technologies are used to steer clear of SQL injection attacks at the application level. This paper encloses dominant and powerful technologies for the prevention of SQL injection attacks. This paper, concludes that automated technique used for the prevention and detection of SQL injection attack in a stored procedure is generally used. For databases systems of small size, the graph control method is also a wise choice. In the forthcoming work, we can use more powerful technologies to achieve more accuracy in the approach of SQL injection.

REFERENCES

[1] C.Anley,Advanced SQL Injection in SQL Server Applications, 2002, , accessed January 21, 2007. [2] N. Audsley, I. Bate, S. Crook-Dawkins, Automatic code generation for airborne systems, in: IEEE Aerospace Conference, New York, NY, 2003, pp. 6_2863– 6_2873. [3] S. Barnum, G. McGraw, Knowledge for software security, Security and Privacy Magazine, IEEE 3 (2) (2005) 74–78. [4] M. Bordin, T. Vardanega, Real-time Java from an automated code generation perspective, in: International Workshop on Java Technologies for Real-Time and Embedded Systems, Vienna, Austria, 2007, pp. 63–72. [5] R.E. Bryant, S. Jha, T.W. Reps, S.A. Seshia, V. Ganapathy, Automatic discovery of API-level exploits, in: 27th International Conference on Software Engineering (ICSE'05), St. Louis, MO, 2005, pp. 312–321. [6] G. Buehrer, B.W. Weide, P.A.G. Sivilotti, using parse tree validation to prevent SQL injection attacks, in: 5th International Workshop on Software Engineering and Middleware, Lisbon, Portugal, 2005, pp. 106–113. [7] Y. Cheon, G.T. Leavens, A simple and practical approach to unit testing: the JML and JUnit way, in: 16th European Conference on Object-Oriented Programming, Spain, 2002, p. 29. [8] D. Florescu, C. Hillery, D. Kossmann, P. Lucas, F. Riccardi, T. Westmann, J. Carey, A. Sundararajan, The BEA streaming XQuery processor, The VLDB Journal 13 (3) (2004) 294–315. [9] W.G.J. Halfond, A. Orso, AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks, in: 20th IEEE/ACM International Conference on Automated Software Engineering, Long Beach, CA, USA, 2005, pp. 174–183. [10] W.G.J. Halfond, A. Orso, Combining static analysis and runtime monitoring to counter SQL-injection attacks, in: Third International Workshop on Dynamic Analysis, St. Louis, MO, 2005, pp. 1–7. [11] W.G.J. Halfond, A. Orso, P. Manolios, Using positive tainting and syntax-aware evaluation to counter SQL-injection attacks, in: 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Portland, Oregon, 2006, pp. 175–185. [12] W.G.J. Halfond, A. Orso, P. Manolios, WASP: protecting web applications using positive tainting and syntax-aware evaluation, IEEE Transactions on Software Engineering 34 (1) (2008) 65–81. [13] W.G.J. Halfond, J. Viegas, A. Orso, A classification of SQL-injection attacks and countermeasures, in: International Symposium on Secure Software Engineering Raleigh, NC, USA, 2006. [14] D. Hovemeyer, W. Pugh, finding bugs is easy, in: 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications, Vancouver, BC, Canada, 2004, pp. 92–106. [15] M. Howard, D. LeBlanc, Writing Secure Code, second ed., Microsoft Corporation, Redmond, 2003. [16] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, S.-Y. Kuo, Securing web application code by static analysis and runtime protection, in: 13th International Conference on World Wide Web, New York, NY, 2004, pp. 40–52. [17] G. Keizer, One-at-a-time Hacker Grabs 22,000 IDs from University of Missouri, first ed., Retrieved Issue 1, vol. 1, 2007, accessed June 30, 2008.