SQL Injection: Detection and Prevention Techniques

Pooja
Research Scholar, Department of Computer Science & Applications, Kurukshetra University, Kurukshetra
Email: poojachoppra@gmail.com
Monika
Assistant Professor, Department of Computer Science & Applications, Kurukshetra University, Kurukshetra
Email: monikaporiye@gmail.com

Abstract
As the use of internet is increasing rapidly, the attacks on web applications are also increasing as well. Nowadays SQL injection attack is a major issue of web applications. It allows unrestricted access to the database. The successful execution of SQL injection leads to a loss of integrity and confidentiality. In this paper, a review of different types of SQL injection attacks, their detection and prevention techniques are presented. This paper will help the researchers decide the technique of interest for further studies.

Keywords: Detection, Prevention, SQL injection attacks, Tautology Attack

1. Introduction

SQL injection attack is a type of security vulnerability that target database connected web applications. In this attack, the attacker inserts a malicious SQL query into the web application to manipulate data or even to gain access to the back-end databases. SQL injection is on the top in the list of web application vulnerability [1]. This vulnerability is mainly occurring due to weaknesses present in source codes. The other reasons of this vulnerability may be the weakness of the programming language or improper input validation. A successful SQL injection attack can update, alter or delete data stored in the back-end databases, read sensitive information from the database and perform administrative operation on the database such as shutdown the DBMS. For example-

*Original Query: SELECT *FROM Login WHERE User_id='ram' and password='123'*

*Injected Query: SELECT *FROM Login WHERE User_id='' OR 1=1; /*' and password='*/-'*

In this query 1=1 is evaluated always true. And rest part of the query is evaluated as a comment. The query is executed and the attacker can access the database. Thus, SQL injection attack harms the database security and because the SQL databases store sensitive information, so loss of confidentiality is a common problem with SQL injection vulnerabilities. When the weak SQL query is used to view user name and password, then the other user may access the data with no previous knowledge of the password. This vulnerability affects the authentication. The successful exploitation of SQL injection vulnerability may change the authorization information. To change or delete the sensitive information harms the integrity. To resolve these problems, an efficient solution is necessary.

This paper is organized in the following sections. Section 2 describes the SQL injection attack, section 3 describes the literature review of SQL injection detection and prevention techniques and section 4 concludes the paper.

2. SQL Injection Attack

There are various types of SQL injection attacks. The following discussed attacks violate the security of web applications.

2.1 Tautology Attack

This attack focuses to inject a SQL query into conditional statements to return all true values. It is done by simply making the "WHERE" clause always true for every query. Generally the tautology attack is used to be in bypassing authentication pages and in extracting data from the databases. For example:

*Original Query: SELECT *FROM User_info WHERE Username='Mohan' and password='133045'*

*Injected Query: SELECT *FROM User_info WHERE Username=''OR 1=1 and password='1456'*

Using this injected query the attacker can return all true values for username and password of all stored users in the database.

2.2 Logically Incorrect Query

The objective of the attacker is to gather all possible information about the structure and the types of the back-end databases in a web application. For this purpose, an attacker inputs a manipulated query to generate an error message from the database. Error message provides the information about table name, field name and sometimes the condition of failure. For example:

*SELECT *FROM User_unit1 WHERE Username='xyz' HAVING 1='1'; -- and password='15672'*

Error generated: "Column 'User_unit1, User_id' is invalid in the select list because it is not contained in either an aggregate function or in the GROUP BY CLAUSE". The error shows the table name, User_unit1 and one of the column names, namely User_id.

*SELECT *FROM User_unit2 WHERE Username='n' HAVING 1='1'; -- and password='15672'*

Error generated: "Column 'User_unit2, User_id' is invalid in the select list because it is not contained in either an aggregate function or in the GROUP BY CLAUSE". This error message displays the table with column name 'Userinfo.Username' [2].

Thus, the attacker can extract all the field names of the table. The two types of error message returns, logical and syntactical. The name of the fields is displayed by the logical errors which extracts the fields or table names while syntactical errors informs which parameters are vulnerable for an injection attack.

2.3 UNION Queries

Code injection and manipulation of information are the main objectives of these attacks. For this purpose a UNION operator is used for data extraction from different tables. Attacker joins the injected query by the UNION operator, so that they can get data about other tables from the application.

SELECT Name, Phone FROM Users WHERE id=$id

Id value can be injected by following query:

$id=UNION ALL SELECT CreditCardNumber, 1FROM CreditCardTable

SELECT Name, Phone FROM User WHERE id=1UNION ALL SELECT CreditCardNumber, 1 FROM CreditCardTable

This query will join the result of the original query with all the credit card users.

2.4 Piggy-Backed Queries

This attack injects the additional queries to the original query. The attacker uses the query delimiter, such as ';' to add the extra query along with the original query. For example:

SELECT accounts FROM Customers WHERE User_id= 'ram' and password= '123'; drop table Customers

If the database gives permission to run multiple queries in the same line, then the database accepts the both queries and executes them because of ";" character. The query before the ';' is the original query and after the ';' is the SQL injection attack. When the database will execute the second query, then customers tables will be dropped and thus loss of valuable data. They perform manipulation operations by using INSERT, UPDATE and DELETE clause.

2.5 Inference

In this attack, the two types of attack "*Blind injection*" and "*Timing attack*" is used, to change the behavior of a database or application.

2.5.1 Blind Injection

Sometime developers hide the error details which help the attacker to get the information about the database. In this situation, the attacker faces a generic page provided by the developer, instead of an error message. An attacker can still steal data by performing queries that have a Boolean result. For example:

SELECT accounts FROM Customers WHERE User_id='karan' and 1=0-- AND passwd= AND pin=0

SELECT accounts FROM Customers WHERE User_id='karan' and 1=1-- AND passwd= AND pin=0

If the application is secured both queries would be unsuccessful because of input validation. But if there is a weak input validation, then the attacker submits their first query and receives an error message, because of "1=0" which is not true. Then the second query is submitted. If there is no error message, because of "1=1" which is always true, then the attacker searches the field vulnerable to injection.

2.5.2 Timing attack

In this type of attack, the attacker observes the database delays in the database responses and gathers the information. This attack is similar to blind injection & attacker can measure the time that a page takes to load, to determine if the injected statement is true. An if-then statement is used for injecting queries. WAITFORE is a keyword, which causes the database to delay its response by a specified time. For example:

Declare@ varchar (7000) select @ = db_nameO if (ascii (substring@, 1, 1)) & (power (2, 0)) > 0 waitfore delay '0:0:5'

If the first bit of the first byte in the name of the current database is 1, then database will stop for five seconds. Then the injected code is able to generate a delay in response time when condition is true.

2.6 Stored Procedure

Stored procedure could be coded by programmer, therefore, stored procedure is as inject able as web application forms. Different database has their different set of stored procedures, so it is difficult to use stored procedures without knowing what database is used. The attacker has the ability to run the commands on the operating system of the server.

2.7 Alternate Encoding

In this, the attacker modifies their injection strings, to avoid the signature and filter based checks. To escape from the various detection systems just by modifying the expression of the injected SQL query. They use ASCII, BASE 64, HEX or Unicode as encoding techniques.

3. Literature Review

SQL injection attack exploits the security vulnerabilities in the web applications. Several approaches have been proposed to detect and prevent this vulnerability. Many researches have been carried out on detection and prevention techniques of SQL injection and the key points from these researches are mentioned in the following sections.

3.1 SQL Injection Detection

To detect SQL injection attack Query Tokenization [3] is used, that is implemented by the query parser method. The original query and the injected query are tokenized separately. All strings before a space, single quote or double dashes form a token. An array is created from these tokens where each token is an element of the array. The original and the injected query create two different arrays. To detect the SQL injection attack both array lengths are compared. If the length of arrays is equal, then no SQL injection and if they are different then there is SQL injection.

An automatic approach is proposed [4] in which to test the web service a representative workload is used and a large set of SQL/Xpath injection attack are applied to uncover vulnerabilities. The structure of SQL/Xpath commands issued in the presence of attack is compared to the previously learned commands to detect the vulnerabilities. CIVS-WS tool provides the identification of vulnerable parameter and of vulnerable lines in the code.

[5] Mutation based SQL injection vulnerability checking (MUSIC) is used to test the SQL injection vulnerability using Mutation based testing. It injects syntax fault to see if any misshapen exists. It can determine whether the statement contains the misshapen by comparing the output. The generated mutant can be killed only with the test cases containing SQL injection attacks and nothing less.

A grammar based algorithm models the string values as CFGs (Context Free Grammar) and string operations as language transducers following minimization [6]. The input string is highlighted in this technique and labels are assigned to them, then it works on input string accordingly. It assigns the direct label to the strings that come directly from the user side such as GET requests. It assigns indirect labels to strings that are coming from the database side. The labeled strings are concluded to find the contexts and then the security of each string in aspect of syntax is checked by regular language & context free language.

[7] Both static and dynamic analyses are combined to detect the SQL injection attack, the static SQL queries

are compared to the dynamically generated queries, after removing the attribute value. Through experiments the effectiveness of this approach has been tested and validated in web applications.

[8] Static application code analysis and the runtime validation are combined. In static analysis phase, program analysis technique is used to represent the SQL queries as Finite State Automata and show them as a SQL graph. In runtime validation phase, the dynamically generated SQL queries are checked with the static data structure for compliance and labels them safe/unsafe. In this technique code modification is not required and SQL graph and SQL query validation are used in parallel to optimize the runtime analysis.

3.2 SQL Injection Prevention

In Negative tainting approach two modules: prevention module and attack database are used [9]. The Prevention module works as different layer. When the query sent to the database server, first it gets filtered by this layer. This module takes the query from the application layer and analyzes it. If it found SQL injection, it blocks the query and generates alarm message to the application program. If there is no SQL injection, query is forwarded to the database server. In attack database module symptoms of all known SQL injection attack is stored in the linked list structure. Symptoms are converted into token, token are converted into integer values and then a primary list is formed from these integer values. Similarly a secondary list is formed from the incoming query. The secondary list is compared with the primary list. If any match is found, it is a SQL injection attack. This technique is able to stop all known attacks except stored procedure and character encoding. In future to reduce the time required for pattern matching, multithreading can be used.

Randomized SQL queries [10] are used to check if there is any malicious statement or not and then terminate them. For this purpose, a proxy server is used between web server and database server to decode the random SQL query and then forward the decoded query with the standard set of keywords to the database for computation. The keyword without randomization concludes SQL injection attack. De-randomization element and the communication protocol are the two primary components between web server and database server. In this the attacker cannot do the SQL injection attack without knowing the random key. The best point of this technique is that it does not affect the performance.

[11] The static analysis and runtime monitoring are combined to propose a prevention technique. A model based approach is used to detect the malicious query before execution. A model of legitimate queries is built in its static part and it inspects dynamically generated queries in the dynamic part with the help of runtime monitoring and then compared them with the statically-built model. If the SQL statement meets with the requirement, then there is no SQL injection attack but if SQL statement does not meet with the requirement, there is a SQL injection attack. Recognizing the hotspot, Creating SQL query models, Instrument application and Runtime monitoring is the main tasks of this approach. The result of this technique shows that it is able to stop all the attempted attacks. The problem with this technique is that it cannot support segmented queries.

Preventing SQL Injection Attack in Web Application (PSIAW) technique is presented for preventing authentication against SQL injection [12]. Normally, a login table has two columns username and password. In this two additional columns are used for the hash value of username and password. When the user logins to the database then hash value of username and password is calculated. If they are equal, the user is granted to access the data. SQL Query component is the main component of PSIAW where the hash value of username and password is calculated. The drawback of this technique is that other SQL injection attack except authentication cannot be prevented by this technique.

Query tokenization and adaptive method are combined to develop a multi-level prevention technique [13]. Static analysis of application code is used in the form of an ordered sequence of tokens of malicious query. To capture all malicious SQL query, this database is authenticated against the differences in incoming SQL query structure at runtime before sending the query to the database server for execution. This technique can be implemented on multiple platforms that using asp.net programming language. The developed technique helps to secure web application from SQL injection based on the input validation.

The input from the user is extracted from the generated query by the web application and then this data is authenticated from the syntactic perspective of the generated query in SQL proxy based blocker technique [14]. A genetic algorithm is used for this purpose. In this approach the source code of the application is not required and no need to learn the authentication process.

[15] Static and runtime analysis approaches are combined to develop a solution to detect and prevent the SQL injection attack. Tracking methods are used to trace and monitor the execution of all received queries by runtime analysis approach. The developer creates a prepared set of expected changes and result of affected objects is compared to this set. This comparison decides the existence of SQL injection attack. Static approach performs a string comparison between the received and previous expected SQL queries. The result shows that it can detect and prevent all types of SQL injection attacks. In future, this technique needs to be enhanced by decreasing the time delay.

4. Conclusion

SQL injection attack is a very serious problem of web applications. Finding the efficient solution of this problem is essential. Researchers have developed many techniques to detect and prevent this vulnerability. There is no appropriate solution that can prevent all types of SQL injection attacks. This paper gives a review of different types of SQL injection attacks on the database. The SQL injection detection and prevention technique are summarized and their strength and weakness are also discussed.

References

[1] OWASP, "Top 10 2013-Top 10," 2013. [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-Top_10.

[2] J. Fonseca, M. Vieira and H. Madeira, "Vulnerability & attack injection for web applications," in *International Conference on Dependable Systems & Networks*, Lisbon, 2009.

[3] L. Ntagwabira and S. L. Kang, "Use of Query Tokenization to detect and prevent SQL Injection Attacks," in *International Conference on Computer Science and Information Technology* , Chengdu, 2010.

[4] N. Antunes, N. Laranjeiro, M. Vieira and H. Madeira, "Effective Detection of SQL/XPath Injection Vulnerabilities in Web Services," in *International Conference on Services Computing*, Bangalore, 2009.

[5] H. Shahriar and M. Zulkernine, "MUSIC: Mutation-based SQL Injection Vulnerability Checking," in *The Eighth International Conference on Quality Software*, Oxford, 2008.

[6] G. Wassermann and Z. Su, "Sound and Precise Analysis ofWeb Applications," in *ACM SIGPLAN Conference on Programming Language Design and Implementation*, California, 2007.

[7] J. G. Kim, "Injection Attack Detection Using the Removal of SQL Query Attribute Values," in *International Conference on Information Science and Applications* , Jeju Island, 2011.

[8] M. Muthuprasanna, K. Wei and S. Kothari, "Eliminating SQL Injection Attacks - A Transparent Defense Mechanism," in *International Symposium on Web Site Evolution*, Philadelphia, 2006.

[9] A. S. Gadgikar, "Preventing SQL Injection Attacks Using Negative Tainting Approach," in *International Conference on Computational Intelligence and Computing Research*, Enathi, 2013.

[10] S. W. Boyd and A. D. Keromytis, "SQLrand: Preventing SQL Injection," in *Applied Cryptography and Network Security*, Berlin Heidelberg, Springer, 2004, pp. 292-302.

[11] W. G. J. Halfond and A. Orso, "AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks," in *international Conference on Automated software engineering*, New York, 2005.

[12] M. Gandhi and J. Baria, "SQL INJECTION Attacks in Web Application," *International Journal of Soft Computing and Engineering (IJSCE)*, 2013.

[13] N. A. A. Othman, F. H. M. Ali and M. B. M. Noh, "Secured web application using combination of Query Tokenization and

Adaptive Method in preventing SQL Injection Attacks," in *International Conference on Computer, Communications and Control Technology* , Langkawi, 2014.

[14] A. Liu, Y. Yuan, D. Wijesekera and A. Stavrou, "SQLProb: a proxy-based architecture towards preventing SQL injection attacks," in *ACM symposium on Applied Computing*, New York, 2009.

[15] J. O. Atoum and A. J. Qaralleh, "A Hybrid Technique for SQL Injection Attacks Detection and Prevention," *International Journal of Database Management Systems (IJDMS)*, 2014.