www.ioirp.com
International Journal of Innovative Research in Computer Science and Engineering (IJIRCSE)
ISSN: 2394-6364, Volume – 2, Issue – 1. January 2016

Defensive Approach to Prevent SQL Injection Attacks

S Gowthami ,
PG Scholar,
Department of Information Technology,
Kongu Engineering College, Perundurai, Erode.

K R Prasanna Kumar,
Assistant Professor,
Department of Information Technology,
Kongu Engineering College, Perundurai, Erode.

A P Ponselvakumar,
Assistant Professor,
Department of Information Technology,
Kongu Engineering College, Perundurai, Erode.

***Abstract*—Web applications that are deployed by organizations and firms for e-business tasks should have high reliability, efficiency and confidentiality. Those applications are written in script languages like PHP, JSP, ASP, etc. These database driven web applications establish connection to database in order to retrieve data, store data and put them in the web. SQL injection is the one of the most common attack in the web application. In this paper, SQL injection attack and its variants are discussed. A brief discuss on the existing approaches for detecting SQL injection attacks are presented. An experimental analysis shows that the proposed system is best for protecting web applications from SQL injection attacks.**

***Keywords*: security, parse tree, malicious, tokenization, injection attacks, script**

I. INTRODUCTION

Web applications often connect to database to store, retrieve, modify and delete the data. The database contains more sensitive and confidential data. The attacker gains access to these data by using SQL injection attack.web applications are of two types. One is the presentation oriented and the other is service oriented.

1.1 Web Application Architecture

A web application is the client/server software that deals with the user requests coming from clients such as web browsers. To serve the user requests, it requires accessing system resources such as databases and files at the server end. Figure 1 shows a simplified architecture of a web application. The system resources are a part of trusted environment and often contain security critical data.

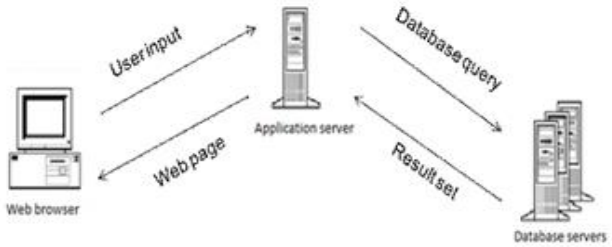

Figure 1 Web application architecture

Hence the resources need appropriate protection to maintain their confidentiality and integrity. It cannot be directly used to access the system resources. Checks should be performed to validate the user input before it can be used in the trusted environment. Lack of validity checks or inadequate checks can result in exploitable vulnerabilities.

1.2 SQL Injection Attacks

SQL injection attacks are one of the most common techniques used by the attackers, to attack the data server, the web server and the network. SQLIA poses the serious threat to the security of the web applications.

SQL injection attacks are performed through input fields. Attackers add the maliciously crafted inputs which performs the unauthorized operations. When the user submits the inputs, the SQL queries are generated. When the attacker adds the malicious input, the SQL query generated will be different from query generated for normal input. These attacks cause the loss of confidentiality, reliability and efficiency. These attacks are mainly due to flaws in input validation and query evaluation. SQL injection attacks are detected by existing approaches and the proposed method will be efficient.

1.3 Variants of SQL Injection Attacks

The literature survey reveals the different variants of SQL injection attacks. The following are the different kinds of SQL injection attacks based on the vulnerabilities.

1.3.1 Bypassing Web Application Authentication

This is the most common type of attack adopted by the attackers to bypass authentication in web applications. In this category of attack, an attacker exploits an input field that is used in a query's 'where' condition part. Attacker adds the OR condition with the input which makes the 'where' condition to be always true. An example for this type:

Query1: SELECT * FROM Users WHERE username='admin' AND password='pass';

Query2: SELECT * FROM Users WHERE username='admin' AND password='pass' OR 1=1;

Query1 is the normal query generated for normal input. And Query2 is the abnormal query generated after submitting the malicious code. This abnormal query will return all the records in the Users table.

1.3.2 Getting Knowledge of Database Fingerprinting

This type of attack is considered as pre-attack preparation by an attacker. This category of attack is made by entering some inputs by which it generates an illegal or the logically incorrect queries. The error messages show the names of the tables and the columns in the database that cause error. The attacker also comes to know about the application database used in the backend server. The following example shows this type of attack.

Query = SELECT * FROM Users WHERE username='admin' AND password='convert (select host from host)';

The query generated will not be executed but the error message will be displayed. The error message contains the details about the table and columns in the table. The attacker uses this information to start attacking the database easily.

Error message for this query:

ERROR 1064: You have an error in your SQL syntax. Check the manual that corresponds to your MySQL server version for the right syntax to use near 'username='admin' AND password= 'convert (select host from host)' at 1.

1.3.3 Injection with UNION query

In this type of attack, the attacker extracts data from a table which is different from the one that was intended in the web application by the developer. An example for this type of attack:

Query: SELECT * FROM Users WHERE username='' UNION SELECT balance_amt FROM Customer_savings WHERE accno=657654 – AND password='pass';

The above example returns the balance amount from Customer_savings table. The attacker can add any SQL query using this UNION keyword. The attacker can also delete table contents and drop the table.

1.3.4 Damaging with additional injected query

This type of attack is generally very harmful. An attacker enters input such that an additional injected query is generated along with the original query. This type of attack is generally called as piggy-backed query attack. Adding the malicious queries into the input causes generation of abnormal queries. The following example shows the harmfulness of this attack.

Query: SELECT *FROM Users WHERE username='admin' AND password=''; DROP TABLE Users;

In the above example, the table Users will be dropped due to additional query added to the input field.

1.3.5 Remote execution of stored procedures

This type of attack is performed by executing the procedures, stored previously by the web application developer. An example for this type of attack:

Query: SELECT * FROM Users WHERE username=''; SHUTDOWN; -- password='';

The above query results in shutting down of the database server. This leads to denial of service. This type of attack is more harmful to web applications. Based on the above variants of the SQL injection attacks, the attack detection approaches are discussed below.

II. RELATED WORKS

Various approaches have been proposed to the confrontation of the threat of web applications. The following are the approaches that have proposed for detecting and preventing SQL injection attacks.

2.1 Web framework

A web framework is a software framework that is designed to support the development of dynamic websites, web applications and web services. It uses a filtering method to remove special characters. Recently, some web frameworks have provided a wider variety of prevention methods than ever before. PHP provides Magic Quotes, which works when any combination of 4 special characters ', '', /, NULL exists in the data field of the POST, GET and COOKIES pages. It automatically adds a '\' in front of the special character to prevent SQL injection attacks. However, Magic quotes only works for the four special characters.SQL injection attacks with other symbols are not detected.

2.2 JDBC-Checker

Carl Gould et al., proposed a Static Analysis method in order to detect SQL injection vulnerability. They uses Java String Analysis (JSA) library to validate the user input type dynamically and prevent SQL injection attacks. However, if malicious input data has the correct type or syntax, it cannot protect against the SQL injection attack. Also, the JSA library only supports the Java programming language.

2.3 WASP

Halfond et al presented Web Application SQL Injection Preventer (WASP), a method for detecting and preventing SQLIAs on the server. WASP utilizes positive taint analysis and syntax-aware evaluation. Using positive taint, WASP fetches trusted values from a metastring library, and tracks them through the application to identify trusted parts of a query. This technique is not fully automated and requires a white-list.

2.4 Zhendong Su and Gary Wassermann's Approach

This approach uses a static analysis method which was combined with automated reasoning. This method assumes that there is no tautology in an SQL query generated dynamically, which was verified. Thus, this method is efficient in detecting SQL injection attacks, but other SQL injection attacks except for tautology cannot be detected.

2.5 Dynamic query structure validation

Sruthy Manmadhan et al designed the dynamic query structure validation which is done through checking query's semantics. It detects SQL injection by generating a benign query from the final SQL query generated by the application and the inputs from the users and then comparing the semantics of safe query and the SQL query. The main focus is on stored procedure attacks in which getting query structure before actual execution is difficult.

2.6 Query tokenization

Anjugam presented a lightweight method to prevent SQL injection attacks by applying query tokenization technique to convert SQL queries into number of useful tokens and then encrypting the table name, fields, literals and data on the query using AES encryption algorithm. This approach avoids memory requirements to store the legitimate query in repository and facilitates fast and efficient accessing mechanism with database.

2.7 Entropy and message authentication code

Diksha G. Kumar designed a new technique using entropy and message authentication code. At the client side, the size of the input is verified and checks for injection specified characters and keywords, if not so, query will be submitted to the server. In the server side, the entropy is calculated for all the queries and static MAC value is calculated. Dynamic MAC value is calculated after submitting the user and compared with static MAC value.

2.8 Flag sequencing

Manveen Kaur proposed a Flag Sequencing method in which the query is converted into flags and flags are separated and converted to integer values. Then the structures of the input query and previously stored query are compared. There are also some complexities involved such as FLAG separation, FLAG to integer conversion and the searching process in link list. Complexity of FLAGS to integer conversion is O(n) where n is the total number of literals in all FLAGS of query.

2.9 Adaptive algorithm

Ashish John dealt an Adaptive algorithm to detect SQL injection attacks which combines two methods. Parse tree validation technique is used to find the SQL injection query by comparing query length and code conversion method is used for converting the user input to code like ASCII, binary, etc and searching for the availability of converted input in the database. The disadvantages of this method are (i) Code conversion to each and every user input is more time consuming as well as the database size will also increase. (ii)Parse tree validation technique will raise false alarm even if legitimate user is having blank space in his/her input.

III. PROPOSED METHOD

In this section, a novel method is proposed that make use of dynamic and static analysis. The static analysis collects the normal queries from the hotspots in the web application. These queries are used for analyzing the dynamically generated queries. The normal queries collected are the fixed queries (FQ). These queries do not have inputs. The dynamic analysis involves the dynamic query evaluation which uses the fixed queries collected for evaluation. The proposed method contains following phases.

i. Input removal
ii. Parse tree validation

3.1 Input removal

The input removal phase removes the inputs from the dynamic query (DQ). The dynamic queries generated may contain malicious input in it. The inputs are removed by using the following Algorithm-1. The Algorithm-1 removes the inputs in the dynamic query and returns the dynamic query without inputs (DDQ). Thus the dynamic query without inputs is passed to next phase for evaluating it.

Example:

DQ: SELECT * FROM Users WHERE username='admin' AND password='pass';

DDQ: SELECT * FROM Users WHERE username='?' AND password='?';

The following is the Algorithm for input removal phase.

```
Algorithm separate( dynamic query)
{
Input: dynamic query(DQ);
Output: dynamic query without input(DDQ);
Quotation_status={quot_end, quot_start};
Current_status: quot_end;
Do while( not null(input))
{
for each( char in input)
{
if(char is quote)
{
Add char to output;
if(current_status=quot_end)
{
current_status=quot_start;
}
else{
current_status=quot_end;
}}
else
{
if(current_status=quot_end)
{
Add char to output;
}}}}
```

3.2 Parse Tree Validation

The dynamic query without input received from input removal phase is evaluated using the fixed query. The parse tree for the fixed query (FQ) and the dynamic query without input (DDQ) are constructed. The parse tree that are constructed, are compared. If the two parse trees are different, the dynamic query is considered as attack query. Or otherwise it is considered as safe query.

Example:

FQ: SELECT * FROM Users WHERE username='?' AND password='?';

DDQ: SELECT * FROM Users WHERE username='?' AND password='?' OR '?'='?';

The parse tree for the fixed query (FQ) is,

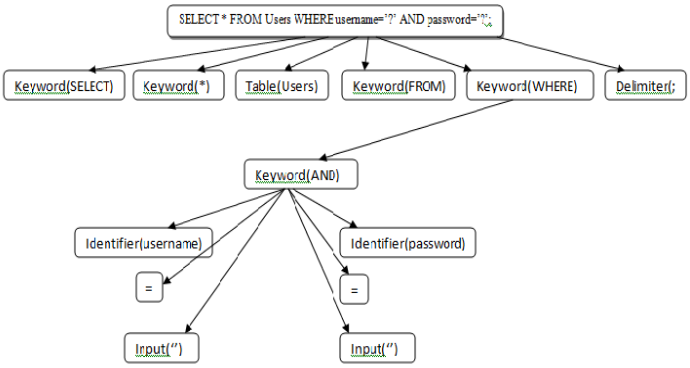

Figure 3.1 Parse tree for fixed query

The parse tree for the dynamic query without input is as follows,

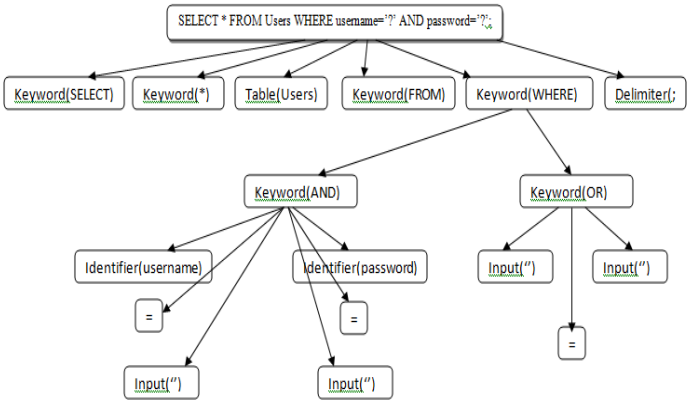

Figure 3.2 Parse tree for dynamic query without inputs

By comparing the parse trees of the fixed query (FQ) and dynamic query without inputs (DDQ), they are different. Thus, the dynamic query is considered as attack query.

IV. RESULT ANALYSIS

The proposed method is evaluated using the following metrics:

Detection rate

The detection rate is defined as the number of intrusion instances detected by the system (True Positive) divided by the total number of intrusion instances present in the test set.

False Alarm Rate

False Alarm Rate is defined as the number of 'normal' patterns classified as attacks (False Positive) divided by the total number of 'normal' patterns.

4.1 Experimental Setup

For evaluating the proposed method, the four different web applications are used from AMNESIA test beds. There are two sets of queries. One is the Legit set which contains the query with normal inputs that look

like attack patterns. These inputs are taken to analyze the false positive rate occurred in the proposed method. The second is Attack set which contains the queries with attack patterns.

4.2 Result Analysis

The existing methods like AMNESIA [1] and SQLIPA [4] are used for comparing the performance of the proposed system. The following Table 4.1 shows the detection rate and false positives of proposed approach. Four types of web applications are used to find the performance of the systems.

Table 4.1 Attack detected in different web applications

Web application	Proposed Method	Detection Rate	False Positive
Bookstore	867/867	100%	1
Portal	1145/1145	100%	0
Employee Directory	1589/1589	100%	0
Checkers	892/892	100%	1

Table 4.2 Comparison between proposed and existing approaches

Attacks	SQLIPA	Amnesia	Proposed
Tautology	Yes	Yes	Yes
Logically incorrect	No	Yes	Yes
Blind injection	Yes	Yes	Yes
Union queries	No	Yes	Yes
Piggy-backed	No	Yes	Yes
Timing attack	No	Yes	Yes
Alternate encodings	No	Yes	Yes
Stored procedure	No	Yes	Yes

V. CONCLUSION

SQL injection attacks cause major impact to the security of the web applications. The problem will exist when the input is not filtered properly. The proposed method detects all five different kinds of SQL injection attacks. By comparing with existing approaches, the proposed method is more efficient technique in protecting web applications from SQL injection attacks. Future evaluation work can be extended to concentrate on the precision of the proposed method and on the other attacks like XSS, CSRF, HTTP Response, etc.

References

[1] "The Open Web Application Security Project, OWASP TOP 10 Project." http://www.owasp.org/.
[2] G. Wassermann, Z. Su, "An analysis framework for security in web applications", In Proceedings of the FSE Workshop on Specification and Verification of Component-Based Systems, SAVCBS, pp. 70– 78, 2004.
[3] C. Gould, Z. Su, P. Devanbu, JDBC checker: " A static analysis tool for SQL/JDBC applications", in Proceedings of the 26th International Conference on Software Engineering, ICSE, pp. 697–698, 2004.
[4] W. G. Halfond and A. Orso, 'AMNESIA: Analysis and Monitoring for Neutralizing SQL Injection Attacks', Proceedings of the IEEE and ACM International Conference on Automated Software Engineering (ASE), 2004.
[5] Halfond WG, Orso A, Manolios P, 'Using Positive Tainting and Syntax-Aware Evaluation to counter SQL injection attacks', Proceedings of the 14th ACM SIGSOFT International Symp. on Foundations of Soft. Engg., pp.175-185, 2006.
[6] Shaukat Ali, Azhar Rauf, Huma Javed, 'SQLIPA: An Authentication Mechanism against SQL Injection', European Journal of Scientific Research, Vol.38 No.4, pp.604-611, 2009.
[7] Sruthy Manmadhan and Manesh T, 'A method of detecting SQL injection attack to secure web applications', International Journal of Distributed and Parallel Systems, Vol.3,No.6, pp.45-54, 2012.
[8] Jaskanwal Minhas, Raman Kumar, 'Blocking of SQL Injection Attacks by Comparing Static and Dynamic Queries', IJCNIS, Vol.5, No.2, pp.1-9, 2013.
[9] Anjugam S, A Murugan, 'Efficient Method for Preventing SQL Injection Attacks on Web Applications Using Encryption and Tokenization', IJARCSSE, Vol.4, No.4, pp.124-132, 2014.
[10] Ashish John, Ajay Agarwal, Manish Bhargwaj, 'An adaptive algorithm to prevent SQL injection', AJNC, Vol.4, pp.12-15, 2015.
[11] Manveen Kaur, 'SQL Injection attacks Its Prevention by Flag Sequencing Method', IKSPCEIS, Vol.6, No.2, pp.102-110, 2015.