SQL Injection Detection and Prevention using Input Filter Technique

Shruti Bangre, Alka Jaiswal

Abstract:- SQL injection attacks, a class of injection flaw in which specially crafted input strings leads to illegal queries to databases, are one of the topmost threats to web applications. A number of research prototypes and commercial products that maintain the queries structure in web applications have been developed. But these techniques either fail to address the full scope of the problem or have limitations. Based on our observation that the injected string in a SQL injection attack is interpreted differently on different databases, in this paper, we propose a novel and effective solution to solve this problem. It has been proposed to detect various types of SQLIA. This method checks the attribute value for single quote, double dash and space provided by the user through the input fields. When attacker is making SQL injection he should probably use a space, single quotes or double dashes in his input. Depending on the no of space, double dash and single quote the count value of the input field (having default count as1) will get increased by 1 respectively. The fixed count value and the dynamically generated count value of the input parameters are then compared. If both the count values are same, there is no SQLIA and if they vary that means some SQL code has been injected through the input fields. Finally such attempt will be recorded separately and will be blocked to access the database.

Index Term:-SQLIA, attribute, etc.

I. INTRODUCTION

SQL Injection attacks target databases that are accessible through a web front-end, and take advantage of flaws in the input validation logic of Web components such as CGI scripts. In the last few months application-level vulnerabilities have been exploited with serious consequences by the hackers have tricked e-commerce sites into shipping goods for no charge, usernames and passwords have been harvested and confidential information such as addresses and credit-card numbers has been leaked. The reason for this occurrence is that web applications and detection systems do not know the attacks thoroughly and use limited sets of attack patterns during evaluation. SQL Injection attacks can be easily prevented by applying more secure authentication schemes in login phase itself. Most of the attacks made on the web target the vulnerability of web applications. Vulnerabilities of web ability to obtain and charge the sensitive information, such as military systems, banks, and e-business, etc are exposed to a great security risk. Many divisions are researching a variety of methods to detect and prevent SQLIAs, and the most preferred techniques are Web Framework, Static Analysis, Dynamic Analysis, Combined Static and Dynamic Analysis, and Machine Learning Techniques. The Web Framework[2,3] provides filtering methods using the user's input data. However, it is only able to filter special characters therefore, other attacks cannot be prevented. Static Analysis methods[4-6] analyzes the input parameter type therefore it is more effective than filtering methods, but attacks using the correct parameter types cannot be detected. Dynamic analysis[7-9] can scan vulnerabilities of web applications without rewriting it however this method is also not able to detect all SQLIAs. Combined Static and Dynamic Analysis[10-13] can compensate for the weaknesses in each method and is highly proficient in detecting SQLIAs. The combined usage of Static Analysis and Dynamic Analysis method is very complicated. Furthermore, MachineLearning method[17,18] can detect unknown attacks, but results may contain many false positives and false negatives.

This paper proposes a simple and effective method to accurately detect and prevent SQLIAs by using SQL query parameter counter. Furthermore, the effectiveness of this method in web applications has been tested and validated. The rest of this paper is organized as follows. Section 2 reviews the architecture of web application and SQLIAs. Section 3 discusses the related work. Section 4 explains the proposed method which uses a combination of SQL query parameters removal and Combined Static and Dynamic Analysis methods for the detection of SQLIAs. Section 5 elaborates the experiment results using the proposed method, and Section 6 ends with a conclusion.

II. WEB APPLICATION AND SQLIAS

In order to understand the SQLIAs, there will be a brief explanation on the architecture and processes of web applications. Also, there will be a discussion on possible SQL-Injection vulnerabilities and attacks made on these web applications will be discussed.

A. Web Application Architecture

Although web application can be classified as programs running on a web browser, web applications generally have a Tree-tier construction as shown in Figure 1[10,12].

1) Presentation Tier: receives the user's input data and shows the result of the processed data to the user. It can be thought of as the Graphic User Interface (GUI). Flash, HTML, Javascript, etc. are

all part of the presentation tier which directly interact with the user.

2) CGI Tier: also known as the Server Script Process, is located in between the presentation tier and database tier. The data inputted by the user is processed and stored into the database. The database sends back the stored data to the CGI tier which is finally sent to the presentation tier for viewing. Therefore, the data processing within the web application is done at the CGI Tier. It can be programmed in various server

Manuscript Received on May 25, 2012
Shruti Bangre, RCET, Bhilai, India.
Prof. Alka Jaiswal, RCET, Bhilai, India.

script languages such as JSP, PHP, ASP, etc.

3) Database Tier: stores and manages all of the processed user's input data. All sensitive data of web applications are stored and managed within the database. The database tier is responsible for the access of authenticated users and the

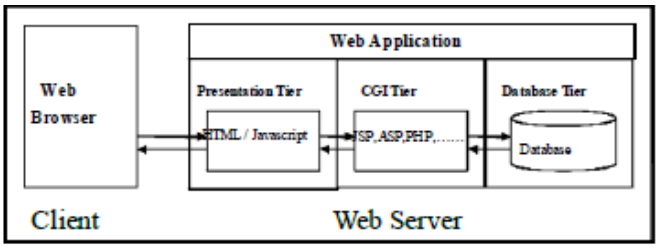

Fig. 1. Web Application Architecture

B. SQL-Injection Attacks

SQL-Injection vulnerabilities and attacks occur between the Presentation tier and the CGI tier. Most vulnerabilities are accidentally made in the development stage.

The data flow of each tier using normal and malicious input data are as shown in Figure 2. It depicts the user's authentication step. When an authenticated user enters its ID and Password, the Presentation tier uses the GET and POST method to send the data to the CGI tier. The SQL query within the CGI tier connects to the database and processes the data.

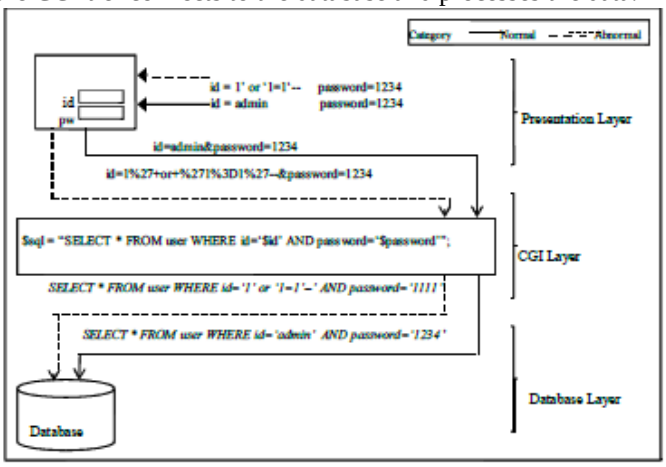

Fig. 2. SQL normal and SQL Injection Attack data flow

When a malicious user enters an ID such as 1' or '1=1'--, the query within the CGI tier becomes SELECT * FROM user WHERE id='1' or '1=1'—' AND password='1111';. After the --, the rest of the sentence becomes a comment and because or '1=1' is always true, the authentication step is bypassed. SQL Injection attacks are malicious data that changes the normal SQL query to a malicious SQL query and allows anomalous database access and processing. Most web applications use data filters to prevent these kinds of SQL injection attacks. However, there are many methods of SQL injection attacks which can bypass data filters which make it difficult to effectively defend the database from attacks. Therefore, a more effective way of detecting and preventing SQL injection attacks is necessary.

III. RELATED WORK

In order to detect and prevent SQL injection attacks filtering and other detection methods are being researched. This section explains the related work.

A. Web Framework

Recently, some Web Frameworks offer various SQL Injection attack prevention methods. PHP provides Magic Quotes[2]. It is a function used when any combination of 4 special characters ',",/, NULL is in the data sent from POST, GET, COOKIES. It automatically adds a '\' in front of the special characters to prevent SQLIAs. However, Magic quotes only support the four special characters. Therefore, other attacks methods that can bypass the Magic quotes exist. Also web applications must be rewritten in order to configure the Magic Quotes function. The Validator[3] in Struts verifies the user's input data using predefined rules. If the special characters used to execute a SQLIA are not predefined in the Validator, it will vulnerable to SQLIAs. In addition the setup of the Validator is very complicated.

B. Static Analysis

Static Analysis analyzes the SQL query sentence within a web application to detect and prevent SQL Injection attacks. It also requires rewriting of web applications. The focus of the static analysis method is to verify the user's input type to reduce potential SQL Injection attacks rather than to detect SQL injection attacks, the JDBC-Checker[4] uses Java String Analysis (JSA) library to validate the real time user's input and prevent SQL Injection attacks. However, if the malicious input data has the correct type and syntax, it cannot be prevented. JSA Library only supports Java. Wassermann[5] used Combined Automated Reasoning and Static Analysis method to prevent SQL Injection attacks. This method assumes there is no tautology in the real time SQL Query and verifies it. Thus this method is efficient in detecting SQL Injection attacks, but other SQL Injection attacks without tautology cannot be detected. Stephen[6] created a fix generation SQL query by collecting plain text SQL statements, SQL queries, and execution calls to verify user input types. This method does not directly prevent or detect SQL injection attacks, but by deleting previous vulnerabilities in SQL query syntax, it prevents attacks from SQLIAs. This method is only available for web applications written in java, and requires AST and SQL parser ZQL libraries.

C. Dynamic Analysis

Dynamic analysis, unlike static analysis, can locate vulnerabilities of SQL injection attacks without making any adjustments to web applications. Open source program Paros[7] scans not only SQLIAs vulnerabilities, but also other vulnerabilities within the web application. Paros is not perfect because it uses predetermined attack codes to scan and uses HTTP response to the success-rate of the attack. Sania[8] finds and collects SQL injection attack vulnerabilities between the web application and databases. Then, it proceeds to generate SQL

Injection attack codes. After attacking with the generated code, it collects the SQL query from the attack. After the normal SQL query is compared and analyzed with the SQL query collected from the attack using the parse tree, the success rate of the attack is verified. Since a parse tree is used, Sania is more accurate than using an HTTP response verification. Yonghee Shin[9] proposed to use Input Flow Analysis and input validation analysis to build a white-box, and generated test input data to locate SQL Injection vulnerabilities. Dynamic analysis method is advantageous because no web application adjustments are necessary. However, the vulnerabilities found in the web application must be manually fixed by the developers and not all of them can be found without predefined attacks.

D. Combined Static and Dynamic Analysis

Combined Static and Dynamic analysis method utilizes the advantages of both static analysis and dynamic analysis method to detect SQL injection attacks. SQLCheck[10] defines SQL Injection attacks and proposes a sound and complete algorithm based on context-free grammars and compilerparsing techniques. AMNESIA[11] finds hotspots produced in SQL queries inside web applications and generates all possible SQL queries. Generated Static SQL queries and user's input data are analyzed and classified using the JSA Library. Buehrer[12] compares the static SQL query and dynamic SQL query generated by the user using parse tree to detect SQL injection attacks. Wei[13] proposed to compare and analyze stored procedure in web applications with runtime user's input SQL query using a control flow graph to detect SQL injection attacks.

E. Instruction-Set Randomization

Instruction-set randomization method inputs random values into the runtime SQL query statement of a web application and checks for volatility to detect SQL injection attacks. SQLrand[14] places a proxy between the web server and the database server and randomizes SQL queries. If the random value can be predicted, this method is not effective.

F. SQL Query Profiling

Jae-Chul Park[15] profiles the SQL query of web application and compares dynamic SQL query generated in runtime using Pairwise sequence alignment of amino acid formulate method to detect SQL injection attacks. This method is advantageous because it can detect SQL injection attacks without rewriting the web application, but must be profiled after each change.

G. Machine Learning

Valeur[16] proposed an intrusion detection system using the machine learning method. The SQL queries generated in a web application are learned to generate models. Then runtime SQL queries are compared to the generated model to check for discrepancies. If a poor training set is used, many false positive and negative results may occur. WAVES[17] uses a web crawler to find vulnerabilities in web applications and generates attack codes by utilizing pattern list and attack techniques. Using the generated attack codes, the SQL injection attack vulnerabilities can be found. Since this method uses machine learning, it is more effective than the traditional penetration testing. However, this method cannot detect all vulnerabilities.

IV. PROPOSED METHOD

This paper proposes a new SQL injection attack detection method that utilizes both Static and Dynamic Analysis. This method compares and analyzes by counting the attribute (parameter) which is inputted by user. Every programmer knows that how many parameters will use for particular table. In this method the total number of parameter will save in a variable. In run time every parameter has one temporary variable which already stores 1 by default. Now every parameter value will check for space, single quote and double dash, because every type of attack needs the all three character. Whenever anyone of them found the temporary variable will increase by 1. When searching is complete, add temporary variable`s value of each parameter and result will compare with total number of parameter. If it will same means there is no sql injection attack. And if it will not same some means SQL code has been injected through the input fields. Finally such attempt will be recorded separately and will be blocked to access the database.

The proposed method is shown in figure 3.

V. EXPERIMENT AND EVALUATION

A. Experiment Method

The proposed algorithm for the detection of SQL Injection can be implemented on real web applications. It can store total number of parameter in web applications and make a list to compare with the real time generated value of parameter . Also, it can profile legitimate dynamic query generated by normal users between the web application and the database server and compare it with the dynamically generated query to detect attacks this paper we implement a simple parameter checking function in the web application. The web application used for the experiment is the " gotocode" which is used in experiments of other research papers[8,10,11,18]. Furthermore, for an accurate experiment, the web application vulnerability " Paros 3.2.13" [7]was used for the collection and scanning.

B. Experiment Result Analysis

This paper compares the detection rate of the proposed method with other researches under the same conditions. It also compares detection and prevention methods of particular attack forms. Furthermore, it compares and analyzes additional elements of the attack and prevention methods.

TABLE1

EXPERIMENT RESULTS

Web Applications	Proposed Algorithm		SQLCheck[11]		AMNESIA[12]	
	Attack / Detection	Detection Rate(%)	Attack / Detection	Detection Rate(%)	Attack / Detection	Detection Rate(%)
Employee Directory	247 / 247	100	3937 / 3937	100	280 / 280	100
Events	87 / 87	100	3605 / 3605	100	260 / 260	100
Classifieds	319 / 319	100	3724 / 3724	100	200 / 200	100
Portal	288 / 288	100	3685 / 3685	100	140 / 140	100
Bookstore	366 / 366	100	3473 / 3473	100	182 / 182	100

a. Attack Detection Rate Analysis

The experiment to evaluate the detection rate for SQL Injection, we used 5 types of web applications which is the same method as SQLCheck and AMNESIA. "Paros 3.2.13" was used to compare the attack frequency and detection frequency to calculate the detection rate. As shown in table 2 there is no difference in the detection rate compared to other researches. This is because the SQL Injection detection methods compare the static SQL queries with the dynamic SQL queries for detection rather than using machine learning or statistical methods. It results in a high detection rate. As a result, the detection rate cannot be used to judge the efficiency of the detection method

b. Comparison of Detection and Prevention Methods by Attack Form

W.G. Halfond[19]classified SQL Injection attacks into various types and used them to compare the efficiency of methods for detection and prevention. Therefore, we use the method of W. G. Halfond to compare the efficiency of the proposed method and other SQL Injection detection methods. The results are shown in table 2. As you can see in table 3, the JDBC-Checker, Tautology-checker, WebSSARI and Java Static Tainting use static analysis method. The Static analysis method only analyzes the static SQL queries implemented inside the web application and therefore the efficiency each method is distinctly different. IDS and WAVES use machine learning anomaly detection method which needs a large amount of SQL Injection for learning. The detection rate of the method depends on the learning stage. The algorithm proposed in this paper along with AMNESIA, SQLCheck and SQLGuard use both static and dynamic methods to detect and prevent SQL Injection attacks. This method compares all the static SQL queries with the dynamic SQL queries generated in real-time. It detects attacks by comparing the structureand the grammar of the query. If the dynamically generated query has a different structure or if it uses different grammar, it will be detected. However, AMNESIA, SQLCheck and SQLGuard use Parse trees to transform the static and dynamic SQL queries. As a result these methods cannot detect Stored Procedure type attacks and also, it is complex to transform all the SQL queries into Parse trees in real-time. Furthermore, the grammar and structure used in various Database Management Systems(DBMS) are different which makes the generation of parse trees dependent to the DBMS. On the contrary, the algorithm proposed in this paper does not use complex analysis methods such as Parse trees. It uses a very simple method which compares the queries after the removal of the attribute values. Therefore, it can be implemented into any type of DBMS and it is able to detect SQL Injection attacks which include Stored Procedure type attacks. The dynamic analysis method is not a solution for the detection and prevention of SQL Injection attacks. It is a method to find vulnerabilities of web applications. Therefore, it will not be mentioned for comparison. For instance the JDBC-Checker reduces the chance of SQL Injection attacks by checking the type of the SQL queries which has no relation with detection and prevention.

c. Comparison of Additional Elements used for Detection

The comparison of additional elements used for detection is shown in table 3. In the proposed algorithm, the additional elements vary by the development method[19]. When using the profiling method, the Proxy is needed as a element and when using the SQL query checking function, the developer learning and source code adjustment element is needed. When using the SQL query list method, there is no need for any additional elements

TABLE 2
DETECTION AND PREVENTION METHODS OF VARIOUS SQL INJECTION ATTACKS

Detection/Prevention Method	Tautologies	Illegal/Incorrect Queries	Union Queries	Piggy-Backed Queries	Stored Procedures	Inference	Alternate Encodings
AMNESIA[12]	z	z	z	z	x		
CSSE	z	z	z	z	x		
IDS	{	{	{	{	{	{	{
Java Dynamic Tainting	-	-	-	-	-	-	-
SQLCheck[11]	z	z	z	z	x		
SQLGuard	z	z	z	z	x		
SQLrand	z	x	z	z	x		
Tautology-checker	z	x	x	x	x	x	x
Web App. Hardening	z	z	z	z	x		
JDBC-Checker	-	-	-	-	-	-	-
Java Static Tainting	z	z	z	z	z		
Safe Query Objects	z	z	z	z	x		
Security Gateway	-	-	-	-	-	-	-
SecuriFly	-	-	-	-	-	-	-
SQL DOM	z	z	z	z	x	z	z
WAVES	{	{	{	{	{	-	{
WebSSARI	z	z	z	z	z	z	z
Proposed Method	z	z	z	z	z	z	z

Symbols: 'z' defines that detection and prevention is possible 'x' defines that detection and prevention is impossible
'{' defines that detection and prevention is partially possible '-' defines that there is no relation

TABLE 3
ANALYSIS OF ADDITIONAL ELEMENTS OF EACH DETECTION AND PREVENTION METHOD

Detection Method	Source Code Adjustment	Attack Detection	Attack Prevention	Additional Elements
AMNESIA	Not needed	Automatic	Automatic	N/A
CSSE	Not needed	Automatic	Automatic	Custom PHP interpreter
IDS	Not needed	Automatic	Report generation	IDS system training set
JDBC-Checker	Not needed	Automatic	Source code adjustment proposed	N/A
Java Dynamic Tainting	Not needed	Automatic	Automatic	N/A
Java Static Tainting	Not needed	Automatic	Source code adjustment proposed	N/A
Safe Query Objects	Needed	N/A	Automatic	Developer learning
SecuriFly	Not needed	Automatic	Automatic	N/A
Security Gateway	Not needed	Detailed manual	Automatic	Proxy filter
SQLCheck	Needed	Partially automatic	Automatic	Key management
SQLGuard	Needed	N/A	Automatic	N/A
SQL DOM	Needed	Automatic	Automatic	Developer learning
SQLrand	Needed	Automatic	Automatic	Proxy, Developer learning, Key management
Tautology-checker	Not needed	Automatic	Source code adjustment proposed	N/A
WAVES	Not needed	Automatic	Report generation	N/A
Web App. Hardening	Not needed	Automatic	Automatic	Custom PHP interpreter
WebSSARI	Not needed	Automatic	Partially automatic	N/A
Proposed Algorithm (SQL Query Check Function)	Needed	Automatic	Automatic	Developer learning
Proposed Algorithm (SQL Query Profiling)	Not needed	Automatic	Automatic	Proxy
Proposed Algorithm (SQL Query Listing)	Not needed	Automatic	Automatic	N/A

VI. CONCLUSION

This paper proposes a SQL Injection detection method by comparing the static SQL parameter with the dynamically generated parameter value after counting some characters of the attribute values. Furthermore, we evaluated the performance of the proposed method by experimenting on a vulnerable web application. We also compared our method with other detection methods and confirmed the efficiency of our proposed method. The proposed method simply counts some characters i.e. space, double dash and single quote of the attribute value in the web pages for analysis which makes it independent from the DBMS. Complex operations such as Parse trees or particular libraries are not needed in the proposed method. Though this method cannot detect all attacks on vulnerabilities on web applications, since it uses both static analysis and dynamic

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

analysis, it can effectively detect SQL Injection attacks on web applications. The proposed method can not only be implemented on web applications but it can also be used on applications connected to databases. Furthermore, it can be implemented to be used for SQL query profiling, SQL query listing and detection program modularization.. Future work is needed on this research for not only SQL Injection attacks but also other web application attacks such as XSS.

REFERENCES

[1] The Open Web Application Security Project, "OWASP TOP 10 Project" http://www.owasp.org/ .
[2] PHP, magic quotes, http://www.php.net/magic_quotes/ .
[3] Apache Struts project, Struts. http://struts.apache.org/ .
[4] 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.
[5] 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.
[6] Thomas. S, Williams. L, "Using Automated Fix Generation ot Secure SQL Statements", In Proceeding of the 29th international Conference on Software Engineering Workshops (ICSEW. IEEE Computer Society), pp. 54, 2007
[7] Paros. Parosproxy.org, http://www.parosproxy.org/
[8] Kosuga. Y, Kernel. K, Hanaoka. M, Hishiyama. M, Takahama. Yu, " Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection" , In Proceedings of the Computer Security Applications Conference 2007, pp. 107-117, 2007.
[9] Yonghee Shin, "Improving the Identification of Actual Input Manipulation Vulnerabilities", 14th ACM SIGSOFT Symposium on Foundations of Software Engineering ACM, 2006.
[10] Z. Su, G. Wassermann, " The Essence of Command Injection Attacks in Web Applications" , In Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 372-382, 2006.
[11] Halfond W. G, Orso. A, "AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks", In Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering, pp. 174-183, 2005.
[12] Buehrer. G, Weide. B. W, Sivilotti. P A, "Using Parse Tree Validation to Prevent SQL Injection Attacks", In Proceedings of the 5th international Workshop on Software Engineering and Middleware, pp. 105-113, 2005.
[13] Wei. K, Muthuprasanna. M, Kothari. S, "Preventing SQL injection attacks in stored procedures", Software Engineering Conference 2006. Australian, pp. 18-21, 2006.
[14] S. Boyd, A. Keromytis, "SQLrand: Preventing SQL injection attacks", Applied Cryptography and Network Security LNCS, Volume 3089, pp. 292-302, 2004.
[15] Jae-Chul Park, Bong-Nam Noh, "SQL Injection Attack Detection: Profiling of Web Application Parameter Using the Sequence Pairwise Alignment", Information Security Applications LNCS, Volume 4298, pp. 74-82, 2007.
[16] F. Valeur, D. Mutz, G. Vigna, "A Learning-Based Approach to the Detection of SQL Attacks", In Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment, pp 123-140, 2005.
[17] Huang. Y, Huang. S, Lin. T, Tasi. C, "Web application security assessment by fault injection and behavior monitoring", In Proceedings of the 12th international Conference on World Wide Web, pp 148-159, 2003.
[18] GotoCode, http://www.gotocode.com/ .
[19] W. G. Halfond, J. Viegas, A. Orso, "A Classification of SQL-Injection Attacks and Countermeasures", In proceeding on International Symposium on Secure Software Engineering Raleigh, NC, USA, pp. 65-81, 2006.