2009 IEEE International Advance Computing Conference (IACC 2009)
Patiala, India, 6–7 March 2009

Combinatorial Approach for Preventing SQL Injection Attacks

R. Ezumalai, G. Aghila
Department of Computer Science, Pondicherry University, Puducherry, India.
ezumalai1984@gmail.com, aghila.csc@pondiuni.edu.in

Abstract - **A combinatorial approach for protecting Web applications against SQL injection is discussed in this paper, which is a novel idea of incorporating the uniqueness of Signature based method and auditing method. The major issue of web application security is the SQL Injection, which can give the attackers unrestricted access to the database that underlie Web applications and has become increasingly frequent and serious. From signature based method standpoint of view, it presents a detection mode for SQL injection using pair wise sequence alignment of amino acid code formulated from web application form parameter sent via web server. On the other hand from the Auditing based method standpoint of view, it analyzes the transaction to find out the malicious access. In signature based method it uses an approach called Hirschberg algorithm, it is a divide and conquer approach to reduce the time and space complexity. This system was able to stop all of the successful attacks and did not generate any false positives.**

Index terms – Security, SQL Injection, Needleman-Wunsch Algorithm, Hirschberg Algorithm, DBMS Auditing.

I. INTRODUCTION

Today's modern web era, expects the organization to concentrate more on web application security. This is the major challenge faced by all the organization to protect their precious data against malicious access or corruptions. Generally the program developers show keen interest in developing the application with usability rather than incorporating security policy rules. Input validation issue is a security issue if an attacker finds that an application makes unfounded assumptions about the type, length, format, or range of input data. The attacker can then supply a malicious input that compromises an application. When a network and host level entry points are fully secured; the public interfaces exposed by an application become the only source of attack. The cross site scripting attacks, SQL Injections attacks and Buffer Overflow are the major threat in the web application security through this input validation security issues [11]. Especially SQL Injection attacks breach the database mechanism such as Integration, Authentication, Availability and authorization [8]. Since 2002, nearly 50% of total cyber vulnerabilities were input validation vulnerabilities.

Since 2002, 20% of the input validation issues are SQL Injection vulnerabilities (SQLIVs) and, therefore, 10% of total cyber vulnerabilities since 2002 [18]. SQL injection attack involves placing SQL statements in the user input for corrupting or accessing the Database [11]. Even the SQL Injection attacks can bypass the security mechanism such as Firewall, cryptography and traditional Intrusion detection systems. If the trend of providing web-based services continues, the prevalence of SQLIVs is likely to increase.

The most worrying aspect of SQL Injection attack are; it is very easy to perform, even if the developers of the application are well known about this type of attacks. The basic idea behind in this attack is that the malicious user counterfeits the data that a web application sends to the database aiming at the modification of the SQL Query that will be executed by the DBMS software. Input validation issues can allow the attackers to gain complete access to such databases. Technologies vulnerable to SQL Injection attacks are dynamic Script languages like ASP, ASP.net, PHP, JSP, CGI, etc. In addition, all types of database have been severely vulnerable in such type of SQL Injection attacks [18].

Researchers have proposed a different techniques to provide a solution for SQLIAs (SQL Injection attacks), but many of these solutions have limitations that affect their effectiveness and practicality. Researchers have indicated that solution to these types of attacks may be based on defense coding practices. But it's not efficient because of three reasons. First, it is very hard to bring out a rigorous defensive coding discipline. Second, many solutions based on defensive coding address only a subset of the possible attacks. Third, legacy software poses a particularly difficult problem because of the cost and complexity of retrofitting existing code so that it is compliant with defensive coding practices. In this work, an attempt has been made to increase the efficiency of the above techniques by a combinatorial approach for protecting web application against SQL Injection attacks.

The remainder of the paper is organized as follows: Section 2 contains background and related work; Section 3 describes our proposed approach. Section 4 describes the conclusion and future work.

II. BACKGROUND AND RELATED WORK

SQL Injection is one of the main issues in database security. It affects the database without the knowledge of the database administrator. It may delete the full database or records or tables without the knowledge of the respective user or administrator. It is a technique used to exploit the database system through vulnerable web applications [9]. These attacks not only make the attacker to breach the security and steal the

978-1-4244-1888-6/08/$25.00 © 2008 IEEE

entire content of the database but also, to make arbitrary changes to both the database schema and the contents. SQL injection attack could not be realized about information compromization until long after the attack has passed. In many scenarios, the victims are unaware that their confidential data has been stolen or compromised. With the help of simple web browser, SQL Injection attacks can be performed by attackers [12]. The following section describes the attacks with an example.

Generally the Authenticated users have username and password such as

Username: bala
Password: ravi12

The SQL Query format is

Select * from table where usrnme='bala' and psswd='ravi12';

The above query fetches the needed records from the database where usrname and psswd is available in the database or it shows some error messages to the browsers.

Malicious user injects the following SQL Injection in this field:

Username: bala
Password: anything' or '1'='1

Then the dynamic SQL query constructed from the above information is

Select *from table where usrnme='bala' and psswd='anything' or '1'='1';

In this SQL statement '1'='1' is always true in the table and this expression connection with 'OR' to another expression will always return true. The result of this query performs SQL Injection attacks.

A. Authorization by pass (SQL manipulation)

This attack allows the attacker access to the privileges of the first user in the database [15]. This attack would be used to by pass the log on screen. The example of this attack is discussed in the above section.

B. Exploiting SELECT

SQL injection is not only a straight forward attacks but also it has some background trickery attack is present. Most of the time attackers would see some error message and will have to reverse engineer their queries.

Direct Vs Quoted (SQL manipulation) - In SQL manipulation, Direct or quoted are the types of SQL Injection attacks. In direct attack the input data become part of the SQL statement formed by the application. Attacker has to add space (‘ ‘) for manipulate the SQL statement and OR to the input. The error message has been returned if the injection was successful.

SQL = “SELECT Title, Author, Publishing from Books ORDER by “&strInput

All the other possible injections are quoted SQL statements. In quoted injection the injected string has following a quote appended to it similar to the statement shown,

SQL = “SELECT Title, Author, Publishing from Books WHERE ISBN = ‘ ” & strInput & “ ‘ ”

In order to manipulate the SQL statement successfully input string must contain a single quote ‘ before the use of first SQL keyword and ends in a WHERE statement that needs single quote appended to it.

C. Exploiting Insert

Insert Basics - Sites like shopping, when registration, it allows the user to feed inputs and store it. INSERT statement allowing the user input is stored in the back end. The misuse of inserts statements by the attacker results in many rows in the database with corrupt data.

Injecting Subselect - Normally an insert statement looks like this: Insert into TableName Values (‘eOne’, ‘eTwo’, ‘eThree’); Suppose the sample SQL statement is used by the application.

INSERT INTO Table Values (‘ “ & eOne & “ ‘ , ‘ “ & eTwo & “ ‘)”

and the values that are input by the user are:

Name: ‘ + (SELECT TOP 1 Fieldname from TableName) + ‘
Email: --@xxx.com

The resulting SQL statement looks like this

INSERT INTO tableName values (‘ “ ‘ + (SELECT TOP 1 Fieldname FROM tableName) + ’ ‘ , ‘--- @xxx.com’)

In injecting subselect, the first value in the Field will be displayed in place of the user name in the above attack. If TOP 1 is not used, there will be an error message “subselect returned too many rows”. Attacker can go through all the records using NOT In clause.

D. Exploiting System Stored Procedures (Function call)

Attacker uses stored procedures to corrupt the database system and its an most harmful attacks. Database uses stored procedures to perform database Administrative operations. If attacker is able to inject SQL string successfully then attacker can exploit these stored procedures. Access to these procedures depends on the privileges of the user on the database.

SELECT proid, cusid, proname from product where prodname like ‘1’ or ‘1’=’1’;exec master.dbo.xp cmdshell ‘dir’—‘

The above injected SQL query will return all the rows from the table as well as execute the operating system command DIR.

The following subsection describes in detail about the related works based on SQL Injection Attacks. A number of

researches had been taken to provide solutions for SQL Injection Attacks.

Xiang Fu et al [1], propose the design of a static analysis framework (called SAFELI) for identifying SIA (SQL Injection attacks) vulnerabilities at compile time. SAFELI statically monitoring the MSIL (Microsoft Symbolic intermediate language) byte code of an ASP.NET Web application, using symbolic execution. SAFELI can analyze the source code information and will be able to identify very delicate vulnerabilities that cannot be discovered by black-box vulnerability scanners. The main limitation of Xiang et al's work is that this approach can discover the SQL injection attacks only on Microsoft based product.

Buehrer et al [12], propose the mechanism which filters the SQL Injection in a static manner. The SQL statements by comparing the parse tree of a SQL statement before and after input and only allowing to SQL statements to execute if the parse trees match. They conducted a study using one real world web application and applied their SQLGUARD solution to each application. It is stopped all of the SQLIAs without generating any false positive results, their solution required the developer to rewrite all of their SQL code to use their custom libraries.

Wassermann and Su [6], propose a static framework to analyze and filter the user inputs. According to them, their approach has restricted to discover only logic-based attacks. i.e. attacks that always result in true or false SQL statements.

Livshits and Lam [21], propose another static analysis approach for finding the SQL injection using vulnerability pattern approach. Vulnerability patterns are described here in this approach. The main issues of this method, is that it cannot detect the SQL injection attacks patterns that are not known beforehand.

Konstantinos et al [3], propose a mechanism to detect SQL injection with novel-specification based methodology. This approach utilizes specifications that define the intended syntactic structure of SQL queries that are produced and executed by the web application. The main limitation of this paper is the computational time overhead to compares the SQL statement with the predefined structure at run time.

William G.J. Halfond and Alessandro Orso [2], propose a mechanism to prevent SQL injection at run time. AMNESIA uses a model based approach to detect illegal queries before it sends for execution to database. In its dynamic method, the technique uses run time monitoring method to inspect each and every query which is passed to its techniques. The main issue of the AMNESIA is it requires the modification of the web application's source code.

William G.J. Halfond, Alessandro Orso, Panagiotis Manolios [13], proposed the mechanism to keep track of the positive taints and negative taints. This work outlined a new automated technique for preventing SQLIAs based on the novel concept of positive tainting and on flexible syntax-aware evaluation. It will check the SQL statement with this taints and if it finds any suspicious activity, it will generate the alarm. The advantage of this mechanism, that it does not require any modification of the run time system even at application level and imposes a low execution overhead.

Marco Cova et al [19], propose a mechanism to the anomaly-based detection of attacks against web applications. Swaddler analyzes the internal state of a web application and learns the relationships between the application's critical execution points and the application's internal state. By doing this, Swaddler is able to identify attacks that attempt to bring an application in an inconsistent, anomalous state, such as violations of the intended workflow of a web application. The main limitation is the overhead grows linearly as the number of executed basic blocks increases. This is due to instrumentation and detection overhead associated with each basic block in the program.

Many authors discussed various techniques to prevent SQL Injection attacks through many ways in static and dynamic analysis and also in DBMS auditing methods. But all these methods reported to have a lot of pros and cons of its own proposal. In this paper, a new attempt has been proposed and worked out against SQL Injection attacks.

III. Our Approach

Our approach against SQLIAs is based on Signature based approach, which has been used to address security problems related to input validation. This approach describes three modules which are used to detect the security issues. Monitoring module has got the statement from the web application which can decide whether it can send the statement to database for execution. Analysis module uses Hirschberg algorithm to compare the statement from the specifications. Specifications comprise the predefined keywords and send it to analyze module for comparisons. It analyzes the comparisons as well as database transaction. If it finds any suspicious activity, it acts as an active agent to stop the transaction and audit the attacks. If both analysis module and auditing module has satisfied, it provides the complete transaction. The following figure 1 clearly depicts the architecture of the system to prevent the SQL Injection attacks using the new combined approach. The following section outlines each module's work in detail.

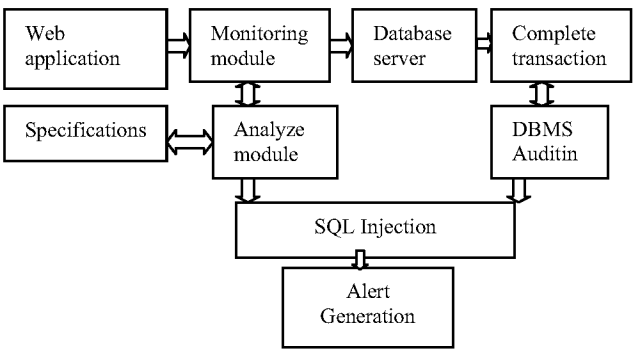

Figure 1: Combinatorial Approach for Preventing SQL Injection Attacks

A. *Monitoring Module*

In monitoring module, it gets an input from the web application and sends it to analysis module for further checking. If analysis module find any suspicious activity, it generate error message to monitoring module to block the further transactions.

B. *Specifications*

Specifications comprise the predefined keywords and send it to analysis module for comparisons. These modules have all predefined keywords which is stored in the database.

C. *Analysis Module*

Analysis module gets an input from the monitoring module and it finds a hot spot from the application and it uses Hirschberg algorithm for string comparison. An advantage of Hirschberg algorithm, it uses Divide and conquer methodology to prevent SQL Injection attacks. The time complexity of this algorithm is O(nm) and space complexity is O(min(nm)).

Hot Spot

Hot spot is that line where it gets the input from the user and vulnerable in execution. This step performs a simple scanning of the application code to identify hotspots. For the example servlet in prg below, the set of hotspots would contain a single element: the statement at line 6,7,8. (In Java-based applications, interactions with the database occur through calls to specific methods in the JDBC library,2 such as

java.sql.Statement.execute(String).)

```
public class Show extends HttpServlet {
1. public ResultSet getUserInfo(String login, String password)
{
2. Connection co = DriverManager.getConnection ("MyDB");
3. Statement stmt = co.createStatement();
4. String queryString = "";
5. queryString = "SELECT info FROM userTable WHERE ";
6. if ((! login.equals("")) && (! password.equals(""))) {
7. queryString += "login='" + login + "' AND pass='" +
password + "'";}
8. ResultSet tempSet = stmt.execute(queryString);
9. return tempSet;
}
...
}
```

This step identifies the hot spot(6,7,8)and it divides the hot spot in to tokens and it send it to query validation phase.

Hirschberg Algorithm

Hirschberg's algorithm is a divide and conquer version of the Needleman-Wunsch algorithm. Hirschberg's algorithm is commonly used in computational biology to find maximal global alignments of DNA and protein sequences. Hirschberg's algorithm is a generally applicable algorithm for finding an optimal sequence alignment. BLAST and FASTA are suboptimal heuristics.

If x and y are strings, where |x| = n and |y| = m, the Needleman-Wunsch algorithm finds an optimal alignment in O(nm) time, using O(nm) space. Hirschberg's algorithm is a clever modification of the Needleman-Wunsch Algorithm which takes O(nm) time, but needs only O(min{m,n}) space.

For pair wise sequence, Needleman-Wunsch algorithm and Smith-Waterman algorithm are mostly used. Both use dynamic programming based on identical Mathematical background(analysis of most common characters) in order to locate the most appropriate sequence alignment,

But in order to reduce the time and space complexity, present a Hirschberg algorithm to provide a comparison of strings. But Hirschberg algorithm takes time complexity as O(nm) in worst case and space complexity is O(min(nm)) for two sequences,

$$F(I,j)= \text{Max}\{ F[i-1, j-1)+s(xi,yi), F(i,j-1)+d, F(i-1,j)+d)\}$$

Table 1: Hirschberg Algorithm

F(i,j)	A	C	T	G	A
A	F(i-1,j-1)	F(I,j-1) +d (d-gap penalty)			
C	F(i-1,j)+d (d-gap penalty)				
T					

The above equation describes the algorithm. There are three paths in the scoring matrix for reaching a particular position i, j: (1) a diagonal move from position i-1, j-1 to position i, j with no gap penalties, (2) a move from any position in column j to i, j, with a gap penalty, or (3) a move from any position in row i to i, j with a gap penalty. But this system could not consider the gap penalty. It directly match between the two sequences. For two sequences a = a1 a2 … an and b = b1 b2 … bn, where Sij =S(a1 a2 … ai , b1 b2 … bj), then:

Where Sij is the score at position i in sequence a and position j in sequence b, s(ai bj) is the score for aligning the characters at positions i and j, d is the penalty for a gap of length x in sequence a, and d is the penalty for a gap of length y in sequence b.

Sij is a type of running best score as the algorithm moves through every position in the matrix. Eventually, all of the matrix positions (all Sij values) are filled. If a global alignment that involves all of the sequences is required, the matrix score in the last row and column is used as the alignment score. Use of the dynamic programming method requires a scoring system for the comparison of symbol pairs, and a scheme for insertion/deletion (GAP) penalties.

Table 2: Using Hirschberg Algorithm to find the similarities

(I,J)	SELECT	*	Fro m	Prod	Where	Usr	=	“		“	And	Pss	=	“		“	;
SELECT	1																
*		1															
From			1														
Prod				1													
Where					1												
Usr						1											
=							1										
“								1									
					DIVIDE	AND	CONQUER										
“										1							
And											1						
Pss												1					
=													1				
“														1			
“																1	
;																	1

This method is based on genetic distances between sequence pairs. The genetic distance between two sequences is the fraction of aligned positions in which the sequence has been changed.

In our proposed system, it maintains the table as it is like gene comparison that it contains keywords which are present in horizontal or vertical line and it will compare the incoming tokens with this predefined values using this algorithm for identity.

In this SQL statement, this is an hot spot which has been identified by the analyze module and it send to this table for detect and prevent SQL Injection attacks.

SQL= “SELECT * FROM prod WHERE Usr= ”& tUsername&”’AND Pss = ‘”&tPassword&””” ;

As per the Hirschberg algorithm, it divides the token and it checks the each token with the predefined tokens using divide and conquer methodology.

SQL Injection code

Select *from prod where usr=”bala” and pss=” anything” or '1'='1';

This analyzer module detects SQL Injection taken place after (anything”) this token to prevent SQL Injection attacks.

The above algorithm uses Divide and conquer methodology as described here. X strings are stored in horizontal line where as Y strings are stored in vertical line. Being a divide and conquer methodology, it divide the (one strings) problem in two sub problem. It compares one sub problems with predefined data and it compares another sub problem with another set to match comparison and it combines the sub problem solutions to main problem solutions. So, it takes time complexity as O(nm) and it needs space complexity is O(min(m,n)).

D. DBMS Audit

Auditing is a facility of the DBMS that enables DBAs to track the use of database resources and authority [11]. When auditing is enabled the DBMS will produce an audit trail of database operations. Each audited database operation produces an audit trail of information including what database object was impacted, who performed the operation, and when. Depending on the level of auditing supported by the DBMS, an actual record of what data actually changed also may be recorded. But it has some limited functionality to predict the attacks. It is very useful to find that what type of operation has been made. For example End user is a student that he can log on to the college server that he can see his personal or his academic record. Only option given to the user is to check their information. i.e (Select operation). Instead of select operation, any deletion or update operation is made; attackers could login in to the system and to do some malicious actions. So this auditing method try to block not only SQL Injection attacks and also some other attacks [18]. The restriction of this DBMS auditing method is to prevent the attackers view some other records, that select operation has been made. It never generates any alarms. Signature based method itself work effectively against SQL Injection attacks well and also this DBMS auditing also provides an support to this method to work effectively against SQL Injection attacks.

The Integrated system is under development and the partial results shows encouraging output.

IV. CONCLUSION

This paper presented a novel highly automated approach for protecting Web applications from SQLIAs. Our approach consists of 1) identifying hot spot from the application, 2) using Hirschberg algorithm to find out the SQL Injection attacks. 3) Using DBMS auditing methods to find out the transactions. Hirschberg algorithm uses divide and conquer approach to detect the SQL Injection attacks in order to reduce the time and space complexity and it provides the complete execution after analyzing the DBMS auditing. Our approach also provides advantages over the many existing techniques whose application requires customized and complex runtime environments: Since, it is defined at the application level, requires no modification of the runtime system, and imposes a low execution overhead.

REFERENCES

[1] Xiang Fu, Xin Lu, Boris Peltsverger, Shijun Chen, "A Static Analysis Framework For Detecting SQL Injection Vulnerabilities", IEEE Transaction of computer software and application conference, 2007.

[2] William G.J. Halfond, Alessandro Orso,Panagiotis Manolios, "WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation", IEEE Transaction of Software Engineering Vol 34, No1, January/February 2008.

[3] Konstantinos Kemalis and Theodoros Tzouramanis, "Specification based approach on SQL Injection detection", ACM, 2008.

[4] Stephen Thomas and Laurie Williams "Using Automated Fix Generation to Secure SQL Statements", International workshop on Software engineering and secure system ", IEEE, 2006.

[5] V. Benjamin Livshits and Monica S. Lam, "Finding Security Vulnerabilities in Java Applications with Static Analysis", ACM, 2005.

[6] Z. Su and G. Wassermann, "The Essence of Command Injection Attacks in Web Applications", 33rd ACM SIGPLAN, SIGACT Symposium on Principles of Programming Languages, Charleston, South Carolina, USA, 2006, pp. 372-382.

[7] SruthiBandhakavi, "CANDID: Preventing SQL Injection Attacks using Dynamic Candidate Evaluations", ACM, 2007.

[8] Ashish kamra, Elisa Bertino, Guy Lebanon, "Mechanisms for database intrusion detection and response", Data security & privacy, Pages 31-36, ACM, 2008.

[9] Christina Yip Chung, "DEMIDS: A Misuse Detection System for Database Systems", Integrity and internal control information systems, Pages: 159 - 178, ACM, 2008.

[10] Y. Huang, F. Yu, C. Hang, C. H. Tsai, D. T. Lee, and S. Y. Kuo. "Securing Web Application Code by Static Analysis and Runtime Protection", In Proc. of the 13th Intl. World Wide Web Conference (WWW 04), pages 40–52, May 2004.

[11] David Geer, "Malicious Bots Threaten Network Security", IEEE, Oct 8, 2008.

[12] G.T. Buehrer, B.W.Weide and P.A..G.Sivilotti, "Using Parse tree validation to prevent SQL Injection attacks", In proc. Of the 5th International Workshop on Software Engineering and Middleware(SEM '056), Pages 106-113, Sep. 2005.

[13] W.G. J. Halfond and A. Orso, "Combining Static Analysis and Run time monitoring to counter SQL Injection attacks", 3rd International workshop on Dynamic Analysis, St. Louis, Missouri, 2005, pp.1.

[14] N. Jovanovic, C. Kruegel, and E. Kirda, "Pixy: A Static Analysis tool for detecting web application vulnerability", In 2006 IEEE Symposium on Security and Privacy, May 2006.

[15] O. Maor and A. Shulman, "SQL Injection Signature Evasion", White paper, Imperva, Apr 2004.

[16] R. McClure and I. Kru ger, "SQL DOM: Compile Time Checking of Dynamic SQL Statements", In proc of the 27th Int. Conference on Software engineering (ICSE 05), pages 88-96, May 2005.

[17] A. Nguyen-tuong, S. Guarnieri, D. Greene, J.Shirley, and D. Evans, "Automatically hardening web applications using Precise Tainting", In Twentieth IFIP Intl, Information security conference(SEC 2005), May 2005.

[18] Xin Jin, Sylvia Losborn, "Architecture for data collection in database intrusion detection system", Secure data management, Springerlink, 2007.

[19] Marco Cova, Davide Balzarotti, Viktoria Felmetsger, and Giovanni vigna, " Swaddler: An approach for the anamoly based character distribution models in the detection of SQL Injection attacks", Recent Advances in Intrusion Detection System, Pages 63-86, Springerlink, 2007.

[20] Mehdi Kiani, Andrew Clark,George Mohay, "Evaluation of Anomaly Based Character Distribution Models in the Detection of SQL Injection Attacks", Third International Conference on Availability, Reliability and Security - Volume 00, , Issue, 4-7 Page(s):47-55, IEEE, March 2008.

[21] V.B. Livshits and M.S. Lam, "Finding Security vulnerability in java applications with static analysis", In proceedings of the 14th Usenix Security Symposium, Aug 2005.