2012 IACSIT Coimbatore Conferences
IPCSIT vol. 28 (2012) © (2012) IACSIT Press, Singapore

Development of an Effective Runtime Defense Algorithm for Web Application Security

Sonal Telang [1+], Prof. Ajit Kumar Shrivastava [2], Dr. Manish Manoria [3]

[1] M-Tech (Scholar) TIEIT

[2] Head of Department (CSE), TIEIT

[3] Prof. and Director, TIEIT

Abstract: Web application plays an important role in different fields like finance sector, business, e-shopping etc. There is no. of web application vulnerabilities such as SQL injection, Buffer overflow etc. Above these SQL Injection vulnerabilities are very harmful for web applications. In literature survey there are number of technique used to prevent SQLIA in application level, but not in database level. SQL injection attacks occur due to vulnerabilities in the design of queries where a malicious user can take advantage of input opportunities to insert code in the queries that modify the query-conditions resulting in unauthorized database access. In this paper we design an effective algorithm to prevent stored procedure from SQLIA in database level. Hirschberg's algorithm is used to prevent the stored procedure, which reduces both time as well as space complexity. We also analyze several aspects which have been discussed further.

Keywords: SQL injection, Hirschberg's Algorithm, Database security, DBMS audit

1. Introduction

Most of the web applications contain security vulnerabilities which enable attackers to exploit them and launch attacks. Architectural approaches [1], [2], [3] attempt to prevent malicious code execution by making certain pages of memory non-executable. This protection methodology is effective for many of the circumvent traditional attacks; however, attackers still manage to them [5] [6]. As a result of the attacks confidentiality, integrity and availability of information are lost. 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[17] [18].SQL Injection Attacks (SQLIA's) constitute an important class of attacks in web application. By leveraging insufficient input validation, an attacker could obtain direct access to the database underlying an application. Recently the incident of SQLIA is so high that in year 2008 it increases by 134% and become a predominant type of web vulnerabilities [15] [16]. The vulnerability exploited by SQLIA is based on maliciously updating the actual query conditions that are dependent on user-inputs. SQL injection leads to violation of privacy, security and integrity of data stored in the database.

The rest of this paper is arranged as follows: Section 2 introduces SQL injection attacks; Section 3 describes about SQLIA in stored procedure; Section 4 shows the evolution and recent scenario; Section 5 describes the proposed work and Section 6 describes Conclusion and outlook.

2. SQL Injection Attacks:

With the recent rapid increase in interactive web applications that employ back-end database services, an SQL injection attack has become one of the most serious security threats [7]. It attempts to modify the parameters of a Web-based application in order to alter the SQL statements that are parsed to retrieve data from the database. Attackers would send SQL to interact with RDBMS servers or modify existing SQL to

+ Corresponding author. Tel.: +(0755-2559997);
E-mail address: sonaltelangmt@gmail.com.

retrieve unauthorized information without any authentication. The result of these attacks is often disastrous and can range from leaking of sensitive data to the destruction of database contents [8] There is very little emphasis on securing objects residing in the database layer such as stored procedure which are also vulnerable to SQLIA. Attacker injects malicious SQL codes through the web application and causes unexpected behavior from the database. It has been estimated that at least 50% of the large e-commerce sites and about 75% of the medium to small sites are vulnerable to SQLIA. Consider the following, in order to login to the website, the user inputs his name and password, by clicking on the submit button the following SQL query is generated:

SELECT * FROM user_table WHERE user_id= ‘sonal’ and password= ‘2345’

if the user input the following user_id: (“‘or 1=1 -- ‘”)

SELECT * FROM user_table WHERE user_id= “ ‘ or 1=1 -- ‘ “ and password = ‘2345’

This is always returns true results. (‘) called the quotation mark tells the parser that the username string is finished, (“or 1=1 -- “) called fragment appended to the statement which always true, (--) comment mark which tells the parser that the statement is finished and password will not be checked.

Stored procedure are very harmful against SQLIA because it is a set of one or more SQL statements that are together stored in database. A procedure is logically a grouped set of SQL statements that perform a specific task and stored in the data dictionary. Once it is modified, all clients automatically get the new version [12]

3. Evolution and Recent Scenario

Various SQL prevention techniques have been proposed are as follows:

- Ke Wei, M Muthuprasanna and S. Kothari [12], proposed the prevention technique in stored procedure, combines static application code analysis with runtime validation to eliminate the occurrence of SQLIA. The limitation of the given technique is that it uses session id which is used to separate the user inputs from the SQL statement that might be guessed by the attacker.
- R. Ezumalai, G. Aghila [4] proposed a signature based approach to block SQLIA. This technique is used to address security problems related to input validation.
- William G.J Halfond and Alessandro Orso [8] proposed WASP (Web Application SQl-Injection Preventer) prototype tool is based on dynamic tainting which marks and tracks at runtime.
- Buehrer et al [13], 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 statement to execute if the parse trees match. It is stopped all of the SQLIAs without generating any false positive results.
- In Ankit Anchlia et al [9] proposed to test the applications that interact with the databases using SQL queries embedded in the code. There are automated ways to test the applications written in imperative and structured languages. However, the methodologies to test the applications with embedded SQL queries are still in the nascent phase.
- In 2010, Michelle Ruse, et al. [10] proposed a novel technique to identify the possibilities of such attacks. Technique is based on automatically developing a model for a SQL query such that the model captures the dependencies between various components (sub-queries) of the query.
- In 2010, Atefeh Tajpour et al. [11] proposed type of SQL injection attack and also different approaches which can detect or prevent them are presented. Finally they evaluate these approaches against all types of SQL injection attacks and deployment requirements.

4. Proposed Work

New framework is proposed for stored procedure, describes three modules. Supervision module gets the input from the user and send it to the analyze module. Analyze module uses Hirschberg’s Algorithm for comparing the two strings of SQL statements from the specifications. The following figure 1 clearly gives the framework to prevent SQLIA in stored procedure. The following section gives the detail.

4.1. Supervision Module:

This module is used to do the supervision of the user input and send it to the analyze module. It incorporated between web server and web applications.

4.2. Specifications:

This module contains all the predefined keywords which are stored in database. It includes the predefined keywords and sends it to the analysis module for comparison.

4.3. Analyze Module:

Analyze module is very important module in the given framework. It uses Hirschberg's algorithm for comparison of two sequences. It is a divide and conquer version of Needleman–Wunsch algorithm. This algorithm is generally applicable for finding an optimal sequence alignment.

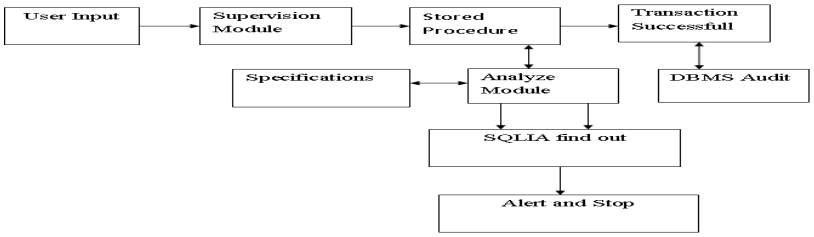

Fig1.Proposed Framework for preventing SQLIA

Hot Spot

Hot spot is that line where it gets the input from the user and vulnerable in execution. This performs a simple scanning of the application code to identify hotspots. Consider the following stored procedure. This step identifies the hot spot (7) and it divides the hot spot into tokens and it sends it to query validation phase.

```
1.  CREATE PROCEDURE SQLIALoginNormal
2.  @userName VARCHAR(50),
3.  @password VARCHAR(50)
4.  AS
5.  BEGIN
6.  DECLARE @sql VARCHAR(400);
7.  SET @sql='SELECT * FROM users_table
    WHERE user_id='''+@userName+''' AND
    Password='''+@password+'''';
8.  PRINT @sql;
11. EXEC (@sql);
12. --select 5
13. END
```

Table1 pseudo code for stored procedure

Hirschberg's Algorithm:

It is a dynamic programming algorithm that finds the least cost sequence alignment between two strings. This is a generally applicable for finding an optimal sequence alignment [14]. 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 takes O (nm) time, but needs only O (min {m, n}) space.

F(i,j)	S	E N	D
A	F(i-1,j-1)	F(i,j-1)+d	
N	F(i-1,j)+d		
D			

Table2 Hirschberg's Algorithm

$F(i,j) = \max\{F[i-1,j-1)+t(x_i, y_i), F(i,j-1)+px, F(i-1,j)+py)\}$

$F(i,j) = 1$ if $(x_i = y_i)$ otherwise 0 , $t(x_i, y_i)$ - score for aligning the characters at positions i and j. p is the

penalty for a gap. There are three paths in the scoring matrix for reaching a particular position i, j: (1) a diagonal move from position i-i, 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 matches between two sequences. For two sequences a = al a2 ... an and b = bl b2 ... bn, where Sij =S (al a2 ... ai , bi 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 of through every position in the matrix. Eventually, all of the matrix positions (all Sij values) are filled. F (i, j) is a type of running best score as the algorithm moves through every position in the matrix. But in our approach, gap penalty has ignored. If X_i=Y_i, then plot "1" else "0" till all the character has been visited. The system maintains the table which contains keywords which are present in horizontal or vertical line and it will compare the incoming tokens with these predefined values using this algorithm for identity. In this SQL statement, this is a hot spot which has been an identified by the analyze module and it send to this table for detect and prevent SQLIA.

SQL= "SELECT * FROM user_table WHERE User_id= “Usename&"'AND Password = "'&Password&"';

As per the algorithm, it divides the token and checks the each token with the predefined tokens.

Select *from user_table where user_id= “‘or 1=1 -- ‘” and password=" anything";

This analyzer module detects SQL Injection taken place and passes this token to prevent SQLIA. X strings are stored in horizontal line & Y strings are stored in vertical line. Being a divide and conquer methodology, it divide the 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. It takes time complexity as O (nm) and it needs space complexity O (min(m ,n)).

d) DBMS Audit: Auditing is the facility of DBMS that enables DBA to track the use of database resources and authority [7]. This module is used to audit the information including what database object was impacted, who and when the operation is performed. It records the actual data which is changed. This DBMS auditing tries to block not only SQLIA but also some other attacks.

i, j	SELECT	*	FROM user_table	WHERE	user_id	=	“		“	and	password	=	“	“	;
SELECT	1														
*		1													
FROM			1												
user_table			1												
WHERE				1											
user_id					1										
=						1									
“							1								
			Divide	and	Conquer										
“									1						
and										1					
password											1				
=												1			
“													1		
“														1	
;															1

Table 3 Using Hirschberg’s Algorithm to find the similarities

5. Results

The results we analyze when the above algorithm is used then the demo result are shown.

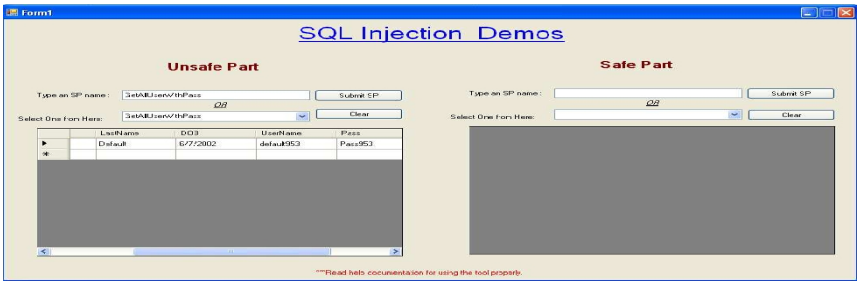

Fig 2. SQL Query in Unsafe Mode

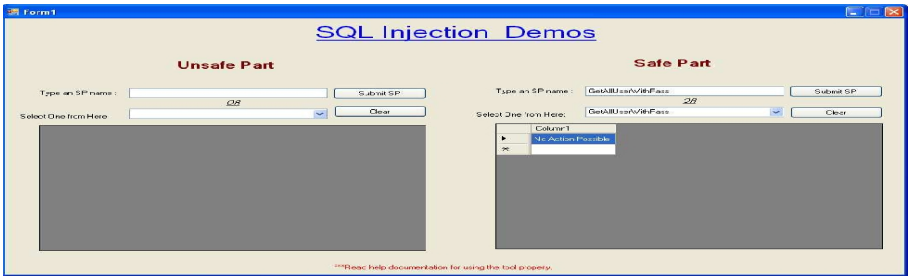

Fig3. SQL Injection Demo

6. Conclusion and Outlook

SQLIA occur due to vulnerabilities in the design of queries where a malicious user can take advantage of input opportunities to insert code in the queries that modify the query-conditions resulting in unauthorized database access. In this paper we design an effective algorithm to prevent stored procedure from SQLIA in database level. Divide and Conquer strategy is used, which reduces both time as well as space complexity. We also analyze several aspects which have been discussed further.

References

[1] "A Detailed Description of the Data Execution Prevention (dep) Feature in Windows xp Service Pack 2, Windows xp Tablet pc ed. 2005, and Windows Server 2003," http://support.microsoft.com/ kb/875352, Dec. 2006.

[2] "Pax Pageexec Documentation," http://pax.grsecurity.net/docs/ pageexec.txt, Dec. 2006.

[3] Intel Corporation, IA-32 Intel Architecture Software Developer's Manual Volume 3A: System Programming Guide, Part 1. Intel Corp., publication number 253668, 2006.

[4] "*Buffer Overflow Attacks Bypassing dep (nx/xd bits)—Part 2: Code Injection*,"http://www.mastropaolo.com/, Dec.06

[5] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, "*StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks*," Proc. Seventh USENIX Security Conf., pp. 63-78, 1998.

[6] H. Etoh, "Gcc Extension for *"Protecting Applications from Stack-Smashing Attacks*," http://www.trl.ibm.com/projects/security/ ssp/, Dec. 2006.

[7] R. Ezumalai, G. Aghila "*Combinatorial Approach for Preventing SQL Injection Attacks"*, IACC IEEE 2009

[8] William G.J Halfond, Alessandro Orso, P. Manolios, "*WASP: Protecting Web Application Using Positive Tainting and Syntax-Aware Evaluation* ", IEEE transaction of Software Engineering Vol 34, No 1, January/February 2008.

[9] Ankit Anchlia, Sheela Jain," *A novel Injection Aware Approach for the Testing of Database Applications*", 2010 International Conference on Recent Trends in Information, Telecommunication and Computing, IEEE.

[10] M.Ruse, Tanmoy Sarkar & Samik Basu, "*Analysis & Detection of SQL Injection Vulnerabilities via Automatic TestCase Generation of Programs*", 2010 10th Annual International Symposium on Applications and the Internet, IEEE.

[11] A.Tajpour, M. JorJor zade Shooshtari, "*Evaluation of SQL Injection Detection & Prevention Techniques*",2010 II International Conference on Computational Intelligence, Communication Systems and Networks, IEEE.

[12] Ke Wei, M. Muthuprasanna, Suraj Kothari, "*Preventing SQL Injection Attacks in Stored Procedures*", Proceedings of the 2006 Australian Software Engineering Conference (ASWEC'06), IEEE

[13] G.T. Buehrer,B. W. Weide, & A. G. Sivilotti "*Using Parse Tree Validation to Prevent SQLIA"ACM-I 2005*

[14] Ezumalai R, Aghila G "*Prevention of Web Attacks Using Hirschberg Algorithm".*

[15] http://www.sans.org/reading_room/whitepapers/securecode/sql-injection-modes-attack-defence-matters_23

[16] *"Top ten most critical web application vulnerabilities"* www.owasp.org/index.asp

[17] MITRE. Common vulnerabilities and exposures list. http://cve.mitre.org/.

[18] David Geer, *"Malicious Bots Threaten Network Security*", IEEE, Oct 8,2008