IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 17, Issue 1, Ver. III (Jan – Feb. 2015), PP 19-24
www.iosrjournals.org

SQL Injection Prevention by Adaptive Algorithm

Ashish John
Dept. of Computer Science and Engineering, SRM University, NCR Campus

Abstract: *An SQL Injection is one of the most dangerous security issues. SQL injections are dangerous because they are a door wide open to hackers to enter your system through your Web interface and to do whatever they please - i.e. delete tables, modify databases. The principal behind SQL injection is pretty simple. When an application takes user data as an input, there is an opportunity for a malicious user to enter carefully crafted data that causes the input to be interpreted as part of a SQL query instead of data. Databases are attractive targets because they typically contain critical application information. SQL injections are a programming error and they have nothing to do with your web site hosting provider. So, if you have been searching for a secure JSP hosting, PHP hosting or any other type of web hosting packages, you need to know that prevention of an SQL injection is not a responsibility of your web site hosting provider but of your web developers. In this paper, we had firstly surveyed different SQL Injection methods and then different techniques against SQL Injection and analyzed their advantages and disadvantages and proposed a novel and effective solution to avoid attacks on login phase.*

Keywords: *SQLIA, Parse Tree Validation, Code Conversion, Static Query;*

I. Introduction

The Internet has just entered the Middle Ages. The simple security model of the Stone Age still works for single hosts and LANs. But it no longer works for WANs in general and Internet in particular [1]. The lack of adequate knowledge and understanding of software and security engineering leads to security vulnerabilities, e.g. by inappropriate programming, getting even worse under deadline pressure and rush to market issues. Some solution may be effective today, but as technology changes, new risks and challenges appear. Moreover, different solutions must be combined to be effective against different types of attacks and the security of the system must be constantly monitored. A database-driven Web application commonly has four tiers namely presentation tier, logic tier, application server and data tier.

The presentation tier is the topmost level of the application. It displays information related to such services as browsing merchandise, purchasing, and shopping cart contents, and it communicates with other tiers by outputting results to the browser/client tier and all other tiers in the network.

The logic tier is pulled out from the presentation tier, and as its own layer, it controls an application's functionality by performing detailed processing.

An application server in an n-tier architecture is a server that hosts an application programming interface (API) to expose business logic and business processes for use by applications.

The data tier consists of database servers. Here, information is stored and retrieved. This tier keeps data independent from application servers or business logic. Giving data its own tier also improves scalability and performance.

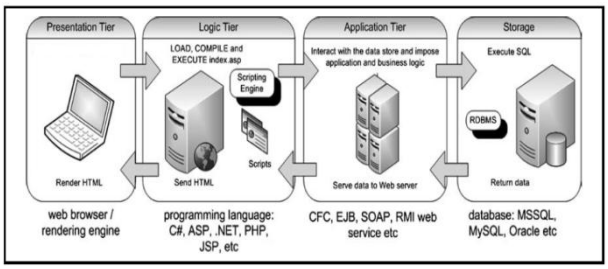

Fig.1 Architecture of web application

The back-end database often contains confidential and sensitive information such security numbers, credit card number, financial data, medical data. Typically the web user supplies information, such as a username and password and web applications receive user request and interact with the back-end database and returned relevant data to the user[2]. Some of the commonly performed web attacks are: Injection attacks, XSS Attack, CSRF Attack, Security Misconfiguration etc. According to OWASP (Open Web Application Security Project) Injection attack is at the first place of the top 10 web attacks that are executed in 2013[3]. SQL injection is a method for exploiting web applications that use client-supplied data in SQL queries. SQL Injection refers to

the technique of inserting SQL meta-characters and commands into Web-based input fields in order to manipulate the execution of the back-end SQL queries[4]. The SQLIA occurs when an intruder changes the structure of the query by inserting any SQL commands. This paper proposes a very simple and effective method to detect SQL Injection Attacks which uses the combination of Parse Tree Validation Technique and Code Conversion Method. The rest of the paper is organized in the form of different sections. Section 2 describes the SQLIA and its categories. Section 3 discusses the related work. Section 4 explains the proposed method to detect and prevent SQLIAs. Section 5 describes the results with some discussion. Section 6 concludes this paper.

II. SQL Injection

SQL injection is a technique (like other web attack mechanisms) to attack data driven applications. The attacker takes the advantage of poorly filtered or not correctly escaped characters embedded in SQL statements into parsing variable data from user input. The attacker inject arbitrary data, most often a database query, into a string that's eventually executed by the database through a web application (e.g. a login form).

2.1 SQL injection method

Here are some methods through which SQL statements are injected into vulnerable systems
- Injected through user input.
- Injection through cookie fields contain attack strings..
- Injection through Server Variables.
- Second-Order Injection where hidden statements to be executed at another time by another function.

2.2 SQLIA Types

There are various types of SQL Injection available. Some them that are highly used are described and explained here with there SQL codes and explanation.

For the simplicity and generalization of our work we had created database named 'security' which consist following tables:

Table_name
emails
Referrers
uagents
Users

Table 1: list of table in database - security

From the given table name we can easily identify that table name 'users' may contain some interesting data. The different fields and data of 'user table' is shown below:

ID	username	password
1	some	some
2	Admin	admin
3	admin1	Password
4	Administrator	123456
5	administrator1	abc123

Table 2: entries in table users

2.2.1 Tautology Attack

Purpose:
- Identify injectable parameters
- Bypass authentication
- Extract data

In logic, a tautology is a formula which is true in every possible interpretation. In a tautology-based attack the code is injected using the conditional OR operator such that the query always evaluates to TRUE. Tautology-based SQL injection attacks are usually bypass user authentication and extract data by inserting a tautology in the WHERE clause of a SQL query. The query transform the original condition into a tautology, causes all the rows in the database table are open to an unauthorized user.

Eg.:

Attacker's Input:

User ID: admin
Password: ' or '1'='1

Backend Process:
Select * from table where userid='admin' and pass='' or '1'='1';

Here sql statement had been modified that an OR operator is added in the statement such that if one part of the statement is it will return true thereby allowing the attacker to get successful login the that account.

2.2.2 Piggy-backed Queries / Statement Injection Attack
Purpose:
- Extract data
- Modify dataset
- Execute remote commands
- Denial of service

This type of attack is different than others because the hacker inject additional queries to the original query, as a result the database receives multiple SQL queries. The first query is valid and executed normally, the subsequent queries are the injected queries, which are executed in addition to the first. Due to misconfiguration a system is vulnerable to piggy-backed queries and allows multiple statements in one query.
Attacker's Input:

User ID: some
Password: ' ; drop table users --

Backend Process:
Select * from table where userid='some' and pass='';drop table users -- ;

Here, due to the vulnerable field attacker was able to add query along with existing query. This added query will drop the users table.

2.2.3 Union Query
Purpose:
- Bypassing authentication
- Extract data

This type of attack can be done by inserting a UNION query into a vulnerable parameter which returns a dataset that is the union of the result of the original first query and the results of the injected query. The SQL UNION operator combines the results of two or more queries and makes a result set which includes fetched rows from the participating queries in the UNION.
Basic rules for combining two or more queries using UNION :
1) Number of columns and order of columns of all queries must be same.
2) The data types of the columns on involving table in each query should be same or compatible.
3) Usually returned column names are taken from the first query.
By default the UNION behalves like UNION [DISTINCT] , i.e. eliminated the duplicate rows; however, using ALL keyword with UNION returns all rows, including duplicates. The attacker who try to use this method must have solid knowledge of DB schema.
Attacker's Input:

User ID: ' union select database();--
Password: abcd

Backend Process:
Select * from table where userid='' union select database(); -- and pass='abcd';

This will return the database name ie security. A thorough knowledge of sql will enable attacker to extract information for different table of the current database.

2.2.4 Illegal/Logically Incorrect Queries
Purpose:
- Identify injectable parameters
- Identify database
- Extract data

In this type of injection an attacker is try gather information about the type and structure of the back-end database of a Web application. The attack is considered as preliminary step for further attacks. If an incorrect query is sent to a database, some application servers returns the default error message and the attacker takes the advantage of this weakness. They inject code in vulnerable or injectable parameters which creates syntax, type conversion, or logical error. Through type error one can identify the data types of certain columns. Logical error often expose the names of tables and columns.

Attacker's Input:

date: 29a/10/2014

Generated Error:

PLS-00306: wrong number or types of arguments in call to 'USERS'
ORA-06550: line 1, column 7:
*from this error attacker receives table name (users) and database (oracle) being used.

2.2.5 Stored Procedures

Purpose:

- Privilege escalation
- Denial of service
- Execute remote commands

A stored procedure is a subroutine available to applications that access a relational database system. Extensive or complex processing that requires execution of several SQL statements is moved into stored procedures, and all applications call the procedures. One can use nested stored procedures by executing one stored procedure from within another. Stored procedures type of SQL injection try to execute store procedures present in the database. Most of the database have standard set of procedures (apart from user defined procedures) that extend the functionality of the database and allow for interaction with the operating system. The attacker initially try to find the database type with other injection method like illegal/logically incorrect queries. Once an attacker determine which databases is used in backend then he try to execute various procedures through injected code. As the stored procedure are written by developers, therefore these procedures does not make the database vulnerable to SQL injection attacks. Stored procedures can be vulnerable to execute remote commands, privilege escalation, buffer overflows, and even provide administrative access to the operating system. If an attacker injects ';SHUTDOWN; -- into either the User ID or Password fields then it will generate the following SQL code :

Attacker's Input:

User ID: abcd
Password: ' ;SHUTDOWN;--

Backend Process:

Select * from table where userid='abcd' and pass=''; SHUTDOWN;--

III. Related Work

3.1 Parse Tree Validation Technique [5]

The technique is based on comparing, at run time, the parse tree of the SQL statement before inclusion of user input with that resulting after inclusion of input.

3.2 Code Conversion Method [6]

1. Converting User input to code like ASCII, binary, hexa etc.
2. Searching the availability of converted input in Data table and returns valid Userid and Password.

IV. Proposed Method

The proposed method consists of the best features of both parse tree validation technique and code conversion method. In this method we parse the user input and check whether its vulnerable, if there is any chance of vulnerability present then code conversion will be applied over that input. In this way, we can detect and prevent SQL Injection using a single code. Below is the algorithm for the proposed method:

For web pages that saves data to the database:

begin()

- get user input
- compare with generalized SQL Query
- if length mismatch
  - display – possibly an attack

- newvariable= hexa(user input)
- set counter =1

o else

- display – safe input
- newvariable= user input
- set counter = 0

o display – value to be stored in database 'newvariable'

end;

For web pages that only retrieves from the database:

begin()

o if counter = 0

- display - variable

o else

- variable = ascii(newvariable)
- display - variable

end;

From the value of counter we can come to know whether the user input is converted or not.

V. Implementation

After implementing the proposed algorithm in php with mysql, we have obtained a more secured input. We were able to sanitize the user input easily without putting any extra load to the processor or to the database. Fig 2 & fig 3 show the changes between a safe user input and a vulnerable user input.

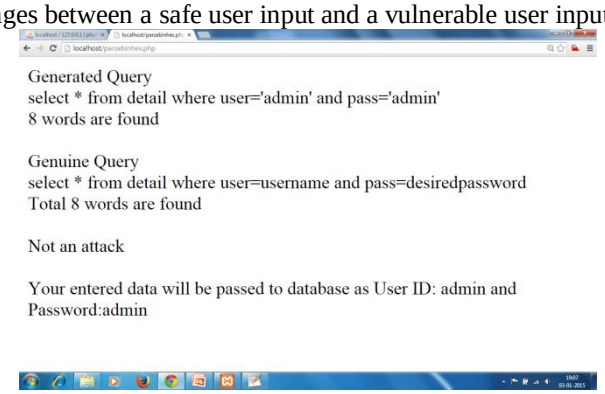

Fig2: safe input

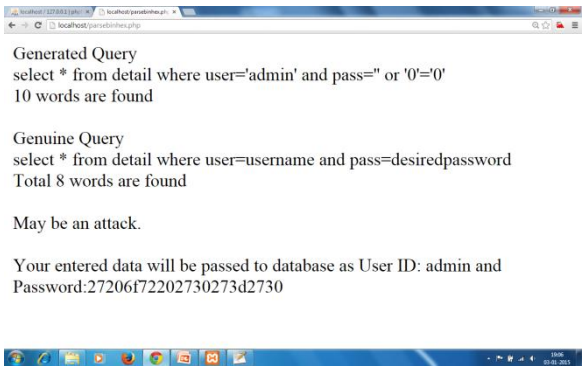

Fig3: vulnerable input

VI. Result and Discussion

After the implementation of various types of SQL injection Attacks the results received showed how important and crucial data is received by modifying the query. This loss of data causes loose to a company in millions. We had implemented various attacks in order to get an in depth knowledge of how these attacks work. Then the results are obtained after implementing the attacks. After studying and implementing some of the available methods we had come to the below mentioned result:

- Code Conversion to each and every user input is more time consuming as well as the database size will also increase.

- Parse tree validation technique will raise false alarm even if legitimate user is having blank space in his/her input.

Since available methods are not sufficient on their own to stop SQL injection attacks so in the present scenario more than one method is used so as to ensure higher security level.

VII. Conclusion and Future work

By conducting a comprehensive survey on existing techniques, we have realized that many SQL injection countermeasures have their limitations. Understanding and identifying the working mechanisms, as well as advantages and disadvantages of current techniques will benefit the work in this area. We had combined available SQL injection prevention methods to get a higher level of security.

Besides the text field, SQL Injection attacks can be performed through cookies or through server variables. As this work is a part of my M. Tech. thesis work, our future work will be preventing SQLI attacks that are being performed by any other mean.

References

[1]. Oppliger, R., "Internet security enters the Middle Ages," Computer , vol.28, no.10, pp.100,101, Oct 1995 doi: 10.1109/2.467613
[2]. http://www.w3resource.com/sql/sql-injection/sql-injection.php
[3]. www.owasp.org
[4]. W.G.J. Halfond, A. Orso, "AMNESIA: analysis and monitoring for Neutralizing SQL-injection attacks," 20th IEEE/ACM International Conference on Automated Software Engineering, Long Beach, CA, USA, 2005, pp. 174–183.
[5]. Michele Spagnuolo,Politecnico di Milano,Milan "Using Parse Tree Validation to Prevent SQL Injection Attacks"
[6]. Indrani Balasundaram, E. Ramaraj "An Efficient Technique for Detection and Prevention of SQL Injection Attack using ASCII Based String Matching" International Conference on Communication Technology and System Design 2011 © 2011 Published by Elsevier Ltd. Selection and/or peer-review under responsibility of ICCTSD 2011
[7]. Shruti Bangre, Alka Jaiswal "SQL Injection Detection and Prevention Using Input Filter Technique" International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-1, Issue-2, June 2012
[8]. Jaskanwal Minhas and Raman Kumar "Blocking of SQL Injection Attacks by Comparing Static and Dynamic Queries" I. J. Computer Network and Information Security, 2013, 2, 1-9 Published Online February 2013 in MECS (http://www.mecs-press.org/) DOI: 10.5815/ijcnis.2013.02.01
[9]. W. Halfond, J. Viegas, and A. Orso. A Classification of SQL-Injection Attacks and Countermeasures. Proceedings of the IEEE International Symposium on Secure Software Engineering (ISSSE), 2006.
[10]. "A Survey of SQL Injection Defense Mechanisms By Kasra Amirtahmasebi", Seyed Reza Jalalinia and Saghar Khadem, Chalmers University of Technology, Sweden Presented at: Institute of Electrical and Electronics Engineers in 2009
[11]. William G.J. Halfond, Jeremy Viegas, and Alessandro Orso "A Classification of SQL Injection Attacks and Countermeasures".