A Propose Model for Prevention of Attack SQL Injection

[1]Dr. T. N. Sharma, [2]Amitesh Kumar, [3]Sumitra Singar, [4]Vishakha Singhal
[1]Director Logicpace pvt. Ltd.
[2,4]Dept. of Computer Engineering
[3]Dept. of Electronics & Communication
[2,3,4]Poornima Institute of Engineering & Technology

Abstract
SQL attacks are growing concerns for the security threats to Web applications. The attackers use the SQL to manipulate the applications and manage to penetrate into the databases underlying the applications. This leads to unauthorized access to the potentially sensitive information. In this paper, we intend to discuss various methods through which SQL injection can be done. Also, we purport the methods to prevent the attack of SQL injection.

Keywords
SQL Injection, tautology, cyber crime, Union Queries, SQLIA.

I. Introduction

Internet is a widespread information infrastructure. Unaware of the security and privacy, the internet is becoming a repository of information. One of the most threads to web application is SQL injection attack. They are regularly employed by spiteful users for a variety of reasons similar financial scam, theft of private data, website vandalism, disrupt, etc. Web applications that are susceptible to SQL injection may allow an attacker to gain complete access to their essential databases. Because these databases often contain delicate consumer or user information, the resulting security abuses can include identity theft, loss of private information and fraud [1]

The SQL are used to form dynamic queries using the inputs from web applications. These dynamic queries are then passed to the data set for execution. In web applications, queries are represented as the strings. Whenever the attacker attacks the application, he passes a SQL code in the form of string as input to the application. The query consists of malicious code. However, the database will treat the query as input from an authorized source and will execute it. Thus, the malicious code will be executed which can result in revealing critical information.

There has been a lot of attention to the SQL injection vulnerabilities; many proposed solutions fail to address the full scope of the problem. There are myriad techniques through which SQL injection attacks can be done. Therefore, the solutions proposed either detect or prevent the SQL injection attacks' subset.[2]

This paper describes the details of the possible SQL injection attack techniques and concepts. Also, it proposes a way to prevent the SQL attacks.

II. What is SQLIA?

SQLIA is a technique in which malicious SQL statements are supplied into a web form fields, in web applications, with an intention to hack into the database and dump the database contents to the attacker. The attacker influences the queries passed to the database.

The SQL Injection can be used using various methods. Say, for example, if we wish to log in to a website as an administrator and the URL is as follows:

http://www.ABC.com/tuts/login.php?username=sing&password=pass

Applying a small change in the above URL as follows:

http://www.ABC.com/tuts/login.php?username=dont&password=know'OR'1'='1

This change will allow the user to log in into the database without knowing the actual username and password and access all the data as per the privileges.

III. Classification of SQL Injection Attacks

The SQL Injection attacks can be performed using various techniques. Some of them are specified as follows:

A. Tautologies

Goal: The main goal of tautology-based attack is to inject code in conditional statements so that they are always evaluated as true. Using tautologies, the attacker wishes to either bypass authentication or insert injectable parameters or extract data from the database. Whenever a conditional statement is injected with cod so that the result is true, then its evaluation and result depends on the way that the query is evaluated inside the application. The exploiter mainly focuses on the WHERE clause to inject the code. The attack is only successful if and only if all the records are returned. It performs some action if at least one record is projected.

For example, the attacker submits "or 1=1—for the username field. So the query becomes:

SELECT Login_ID FROM users_ID WHERE user_ID=" or 1=1— AND pass_word=' '

The WHERE clause in the query is a tautology as it contains the injected code 1=1. Now, the database checks for each row if the tautology exists. Since, the WHERE clause is always true so, it will return all the rows from the database.[3]

B. Illegal/Logically Incorrect Queries

Goal: The main goal of the Illegal/Logically Incorrect Queries based SQL Attacks is to gather the information about the back end database of the Web Application.

The Illegal/Logically Incorrect Queries based SQL Attack is considered as the basis step for all the other techniques. The vulnerability leveraged by this technique is that the default error page is over-descriptive. In fact, the error message returned by the application reveals the vulnerable/injectable parameters to the attacker. When performing the SQL Attack, the attacker intends to inject such codes that can cause certain syntax, type conversion or logical errors. The syntactical error helps to identify the injectable parameters. The logical errors often reveal the names of the tables and columns that caused the error [6]

For example,

SELECT login_ID FROM users_ID WHERE login_ID=" AND pass_word=" AND ID= convert (int (SELECT top 1 user_ID FROM sysobjects WHERE ID='u'))

In the attack string, the injected select query attempts to extract the first user table (ID='u') from the database table (assume the application is using the Microsoft SQL Server, for which the metadata table is sysobjects). The query then tries to convert this table name into an integer. Since this is not a legal type alteration, the database throws an exception.

C. Union Query

Goal: The main goal is to trick the database to return the results from a table different to the one intended. This technique is mainly used to bypass authentication and extract data.

In union-query attacks, an attacker exploits the vulnerable parameter to change the data set returned for a given query. The attackers inject a statement of the form: UNION SELECT <rest of the injected query> to return the results from a different table. The result of this query is that the database returns a dataset which is union of both the queries [6].

For example, an attacker could inject the text 'UNION SELECT user_ID, password FROM user_info WHERE user_name='abc'— into the login field which produces the following query:

SELECT * FROM users_ID WHERE login_ID='' UNION SELECT pass_word FROM user_info WHERE user_ID='abc'— AND pass_word=''

Assuming that there is no login equal to '', the unique first query proceeds the null set while the second query returns data from the user_info table. In this case, the query will return the password for username abc. The database takes the result of both the queries, union them and displays the result. In some cases, the query returns the value of password along with the username.

D. Piggy-Backed Query

Goal: The attacker wishes to execute remote commands or add or modify data.

In this type of attack, the attacker does not intend to make changes in the original query but inject additional queries. This is different from other types because the attackers are not trying to modify the original proposed query; instead they are trying to include a new and distinct queries that 'piggy-back' on the original query. Thus, the database receives multiple SQL queries. The first is the proposed query by the application which is performed as normal; the succeeding ones are the injected queries, which are performed in addition to the first. If successful, the attackers can virtually insert any type of SQL command and have them executed with the original query. Vulnerability of this kind of attack is dependent on the kind of database [8].

For example, if the attacker inputs [';drop table users --] into the password field, the application generates the query:

SELECT Login_ID FROM users_ID WHERE login_ID='john' and pass_word=''; DROP TABLE users – 'AND ID=1223

After executing the first query, the database encounters the query delimiters (;) and execute the second query. The result of executing second query would result into dropping the table users, which would likely destroy valuable information. There could be other types of queries which could insert new users into the database or execute stored procedures. Only scanning of delimiter to prevent this kind of attack is not appropriate as not many databases require special characters to mark the end of a query.

E. Stored Procedure

Goal: The main goal of stored Procedure SQL Attack is to perform privilege escalation and try to execute the SQL procedures.

SQLIAs of this type try to execute stored procedures present in the database. These days, most vendors ship databases with a standard set of stored procedures to extend the functionality of the database and allow for interaction with the operating system. Therefore, once the attacker determines which backend database is in use, SQLIAs can be crafted to execute the stored procedures provided by that precise database, with procedures that interrelate with the operating system [1].

D. Alternate Encodings

Goal: The main goal is to escape detection.

In this attack, the injected text is modified so as to avoid detection by defensive coding practices and also many programmed prevention techniques. This type of attack is used in unification with other attacks. In other words, substitute encodings do not provide any single way to attack an application; they are simply an enabling technique that allows attackers to evade detection and prevention techniques and exploit vulnerabilities that might not otherwise be replaceable. These elusion techniques are often necessary because a common defensive coding practice is to scan for certain known "corrupt characters" such as single quotes and comment operators [3].

Example:

$login = mysql_query("SELECT * FROM user WHERE (User_ID= '" . mysql_real_escape_string($_POST['User_ID']) . "') and (pass_word = '" . mysql_real_escape_string($_POST['pass_word']) . "')");

To evade this defence, attackers have employed alternate methods of encoding their attack strings (e.g., using hexadecimal, ASCII, and Unicode character encoding). Common scanning and detection techniques do not try to evaluate all specially encoded strings, thus allowing these attacks to go undetected.

IV. Proposed Model for Prevent SQL Attacks

We considered the stages in the process of risk analysis as defined by Connolly et al. suitable as a base for choosing steps in our method, described below.

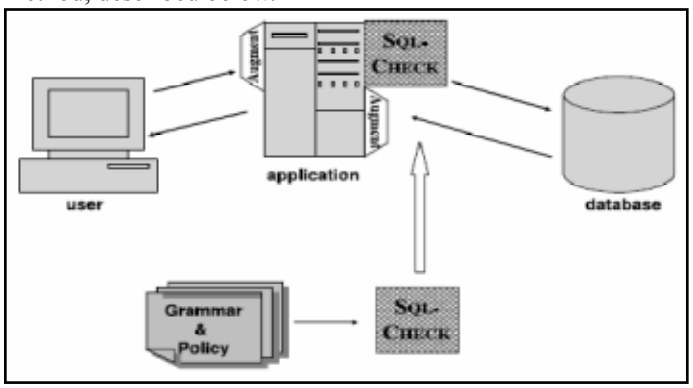

Fig. 1: System Architecture of SQLCHECK

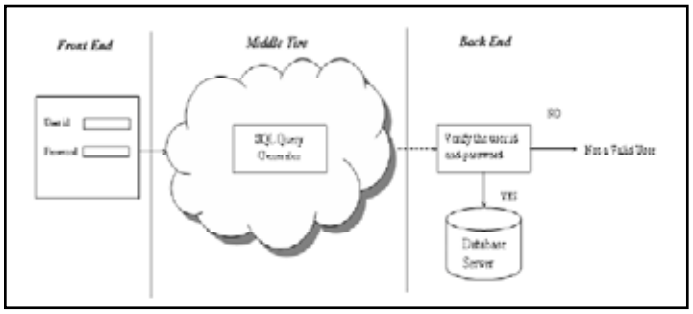

Fig. 2: Basic Model for Web Application

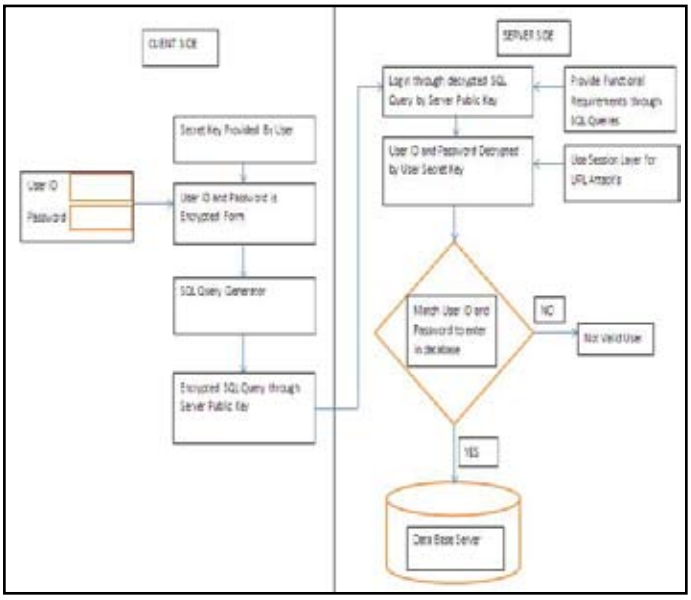

Fig. 3: Login and Verification Phase

A. In First Step

We will profile web applications by exploring their scope, implementations, architecture, inherent components and communication principles. We also identify assets that we believe many web applications share.

B. In Step Two

Basic principles of RDBMS systems as well as SQL will be explored. In addition, we identify security issues that we consider relevant.

C. In Step Third

Involve studying general aspects of Web security.

D. In Step Four

Constitutes our survey over SQL injection in which we classify the area in terms of general criteria. Along the way, we try to capture as many general aspects as we can: definitions, characteristics, conditions, susceptibilities, methods of attack and countermeasures. The different ways in which SQL injection-related attacks may be carried out on web applications are examined by different security services, e.g. availability, confidentiality and integrity.

E. In Step Five

We analyse our security model constructed in step four.

- User Input from predefined functions.
- Bind Variables Mechanism.
- Parameterized Statements.
- Input Validation.

V. Prevention Methodology

The tautological-based SQL Attacks are of the following two types:

Access through login page or user input by the use of 1'or'1' ='1

Access through URL

example:- http://www.bsnl.gov.in/userid=%2012%8 or 1=1

The prevention methodology proposed is as the following code: if(Login())$User_ID=stripslashes($User_ID);$User_ID=mysql_real_escape_string($User_ID); mysql_query("SELECT * FROM Registration WHERE name='{$ User_ID }'");

$sub = addcslashes(mysql_real_escape_string("%str"), "%_"); // $sub == \%str_ mysql_query("SELECT * FROM Report WHERE subject LIKE '{$sub}%'");

$login = mysql_query("SELECT * FROM user WHERE (User_ID= '" . mysql_real_escape_string($_POST['User_ID']) . "') and (pass_word = '" . mysql_real_escape_string($_POST['pass_word']) . "')");

VI. Conclusion

The proposed model gives new technique for preventing SQL Injection attack. Normally attacker tries to complicate the middle layer technology by reforming the SQL queries. SQLIPA uses user name, password and their hash values for authentication process. The SQL Injection Attack is tested on sample data of different condition of attack on web pages. SQL injection Prevention is covered in this paper by all the powerful techniques. We use PHP technology for testing SQL injection Attacks.

References

[1] William G.J.Halfond, Alessandro Orso,"AMNESIA:Analysis and Monitoring for Neutralizing SQL-Injection Attacks".

[2] X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, L. Tao, "A Static Analysis ramework for Detecting SQL Injection Vulnerabilities", OMPSAC 2007, pp. 87-96, 24-27 July 2007.

[3] S. Thomas, L. Williams, T. Xie,"On automated prepared statement generation to remove SQL injection vulnerabilities", Information and Software Technology 51, pp. 589–598, 2009.

[4] M. Ruse, T. Sarkar, S. Basu,"Analysis & Detection of SQL Injection Vulnerabilities via Automatic Test Case Generation of Programs", 10th Annual International Symposium on Applications and the Internet, pp. 31-37, 2010.

[5] Shaukat Ali, Azhar Rauf, Huma Javed,"SQLIPA: An authentication mechanism Against SQL Injection".

[6] Roichman, A., Gudes, E.,"Fine-grained Access Control to Web Databases", In: Proc. of 12th SACMAT Symposium, France, 2007.

[7] K. Amirtahmasebi, S. R. Jalalinia, S. Khadem,"A survey of SQLinjection defense mechanisms", Proc. Of ICITST 2009, pp. 1-8, 9-12, Nov. 2009.

[8] G. T. Buehrer, B. W. Weide, P. A. G. Sivilotti,"Using Parse Tree Validation to Prevent SQL Injection Attacks", In International Workshop on Software Engineering and Middleware (SEM), 2005. [II] F.Monticelli., PhD SQLPrevent thesis. University of British Columbia (UBC) Vancouver, Canada.2008.

[9] C. Gould, Z. Su, P. Devanbu,"JOBC Checker:A Static Analysis Tool for SQLlJOBC Applications", In Proceedings of the 26th International Conference on Software Engineering (lCSE 04) Formal Demos, pp. 697-698, 2004.

[10] Shaukat Ali, Azhar Rauf, Huma Javed,"SQLIPA: An Authentication Mechanism Against SQL Injection", European Journal of Scientific Research, Vol. 38, No. 4, 2009, pp. 604-611.

[11] William G.J.Halfond, Alessandro Orso,"A Classification of SQL Attacks and Countermeasures".