Survey on SQL Injection Attacks

Detection and Prevention of attacks

Akhil Gudavalli
Computer Science Department
Utah State University
Logan, Utah
akhilgudavalli@aggiemail.usu.edu

***Abstract*— SQL injection is an attack technique that exploits a security vulnerability occurring in the database layer of an application and a service. This is most often found within web pages with dynamic content. It is the one of the most commonly exposed attack on the internet. The paper aims to put SQL injection attacks into perspective by outlining some of the materials and researches that have been done in the past decade. The paper also aims to clarify some misconceptions about detection and prevention of SQL injection attacks and is useful for software developers and database administrators.**

Keywords—SQL injection attack, web application security, Attack, Authentication, database.

I. Introduction

SQL injection is a type of security exploit in which the attacker adds Structured Query Language (SQL) code to a Web form input box to gain access to resources or make changes to data. An SQL query is a request for some action to be performed on a database. Typically, on a Web form for user authentication, when a user enters their name and password into the text boxes provided for them, those values are inserted into a SELECT query. If the values entered are found as expected, the user is allowed access; if they aren't found, access is denied. However, most Web forms have no mechanisms in place to block input other than names and passwords. Unless such precautions are taken, an attacker can use the input boxes to send their own request to the database, which could allow them to download the entire database or interact with it in other illicit ways.

The reason that SQL injection and many other exploits, such as cross-site scripting, are possible is that security is not sufficiently emphasized in development. For the protection of the integrity of Web sites and applications, experts recommend simple precautions during development such as controlling the types and numbers of characters accepted by input boxes. In this paper, we are going to discuss about a few techniques for detection and prevention of SQL injection attacks.

II. Types of SQL Injection Attack

SQL injection attack is classified into many categories. The different types of SQL injection attacks are listed as follows:

A. *Tautologies*

Tautology attacks work through injecting code by one or more conditional SQL statement queries to make the SQL command evaluate as a true condition such as (1=1) or (- -). The most common use of this technique is to bypass authentication on web pages resulting in access to the database.

B. *Illegal/Logically Incorrect queries*

Logically Incorrect queries attack takes advantage of the error messages that are returned by the database for an incorrect query. These database error messages often contain useful information that allow attacker to find out the vulnerable parameter in an application and the database schema.

C. *Union queries*

By this technique, attackers join injected query to the safe query by the word UNION and then can get data about other tables from the application.

D. *Piggy-backed queries*

Piggy-backed queries is a type of attack that compromises a database using a query delimiter, such as ";", to inject additional query statements to the original query. In this method the first query is original whereas the subsequent queries are injected. This attack is very dangerous; attacker can use it to inject virtually any type of SQL command.

E. *Stored Procedures*

In this technique, attacker focuses on the stored procedures which are present in the database system. Stored procedures run directly by the database engine. it is a piece of code which is exploitable. Stored procedure gives true or false values for the authorized or unauthorized clients.

F. *Inference*

Using Inference attack enable the attacker changing the behavior of a database or application. This type of attack can be classified into two well-known techniques, Blind injection and timing attack.

- Blind Injection: This type of SQLIA occurs when programmers forget to hide an error message which causes the database application insecure, this error message help SQLIA to compromise the database through asking a series of logical questions through SQL statements.

- Timing Attacks: This type of attack permits an attacker collects information from a database by observing timing delays in the database's responses. This kind of attack used if condition statement to achieve a time delay purpose.

G. *Alternate Encodings*

In this technique, attackers modify the injection query by using alternate encoding, such as hexadecimal, ASCII, and Unicode. Because by this way they can escape from developer's filter which scan input queries for special known "bad character". For example, attacker use char (44) instead of single quote that is a bad character.

III. RELATED WORKS

Various research has been conducted in the detection and prevention of SQL injection attacks. In this section, I am going to list few techniques that were implemented before 2005 and how they fare against the different SQL injection attack types.

A. *AMNESIA*

This scheme identifies illegal queries before execution. Dynamically-generated queries are compared with the statically-built model using a runtime monitoring.

B. *SQLrand*

A strong random integer is inserted in the SQL keywords.

C. *SQLDOM*

A set of classes that are strongly-typed to database schema are used to generate SQL statements instead of string manipulation.

D. *WebSSARI*

Combination between static analysis and runtime monitoring.

E. *SQLGuard*

The parse trees of the SQL statement before and after user input are compared at a run time. The Web script must be modified.

F. *CANDID*

Programmer-intended query structures are guessed based upon evaluation runs over non-attacking candidate inputs.

G. *SQLIPA*

Using user name & password hash values, to improve security of authentication process.

H. *SQLCHECK*

A key is inserted at both beginning and end of user's input. Invalid syntactic forms are attacks. The key strength is a major issue.

I. *DIWeDa*

To detect various types of intrusions in Web Databases applications.

J. *Automated Apporoaches*

Static analysis to find Bugs and Web vulnerability scanning frameworks are implemented.

The different techniques to detect and prevent SQL injection attacks and their defenses against different types of SQL injection attacks.

Technique	Tautology	Incorrect Queries	Union Queries	Piggy Backed Queries	Stored Procedures	Inference	Alternate encodings
AMNESIA	YES	YES	YES	NO	YES	YES	YES
SQLrand	YES	NO	YES	NO	YES	YES	NO
SQLDOM	YES	YES	YES	NO	YES	YES	YES
WebSSARI	YES	YES	YES	YES	YES	YES	YES
SQLGuard	YES	YES	YES	NO	YES	YES	YES
CANDID	YES	NO	NO	NO	NO	NO	NO
SQLIPA	YES	NO	NO	NO	NO	NO	NO
SQLCHECK	YES	YES	YES	NO	YES	YES	YES
DIWeDa	NO	NO	NO	NO	NO	YES	NO
Automated	YES	YES	YES	NO	YES	YES	NO

.

IV. RECENT RESEARCH(PAST 5-6 YEARS)

In this Section, we are going to discuss about the recent research that is conducted in the recent years. There are a few other techniques implemented to detect and prevent SQL injection attacks. The different techniques and their implementations are discussed below.

A. *Query Tokenization method*

It is a method that detects a single quote, space or double dashes. All strings separated by single quote, space, or double dashes constitute a token. All the tokens grouped together make an array for which a token is an index. The tokenization is done for both the original query and the query with injection. The obtained arrays are compared and if their lengths differ, an injection is detected otherwise there is no injection. As a result, the access to data can be granted or denied once the lengths of the arrays are the same or different respectively.

For Example, Consider the below given query injection.

Original query - Select*from student where userid=100;
Injected query - Select*from student where userid=100 or 1=1;

The Original query is tokenized as follows:

0	1	2	3
Select*from	student	where	userid=100

The Injected query is tokenized as follows:

0	1	2	3	4	5
Select*from	student	where	userid=100	or	1=1

Comparing the two arrays described in the tables above, we observe that their lengths ae different. The arrays have indexes ranging from 0 to 3 and 0 to 5 respectively. Therefore, the lengths of the arrays are 4 and 6 respectively. Thus, lengths are different and according to query tokenization method, it is an SQL injection attack.

This method is called QueryParser method. After identifying the injection attack, the QueryParser method runs the injected query until the length of the query is equal to the original query.

In our example, the length of the original query is 4. So, the Injected query is taken from index 0 to 3. So, the query that is executed is Select* from student where userid=100. Without stopping the flow of execution, the QueryParser method identifies the attack and prevents the attack. This method is used since to make SQL injection attack, an attacker should use a space, double quotes or double dashes in his input.

B. *Removing Query Attributes*

It is a novel method to detect SQL injection attacks based on static and dynamic analysis. This method removes the attribute values of SQL queries at runtime (dynamic method) and compares them with the SQL queries analyzed in advance (static method).

The function f mentioned below removes the query attributes in a given SQL query and outputs the query after the removal of query attributes.

```
Algorithm f(One SQL query)
Enumerate Quotation_Status = { Quot_Start, Quot_End}
Input String=One SQL query;
;Output_String=Null;
Current_Quotation_State=Quot_End;

Do while( not empty of Input String)
{
    Char=Get_Token(Input_String);
    If Char is a quotation character
    {
        Add Char to Output_String;
        If the preceding character is not back slash
            Toggle Current_Quotation_State;
    }
    Else
    {
        If Current_Status is Quota_End than
        {
            Add Char to Output_String;
        }
        Else
        {
            If the preceding character is \ (back slash) then
            Add Char to Output_String;
        }
    }
}

Return Output_String;
```

Let's take the query set as follows.

FQ: Select* from user where id='$id' and password='$password'

DQ1: Select* from user where id='admin' and password='1234'

DQ2: Select* from user where id='1' or '1=1'—' and password='1234'

Static method (Initial Query after removal of query attributes)
FDQ = f(FQ) = Select* from user where id='' and password=''.

Dynamic (Injected Queries after removal of query attributes)
DDQ1 = f(DQ1) = Select* from user where id='' and password=''.

DDQ2 = f(DQ2) = Select* from user where id='' or ''—''1234'.

Now, as per the algorithm we have the SQL queries at runtime and SQL query of the original query after the removal of query attributes. We must compare them to identify the attack. The Comparison is done using the Exclusive OR operator.

If FDQ ⊕ DDQ = 0 then the query is not an attack
Else the query is an SQL injection attack.

In our case,
FDQ ⊕ DDQ1 = 0 and FDQ ⊕ DDQ2 != 0.

So, we can say that DDQ1 is a normal query and DDQ2 is SQL injection attack. This method is used for detection of the SQL injection attacks.

This method is used to detect all the types of SQL injection attacks and hence it is very effective.

C. *TransSQL*

TransSQL is a method where a SQL database is automatically duplicated and stored in LDAP form. Then TransSQL monitors connections between the protected Web Application and the related SQL Database. Whenever a SQL request is sent to the SQL database, TransSQL creates a LDAP-equivalent request of the SQL request and sends the LDAP request to the LDAP database. TransSQL determines whether the SQL request is a SQL injection request or not by comparing the results from both the SQL database and the LDAP database.

TransSQL consists of two phases, preprocessing phase and runtime phase.

Preprocessing phase	Runtime phase
TransSQL retrieves information from a SQL database to produce corresponding LDAP schema and LDIF (LDAP Data Interchange Format) file. Later, TransSQL builds a LDAP equivalent database by importing LDAP schema and LDIF data into the LDAP database	TransSQL intercepts every SQL re-quest between the protected SQL database and a web application and translates it into LDAP equivalent request. A SQL request is a SQL injection request if the results re-turning from both databases have inconsistent responses. After detecting a SQL injection request, the result from SQL database would be replaced with a null result and send it to the web application.

An example of LDAP equivalent request is shown below:

<u>SQL query</u>
Select * from users where Name = 'john';

<u>LDAP equivalent request</u>
Select * from ou = users, dc = example, dc = com where Name = 'john'

Ou, dc, example, com belongs to LDAP database generated from the original database.

D. *Adaptive Intelligent Intrusion Detector Agent*

This method suggests a hybrid approach based on Adaptive Intelligent Intrusion Detection (AIIDA-SQL) for detection of SQL Injection attacks. AIIDA-SQL combines the advantages of Case-Based Reasoning (CBR) systems, such as learning and adaptation, with the predictive capabilities of a combination of Artificial Neural Network (ANN) and Support Vector Machine (SVM). Through these mechanisms, we take advantage of both the strategies in accurately classifying the SQL queries. To classify SQL queries this model utilized a virtualization mechanism which combines clustering techniques and unsupervised neural models to reduce dimensionality of the data.

In terms of CBR, a case is composed of elements of the analyzed SQL query described as follows:

- Problem: describes the initial information available for generating a plan. The problem description consists of: case identification, user session and SQL query elements.
- Solution: states the action carried out to solve the problem (the applied prediction models).
- Final State: describes the state achieved after applying the solution

Evaluation of AIIDA-SQL:

The different classifiers were applied to 705 previously classified queries (437 legal queries and 268 attacks). The classifiers include Bayesian Network, Naive Bayes, AdaBoost M1, Bagging, DecisionStump, J48, JRIP, LMT, Logistic, LogitBoost, MultiBoosting AdaBoost, OneR, SMO, and Stacking. We also applied the AIIDA-SQL model on the classified queries data. AIIDA-SQL gave the best results compared to any of the classifiers on which the test is conducted on. The performance of the classifies on 705 queries is mentioned in the below figure.

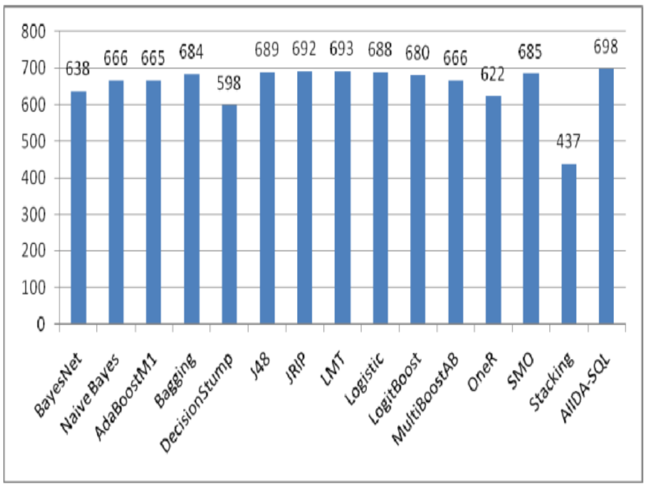

Figure 2. Total number of hits for the different classifiers.

The Accuracy of AIIDA-SQL came out to be 99% as it uses a combination of ANN and SVM which are different artificial intelligence paradigms.

V. Analysis

The below table is the analysis of key elements in the methods for detection and prevention of SQL injection attacks.

Method	Modify Code	Detection	Prevention	Key Elements
Query Tokenization	No	Yes	Yes	QueryParser method
Removal of Query Attributes	Yes	Yes	No	Combination of Static and dynamic analysis.
TransSQL	Yes	Yes	Yes	LDAP database and LDAP equivalent representation of queries
AIIDA-SQL	No	No	No	CBR engine, ANN and SVM

AIIDA-SQL can be considered as an accurate measure for detection of the attacks, but it needs to have a large database of correctly classified queries.

Removal of Query attributes can detect all the types of attacks. Hence, it is more effective than the techniques that were implemented in the last decade in the detection of SQL injection attacks.

Modification of the original queries is not necessary in Query Tokenization and AIIDA-SQL.

VI. CONCLUSION

This paper is a survey of the current research related to SQL injection attacks. The current research trend in this area is applying data science and machine learning techniques to detect and prevent SQL injection attacks. I feel that the attacks can be detected more accurately if we can apply neural networks in prediction of the query whether it is an attack or not.

Acknowledgment

I would like to thank my professor, Dr. Chad Mano, for the patient guidance, encouragement and advice he has provided throughout my time as his student. I have been extremely lucky to have a professor who responded to my questions and queries so promptly. I would also like to thank all the people who helped me complete the survey paper.

References

[1] N. Lambert and K. S. Lin,” Use of Query Tokenization to detect and prevent SQL Injection Attacks,” in Proc. 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), July 2010.

[2] I. Lee, S. Jeong, S. Yeo, and J. Moon, “A novel method for SQL injection attack detection based on removing SQL query attribute values “ Mathematical and Computing Modeling Journal, 2011.

[3] Kai-Xiang Zhang, Chia-Jun Lin, Shih-Jen Chen, Yanling Hwang, Hao-Lun Huang, Fu-Hau Hsu, “ TransSQL: A Translation and Validation-Based Solution for SQL-injection Attacks”, Robot Vision and Signal Processing, First Internatioonal Conference, 2011.

[4] C. Pinzon, J. F. Paz, J. Bajo, A. Herrero, and E.Corchado, “AIIDA-SQL: An Adaptive Intelligent Intrusion Detector Agent for Detecting SQL Injection Attacks,” in Proc. 10th International Conference on Hybrid Intellignt Systems (HIS), IEEE Press, 2010

[5] Atefeh Tajpour, Mohammad JorJor zade Shooshtari, “ Evaluation of SQL Injection Detection and Prevention Techniques”, Computational Intelligence, Communication Systems and Networks (CICSyN), Second International Conference, 2010.