International Journal of Advanced Technology in Engineering and Science
Vol. No.5, Issue No. 04, April 2017
www.ijates.com
ijates
ISSN 2348 - 7550

EVALUATING SQL INJECTION DETECTION AND PREVENTION APPROACHES

Shyamsundar Pushpad, Chirag Juneja, Anmol Chawla

Department of Computer Applications National Institute of Technology
Kurukshetra, Haryana (India)

ABSTRACT

Data is considered to be the most indispensable asset for any individual, and organization, in particular. Any kind of threat to the data implies the whole organization is in jeopardy. Therefore database security is the most critical issue for an organization. Especially, the web applications dealing with personal information of users such as applications for banking, shopping etc. are very much dependent upon the database. So the confidentiality of this information is of the utmost priority. But as the number of applications are increasing, the number of attacks are also increasing with the same pace. Web applications are extremely vulnerable to attacks. Of all the attacks, SQL Injection Attack (SQLIA) is considered to be one of the top threats to the databases. Through the user interaction interface, the attacker injects malicious code to get access to the database. We have put light to some of the attack methods of SQLIA and compared some of the efficient methods or approaches developed for those attacks.

Keywords : Approach, Detection, Prevention, SQL injection attacks.

I. INTRODUCTION

The major issue in the protection of databases is the improper design and development of the database module and the lack of validation controls on the user related data. Due to this, the attackers have an edge to put malicious code in the interaction interface pretending to be a normal user. In this way, a malicious user can get unrestricted access to the databases. Usually, the user gets some fields from which he/she provides information for the database to fetch some other information or to access some kind of account. The information put up by the user goes to the server in the form of SQL query. The attacker uses this interface and adds up some code in a smart way to either get information or to cause damage to the database. Typically, the database's structure is targeted as the attacker gets most of the information related to the data from the schema of the database, for instance, the table name, number of fields, data type of the fields etc. The paper is constructed in the following mentioned manner. Section 2 contains a JSP code snippet for the login process of a web application. In section 3, we have discussed about various types of SQL injection attacks on the web application. Section 4 shows different techniques those are developed for detecting and preventing different attacks. In section 5 we have compared the discussed approaches on the basis of the attacks that they can detect and prevent.

II. EXAMPLE APPLICATION

Before talking about the various SQL injection types, we instigate an example application that have chances

of being attacked by an SQL injection. We use this example in the next section to give SQL injection examples.

```
1. String uname,password,query
2.  uname = getParameter("UserName");
3. password = getParameter("Password");
4. Connection conn.createConnection("MyDataBase");
5. query = "SELECT * FROM users WHERE uname='" +
   UserName + "' AND password='" + Password;
6. ResultSet result = conn.executeQuery(query);
7. if (result!=NULL)
8. displayAccounts(result);
9. else
10. displayAuthFailed();
```

Snippet of servlet implementation

The code in figure 1 is simple and can be prevented from attacks using coding fix but we use this simple example to demonstrate different types of attacks. The code snippet in figure 1 implements the login process for an application. The code in the example uses the input parameters UserName, Password to dynamically build an SQL query and submit it to a database. For example, if a user submits UserName, Password "yashi", "258741369" the application dynamically builds and submits the query:
SELECT * FROM users WHERE uname='yashi' AND password='258741369'
If the UserName, Password matches the corresponding entry in the database, yashi's account information is returned and then displayed by function displayAccounts(). If there is no match in the database, function displayAuthFailed() displays an appropriate error message.

III. SQL INJECTION TYPES

In this section, we provide description of various kinds of SQL injection attacks. Generally attacks are not performed in isolation. On the basis of the goals, attackers perform different types of attacks together or one after another. There are various types of attacks and attackers can do variations on each type of attacks. We present the type of attacks that attackers often use.

3.1 Tautology

A tautology is a logical statement that is always evaluated to true. The main aim of attack is to inject code in conditional statements so that they always evaluate to true. Tautology is injected in the query's where condition. Making the conditional expression into a tautology causes all of the rows in the database table targeted by the query to be returned.

Example: normal query to login for User table:

SELECT * FROM User WHERE UserName='yashi' and Password='123456789'

Example of a malicious query to login for User table:

SELECT * FROM User WHERE UserName='' or 1=1 -- and Password=' '

The injected code 1=1 in condition converts whole WHERE clause into a tautology and force the table to return all information about the users.

3.2 End Of Line Comment

In many SQL languages the end of line comment (--) is used to disable part after -- from being executed. Double hyphen makes everything after that till the end of line, a part of comment. It will not be executed.

Example:

SELECT * FROM User WHERE UserName='yashi' -- and Password=' '

The above query will disable the Password field and make it comment by writing – before it.

3.3 Illegal/Logically Incorrect Queries

Attackers inputs the illegal/logically incorrect queries so that SQL database servers return error messages. These errors contain a plenty of utile information. This attack helps an attacker to collect significant information about the name, type, version, data type and the structure of the back-end database of web application. This attack is mostly used as preliminary for other attack techniques. Additional error information is provided to help the programmers in debugging, further helps the attackers also to get information about the schema further helps the attackers also to get information about the schema.

3.4 Union Queries

Union keyword is used to combine two or more queries. Attackers take advantage of this characteristic to inject second query with the original query. They do this by adding a statement UNION SELECT <remaining injected query>. They can use second query to retrieve the desired information which can be from another table also because they have complete control on the second query. The result of this attack is the union of the results of original query and all the queries attached with it using UNION keyword.

Example:

SELECT * FROM User WHERE UserName ='' and Password='' UNION SELECT * from card where

Account =51512668745 -- AND pass=''

Most probably there will be no user with username='' so first query returns null but the injected query returns data from the card table. Here database would return all the columns from card table for account=51512668745. Database combine the result of both the queries and return it to the application.

3.5 Piggy-Backed Queries

In this type of attack, an attacker injects another query into the original query. In this, attackers don't have intention to modify the original query; instead, they are trying to inject a new and different query at the end of original query using semicolon. Database will executes both the queries one after the other. This is very harmful attack because if attacker get success in injecting the query he/she can add or delete the tables.

Example: If the attacker inputs; drop table User into the Password field, the application generates the query: SELECT * FROM User WHERE UserName ='yashi' and Password='82354668'; drop table user;
After executing first query, the database would execute the injected query which will drop the table User and delete the valuable information.

3.6 Alternate Encoding

In this type of attack, attackers manipulate the query using special characters like Unicode, ASCII code etc. to evade from the detection by SQL attack prevention techniques. This style of attack is known as alternate encodings. This attack is done with another type of attack that is it does not provide a way to attack, it is just helping the attackers to evade the detection and prevention techniques.
Example:
Character (756e696f6e) in hexadecimal is represents the UNION command in ASCII encoding which is one of the SQLIAs techniques so the attackers in this technique will have the ability to hide themselves from the defending mechanisms. [1]

3.7 Stored Procedures

SQLIAs of this type try to execute stored procedures present in the database [2]. Today, most database vendors store databases with a standard set of inbuilt procedures that increase the functionality of the database and allow interaction with the operating system. Therefore, once an attacker able to know which backend database is in use, SQLIAs can be designed to execute stored procedures provided by that specific database, including procedures that interact with the operating system.

VI. SQLIA Detection and Prevention Approaches

4.1 Positive Tainting

This approach is based on dynamic tainting. This method, proposed by Livshits and Lam [3], deals with trusted data, unlike the conventional tainting method which involves untrusted data. This creates a great deal of difference, in the way that, it helps in addressing the inefficiencies caused by incompleteness in identifying pertinent data to be marked. This approach ensures that any kind of injection attack cannot go unnoticed. Though, some false positives may occur, but that too can be found out and eliminated.

4.2 SQLrand

This technique was proposed by Boyd and Keromytis [4]. In this approach, the developers make use of randomized instruction set instead of normal SQL keywords. Here, a proxy is used to append key to the SQL keyword(s). Then the queries to the database are intercepted by a proxy filter that de-randomizes the keywords and converts them into proper SQL queries. As the key is not known to the attacker, so the injected code will be recognized as undefined expressions and keywords. So exception(s) will get created and hence, the injected query is not sent to the database.

The disadvantage of this approach is that if the key is revealed, the attacker will be able to construct successful malicious code and inject it in the database.

4.3 WebSSARI

This tool, developed by Y. Huang, F. Yu, C. Hang, C. H. Tsai, D. T. Lee, and S. Y. Kuo [5], helps in detecting errors corresponding to input-validation by analyzing the information flow. Here, the taint flows' checking is done against some preconditions for sensitive functions by the help of a lattice based static analysis algorithm. This tool inserts runtime guards, by its own, in the probable insecure segments of the code. Also, the overhead involved is less, as the data from static analysis is used to minimize the number of insertions of the guards.

The drawback of this approach is that it presumes that sufficient preconditions can be precisely expressed for the sensitive functions.

4.4 JDBC-Checker

It was proposed by C. Gould, Z. Su, and P. Devanbu [6]. Though, this technique was not developed for preventing SQL injection attacks, but can be used to check the type mismatching(s) in dynamically generated query string(s). One of the basic cause of SQL injection attack, that is, improper type checking of input by user can be detected by this technique. A limitation of this technique is that it would not be able to detect other general forms of the injection attacks.

4.5 Amnesia

AMNESIA, a model-based tool developed by Halfond and Orso, integrates static as well as dynamic analysis in preventing the injection attack(s) [7]. In the static analysis, this tool constructs a set of legally correct and permissible queries that the application may generate. Then, in the dynamic analysis part, runtime scanning is done to check all the queries actually generated by the application against the statically built set of licit queries. All the illegal queries are then spotted and hence blocked from accessing the database.

However, one downside of this technique is that the prevention of injection attacks are mostly reliable on the correctness of the static analysis part. There are methods to develop some queries those may go unnoticeable by this step and perform the injection attack

4.6 SQL DOM

This approach, proposed by R. McClure and I. Kruger, provides a secure way to use database(s) by making use of database query encapsulation which makes it more reliable. This method uses type-checked API instead of the conventional way of concatenation of strings. In this way the query building becomes more systematic. It is

a kind of defensive approach, in which effective coding methodologies like input filtering and detailed type-checking are applied which disallows the attacker to use malicious coding practices. Though this technique is very effective, it does not provide an effective way to curb attacks for the available systems. Rather it requires the developers to grasp and make use of a new programming approach.

V. ANALYSING SQLIA DETECTION AND PREVENTION APPROACHES

We have assessed few SQL injection detection and prevention techniques those are capable of handling almost all of the attacks. We have not focused on the implementation part of these techniques, due to the fact that either the technique was not implemented or the implementation was not available. So, we have analyzed the techniques on the basis of the various attacks that they can detect or prevent. Approaches like SQlrand, those used, on the whole, the concept of a secret key, though did well with avoiding the injection attacks, but still attacker has an edge to get successful. If the secret key is exposed to the attacker, the whole algorithm fails, making the database vulnerable. Other techniques those proved to be very efficient were WebSSARI and AMNESIA. These approaches used two pass algorithms, that is, in the first pass they analyzed the user code statically and in the second phase, they did dynamic analysis. However, both the techniques' efficiency mostly relied upon the static part. If the static part of approach missed some key factors in analyzing the injection query, the dynamic part will fail to prevent those malicious queries from accessing the database. Some approaches were capable of handling the attacks partially, like the JDBC-Checker. This is due to the fact that this technique detects only the type-related errors and there are a lot of vulnerabilities.

Attack types / Techniques	Tautology	Logically incorrect query	Piggy-backed query	Union query	Alternate encoding	End of line comment	Stored procedures
AMNESIA	✓	✓	✓	✓	×	✓	×
SQL rand	✓	×	✓	✓	×	✓	×
Positive tainting	✓	✓	✓	✓	✓	✓	✓
SQl DOM	✓	✓	✓	✓	✓	✓	×
JDBC-CHECKER	p	p	p	p	p	p	p
WebSSARI	✓	✓	✓	✓	✓	✓	✓

Fig. 2. Comparison of SQL Injection Detection and Prevention Techniques According to the Attack Types

Figure 2 contains a table, where we have shown the capabilities of different techniques and approaches. It basically exhibits which all attacks the discussed techniques can deal with. A few symbols are used in the table like '✓', which represents that a particular technique is capable of detecting and preventing a particular injection attack. Whereas a '×' depicts that a particular approach is not proficient for a particular SQL injection attack. Another symbol used here is 'p', which is for the special case of JDBC-Checker. This technique proves to be inefficient in dealing with the detection and prevention of the SQL injection attacks, totally. It shows that the JDBC-Checker handles the all the attacks partially. For instance, the technique SQLrand eradicates all other

injection attacks like tautology, piggy-backed query injection except for logically incorrect query, alternate encodings and stored procedures. Almost all the approaches are capable of eradicating most of the SQL injection attacks. But still the attacker, if studies deeply about the behavior of the working of database, can be able to formulate smart SQL queries that may prove to be harmful to the organization handling the database.

IV. CONCLUSION

Databases will always remain the cornerstone for almost all corporations. This is also the weak spot as the attackers keep eye on the databases to get unrestricted access and fetch as much information as they can. Hence, the security of databases will always be the prime affair for the organizations. Studying various techniques developed for paralyzing the attacks, will help in detecting the problems prevailing in the approaches. Also, investigating various techniques helps in proper decision making as to which approach should be chosen for a particular web application. Though the contemporary approaches have performed well in different circumstances, there is, still a lot to improve, to eliminate the attacks comprehensively.

REFERENCES

[1] Al-khashab, E., F.S. Al-Anzi and A.A. Salman, 2011. PSIAQOP: Preventing SQL injection attacks based on query optimization process. Proceddings of the 2nd Kuwait Conference on E-Services and E-Systems, April 5-7, 2011, Kuwait, USA.

[2] HALFOND, W. G. J., ORSO, A., AND MANOLIOS, P. 2006. Using positive tainting and syntax-aware evaluation to counter SQL injection attacks. In Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT'06). ACM, New York, 175–185.

[3] V. B. Livshits and M. S. Lam. Finding Security Errors in Java Programs with Static Analysis. In Proceedings of the 14th Usenix Security Symposium, pp 271–286, Aug. 2005.

[4] S. W. Boyd and A. D. Keromytis. SQLrand: Preventing SQL injection attacks. In Proceedings of the 2nd Applied Cryptography and Network Security (ACNS) Conference, pages 292–302, June 2004.

[5] 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 Proceedings of the 12th International World Wide Web Conference (WWW 04), May 2004.

[6] C. Gould, Z. Su, and P. Devanbu. JDBC Checker: A Static Analysis Tool for SQL/JDBC Applications. In Proceedings of the 26th International Conference on Software Engineering (ICSE 2004) –Formal Demos, pages 697–698, 2004.

[7] Halfond, W.G.J. and A. Orso, "AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks." in Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering2005, ACM: Long Beach, CA, USA. p. 174-183.

[8] R. McClure and I. Kruger. SQL DOM: Compile Time Checking of Dynamic SQL Statements. In Proceedings of the 27th International Conference on Software Engineering (ICSE 2005), pages 88–96, 2005