Intrusion Detection & Prevention System: SQL Injection Attacks

[1]Amrita Bhat, [2]Priyanka Mumbarkar

Sardar Patel Institute of Technology
Mumbai, India

Abstract— **Web-based applications are emerging on a larger scale in today's world in various areas covering e-commerce, banking, finance & many more. This increasing demand of web-based applications has also made the Internet as a potential target for different forms of attack thus raising awareness of web application administrators of the need to effectively protect their web applications from being attacked by malicious users. One such attack most commonly used is Sql Injection attack in which the inputs are modified resulting into illegal queries to a database which has become one of the most serious threats to the web applications.**

In this paper, we propose a technique that uses the intrusion detection system combining the static & dynamic phases so as to validate the user input.

Keywords: **SQL injection attack, SQL query, a combined dynamic and static method, Web application**

I. INTRODUCTION

In today's world, internet is playing an important role since it is one huge interconnected network which plays a vital role in our day to day lives. Web applications are widely used in financial, commercial and across multiple business areas. These web applications work by taking the input from the user, processing them & providing the resultant output for the same. Since the input data obtained by the user can contain malicious data turning into an attack. Thus, this data must be validated before processing so as to confirm that the data is not malicious and is obtained from the valid user. Sometimes, it might happen that there are no strong validation checks performed which can affect the security of the data. Thus, the lack of strong types and invalidated database access control can permit attacks to exploit the vulnerabilities to launch particular attacks. These attacks might cause a serious leak of sensitive data & might also lead to file corruption.

Many research and studying is undertaken by various researchers so as to come up with various detecting and prevention of these attacks and most preferred techniques are web framework, static analysis, dynamic analysis, combined static and dynamic analysis, and machine learning techniques.

The static analysis method [1] involves the inspection of computer code without actually executing the program. The main idea behind static analysis is to identify software defects during the development phase. Static analysis is applied to find potential violations matching a vulnerability pattern, so it is more effective than the filtering method. But attacks having the correct parameter types cannot be detected. The main limitation of the method is that it cannot detect SQL injection attacks patterns that are not known beforehand, and explicitly described in the specifications. The dynamic analysis [2] can be seen as the next logical step of static analysis. It inspects the behavior of a running system and does not require access to the internals of the system; however this method is not able to detect all SQL injection attacks. A combined static and dynamic analysis method [3] can compensate for the weaknesses of each method and is highly proficient in detecting SQL injection attacks. The combined usage of a method of static and dynamic analysis is very complicated.

II. SQL INJECTION

SQL injection is a code injection technique used to attack database systems through vulnerable web applications [4]. The technique not only allows the attacker to steal the entire content of relational databases but also to make arbitrary changes to both the database schema and the contents. Relational database server products have no mechanism to deal with SQL injection as the problem is rooted not in the database server itself but in vulnerable applications with excessive privileges granted to users. In many cases, a victim of an SQL injection attack does not even know that information is compromised until long after the attack has passed. Perhaps he may receive an angry e-mail from a customer whose credit card number may have been stolen or from the attacker himself seeking some form of blackmail. In many instances, victims are not aware that their confidential / critical data has been stolen. While the details of SQL injection attacks vary among implementations of relational database systems (RDBMS), both commercial and open source RDBMSs are potentially susceptible to attack.

Most SQL injection attacks are executed through an application that takes inputs supplied by the user for query parameters. The attacker supplies a carefully crafted string to form a new query whose results are very different from what the application developer intended. For example, consider a script on a web site that takes a search parameter like Zip code to return selected results from a database. A very simple attack may be possible by providing something, like "1 OR 1=1" in the text field, which causes the SQL server to return true and returns all records from a particular table. An attacker can often gain access to anything available with the

script's privileges, which is often full access to one or more databases.

While SQL injection attacks could be executed against any application, web applications are the most commonly vulnerable. The attacker can easily explore a site for vulnerabilities without being caught or having to work through sophisticated network intrusion techniques as most prospective targets leave their web site applications wide open. Fire- walls and traditional network intrusion detection systems are not useful against SQL injection. Some signature-based detection systems have been designed for web servers to protect vulnerable scripts from malicious input code. However, these signature-based systems are inherently susceptible to evasion methods that take advantage of the expressiveness of the SQL language or alternate character encodings. Remarkably, writing scripts that are not vulnerable to SQL injection is as simple as passing all user-provided text through a string. As past experience has shown, vulnerable scripts are found in most places and SQL injection affects every database on every platform. Attacks can be used to gain information disclosure, to bypass authentication mechanisms, to modify the database, and, in some cases, to execute arbitrary code on the database server itself.

A. AND/OR ATTACK

Web programmers often take string values entered by an Internet user on a form that represents user names and passwords and place them directly into the SQL statement to be run against a database. A simple example of SQL statement that may be used is as follows:

SELECT username, password FROM Login
WHERE username = 'username' AND password = 'password';

In this example, the values *username* and *password* are obtained from the form submitted via the web application. This username and password obtained from the form are matched with the username and password in the Login table. If any rows are returned, the user is authenticated.
However, if the web programmer does not validate these values, a hacker may instead pass arbitrary values that the programmer did not originally anticipate. One such attack is the basic attack that involves the AND or OR logic in the SQL predicate. The hacker can specify a valid username such as "Jen Swift" and then specify the password as "' OR '1'='1" in the form. The final test SQL query that uses these values will be:

SELECT username, password FROM Login WHERE username = 'Jen Swift' AND password = '' OR '1'='1';

Provided that "Jen Swift" is a valid user, the user will be considered an authenticated user and this will allow the hacker to log-in and proceed as "Jen Swift", because even though the password string is not empty, '1'='1' is a valid predicate that will always return TRUE. Thus, lets the hacker get the complete access to the victim's information.

Here, the hacker simply needs to do several "probing" tests and check the messages returned to see if this is indeed the case. If the attack is not successful, the attacker simply moves on and tries another method. If it is successful, then the DBMS returns the username and corresponding password; the hacker now has unauthorized access to the database through that username.

B. COMMON ATTACK

As mentioned earlier, SQL allows inline commenting within the SQL "code". This allows two variations of SQL-I comments attacks. One simple technique is assigning username to be a valid username followed by comment characters. For example, we assign username = "admin' --". Then our SQL test query may look like this:

SELECT username, password FROM Login
WHERE username = 'admin' --' AND password = 'password';

Everything after the "--" in the WHERE clause will be ignored, so the hacker can now login as "admin".
This is a method of using comments as a way of ignoring the rest of the query. [5] The variation of the comments attack is using comments as a way of obfuscating the signature of any SQL-I attack to avoid detection. Therefore, the use of C-style comments "/*" and "*/" can be combined with any of the previously discussed attacks as a way of attempting to circumvent signature-based detection. For example, if an application searches a string passed from a form for the UNION keyword to attempt to catch UNION injection attacks (discussed in more detail in a subsequent section), an attacker may choose to use comments to conceal this. For example, instead of using "UNION ALL", the attacker may instead use 'UNION /**/ALL' or 'UN/**/ION A/**/LL'. Both of these are synonymous with "UNION ALL" in the context of an SQL statement. In addition to breaking up keywords, comments may be used in place of spaces. A system using signature-based detection may miss keywords and SQL-I patterns if it is not careful to also consider SQL-I Comments attacks as well.

C. STRING CONCATINATION

Attack SQL has an option to concatenate separate strings or characters to form complete strings. This is accomplished using + or "double pipe" (||), or the function CONCAT (such as in MySQL). These operations can be used to create a variation of the UNION Injection attack by obfuscating the UNION keyword in a string concatenation operation. For example, an attacker may use 'UNI' + 'ON A' + 'LL'in place of "UNION ALL" if he suspects the system looks for the UNION ALL keyword.[5] Another use of string concatenation in an attack is when the attacker suspects the system searches for single quotes ('). Then the attack may choose to use the CHAR() function in conjunction with the string concatenation to issue characters indirectly without using any single quotes. For example, an attack may use CONCAT(CHAR(65),CHAR(68),CHAR(77),CHAR(73),CHAR(78)) to represent 'ADMIN' so that the system will not find a single quote if it was looking for them.

D. UNION INJECTION ATTACK

The UNION Injection attack may be the most dangerous, since it allows the attacker to return records from another table. For example, an attack may modify the SQL query statement that selects from the user authentication table to select another table such as the accounts table.

SELECT username, password FROM userAuthTable UNION ALL SELECT accountNum, balance FROM Accounts;

The use of UNION ALL in this attack allows the attacker access to tables that the SQL query statement was not originally designed for. The resulting rows selected from both tables will appear on the resulting page. The trickiness in this attack lies in the fact that the columns selected from the second table must be compatible in number (the same number of columns as the original table must be selected) and type. When trying to guess the correct number of columns, the attack may simply keep trying to use different number of columns in each attempt until he finds the right number. To match the type, the attack may try to try different types until he stumbles upon the right one or he may simply choose to use NULL instead. The IDPS system discussed later does not return any messages such as response or HTTP status codes and limits internal information being broadcast externally as much as possible.

E. HEXADECIMAL/DECIMAL/BINARY VARIATION ATTACK

Attackers can further try to take advantage of the diversity of the SQL language by using hexadecimal or decimal representations of the keywords instead of the regular strings and characters of the injection text.
For example, instead of using the traditional SQL-I Attack text

1 UNION
SELECT ALL
FROM WHERE

an attacker may substitute this with
1 UNION SELECT ALL FROM WHERE
to attempt to avoid detection by signature-based detection engines. [6] To attempt to avoid detection by signature-based detection engines. [6]
The system that does not look for hexadecimal or decimal characters will be susceptible to this variation of the SQL-I attack.

F. WHITE SPACE MANIPULATION ATTACK

Signature-based detection is an effective way of detecting SQL-I attacks. Modern systems have the capacity to detect a varying number of white spaces around the injection code, some only detect one or more spaces; they may overlook patterns where there are no spaces in between. For example, the SQL-I pattern ' or 'a' <> 'b can be re-written as 'or'a'<>'b 16 containing no spaces in between. A DBMS SQL parser will be able to handle a variable around all of white space characters or keywords. If a signature-based detection method only takes into account the first pattern, it will completely overlook the second one. In additional to the standard space character, white space characters also include the tab, carriage return, and line feed characters.[6] To properly implement signature-based detection, the system must be able to handle white space characters.

III. INTRUSION DETECTION & PREVENTION SYSTEM (IDPS) DESIGN

The system discussed is called the Intrusion Detection and Prevention System (IDPS). The particular system discussed here is an extension of a particular system that protects a web application system from CGI attacks. [7] However, the original system did not guard against SQL Injection attacks directed at databases connected to the system. We discuss how the SQL-I extension of the Intrusion Detection and Prevention System works in more detail.

IDPS Detection Models
There are two models of detection used by this system.

- Signature-based Detection Model
- Anomaly-based Detection Model

A system that implements only one of these models is not as robust as a system that utilizes a combination of them. The signature-based detection fails to detect unknown attacks, while anomaly-based detection will detect unusual activity and behavior. This is the reason why the Intrusion Detection and Prevention System (IDPS) make use of both.

IV. PROPOSED SCHEME

Here we introduce an efficient algorithm which can be used for detecting & preventing the SQLIA. This algorithm basically works as pattern search & match. We divide this module into two different phases viz.

- Static Phase (Signature-based Detection Model)
- Dynamic Phase (Anomaly-based Detection Model /Behavioral)

Static Phase

1. Input query (from UI) compared with the pre-determined patterns stored called as anomaly patterns stored in the Static Pattern List.
2. If matched, considered as an attack and the user is blocked by giving a generic message "invalid username/password."
3. If not matched, enters the dynamic phase.

Dynamic Phase

1. Here, Anomaly Score value is calculated for the user generated SQL Query, If the Anomaly Score value is more than the Threshold value, then an Alarm is given and Query will be pass to the Administrator.

2. If the Administrator receives any Alarm then the Query will be analyzed by manually. If the query is affected by any type of injection attack, then a pattern will be generated and the pattern will be added to the Static Pattern list A.

- Anomaly Score value calculation

As mentioned earlier, in the Static Phase, each anomaly pattern from the Static Pattern List is checked with the user generated query. This anomaly pattern is given an anomaly score. If this score is a 100% match with the Static Pattern List, it is considered as an attack and the user is blocked.
Now, if the anomaly pattern is not 100% match with the ones listed in the Static Pattern List, it enters the Dynamic Phase. If the Anomaly Score value is more than the Threshold value (assume that 50%), then the query will be transferred to the Administrator.

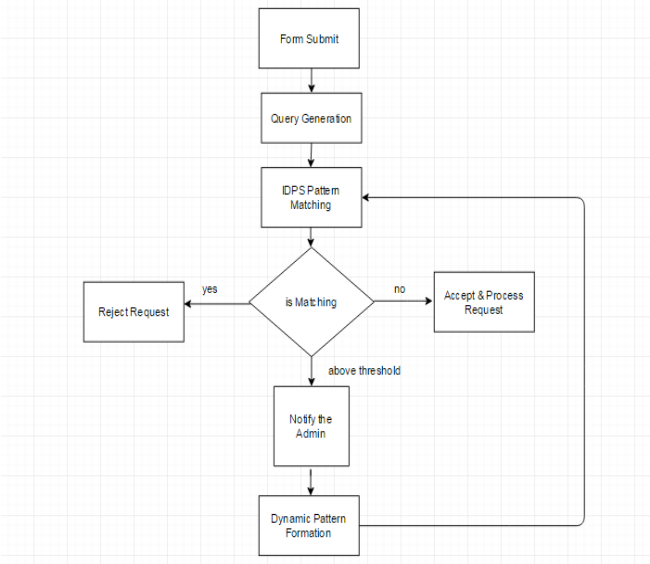

Figure1. Flowchart of Propose Scheme

V. CONCLUSION

SQL Injections is one of the most common techniques used for attacking the web applications. These attacks are used to gain the information i.e. the data stored in the database. These attacks reshape SQL queries, thus altering the behavior of the program for the benefit of the hacker. Thus to avoid this, we use IDPS systems wherein the combination of the static & dynamic phase together can help in detection & prevention of the attacks.

This implementation minimizes the efforts required by the programmer since it eliminates all the malicious queries detecting the already generated attack patterns & prevents the user from entering the system. It also detects the patterns that are prone to form an attack thus calculating the threshold value of that particular query & confirming if it can be a malicious query or not & processes accordingly further.

Future work should focus on evaluating the techniques precision and effectiveness in practice.

REFERENCES

[1.] C. Gould, Z. Su, P. Devanbu, JDBC checker: "A static analysis tool for SQL/JDBC applications", in Proceedings of the 26th International Conference on Software Engineering, ICSE, 2004, pp. 697–698.

[2.] Y. Kosuga, K. Kernel, M. Hanaoka, M. Hishiyama, Y. Takahama, "Sania: syntactic and semantic analysis for automated testing against SQL injection" , in Proceedings of the Computer Security Applications Conference 2007, 2007, pp. 107–117.

[3.] W.G. Halfond, A. Orso, "AMNESIA: analysis and monitoring for neutralizing QL-injection attacks", in Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, 2005, pp. 174– 183.

[4.] K. Spett. Sql injection: Are your web applications vulnerable?http://www.spidynamics.com/ whitepapers/WhitepaperSQLInjection.pdf, 2002.

[5] Webpage "SQL Injection Cheat Sheet" http://ferruh.mavituna.com/sql-injection-cheatsheet-oku Retrieved on 2010-03-01.

[6] File: sql-injection-detection-wp.pdf Website: http://www.f5.com/pdf/whitepapers/sql-injection-detection-wp.pdf Retrieved 2009-10-01

[7] Aulakh, T. Intrusion Detection and Prevention System: CGI Attacks, 2009. San Jose State University master's thesis project.