2011 International Conference on Information and Electronics Engineering
IPCSIT vol.6 (2011) © (2011) IACSIT Press, Singapore

Analyzing SQL Meta Characters and Preventing SQL Injection Attacks Using Meta Filter

Sangita Roy [1+](Corresponding author.), Avinash Kumar Singh [2] and Ashok Singh Sairam[1]

[1] Indian Institute of Technology, Patna, India

[2] Kalinga Institute of Industrial Technology, Bhubaneswar, India

Abstract. SQL injection attacks (SQLIA) are widely used in which an attacker crafts input to the application server to access or modify data on the database server. A common approach for an attacker to launch SQLIA is by modifying the input URL to contain partial SQL queries and trick the server into executing them. In this paper we first identify all those input patterns that can appear in the URL of an attack. Next we proposed to deploy a SQL Meta character filter that parses the input URL to detect attack patterns. The attack patterns are so chosen so that SQL Meta characters that appear in a legal input are not filtered out. We implement the filter using Java servlet and demonstrate its effectiveness.

Keywords: SQL injection attacks, prevention, SQL Meta Characters, URL filtering.

1. Introduction

SQL injection (SQLI) can allow an attacker unrestricted access to the databases and thereby to potentially sensitive information. With a myriad of techniques available to perform SQLI, sanitizing the code can be tedious, cumbersome and time-consuming. Many of the SQL injection vulnerabilities discovered in real application are due to human errors. So developers need to be very careful for their coding practice [2,3]. URL filters are commonly used by enterprises to block websites with objectionable content. In this paper we propose to translate the user input to an URL and use an SQL Meta character analysis filter to validate the input. This will allow a developer to fully concentrate on code development and leave the task of code sanitization to the filter. The paper is organized as follows. Section 2 presents review of different SQL Injection prevention mechanisms. In section 3 we present our URL filtering approach to prevent SQLI. Section 4 shows the implementation details and the result analysis. Conclusion and future work has been discussed in section 5.

2. Analysis of SQL Injection Attacks

In this section we review the different types of known SQLI attacks ([1]). For each such attack we identify a *pattern* of the attack. A pattern or *signature* of the attack is a sequence of characters that will always appear in the URL for that particular attack type. Our aim is to extract a signature for the known SQLI attacks and then use these signatures to prevent such attacks.

2.1. Tautologies

In a tautology-based attack an attack code is injected using the conditional OR operator such that the query always evaluates to TRUE. For example, an attacker can invoke the code using the URL: *http://localhost/?EmpId=1‘ OR ‘1’=’1*. The conditional *OR* statement will get appended to the SQL statement and the query will always evaluate to TRUE. In this example it will render the following statement:

SELECT empinfo FROM EmpTable WHERE EmpID = '1' OR ‘1’=’1'

[+]Corresponding author. Tel.:+918521309190
Email Address: rubyroy21.2005@gmail.com

In this type of attack the injected code will always start with a string terminator (‘) followed by the conditional OR operator. The OR operator will be followed by a statement that always evaluates to TRUE. So the signature for such attacks is the string terminator (‘) and OR.

2.2. Illegal/Logically Incorrect Queries

If an incorrect query is sent to the DBMS then generally it display the error with a description, in this attack attacker uses those information (DBMS Error) to explore the whole database, like grabbing the banner, extracting the columns, table names and records.There are several ways to perform illegal or incorrect queries like incorrectly terminating the string (‘), invalid conversions, using AND operator to perform incorrect logics, using order by, etc.

2.3. Union Query

Union is a command in the database which work for combining two queries, by using UNION command, attacker merges their specially crafted queries with the original query to extract the records from a table other than the one intended by the developer. Continuing with our running example, an attacker can invoke an union-query attack using the URL

http://localhost/?EmpId=‘ UNION <SQL statement>.

This url will render the following SQL statement

SELECT empinfo FROM EmpTable WHERE EmpID = '' UNION <SQL statement>. The second SQL statement will get executed.

The signature of union-query attack is the UNION meta character of SQL.

2.4. Piggy-backed Queries

In this type of attack, attacker binds another SQL statement by terminating the first using”;”. The first query will execute as normal but the subsequent injected queries will also get executed. An example for such an attack was shown in section 2 where the attacker can supply values through the URL such that tables get dropped. The signature of this attack is (;), the database line terminator.

2.5. Stored Procedure

A stored procedure is a set of SQL commands that has been compiled and stored on the database server. Databases often come with a number of preloaded stored procedures. Client applications can execute the stored procedure over and over again without sending it to the database server every time or compiling it. Since the stored procedure contains SQL statement, it can be exploited by an attacker by piggy-backing the attack code. The signature of this attack will be the same as that of piggy-backed queries.

2.6. Blind SQLI

In this attack when the error is not visible in client side and if the application is still vulnerable the attacker inject some SQL code which evaluates as true and false, so by inference the page behavior attacker can perform SQLI. The possible guessing start with the AND operator and some time attacker also uses conditional operators.

2.7. Timing Attack

This is also like blind injection where attacker used SQL time function to observe the behavior, and by using if else condition attacker extract records from the database and exploit the database.

WAITFOR is a function used for delaying the response from the database. In this type of attack the IF ELSE statement is used for injecting queries. So the possible signatures of this attack are WATEFOR, IF, ELSE.

2.8. Alternate Encoding

In this technique, attackers modify the injection query by using alternate encoding, such as hexadecimal, ASCII, and Unicode for bypassing the validations, e.g. char(44) is used in place of single quote(‘) which is a bad character.

Char(), ASCII(), HEX(), UNHEX() etc. used for encoding the actual string

3. Proposed Technique

In server-side architecture, a user invokes the services provided by the application server using a browser. The input provided by the user is usually sent to the application server in the form of a parameter string. The application server uses this input to generate a SQL query to retrieve information from the database or update it. As shown in figure 1, our proposed Meta filter is positioned between the user and application server. The filter intercepts the input from the user, parses it into SQL Meta character tokens. If the input from the user contains any attack signature then the injected input is treated as an attack and an error page is displayed [4], otherwise the input is processed by the application server normally.

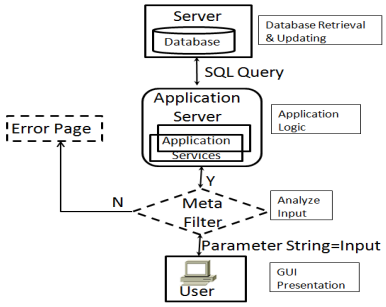

Fig. 1: Server-side Architecture and its interaction with our proposed Meta Filter

As can be seen from figure 1, the SQL query and as such the SQL Meta characters are generated by the application server. Our proposed Meta filter checks for the presence of SQL Meta characters before the input is processed by the application server. Therefore, our proposed solution will not block legal inputs. Moreover, the attack patterns have been so designed so that it is robust to SQL Meta characters that can accidentally occurs in a legal input.

4. Implementation and evaluation

In this section we show how to design a Meta filter using Java code. The input to the filter is the request URL from the user. The detailed steps involved in designing the filter are outlined below:

Step1: Get url request from User

```
HttpServletRequest req1 = (HttpServletRequest)req;
```

Step2: Parse the Input

```
String qry = req1.getQueryString();
```

Step3: Check whether attack pattern is present in input

```
String[] AttackPattern={"'"," CREATE"," create"," ALTER"," alter"," DROP"," drop"," RENAME"," rename"," SELECT"," select"," INSERT","
insert"," UPDATE", " update"," DELETE", " delete", " GRANT", " grant"," REVOKE", " revoke", " char", " int", "@@version","@@VERSION","
exec"," EXEC"," union", " UNION"," WAITFOR", " waitfor"," ORDER BY"," order by",";"};
int i=0,fnd;boolean chk=true;
for(i=0;i<AttackPattern.length;i++){
    fnd=qry2.indexOf(AttackPattern [i]);
    if (fnd!=-1)        {
        chk=false; //Attack Pattern Present
        break;          } }
if (chk==true) {
chain.doFilter (req, res); //Process the query
}
else{
res1.sendRedirect("./error.html");//Redirect to error page
}
```

Fig. 2: JAVA code for Meta Filter

We have designed the filter using java servlet. In step 3 we checked all the SQL meta characters which can appear in the url for SQLIA. In section 3 we have analyzed all the different type of SQLIA and found out the different SQL Meta character required for that attack. If our filter gets any SQL Meta character in the url attack pattern it will forward the request to the error page otherwise it will greet us with the required page.

We implemented the above code on a windows 2008 server and tested it for the different SQLIA mentioned in section 3. Our evaluation shows that we could prevent the above mentioned SQLIA using the meta filter.

5. Conclusion and Future Work

In this paper we proposed a model to prevent SQLI using a simple url filter. A developer only needs to concentrate on the filter to validate inputs from clients. The approach is very easy to deploy because only single filter page is maintained to control the whole application as well as it can take care of new pages get added. The approach was tested using Java servlets. As future work we have to analyze new SQLI signature.

6. References

[1] William G.J. Halfond, Jeremy Viegas, Alessandro Orso. A Classification of SQL Injection Attacks and Countermeasures, *In The Proceeding of the IEEE International Symposium on Secure Software Engineering,* Arlington,VA,USA, March 2006.

[2] Nuno Antunes, Marco Vieria. Comparing the Effectiveness of Penetration Testing and Static Code analysis on the Detection of SQL Injection Vulnerabilities in Web Services, *In The Proceeding of 15th Pacific Rim International Symposium Dependable Computing*, 2009, pp. 301-306

[3] Kasra Amirtahmasebi, Seyed Reza Jalalinia, Saghar Khadem. A survey of SQL Injection Defense Mechanisms, *In The Proceeding of the International Conference for Internet Technology and secured transaction (ICITST 2009),* Nov 2009.

[4] Susanta Nanda, Lap-chung Lam, Tzi-cker Chiueh. Web Application Attack Prevention for Tiered Internet Services, *In The Proceeding of the Fourth International Conference on Information Assurance and Security (IAS 08),* Sept 2008, pp. 186-192