A Novel Approach to Prevent SQL Injection Attack Using URL Filter

Sangita Roy, Avinash Kumar Singh, and Ashok Singh Sairam, Senior Member IACSIT

Abstract—Web services are usually supported by a database at the backend while a frontend takes input from the user, construct SQL statements and access the database. SQL injection is a popular technique used by attackers to exploit unsanitized user input vulnerability by convincing the application to run SQL code that it was not intended to run. Validating all user inputs and checking for vulnerability can be tedious on the part of the programmer. In this work we propose a new approach to prevent SQL injection attack using URL filtering. URL filters are used to validate user input to web forms. In this approach a single filter can be used to validate input to several databases which makes our approach more scalable and efficient. We implement the filter using Java servlet and demonstrate its effectiveness.

Index Terms—SQL injection attacks, prevention, URL filtering.

I. INTRODUCTION

Web services have become hugely popular because it allows an enterprise integration of its numerous Internet-enabled applications. A web service can be remotely triggered by a client using HTTP. Typically the client will send a query, the web service will retrieve the relevant information from an underlying database and send back the response.

The input from the client can be gathered using either the input box present in the web form or the URL of that web form. Next the application will use the input to construct a SQL statement, query the database and send the response back to the client.

Insufficient validation of user input can allow an attacker to induce the application to run SQL code not intended by the developer. Such attacks known as 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 [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 URL 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 II defines SQL injection attack. Section III presents review of different SQL injection prevention mechanisms. In section IV we present our URL filtering approach to prevent SQLI. Section V shows the implementation details and the result analysis. Conclusion and future work has been discussed in section VI.

II. SQL INJECTION

SQL injection is a code injection mechanism in which malicious code is inserted into the input point of a web form to gain access to the database. The primary form of SQL injection consists of direct insertion of code into user-input variables that are concatenated with SQL commands and executed. For example in the following code, the user is prompted to enter a name. The script then builds a SQL query by concatenating hard-coded strings together with a string entered by the user.

```javascript
var UserName;
UserName = Request.form("UserName");
var sql = "select * from UserTable where UserName = '" +UserName+'"';
```

In the above code if the user inputs `Raju'; drop table UserTable--`

It will cause the database to delete the table UserTable.

An indirect attack injects malicious code into strings that are destined for storage in a table or as metadata. When the stored strings are subsequently concatenated into a dynamic SQL command, the malicious code is executed. SQLI occurs because SQL Server will execute all syntactically valid queries that it receives [1].

III. SQL INJECTION PREVENTION MECHANISM

There are number of techniques available in literature to address SQLI attacks. Here we review all the techniques briefly with their advantages and disadvantages [2], [4], [7].

A. Defensive Coding Practices

The defensive coding is for the developer who is responsible for developing the web application. As the coding practice is very much prone to human error, developers always give the extra effort to code safely. The root cause of SQLI is the insufficient input validation and sometimes developers forgot to add checks or did not perform adequate input validation. So there are various...
guide lines proposed to fix this problem [5].

1) Input type checking
SQLI attacks can be performed by injecting commands into either a string or numeric parameter. A simple check of such inputs can prevent many attacks.

2) Encoding of inputs
Injection into a string parameter is often accomplished through the use of meta-characters that trick the SQL parser into interpreting user input as SQL tokens. So the solution is to use the functions that encode a string in such a way that all meta characters are specially encoded and interpreted by the database as normal characters.

3) Positive pattern matching
Input validation should be able to identify all good inputs as opposed to all bad inputs. Because the negative validation is not always possible due to the new type of attack signature. So better solution is to implement the positive validation.

4) Identification of all input points
Developers must check all input points to their application. There are many possible sources of input to an application. If used to construct a query, these input sources can be a way for an attacker to introduce an SQLIA. Simply put, all input sources must be checked.

B. Black Box Testing

The web vulnerability scanner are used for the black box testing, the vulnerability scanner are used for finding the loop holes in the existing application. The vulnerability scanner mainly visits the web application’s input point and simulates the attack against and if the attack is possible or made success then it summarizes it in the form of a report.

C. White Box Testing

The static code analyzers are used for the white box testing, the static code analyzers basically analysis the byte code of the web application with the intension of finding the vulnerability.

D. Run Time Monitoring

For the run time monitoring IDS (Intrusion detection System) can be used, the IDS system is based on a machine learning techniques that is trained using a set of typical application queries. The technique builds models of the typical queries and then monitors the application at run time to identify queries that do not match the model.

IV. PROPOSED TECHNIQUE

In this section we present the URL filter approach to address the problem of SQLI. As shown in figure 2, by filter we mean a program that runs on the server before the servlet or JSP page with which it is associated. A filter can be attached to one or more servlets or JSP pages and can examine the request information going into these resources. After doing so, it can choose among the following options [8].

- **Authentication**-Blocking requests based on user identity.
- **Logging and auditing**-Tracking users of a web application.
- **Image conversion**-Scaling maps, and so on.

- **Data compression**-Making downloads smaller.
- **Localization**-Targeting the request and response to a particular locale.
- **XSL/T transformations of XML content**-Targeting web application responses to more than one type of client.

These are just a few of the applications of filters. There are many more, such as encryption, tokenizing, triggering resource access events, mime-type chaining, and caching.

The great advantage of using filter is that we can make a single filter for many pages, so it enhance the reusability and as well as scalability, the main concern of filter designing is to provide security against the SQLI, generally a attacker launch their attacks with the help of URL modification, because of the in sanitized URL the request directly goes to the database server and the database server will act according that, so the little modification in the URL an attacker can take control all over the application. By placing filter between the database server and the request we can actually secure the web application, and by the reusability factor of the servlet we have to design only one filter for all[6].

Fig. 1 shows that if any request will come for the page1, page2 or any page in the application server then it first goes to the filter then filter check the request if this the valid request then it return back to the same page for that request has come otherwise it divert the request to the default error page, so any changes in the URL will not be considered as the legitimate request and greet with the error page or any message [9], [10].

E. Proposed Architecture

Our proposed architecture is shown in figure 2. The architecture consists of three major building blocks: the clients who send the request, the application server where the logic of the program will be decided and the database server which can store the clients’ data for the future use. In normal 3-tier web application architecture there is no filter...
deployed in application server. The work flow diagram of our proposed model is shown in figure 3. The request from the client is intercepted by the application server and the requested url is send as an input to the filter. The filter checks the URL filter database and if it is a valid url then it returns success to the application server else an error message is displayed. On getting a success response from the filter the application servers constructs the corresponding SQL query and directs it to the appropriate database.

![Work flow diagram of proposed model](image)

**F. Filter Design**

In this section we show how to design a simple filter using Java code. The input to the filter is the request url from the application server. The steps involved in designing the filter are outlined below:

**Step1:** Get url request from application server

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

**Step2:** Construct query from the given url and check whether it is valid

```java
String qry = req1.getQueryString();
ResultSet rs=stmt.executeQuery("select * from fil_url");
//selecting all input fom database
while(rs.next()) {
    String x=rs.getString("input");
    if (qry.compareTo(x) == 0)
        // if both matched
        chain.doFilter (req, res); //valid query
    else
        res1.sendRedirect("/avinash/error.html"); // invalid query, divert to error page
```

**G. Filter Mapping**

We have to little modification in web.xml file which are as following.

```xml
<filter>
    <filter-name>fil</filter-name>
    <filter-class>filter</filter-class>
</filter>
<filter-mapping>
    <filter-name>fil</filter-name>
</filter-mapping>
```

**V. Result and Analysis**

Fig. 4 shows the original web page with the url http://127.0.0.1:8082/avinash/filtercheck?t1=Hopes, which is our input point here. In this url the query string is t1=Hopes, which means that this filtercheck page will display the records from the database which have the values Hopes. The query will be represented like this: SELECT * FROM poems WHERE title= 'Hopes'; Figure 5 shows the case where the attacker has fired a SQLI attack. The url is http://127.0.0.1:8082/avinash/filtercheck?t1='or'1='1. The corresponding SQL query will be SELECT * FROM poems
WHERE title = ‘OR ‘1’=’1; If the url is not sanitized then all the records in the table poems will be displayed. Attacker can also put other query string like http://127.0.0.1.http://18082/avinash/filtercheck?q1=Hopes order by n. The query will be executed like this: SELECT * FROM poems WHERE title = ‘Hopes’ ORDER BY n ; here n is the integer value and will be used to find columns. 

Fig. 6 shows the result where we have prevented all these type of SQL queries through filter which redirects towards an error page rather than the requested page.

We have implemented a filter which can successfully work with any version of Java servlet. For the deployment we have used localhost of apache server. Similarly we can map it globally. The current filter design is fully and perfectly able to block SQL injection attack without any complexity.

We have implemented it only on jsp and servlet. When the web page is designed using jsp our approach is fully able to block the SQLi attack. With the current approach on defensive coding practice where a developer has to check all the input points to validate it, our approach will takes less time for validation. The developer only needs to concentrate on the filter database.

In the current three tier approach, if any new page is added to the web application the developer need to write another new code for validation. However, using the filtering approach the developer only need to update the database.

VI. CONCLUSION AND FUTURE WORK

In this paper we proposed a model to prevent SQLI using a simple url filter. The approach though simple is robust as it isolates the actual database from the attacker. A developer only needs to concentrate on the filter to validate inputs from clients. The approach is scalable because only the url database needs to updated as new pages get added. The approach was test using Java servlet. As future work we propose to implement this approach for different web development languages like PHP and ASP. We also plan to test the filter comprehensively against the various types of SQLI attacks.

REFERENCES


Sangita Roy obtained her B.Tech degree from West Bengal University of Technology, India in the year 2005. She obtained her M.Tech degree from Kalinga Institute of Industrial Technology, Bhubaneswar, Orissa in 2008. Currently she is pursing her Ph.D degree from Indian Institute of Technology, Patna. Prior to this she was working as an Assistant Professor at Kalinga Institute of Industrial Technology, Bhubaneswar. Orissa. Her research interests include steganography, web application security and network security.

Avinish Kumar Singh obtained his B.Sc(IT) and M.Sc(IT) degree from Kumaun University, Nainital in the year of 2007 and 2009 respectively. He has received his M.Tech degree from Kalinga Institute of Industrial Technology in the year of 2011. He worked as an Assistant professor in Gwalior Engineering College, Gwalior. Currently he is pursuing his Ph.D degree from Indian Institute of Information Technology,Allahabad. His research interests include web application security and network security and Biometric. He is also a certified Ethical Hacker.

Dr. Ashok Singh Sairam obtained his B.Tech degree from National Institute of Technology Silchar, India in the year 1993. He obtained his M.Tech and Ph.D degree from Indian Institute Technology Guwahati, India in 2001 and 2009 respectively. Currently he is working as an Assistant Professor at Indian Institute of Technology Patna, India. Prior to this he was working as a senior research officer at Indian Institute Technology Guwahati, India. His research interests include network security, wireless networks and traffic engineering. He is currently working as a chief investigator on a major network security project sponsored by the department of Information Technology, government of India. He has given invited lectures and served as PC member in several international conferences.