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*—SOL 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.

Manuscript received July 30, 2012; revised September 18, 2012. This work was supported in part by the Indian Institute of Technology Patna, India.

Sangita Roy is with Computer Science and Engineering Department, Indian Institute of Technology Patna, India, (e-mail: r_sangita@iitp.ac.in).

Avinash Kumar Singh was with KIIT University, Bhubaneswar, Orissa, India. He is now with the Department of Computer Science, Indian Institute of Information Technology Allahabad, India (e-mail:rs110@iiita.ac.in).

Ashok Singh Sairam is with the Computer Science and Engineering Department, Indian Institute of Technology Patna, (e-mail: ashok@iitp.ac.in).

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.

```
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