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.

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

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).

DOI: 10.7763/IJIMT.2012.V3.284

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].

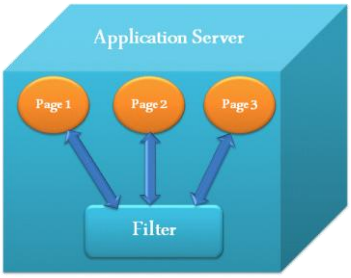


Fig. 1. Communication between filter and application server.

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

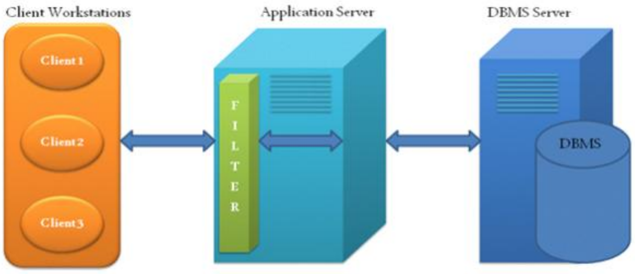


Fig. 2. A simple 3-tire web application architecture with filter

Our proposed architecutre 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.

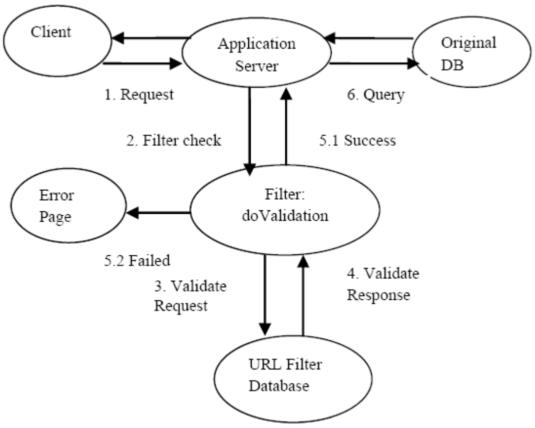


Fig. 3. Work flow diagram of proposed model

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

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

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

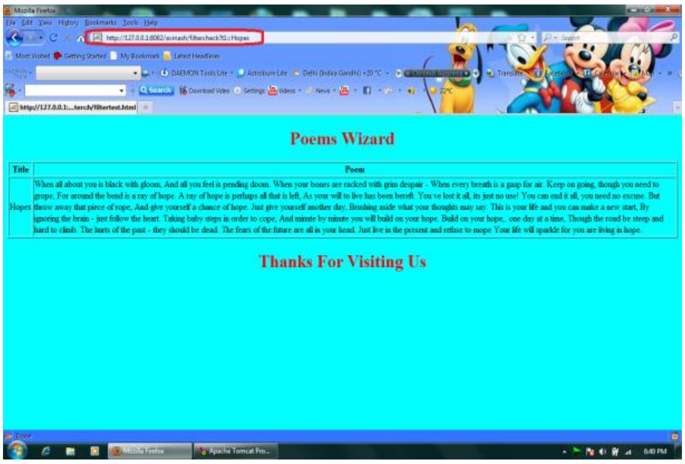
```
String qry = req1.getQueryString();
ResultSet rs=stmt.executeQuery("select * from fil_url");
//selecting all input fom database
while(rs.next()) {
String x=rs.getString("input");
if (qry.compareTo(x) == 0) // if both matched
chain.doFilter (req, res); //valid query
else
res1.sendRedirect("/avinash/error.html");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

G. Filter Mapping

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

```
<filter>
    <filter-name>fil</filter-name>
    <filter-class>filter</filter-class>
  </filter>
<filter-mapping>
   <filter-name>fil</filter-name>
<url-pattern>/*</url-pattern>//this will map all url in the
web application
</filter-mapping>
```

V. Result and Analysis



http://127.0.0.1:8082/avinash/filtercheck?t1=Hopes

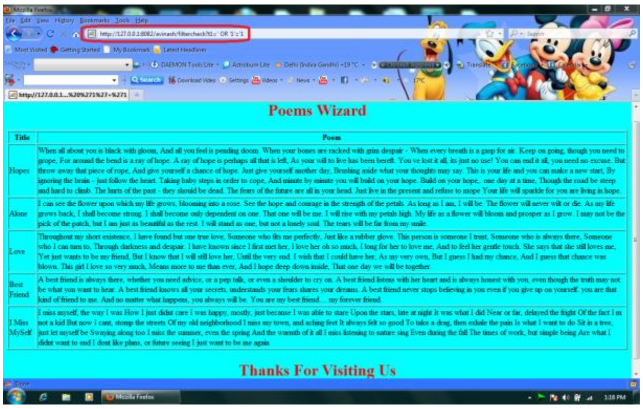
http://127.0.0.1:...terch/filtertest.html

Poems Wizard

Title	Poem
Hopes	When all about you is black with gloom, And all you feel is pending doom. When your bones are racked with grim despair - When every breath is a gasp for air. Keep on going, though you need to grope, For around the bend is a ray of hope. A ray of hope is perhaps all that is left, As your will to live has been bereft. You ve lost it all, its just no use! You can end it all, you need no excuse. But throw away that piece of rope, And give yourself a chance of hope. Just give yourself another day, Brushing aside what your thoughts may say. This is your life and you can make a new start, By ignoring the brain - just follow the heart. Taking baby steps in order to cope, And minute by minute you will build on your hope. Build on your hope,. one day at a time, Though the road be steep and hard to climb. The hurts of the past - they should be dead. The fears of the future are all in your head. Just live in the present and refuse to mope Your life will sparkle for you are living in hope.

Thanks For Visiting Us

Fig. 4. Original web page without putting any command on URL



http://127.0.0.1:8082/avinash/filtercheck?t1=' OR '1'='1

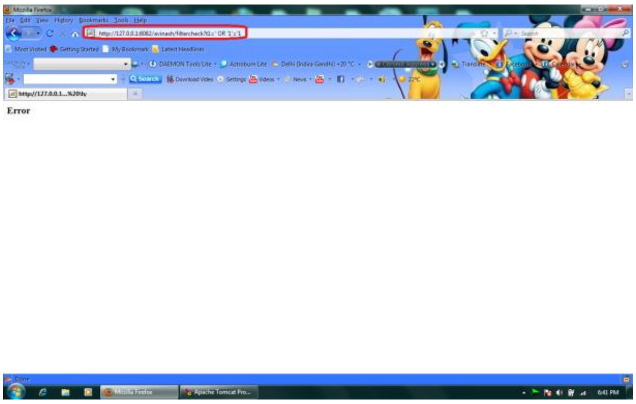
http://127.0.0.1...%20%271%27=%271

Poems Wizard

Title	Poem
Hopes	When all about you is black with gloom, And all you feel is pending doom. When your bones are racked with grim despair - When every breath is a gasp for air. Keep on going, though you need to grope, For around the bend is a ray of hope. A ray of hope is perhaps all that is left, As your will to live has been bereft. You ve lost it all, its just no use! You can end it all, you need no excuse. But throw away that piece of rope, And give yourself a chance of hope. Just give yourself another day, Brushing aside what your thoughts may say. This is your life and you can make a new start, By ignoring the brain - just follow the heart. Taking baby steps in order to cope, And minute by minute you will build on your hope. Build on your hope,. one day at a time, Though the road be steep and hard to climb. The hurts of the past - they should be dead. The fears of the future are all in your head. Just live in the present and refuse to mope Your life will sparkle for you are living in hope.
Alone	I can see the flower upon which my life grows, blooming into a rose. See the hope and courage in the strength of the petals. As long as I am, I will be. The flower will never wilt or die. As my life grows back, I shall become strong. I shall become only dependent on one. That one will be me. I will rise with my petals high. My life as a flower will bloom and prosper as I grow. I may not be the pick of the patch, but I am just as beautiful as the rest. I will stand as one, but not a lonely soul. The tears will be far from my smile.
Love	Throughout my short existence, I have found but one true love, Someone who fits me perfectly. Just like a rubber glove. This person is someone I trust, Someone who is always there, Someone who I can turn to, Through darkness and despair. I have known since I first met her, I love her oh so much, I long for her to love me, And to feel her gentle touch. She says that she still loves me, Yet just wants to be my friend, But I know that I will still love her, Until the very end. I wish that I could have her, As my very own, But I guess I had my chance, And I guess that chance was blown. This girl I love so very much, Means more to me than ever, And I hope deep down inside, That one day we will be together.
Best Friend	A best friend is always there, whether you need advice, or a pep talk, or even a shoulder to cry on. A best friend listens with her heart and is always honest with you, even though the truth may not be what you want to hear. A best friend knows all your secrets, understands your fears shares your dreams. A best friend never stops believing in you even if you give up on yourself. you are that kind of friend to me. And no matter what happens, you always will be. You are my best friend.... my forever friend.
I Miss MySelf	I miss myself, the way I was How I just didnt care I was happy, mostly, just because I was able to stare Upon the stars, late at night It was what I did Near or far, delayed the fright Of the fact I m not a kid But now I cant, stomp the streets Of my old neighborhood I miss my town, and aching feet It always felt so good To take a drag, then exhale the pain Is what I want to do Sit in a tree, just let myself be Swaying along too I miss the summer, even the spring And the warmth of it all I miss listening to nature sing Even during the fall The times of work, but simple being Are what I didnt want to end I dont like plans, or future seeing I just want to be me again

Thanks For Visiting Us

Fig. 5. Web page with the command on URL and the result without using filter.



http://127.0.0.1:8082/avinash/filtercheck?t1=' OR '1'='1

http://127.0.0.1...%20%

Error

Fig. 6. Web page with the command on URL and the result using filter.

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://127.0.0.18082/avinash/filtercheck?t1=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 the all 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

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

[2] A. Tajpour, M. Massrum, and M. Zaman Heydary, "Comparison of SQL Injection Detection and Prevention Techniques," in *Proc. The Proceeding of the 2nd International Conference on Education Technology and Computer (ICETC 10)*, vol. 5, pp. 174-179, June 2010.

[3] N. Antunes and M. Vieria, "Comparing the Effectiveness of Penetration Testing and Static Code analysis on the Detection of SQL Injection Vulnerabilities in Web Services," in *Proc.* 15*th Pacific Rim International Symposium Dependable Computing*, pp. 301-306, 2009.

[4] K. Amirtahmasebi, S. R. Jalalinia, and S. Khadem, "A survey of SQL Injection Defense Mechanisms," in *Proc. the International Conference for Internet Technology and secured transaction (ICITST 2009)*, Nov 2009.

[5] G. J. W. Halfond, Alessandro Orso, and Panagiotis Manolios, "WASP: Protecting Web Applications Using Positive Training and Syntax-Aware Evaluation," in *Proc. IEEE Transaction on Software Engineering (TSE 07)*, vol. 34, pp. 65-81, Jan-Feb 2008.

[6] S. Nanda, L. Lam, and T. Chiueh, "Web Application Attack Prevention for Tiered Internet Services," in *Proc. the Fourth International Conference on Information Assurance and Security (IAS 08)*, pp. 186-192, Sept 2008.

[7] M. Muthuprasanna, Ke Wei, and Suraj Kothari, "Eliminating SQL Injection Attacks - A Transparent Defense Mechanism," in *Proc. the International Symposium on Web Site Evolution (WSE 06)*, pp. 22-30, Sept 2006.

[8] The essentials of Filters. (2010). [Online]. Available: http://www.oracle.com/technetwork/java/filters-137243.html

[9] M86 Security. (2010). Six Steps for Implementing an Effective Web Security Solution. White paper. [Online]. Available: http://www.m86security.com/documents/pdfs/white_papers/business/WP_Six_Steps_for_Effective_Web_Security.pdf.

[10] Osterman Research. The Critical Need to Secure the Web in Your Company. (2010). White paper, Webroot, Inc., [Online]. Available: http://www.ostermanresearch.com/whitepapers/or_or0210a.pdf.

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 pursuing 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.

Avinash Kumar Singh obtained his B.Sc(IT) and M.Sc(IT) degree from Kumaun University, Nanital 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 invesstigator 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.