Preventing Injection Attack by Whitelisting Inputs

Gopi Krishnan S
Research Associate
Society for Electronic Transactions and Security, Chennai, India
gopikrishnans@setsindia.net

Sandeep K
PG Research Scholar
SRM Univerisity, Chennai, India
k.sandeep440@gmail.com

***Abstract*—Usage of web applications becomes larger than just web pages. Usage of dynamic pages brought lot of vulnerabilities. The web technology offers social networking, online shopping, electronic library, and etc. This paper proposes a possible technique to prevent injection attacks by whitelisting inputs. This model was developed using PHP (Hypertext Preprocessor) to makes our developers easy. The input elements are divided into text input and list input elements. The text input element is verified using regular expression. And the list input element verified comparing keys defined and submitted.**

***Index Terms*—web security; injection attack; validation; php**

I. Introduction

Day to day increase in usage of web application introduced attacks on web applications and web server. A vulnerability of single page in a server can make it exploitable. Injection attack is one of the major attack to which most of the pages are vulnerable. According to the Open Web Security Project (OWASP); Injection attack is top most attack to which most of the pages are more vulnerable.

II. Web Injection Attack

Injection attack is act of injecting malicious data inside POST and/or GET data to exploit the web server. To launch the attack the hacker has to trick the input box or URL by writing/appending combinations of malicious inputs. To make this task simple lot of free tools are available on the Internet. The different types of injection attacks are discussed below.

A. Code Injection

A malicious code was injected to the web page's input box. The injected code executes on client side or server side. The login, registration and guest book are ideal place for code injection.

Figure 1. Code injection attack to steal cookie

Fig. 1 shows the example of code injection. This input will be executed if programmer hasn't validated the inputs. This will redirect to another page with their cookie.

B. SQL Injection

The SQL (Structured Query Language) injection attack is a method used by hackers to exploit the database by using Vulnerabilities in the application software. A successful SQL injection exploit can read and modify sensitive data from the data base. SQL injection is classified into [1] many types some of them are illicit queries, union queries, piggy backed queries, stored procedure. SQL injection occurs when data enters a program from an untrusted source, data used to dynamically construct a SQL query. The main consequences [2] of SQL injections are confidentiality, authentication, and authorization.

To detect and remove [3] the SQL queries there are certain methods. Methods like web framework, static analysis, dynamic analysis, and machine learning method.

These methods has certain draw backs to overcome them Dr. Lee proposed [4] the novel method which is used to detect and remove the SQL injections and queries based on the static and dynamic analysis. By this method they scan web applications in order to extract fixed SQL queries and make a list to be compared with each generated dynamic SQL queries. They use rules of static and dynamic SQL queries this method removes the attribute value in SQL queries and make it independent DBMS (Database Management System).

Fig. 2 shows the example to bypass authentication via SQL Injection. If the developer failed to validate the input, it will return true as the SQL statement returns true

Manuscript received January 10, 2013; revised March 12, 2013.

©2013 Engineering and Technology Publishing
doi: 10.12720/lnit.1.3.132-135

for or operation with a statement that always return true. So, a malicious user will get access to the contents.

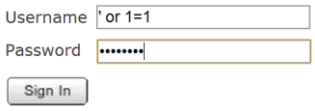

Figure 2. SQL Injection attack to bypass authentication

C. Command Injection

Consider a website is having search option. Developer uses grep command to find the file with the content for programming simplicity. The find text box will get the text to search and gives in exec function as parameter.

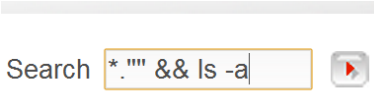

Figure 3. Command Injection to execute malicious command

Fig. 3 shows a sample command injection. The text typed in text box goes as argument to grep command inside a exec function. The search query must be framed as exec("grep $_POST['query'] $search"); But the injected malicious string will expanded to list all the files in workdirectory as exec("grep *."" && ls -a"); in place of "ls -a" any other malicious kind of linux command can be substituted. That will executed with the privileges of web server or application.

III. Preventing Web Injection Attacks

A. Related Work

Prithvi Bisht et al [5] proposed a dynamic evaluation mechanism to prevent SQL injection attack. It works by escaping the special characters. The injection involves special characters such as single quotes, double quotes, ampersand, and etc.

Ramakanth Dorai et el [6] discussed various possible ways and issues of SQL injection attack. They provided solution based on Mod-Security, Green-SQL, PHP's string escaping feature, user access control management, and encrypting data. Sandeep et al [7] developed a mechanism to prevent the SQL injection attack by syntactic and semantic analysis of SQL query that sent from web application to database service. Before executing the SQL injection, the query is analyzed.

Shuo Tang et el [8] proposed a layer that integrates in browser and provides automated security mechanisms to protect the web pages from injection attack. But this can easily bypassed using tools such as burpsuite. For example the information passed through this page can be sniffed using a tool called "burpsuite". It acts as a proxy server and allows us alter the data before it sent to the internet. So, the protection only in client side does not ensure safety.

B. Contribution

A public class called *SecureField* was developed using PHP (hypertext preprocessor) which takes the arguments of *FieldHTML* and *LstOpts*. It is mandatory when developer create a combo list box. The constructor identifies the type of element and builds the html tags accordingly with the provided attributes; while other methods help the constructor by providing necessary functionality. If the type given *FieldHTML* is list the html is constructed for a combo-box. This framework enforces the input pattern on the developers. Rather than having complex systems that consumes heavy resource, a simple enforcement of pattern can prevent the injection attack. Again the part is choosing a correct pattern. The standard set of ready made patterns are available that can used.

C. Building HTML

If textbox or text area needs to be created then, the attributes are provided in *$key=>$value* array format. The elements provided are type, id, class, pattern, and title. The pattern and title are the mandatory attribute of this type of object. The element is not created if pattern or title is not provided. While creating a list boxes the *LstOpts* is mandatory in place of pattern or title. The list box needs attributes such as id and class. The uses same format as textbox or text area. The LstOpts is provided as array in *$value=>$label* format.

The feature of HTML5 (hypertext markup language) is used here to validate the client side inputs based on the specified pattern. The pattern is specified in form of regular expression. The form element objects are created in a separate PHP script file. The form element is deployed calling *object→echoHtml()*. This script is included in both file where to deploy form element and where to post the element. The element value passed through $_POST is again verified in server side. To verify validity of the $_POST data the method *object→isValid()* is called for each object to get verified. It returns true if given input is valid.

D. Server Side Validation

Even the inputs are validated on client side; it's very safe to verify them on server side too. A man in the middle attack can alter the content of date sent by the user. Burpsuite is a tool that can be used to alter the sent by them to a server without their knowledge. Intruder module in the burpsuite exactly locates and shows the $_POST and $_GET contents the hacker. In addition it also provides an option to alter the content before it reaches the server. The server only validation needs AJAX or validated after entire form is filled and submitted. This increases content loaded by AJAX requests. In other case the user get stressed when they receive a validation error after filling the entire form. Twitter social networking registration site is very good example for AJAX based server side only post-validation.

But matrimonial networking registration requires too many inputs to be fed. The stress will in more as many validation errors thrown after post-validation at server side. To validate the textbox, regular expression is used on both client and server. An attacker can also exploit the regular expression verification process by giving the more characters that matches the regular expression. That

©2013 Engineering and Technology Publishing

causes the server to server check he character for long time.

For example the expression /^[a-z][a-z0-9]*/ used to match the user name, that can matches any number of alphanumeric string. This expression exploitation can be avoided by changing the regular expression that matches only limited length of character. The example given above can be rewritten as /^[a-z][a-z0-9]*{,16}/. This matches the string of length from three to 16. The username cannot go beyond 16 characters.

The option combo list boxes are also easily used to inject an invalid data. Using Google chrome debugger the value of a particular select option can be changed and a malicious content can be injected through global $_POST array variable. So the combo-list box is also validated for its $_POST array data. The submitted option is cross checked with the list of available values in the object for which it is created.

IV. Experimental Results

This class was utilized in web application that developed to administrate Linux based firewall appliance. The pattern loaded in client browser that supports HTML5 validates and stop the form submission if the entered input does not match the regular expression.

Figure 4. Client side validation error

Fig. 4 shows the validation error stopped the form submission at client side. This regular expression based input validation works even with java script disabled. A public class called *SceureField* was developed using PHP (hypertext preprocessor) which takes the arguments of *FieldHTML* and *LstOpts*.

Figure 5. Using Google chrome debugger to modify source

An attacker can inject the malicious content by editing the source using a debugger tool. Here Google chrome is used to inject the malicious content. Fig. 5 shows the usage of debugger tool to modify the source. The representation of the dual validation is given below. A class name *SecureField* has member function and member data namely *Sfmf*, *Sfmd* respectively. The constructor constructs the necessary html tags to echo the particular field on html page. Then, the function *EchoHtml* is called to print the actual html tags wherever necessary. The validity and integrity of user input can be verified using function *IsValid*. The function should be called wherever the input of *SecureField* got posted.

```
SecureField ◄ {Sfmf ,Sfmd}

Sfmf = < f:Construct, f:EchoHtml, f:IsValid >
Sfmd = <@Name, @Type, @Regx, @Attr, @Htm5, @Opts >

f: Construct(@HtmlData, @LstOpts = Ø)
f: @htmlString → EchoHtml()
f: @httpSubmit → IsValid()

┌─Construct(@HtmlData, @LstOpts)
│
│       @this.Name := @HtmlData['name'];
│       @this.Type := @HtmlData['type'];
│       @this.Regx := @HtmlData['regx'];
│       @this.Attr := @HtmlData['attr'];
│       @this.Opts := @LstOpts;
│
│       @this.Htm5 := @this.Type + @this.Name + @this.Regx;
│       @this.Htm5 := @this.Htm5 + V(@this.Attr);
│       @this.Htm5 := @this.Htm5 + V(@this.Opts);
│ ▪
├─EchoHtml(void)
│
│       echo @this.Htm5;
│       (@this.Type ≠ 'list' ∧ @this.Regx = Ø)?
│               echo 'WARNING! No pattern';
│ ▪
├─IsValid(void)
│
│       @status := false;
│       (@this.Type = 'list')?
│               array_key_exist(POST[@this.Name],
@this.Opts)?
│                                       @status := true;
│       : pregMatch(POST[@this.Name], @this.Regx)?
│                                       @status := true
│       return @status;
│ ▪
└──────────────────────────────────────────■
```

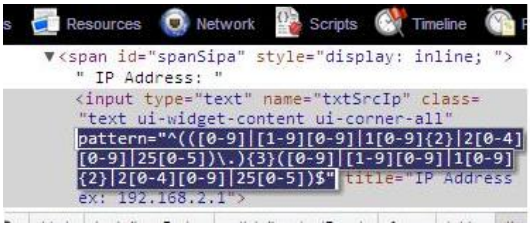

Figure 6. Modifying source to Remove Regular Expression

Fig. 6 shows the code modification. Today most of the web applications validate the textbox using java script. This HTML5 regular expression based validation even works with java script disabled. Mostly the combo- list box inputs are assumed to be valid in most web application. However, it is possible to utilize select option list to inject malicious code.

Fig. 7 shows editing the source of combo-list box to inject malicious data. In both cases the proposed system validates all the elements and displays form elements with invalid $_POST/$_GET content and exit the process.

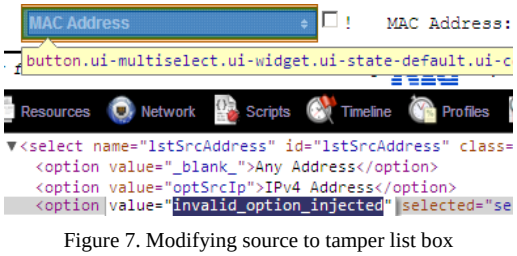

Figure 7. Modifying source to tamper list box

©2013 Engineering and Technology Publishing

Fig. 8 shows the failback page responded by our system about the invalid input to the form elements called *lstSrcAddress* and *txtSrcMac*. Currently the system accept the textbox with input either match regular expression or empty data.

```
Error @lstSrcAddress => $invalid_option_injected;
Error @txtSrcMac => $invalid_mac_Addr;
```

Figure 8. Error message echoed if any invalid input

V. Conclusion

A programming approach to prevent the injection by whitelisting the inputs. Whitelisting inputs is done for both text and list inputs to prevent malicious strings. Based on experiments, the whitelisting is simple and effective for bot validation and preventing malicious strings. But for text input elements, everything is still depends on the regular expression used. Future work includes expanding our experiment with efficient dynamic list elements and other input elements on rich Internet applications

References

[1] K. Ahmad and K. P. Yadav, "Classification of SQL injection attacks," *Visual Soft Research and Developement-Technical and Non-Technical Journal*, vol. 1, no. 4, pp. 235-242, 2010.

[2] Owasp Testing Guide, Version 3.0, 2008, pp. 204-207.

[3] P. Ramasamy, "SQL injection attack detection and prevention," *IJEST*, vol. 4, 2012.

[4] I. lee and S. Yeo, "A novel method for SQL injection attack detection based on removing SQL query attribute values," *Mathematical and Computer Modeling*, vol. 55, no. 1-2, pp. 58-68, 2012.

[5] P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan, "Dynamic candidate evaluations for automatic prevention of SQL injection attacks," *ACM Transactions on Information and System Security*, vol. 13, no. 2, pp. 38, February 2010.

[6] R. Dorai and V. Kannak "SQL Injection-Database attack revolution and prevention," *Journal of International Commercial Law and Technology*, vol. 6, no. 4, pp. 224–231, 2011.

[7] S. N. Narayanan, A. P. Pais, and R. Mohandas, *5th International Conference on Information Processing*, pp. 103-112, August 2011.

[8] S. Tang, N. Dautenhahn, and S. T. King, "Fortifying Web-based applications automatically," in *Proc. 18th ACM Conference on Computer and Communications Security*, 2011, pp. 615-626.

Gopi Krishnan S born and brought up in Virudhachalam, Tamil Nadu, India. Completed his M.Tech in information security at Pondicherry Engineering College, Pondicherry, India. Interested in network security, web application security, programming, image processing, and cryptography. He is also interested on Linux operating system and opensource.

He working as Research Associate at Society for Electronic Transactions and Security, Chennai, India. He worked on network perimeter product research and also involved in programming activities. He published Color image cryptography scheme based on visual cryptography, Signal Processing, Communication, Computing and Networking Technologies (ICSCCN), 2011 International Conference on, 21-22 July 2011, pp: 404 – 407.

©2013 Engineering and Technology Publishing