Vol 21, No 2 (2014): Journal of Faculty of Engineering & Technology (JFET), Pages 10-20

SECURITY TECHNIQUES AND SOLUTIONS FOR PREVENTING THE CROSS-SITE SCRIPTING WEB VULNERABILITIES: A GENERAL AAPROACH

M. Junaid Arshad[1], N. Nigar[1], H. Ahmad[2], Amjad Farooq[1], M. Usman Ghani[1], M. Adrees[1]*

[1]CS&E Department, UET, Lahore, Pakistan
[2]Virtual University (VU), Lahore, Pakistan

Abstract

Across the World Wide Web the development of social network sites is directly proportional to the complex user-built HTML contents and such things are rapidly becoming the model rather than exception. The complex user-built web messages are the threats for XSS (cross site scripting) attacks which hits different websites and private user information. In such scenario, the process that prevents web application to attacks from XSS has been of great interests for web researchers. The most of the web applications and private user information have security issues with XSS attacks. By applying such technique the attackers embed their malicious scripts onto the application's outputs. Such contaminated responses from the servers are sent to a client's web browser where it is executed and user's confidential information is shifted to a third party. Currently XSS attacks on server sides are prevented, by thoroughly observing, removing and filtering such malicious contents induced by the hackers. The criticality of XSS attack for social network sites effects even greater because a hacker can attempt more socially engineered attack where the marked user can be fooled by realizing that the attack links are initiating by his 'friends'. The proposed solutions focus on prevention methods for XSS-cross-site attacks both on the client-side and on the server-side by keeping a track of all users' information and requests. We have also discussed various recent XSS attacks in real world and have done analysis that why filtering mechanisms are so abortive and being failed in defending these attacks.

Keywords: Attacks Prevention; Filtering; Security Issues; Cross-Site Scripting-XSS

1. Introduction

[1]Web today has evolved and is still growing at a rapid pace. Complex business

[1]Corresponding Author E-mail: junaidarshad@uet.edu.pk
Phone no.: 0092-42-99029260;

applications are now being delivered over the web. More and more people are using web every day. Many social networking sites have emerged as a result of this rapid growth. As more and more data (both secure and un-secure) is available on the net, it raises a serious concern about the security of cloud computing, social networking and other websites in general. Cyber criminals have become highly effective in stealing data and getting away with it, which makes organizations and businesses around the world more and more vulnerable to cyber crime attacks. Attackers have invented new ways for attacks to exploit vulnerabilities in websites and user confidential data. In this threat climate of attacks, cross sites scripting-(XSS) has got more attention in the recent scientific research (Florian, 2007). XSS attacks are on the top rank for affecting security in the current internet era. These attacks affect user's confidential and sensitive information. Besides these, users are defrauded, authorization schemes are exploited and many more. These attacks have targeted Facebook, LiveJournal, MySpace and Orkut etc (Louw et al., 2009). Many private websites providers, companies and governmental institutions use open source web applications that are also a part of the web site. A web application assembling pages contain information in it and an attacker can gather this information from various sources. Current web applications allow the introduction of malicious contents to be executed into the client's web browser that is then served to malicious user. The most important way to gather information is through the interaction with the web pages. Mostly the primary source to retrieve information is a database that is requested by the user (Florian, 2007). In cross-site scripting attacks, the web application does not do any validation for this information provided by the malicious user and without any filtering inserts this malicious code into web page's output. The web browser starts executing the malicious content and the sensitive data is accessed by malicious code and information is provided to attacker (Figure 1). Broadly speaking, XSS is a method to inject the unauthorized script code in the web page, causing the script to be executed on a victim's web browser.

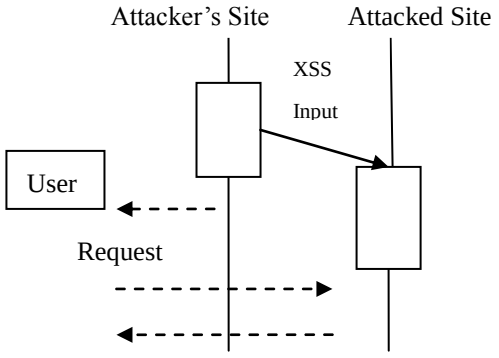

Figure 1: XSS Attack Description

2. Literature Survey

Cross site scripting is a malicious code injection vulnerability which has been found in several web applications (Elad, 2010). Hackers generally use this to control a user's session within the website. The major reason of XSS vulnerabilities is due to the improper validation of user input by the server and then sending back this invalidated input to the user in some exploitable form. XSS is an application-layer web attack technique exploit by malicious hacker. This attack commonly executes scripts on the client-side's browsers rather than on the server side. It has become a high threat caused by the internet security weaknesses of client-side scripting languages. The main idea behind the XSS attacks is that according to the desire of malicious user, client-side execution of a web-application is done.

The failure of a site to validate the server's response causes XSS vulnerability. Web server sends his output to user's web browser having malicious content and it is the core of XSS attacks. The malicious script is executed and returns all sensitive information (Joaquin, 2006). The detail analysis of XSS exploits in different cases shows how the web technology is getting loop holes to make applications more unreliable and insecure. Web results reveal many aspects of this exploit that how they target the large-scale corporation websites. Another major reason of this kind of exploit is HTML and JavaScript. This vulnerability introduces the malicious code on a website that is then served to the users. These attacks are highly ranked in making websites vulnerable. All the sensitive information is fetched using such attacks. XSS can be further explained by various examples. A basic example in which hacker injects his malicious script is online banking site URL and as a result user is met with a fake but identical page. A script is executed in client's browser to confine the cookie of user by malicious page and that cookie is now transferred to the hacker (Jovanovic et al., 2006). Many defense approaches have been used in past, some of these are briefly described below:

2.1 Content Filtering

Using this mechanism, application detects and eliminates all the malicious code in the user's request and then this filtered information is sent to web browser. Another name for this technique is sanitization. This potentially removes the malicious data from user input. Filtering mechanisms are applied after the input is read by the web application and then this filtered output is sent to the client's browser where it is being displayed to the requested user (Louw et al., 2009).

2.2 Browser Collaboration

In this defense mechanism approach, application collaborates with the browser to indicate the scripts which are authorized in the web page and which are not, so that the browser can only execute authorized pages to avoid the XSS attacks (Louw et al., 2009).

2.3 XSS-GUARD

In this technique, a new framework has been designed against XSS attacks on the server side. It works as an intelligent agent which learns a set of scripts dynamically that a web application is expected to generate for any HTML request. This approach identifies the malicious scripts and then these scripts are removed from the server's generated output that is not intended by the web application by means of a robust mechanism (Bisht, 2008).

2.4 Gateway Method

In this technique, a gateway is used at the server side to prevent the reflected XSS and request forgery attacks. Suggested Gateway at the server side protects website and all of its pages against XSS attacks. It stills functions normally while not being attacked. The correctness of this approach has been proved using a software model checker.

2.5 Data Checking at Client Side

In this technique, a track of sensitive information in the web browser is kept and this solution defends to stop the XSS attacks on the client side. If hacker attempts to steal and transfer the sensitive information, the user can decide whether hacker's action should be allowed or not. As a result, the user has an additional protection layer for their confidential information and browsing the internet and it does not solely depend on the security of the web application (Vogt et al., 2007).

2.6 X-Hunter

X-Hunter is a tool that takes input a 'web trace' and scans this input for identifying possible XSS exploits. This tool is not able to provide any defense mechanisms against attacks both in web applications and browsers. It is designed for processing and scanning thousands of URLs given as input. The output of this tool is the XSS exploits isolation in the given input. X-Hunter shows that how real XSS exploits look like and what are the triggering points of these attacks in the web browsers (Elias et al., 2010).

3. Proposed Solution

We have proposed some techniques for the detection and prevention of XSS. They have been briefly described below:

3.1 Unique Identifier Method

In this technique a unique id is assigned to each request for a web page. When a malicious user sends a request to web server having malicious code, it does not have that unique id and its validation fails on server side and no malicious code is executed in client browser and in this way XSS attack can be avoided. As a result the web browser

fetches data/code (usually written in HTML and/or another language) without having any malicious code from web servers and then displays it. This unique identifier method can be further explained by a communication between a client and server and Table 1 presents the comparison of proposed Unique Identifier Technique vs. conventional technique. User requests to server for a web page. If Id attached to the request is validated and matched then server would respond successfully and requested web page is sent to user. In reverse case a warning message is sent to user "Warning!!! Requested page has malicious code in it".

Case 1: ID is matched

Client Request:
Connection: open
GET /mainpage.html HTTP/1.1
Host: www.xss-site.com
Server Response:
HTTP/1.1 200 OK
ContentType: text/html; charset=UTF-8
ContentLength: 538
Date: Fri, 21 Nov 2011 21:31:21 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
LastModified: Wed, 31 Oct 2010 23:10:51 GMT
Unique Id: Matched
AcceptRanges: bytes
Connection: close
Warning Message!

Case 2: ID is not matched

Client Request:
Connection: open
GET /mainpage.html HTTP/1.1
Host: www.xss-site.com
Server Response:
HTTP/1.1 200 OK
ContentType: text/html; charset=UTF-8
ContentLength: 538
Date: Fri, 21 Nov 2011 21:31:21 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 31 Oct 2010 23:10:51 GMT
Unique Id: Not Matched
Accept-Ranges: bytes
Connection: close

Warning Message!

Table1: Unique Identifier Technique vs. Current Technique

	Proposed System	**Current Technique**
Dynamic	Unique key is generated at runtime	No
Secure	More Secure	Less Secure
Time	More time is taken to match a request	Less time
Human Readable	No	No
Support	Server side	Sever side
Filtering	No	Yes

3.2 Information Mapping Method

The Information Mapping Method is a systematic approach for identifying, categorizing and interrelating request. In this technique an information map is placed on server side. When a user requests from server, his request is totally mapped on server side. If request is matched with information map present on web server, response is sent to the user. When a hacker inserts his code to the request, it is modified and when it comes across for matching on server it fails and a warning message is sent to the user as shown in Fig. 2 and Table 2 presents the comparison of proposed Information Mapping Method vs. conventional technique

3.2.1 Information Mapping Algorithm

We have developed an algorithm for request matching with information map. All the requests have been placed on the server in the information map. When a request comes in, server will match that request against all the request patterns defined in information map, according to the following precedence:

- Exact matches
- Suffix matches
- Prefix matches
- The longest match, if multiple request patterns get matched.

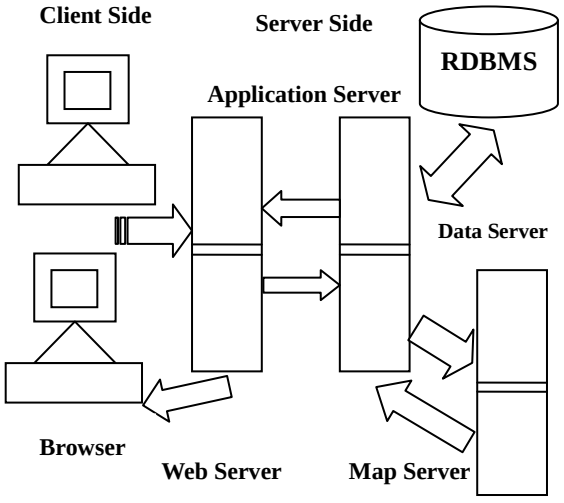

Figure 2: Information Mapping between Client and Server

Table2: Information Mapping Method vs. Current Technique

	Proposed Technique	**Current Technique**
Database Required	Yes	No
Secure	More Secure	Less Secure
Time	More time is taken to map a request	Less time
Human Readable	No	No
Support	Server side	Client side

3.3 Request Encoding

In this technique, client request is sent in encoded form. Encoded scheme is applied on the request using dynamic method. 1st character of the request will tell about encoding scheme but the remaining would be encoded using that encoding scheme (Figure 3). The attacker will not be able to detect encoding method and hence his request will be discarded. When a web page is requested through the means of Hypertext Transfer Protocol (HTTP) from web browser, encoding scheme is applied on run time. The request 1st character shows which encoded technique has been applied and all other remaining

characters would be encoded using that encoded method. And on the server side, it is decoded first and then server depending upon the requested URL, locate that file in its file system. This file can be a program or a regular file. On the other phase, the server based upon its configuration run the program and sends its output as the required page.

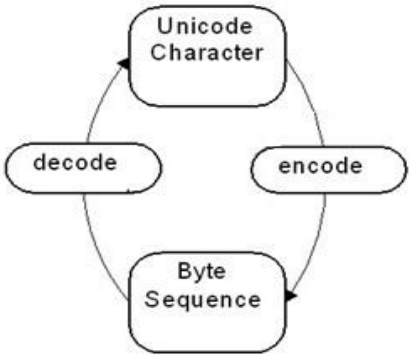

Figure 3: Encoding/Decoding Process

3.4 Request XORING

Method 1: In this technique, a XOR based character is sent at the end of each request. Since each data transmission in the computer is carried out in the form of bits or bytes. A request is also composed off bytes. When a client web browser sends request for a page, request 1st byte is XORED with the second byte and second with the third and so on. The resulting XORED byte is placed at the end of each request. When a malicious user tries to inserts his code he is unaware of this method and when server checks for the last XORED byte in the hacker's code, it fails. And in this way, XSS attacks can be avoided.
For example, the request in byte form is '01010111 01101001 01101011 01101001' can be encrypted as follows:

Request: 01010111 01101001 01101011 01101001
Xoring of 1st and 2nd byte: 01010111 01101001
Result (R1) = 00111110
Xoring of R1 and 3rd byte: 00111110 01101011
Result (R2) = 01010101
Xoring of R2 and 4th byte: 01010101 01101001
Result (R3) = 00111100
R3 (XORED character) is placed at the end of the request, now request sent is:
Request after Xoring: 01010111 01101001 01101011 01101001 00111100

Method 2: In this technique, a key is generated at run time and each byte of request is XORED with that key. When a hacker tries to inject his code, he is does not know about

dynamically generated key and unable to insert his malicious code. This is another way to avoid XSS attacks.
For example, the request in byte form is '01010111 01101001 01101011 01101001' can be encrypted as follows:

Request: 01010111 01101001 01101011 01101001
Dynamically generated key is: 11110001
Result after Xoring is:

$$\begin{array}{ll} & 01010111\ 01101001\ 01101011\ 01101001 \\ \oplus & 11110001\ 11110001\ 11110001\ 11110001 \\ \hline = & 10100110\ 10011000\ 10011010\ 10011000 \end{array}$$

3.5 Script Mapping Method

In this technique, user's request is sent to server and execution of request is decided on runtime whether to execute it or not. For example if a user requests for a page and hacker injects his malicious code to the request to hack the session or cookie of that particular user, response is sent to client's browser and browser decides to execute it or not.

HTML Browser Parser: Here we present a proposed HTML browser parser which would parse the request for making decision whether to execute it or not. The primary goal of this parser is to indicate that how arbitrary data is parsed through browser.

To further illustrate this scenario malicious data is passed through the client's browser using HTML interpretation process. The following Fig. 4 shows that HTML input (arriving through path A) flows through the web browser as it is parsed and interpreted. Browser's HTML parser and lexer process the HTML code and produce a parse tree. In next phase i.e., document generation stage, Input is a parsed tree and this input is given via path B. In this phase, web content described by parse tree are stored and interpreted. After this stage interpreted web contents are submitted to JAVA interpreter for execution via path C. This JS parse tree is submitted to via path D to javaScript Runtime Environment for execution. Our approach uses low-level Document Object Model (DOM) primitives small set that are well documented and all JavaScript-enabled browsers support it. DOM APIs are supplied with both instructions and data via path E. Browser's DOM implementation constructs the un-trusted HTML parse tree and provides to the document generator through the final transition R.

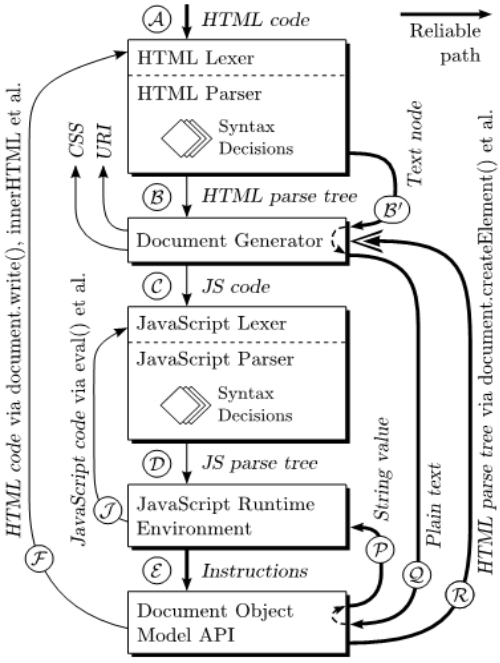

Figure 4: Generalized Browser's HTML Interpretation Process

4. Conclusions

The increasing use of web paradigm is introducing new security threats to the applications and users. Web developers should implement construction frameworks, programming models and secure coding practices to secure the web sites and free of vulnerabilities. At the other end attackers continue to search new ways to exploit web applications. These attacks have been developed to affect important critical systems in industries. In this research paper, we have explained XSS attacks and analyzed that how much great risk is involved in cross-site scripting vulnerabilities on a web application. Currently many solutions have been proposed using interesting approaches to avoid XSS attacks. But these solutions are not so much efficient and do not provide enough security.

We have concluded five more efficient techniques for XSS-attacks prevention. We believe that our proposed techniques are more scalable than other proposed solutions. Our proposed techniques are the one that prevents cross-site attacks to its maximum extent and has the potential for real world deployment given its performance characteristics. We have described the proposed techniques in detail and another contribution is the thorough analysis of the solutions and the system as a whole. We have exposed that how to deploy our proposed techniques for several web applications. They are efficient for a large number of web application and we believe it should be part of best practices. Most importantly, usability of the web sites does not get any hard impact.

References

Bisht, P., (2008), XSS-GUARD: Precise Dynamic Prevention of Cross-Site Scripting Attacks, Lecture Notes in Computer Science.

Elad, E., (2010), Flash Security, AdvancED Flex 4.

Elias A., Antonis K., and Evangelos P. Markatos, (2010), Hunting Cross-Site Scripting Attacks in the Network, In Proceedings of the 4th Workshop on Web 2.0 Security & Privacy (W2SP), Oakland, California.

Florian, K., (2007), Simple cross-site attack prevention, Third International Conference on Security and Privacy in Communications Networks and the Workshops – SecureComm.

Joaquin Garcia-Alfaro, (2007), Prevention of Cross-Site Scripting Attacks on Current Web Applications, Lecture Notes in Computer Science.

Jovanovic, N., Kirda, E., and Kruegel, C., (2006), Preventing Cross Site Request Forgery Attacks, Proceedings of IEEE International Conference on Security and Privacy in Communication Networks.

Louw, M.T., and Venkatakrishnan, V.N., (2009), BluePrint: Robust Prevention of Cross-site Scripting Attacks for Existing Browsers. In Proceedings of the IEEE Symposium on Security and Privacy.

Vogt, P., Nentwich, F., Jovanovic, N., Kirda, E., Kruegel, C., and Vigna, G., (2007), Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis, In Proceeding of the Network and Distributed System Security Symposium (NDSS'07).