IOSR Journal of Engineering
Apr. 2012, Vol. 2(4) pp: 854-857

# Detecting and Prevention Cross –Site Scripting Techniques

Tejinder Singh
*Research Scholar (JJTU), Lecturer BFGI, Deon, Bathinda,*

***Abstract***
*Cross-Site Scripting is one of the main problems of any Web-based service. Since Web browsers support the execution of commands embedded in Web pages to enable dynamic Web pages attackers can make use of this feature to enforce the execution of malicious code in a user's Web browser. To augment the users' experience many web applications are using client side scripting languages such as JavaScript but this growing of JavaScript is increasing serious security vulnerabilities in web application too, such as cross-site scripting (XSS). In this paper, I survey all the techniques those have been used to detect XSS and arrange a number of analyses to evaluate performances of those methodologies.*

***Keywords:*** cross-site scripting, injection attack, JavaScript, scripting languages security, survey, web application security.

## 1. Introduction

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications that enables malicious attackers to inject client-side script into web pages viewed by other users see fig1. An exploited cross-site scripting vulnerability can be used by attackers to bypass access controls such as the same origin policy. Cross-site scripting carried out on websites were roughly 80% of all security vulnerabilities documented by Symantec as of 2007. the malicious script is granted full access to all resources (e.g., authentication tokens and cookies) that belong to the trusted site. Such attacks are called cross-site scripting (XSS) attacks. Notably Facebook, LiveJournal, MySpace and Orkut have all been hit by these attacks. XSS attacks can be self propagating.

### 1.1 Cross-Site Scripting Attacks

Cross-Site Scripting attacks (XSS attacks for short) are those attacks against web applications in which an attacker gets control of the user's browser in order to execute a malicious script (usually an HTML/JavaScript4 code) within the context of trust of the web application's site. As a result, and if the embedded code is successfully executed, the attacker might then be able to access, passively or actively, to any sensitive browser resource associated to the web application (e.g., cookies, session IDs, etc.). Two main types of XSS attacks: persistent and no persistent XSS attacks (also referred in the literature as stored and reflected XSS attacks).

## 2. Cross-site scripting attack mechanism

Users interact with a dynamic web site by clicking on links or filling in and submitting forms, which results in a list of name/value pairs being sent to the server in the form of an http request. The request can contain other information such as a list of cookies, the referrer URL, etc. In general, any data in the request should be considered as untrusted. What most web pages interact with, however, is the list of name/value pairs. Within a J2EE envelopment environment, a dynamic web page receives the input values as Java strings by calling standard methods provided by the Servlet or JSP container. The Java strings can be stored and/or used to form an HTML page as the response to the request. Problem arises when the input string values contain characters that are considered special (markup character) under the HTML specification. For example, suppose a Hello Servlet takes a username input and produces an HTML page that prints the string "Hello" followed by the username: String username = request.getParameter("username");
response.getWriter().println("<html> Hello "+username+"</html>");
If the username is "foo", the following HTML is sent to the browser: <html> Hello foo </html> However, if username is "foo<b>", the following is sent: <html> Hello foo<b> </html> The sub string "<b>" as part of the username will not be displayed as it is treated as an HTML tag. With this observation, a malicious user can produce an input such as "foo <script> ... </script>", the resulting HTML would be: <html> Hello foo <script> ... </script> </html> When a browser receives the HTML, the browser will try to execute the scripts between the script tags. JavaScript programs are treated as entrusted software components that have only access to a limited number of resources
within the browser. Also, JavaScript programs downloaded from different sites are protected from each other using a compartmentalizing mechanism, called the same-origin policy. This limits a program to only access resources associated with its origin site. Even though JavaScript interpreters had a number of flaws in the past, nowadays most web sites take advantage of JavaScript functionality. The problem with the current JavaScript security mechanisms is that scripts may be confined by the sandboxing mechanisms and conform to the same-origin policy, but still violate the security of a system. This can be achieved when a user is lured into downloading malicious JavaScript code (previously created by an attacker) from a

trusted web site. Such an exploitation technique is called a cross-site scripting (XSS) attack.

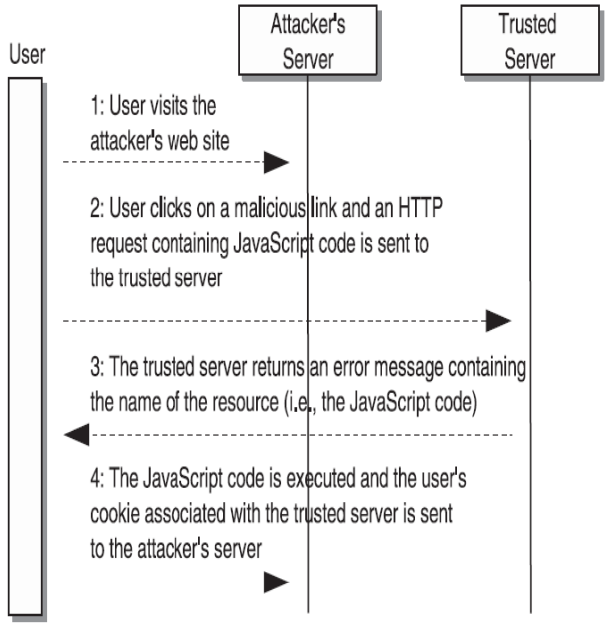


Fig. 1- A typical cross-site scripting

### 2.1 Types of XSS attacks

Three distinct classes of XSS attacks exist:
DOM-based attacks, stored attacks, and reflected attacks . In a stored XSS attack, the malicious JavaScript code is permanently stored on the target server (e.g., in a database, in a message forum, or in a guestbook). In a DOM-based attack, the vulnerability is based on the Document Object Model (DOM) of the page. Such an attack can happen if the JavaScript in the page accesses a URL parameter and uses this information to write HTML to the page. In a reflected XSS attack, on the other hand, the injected code is ‘‘reflected’’ off the web server, such as in an error message or a search result that may include some or all of the input sent to the server as part of the request. Reflected XSS attacks are delivered to the victims via e-mail messages or links embedded on other web pages. When a user clicks on a malicious link or submits a specially crafted form, the injected code travels to the vulnerable web application and is reflected back to the victim’s browser. The reader is referred to for information on the wide range of possible XSS attacks and the damages the attacker may cause. There are a number of input validation and filtering techniques that web developers can use in order to prevent XSS vulnerabilities However, these are server-side solutions over which the end-user has no control.

## 3. Defense approaches

To disallow script execution in untrusted web content, a web application might possibly take one of the following approaches.

**Content Filtering**. The application may attempt to detect and remove all scripts from untrusted HTML before sending it to the browser.

**Browser Collaboration**. The application may collaborate with the browser by indicating which scripts in the web page are authorized, leaving the browser to ensure the authorization policy is upheld.

**Content filtering.** Content filtering is otherwise known as sanitization. This defense technique uses filter functions to remove potentially malicious data or instructions from user input. Filter functions are applied after user input is read by a web application, but before the input is employed in a sensitive operation or output to the web browser.

### 3.1 Cookies and cross-site scripting

A **cookie**, also known as a **web cookie**, **browser cookie**, and **HTTP cookie**, is a text string stored by a user's web browser. A cookie consists of one or more name value pairs containing bits of information, which may be encrypted for information privacy and data security purposes. The cookie is sent as an HTTP header by a web server to a web browser and then sent back unchanged by the browser each time it accesses that server. A cookie can be used for authentication, session tracking (state maintenance), storing site preferences, shopping cart contents, the identifier for a server-based session, or anything else that can be accomplished through storing textual data. As text, cookies are not executable. Because they are not executed, they cannot replicate themselves and are not viruses. However, due to the browser mechanism to set and read cookies, they can be used as Spyware. Anti-spyware products may warn users about some cookies because cookies can be used to track people. Many web applications rely on session cookies for authentication between individual HTTP requests, and because client-side scripts generally have access to these cookies, simple XSS exploits can steal these cookies.

### 3.2 Cryptography

Until modern times cryptography referred almost exclusively to *encryption*, which is the process of converting ordinary information (plaintext) into unintelligible gibberish (i.e., *ciphertext*)[16]. Decryption is the reverse, in other words, moving from the unintelligible ciphertext back to plaintext. A *cipher* (or *cypher*) is a pair of algorithms that create the encryption and the reversing decryption. The detailed operation of a cipher is controlled both by the algorithm and in each instance by a *key*. This is a secret parameter (ideally known only to the

communicants) for a specific message exchange context. Keys are important, as ciphers without variable keys can be trivially broken with only the knowledge of the cipher used and are therefore useless (or even counterproductive) for most purposes. Historically, ciphers were often used directly for encryption or decryption without additional procedures such as authentication or integrity checks**.**

## 4. Proposed method

As we stated a cookie can be stolen and the privacy of its user can be violated. There is some solution to prevent attackers to steal cookies by XSS attacks as mentioned above. Although this methods maybe robust and effective but they cannot prevent the stealing of the cookie in some circumstances. Consider another situation in which the user can get his (her) cookie and change some data stored in it. For Example following Proposed Method steps are:

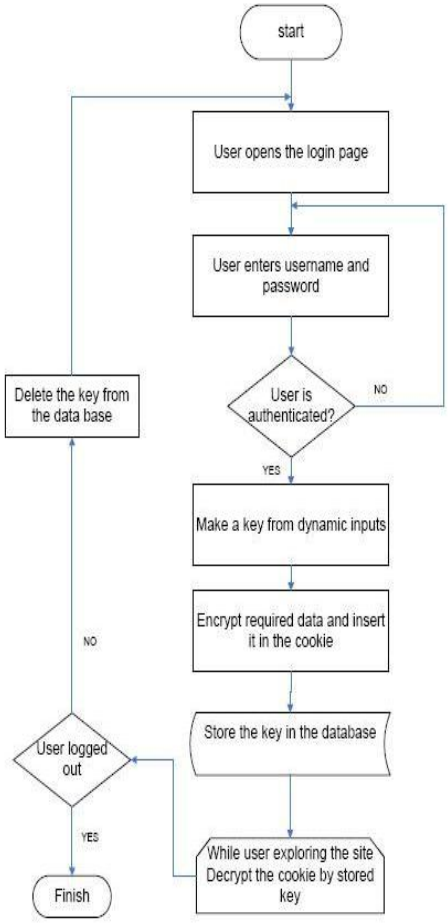


Fig. 2- Our proposed method

### 4.1 Prevention method I

The simplest (and perhaps most performance ) form of prevention for this type of attack is to restrict the valid input to be free of characters that have special meanings under the HTML pecification. For example, if the value of a user input should be a number, and is validated by the web application as such, we are sure that it cannot be used to launch a cross-site scripting attack. A common problem in software development is that developers tend to give too much freedom in terms of what values an input can take. Does an input value have to allow for characters such as < and double/single quotes? Many developers ignore such issues at design time or choose for unnecessary flexibility. A reasonably restrictive input set can often greatly simplify a program.

**Prevention method II**

If it is infeasible to restrict the content of the input, another effective method is to encode/escape the user input on output. The first point is performance. The encoding method requires transferring a string into another where all occurrences of HTML special characters in the original string be replaced with their entity representation (e.g. replace < with <). If a lot of the encoding is needed in each generated HTML page, care should be taken to make sure that the encoding method is performance (Java string manipulations can be slow if not coded properly). The second point is regarding charset. it is important to have the correct charset set in the HTTP response header. In servlet or JSP development, you can do the following respectively to set the charset:
response.setContentType("text/html; charset=..."); <%@ page contentType="text/raw;charset=..." %>

## Conclusion

Cross-site scripting attack is a valid security threat to dynamic web sites. It is regarded as one of the top security flaws existing in today's dynamic sites. With the attack method becoming more mature and automated, and the fact that more dynamic web sites are being set up, we can expect the problem to become worse. While a number of articles have given examples of the attack, as well as testing, prevention, mitigation methods, this paper attempted to focus on the prevention methods, giving more details especially in J2EE development environments. We hope that the technical details provided will help developers understand and protect their applications against this attack. One of the most prolific problems plaguing the security sector today is Cross Site Scripting (XSS). Yet it is rarely taken seriously. XSS exploits web application vulnerabilities which impact on the end user, so few application developers or their organizations pay much attention to XSS. To develop secure web applications, you have to avoid these three pitfalls, insufficient handling of malicious inputs, deficiencies of native execution models, and not enough support for enforcing same origin policies.

## References

[1] Alcorna, W. Cross-site scripting viruses and worms – a new attack vector. Journal of Network Security, 2006(7):7–8, Elsevier, July 2006.
[2] Anderson, A. and Lockhart, H. SAML 2.0 profile of XACML v2.0. Standard, OASIS. February2005.
[3] Amit, Y. XSS vulnerabilities in Google.com. November 2005. http://www.watchfire.com/securityzone/advisories/12-21-05.aspx
[4] Anupam, V. and Mayer, A. Secure Web scripting. IEEE Journal of Internet Computing,2(6):46–55, IEEE, 1998.
[5] Google. Docs & Spreadsheets. http://docs. google.com/
[6] Google. Orkut: Internet social network service. http://www.orkut.com/
[7] Grossman, J., Hansen, R., Petkov, P., Rager, A., and Fogie, S. *Cross site scripting attacks:XSS Exploits and defense.*. Syngress, Elsevier, 2007.
[8] Hallaraker, O. and Vigna, G. DetectingMalicious JavaScript Code inMozilla. *10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05)*, pp.85– 94, 2005.
[9] Cook, Steven. "A Web Developer's Guide to Cross-Site Scripting." 11 Jan 2003. URL: http://www.sans.org/rr/ papers/46/988.pdf (4 May 2004)
[10] Sun Microsystems. "Java Servlet Technology" URL: http://java.sun.com/products/servlet/ (4 May 2004)
[11] ModSecurity: Features: PDF Universal XSS Protection". Breach Security. Retrieved June 6, 2008.
[12] Whitfield Diffie and Martin Hellman, "New Directions in Cryptography", IEEE Transactions on Information Theory, vol. IT-22, Nov. 1976