XSS Attack: Detection and Prevention Techniques

Monika Rohilla
Research Scholar, Department of Computer Science & Applications, KurukshetraUniversity,Kurukshetra
Email: mnrohilla508@gmail.com
Rakesh Kumar
Professor, Department of Computer Science & Applications, Kurukshetra University, Kurukshetra
Email: rakeshkumar@kuk.ac.in
GirdharGopal
Assistant Professor,Department of Computer Science &Applications,KurukshetraUniversity,Kurukshetra
Email: girdhar.gopal@kuk.ac.in

Abstract:
Web applications provide access to online services. Web application's security is the most critical part of web development. The attacker can exploits the vulnerabilities of web applications by injecting the malicious code in application which results in theft of cookies and other credential information. Cross site scripting (XSS) attack is one of the web application vulnerabilities. This paper discusses about various techniques to detect and prevent XSS attacks like sanitization, input validation, web proxy, Browser Enforced Embedded Policy (BEEP), Saner, deDcaota, NOXES etc. The details of these techniques with their shortcomings have been conducted so that one can use these techniques and tools as applicable to avoid the XSS attacks on Web applications.

Keywords- Cross site scripting (XSS), Detection, Prevention, Vulnerability

1. Introduction

XSS attack is a type of vulnerability mainly found in web applications and discovered continuously at alarming rate. XSS attack operates at application layer. XSS attacks are very common, an attacker can easily find out the vulnerable web application. Cross site scripting attacks are those attacks in which attacker inject the malicious script usually client side script from outside the web application environment. Designing a secure web application is not an easy task because all web applications require a user interface so these need to be interactive, accepting and providing the data from users. The original CERT advisory defined cross site scripting attacks as a means by which "malicious HTML tags or script in a dynamically generated page based on invalidate input from untrustworthy sources" [1]. XSS attacks depends on browser capability to distinguish between legitimate content served by a web application and content that has been injected into web application's output. An attacker injects a script into code which executes at client side and affects the environment. There are various client side scripting languages, java script [2] is the most widely used scripting language. Java script can be used to extract the information from browser cookies and send it to the attacker for further use. For example:

<A href="http://www.trusted.com "/>
<SCRIPT>
document.location = 'http://malicious.com/steal cookie.php?' + document.cookie
</SCRIPT>
Click here to collect items
</A>

When any user clicks on the above link, HTTP request is sent to the trusted web site (http://www.trusted.com) by the web browser.

<SCRIPT>
document. location =
"http://www.malicious.com/steal-cookie.php?
+document.cookie
</SCRIPT>

When a trusted web site receives request, server decides to include the required file (script) and browser executes the script. Then cookies set by trusted web site will be sent to http://evil.com.

These cookies will be saved in attacker server. An attacker can use these cookies to impersonate the suspected user with respect to trusted site.

1.1 Vulnerability associated with XSS Attack

Vulnerability is a weakness of web application that can be exploited by an attacker to gain unauthorized access. Cross site scripting has various vulnerabilities, some of them are discussed below.

i. Vulnerability occurs if the incoming input to the web application is not properly validated.
ii. If a malicious url is present in the web site, by clicking on malicious url attacker connects a user to malicious server of his choice for accessing personal information.
iii. Attacker can easily steal the session information.
iv. User unknowingly executes the script when he visits web application.
v. Attacker provides various trusted links, by clicking on those links, user redirected to another web site.
vi. Attacker can hack user account and fetch the credential information, can make misuse of user cookies and place false advertisement to web site.

This paper is organized in following section, section 2 describes the types of attacks and section 3 describes the literature review of cross site scripting attacks detection and prevention techniques. Section 4 presents the conclusion of paper.

2. Types of XSS Attacks:

XSS attack can be of three types-

- Persistent attack
- Non persistent attack
- Document Object Model (DOM) based attack

2.1 Persistent Attack

This is very powerful attack that can be spread to millions of people at the same time. A malicious script is injected in web application and is permanently stored on the server. When a user requests some information from server, injected script is reflected by a server as an error message or search result. Persistent XSS attacks are delivered to users via email or link embedded on some other web page. Persistent attacks are less frequent than non-persistent attack. For exploiting the stored XSS vulnerabilities, firstly we have to find out the vulnerable web site that can be used to carry out an attack. The vulnerabilities which make it possible are difficult to find. There are some web applications that allow sharing of contents and vulnerable to persistent attack like Forums / message boards, blogging websites, social networks, web-based email server consoles and web-based email clients. The malicious code is usually delivered by the attacker in input fields of vulnerable web applications. The damage that persistent attack can do is more destructive than damage done by non-persistent attacks.

2.2 Non Persistent Attack

It is the most common type of XSS vulnerability. It targets the website vulnerability that deals with dynamic property of web application. Every input has potential to be an attack vector. When users submit data, it is immediately processed by web server to generate the result that is sent back to the browser. If a web application has lack of encoding schemes and user input validation methods, an attacker can inject malicious URL with harmful script code. A non-persistent attack is typically delivered via emails, social networking sites and malicious links on the web site.

Here is Php code that can suffer from non-persistent XSS attack.

```
<? php
If(!array_key_exist("name",$_GET) || $_GET['name']==NULL|| $_GET['name']=="")
{
$isempty=true;
}
else{
echo '<pre>';
echo 'hello'. $_GET['name'];
echo '<pre>';
}
?>
```

As you can see that the "name" parameter is not sanitized and echo back to the user. When the user injects a malicious Java Script code, it will be executed by the browser and a malformed result is shown to user.

2.3 DOM Based Attack

DOM (Document Object Model) is a client side injection. Entire code is originated from the server that means it is developer's responsibility to make a safe web application. All XSS attacks are executed at the browser. The main difference between these attacks is where the code is injected into the application. In DOM [3] based attack the code is injected from client side during the run time. The prior condition for a DOM attack, site is vulnerable to attack and contains HTML pages that use data from the document.location, document.url or document.referrer.

3. Literature Review

Various approaches have been implemented for detecting the different types of XSS attacks. There is no standard technique to mitigate and prevent all types of attacks. A variety of techniques can be used for the prevention of client side and server side attacks. Some of these techniques are as follows.

3.1 Detection of XSS attack

A static string analyzer [4] checks the string output of a program with context free grammar. This technique checks the presence of "<script>" tag in the whole document. As web applications more often have their own scripts and also there are several other ways to invoke a JavaScript interpreter, the approach is not at all practical to find XSS vulnerabilities. Web request and server response [5] are used to detect the XSS attack. The web request parameter passed to the HTML parser. They modified the HTML, JAVA script tags, method, method calls and expression with tokens. The script engine is used to detect the server response. For detection of web request, input is analyzed by analyzer and extracts the malicious links and malicious script as a request parameter. The features are extracted based on syntax tree and compared to the white list. If malicious tags and malicious scripts are present, alert message of XSS is generated. There was no requirement of modification in browser or server engine. It has weak validation from client side and server side. The web application vulnerability is detected by a static analysis tool pixy [6]. The authors address the problem of vulnerable web application by means of static source code analysis. The flow sensitivity data flow analysis was used to discover the vulnerable point in program. Non persistent attack is detected by matching the incoming data and outgoing java script using a simple metric like matching the incoming data to HTML java script code. Johns purposed a prototype for detection of reflected XSS attack [7]. The authors define two tasks that require special attention: Script extraction and script parsing. Reliable script extraction: Detection of reflected attack is a very critical task. The browser is used for their implementation, rendering all illegitimate HTML tags. The identification of all the script is a non-trivial task [8]. HTML rendering engine of Mozilla Firefox browser is modified for exactly matching the script that is executed by the browser. Java script parser can easily find out the java script string constant that are valid for java script code. Their implementation is tested on HTML based attack vector [8] and they detect all the attacks reliably but cannot detect other than HTML based attacks. Script parsing: Both of their detectors rely on java script tokenizer for preprocessing. The implementation is done in Ruby. The tokenization step in XSS detector is a bottleneck in performance. Jevitha using machine learning algorithm [9] (Naïve Bayes, Support vector Machine and J48 Decision Tree) they classify a normal page and malicious page based on the feature extracted from URL and java script code. The web pages are collected which are infected by java script. The authors extract the java script code and URL from web pages. The database is created according to some specified features which are identified from java script and URL. This database is used for training the

algorithm. After experiment, the authors find that J48 provides better performance with respect to FPR (False Positive Rate). But time required for file creation was greater than NB (Naïve Bayes) classifier but less than SVM (Support Vector Machine).

3.2 Prevention of XSS Attack

Prevention of XSS attacks can be distinguished according to prevention techniques employed at server side, client side and at the both sides.

3.2.1 Server side prevention of XSS attack

To prevent the cross site scripting in server side, White box vulnerability scanner and black box [10] web application testing tool is purposed by various researchers. To identify the technical flaws, scanner is the best. A proxy firewall AppShield [11] is used to mitigate the XSS attack by learning from the traffic to a specific application. AppShield is a plug and play application that provides limited protection from attacks because it has lack of security policies. A major drawback of this solution is that it protects web application at deployment phase rather than development phase. Server side sanitizer prevents XSS attack by checking existing sanitization correctness. It generates input encoding to match usage context. In sanitization, clean up the untrusted input that might contain java script code. Correct sanitization is a challenging task in web application. It is difficult to guarantee that all part of web application are covered [12][13]. Firstly, find all the paths through which attack can be done. A single input might appear in different context in the output of application[14]. HTML input filter [15] is used against the security of web application in browser side. A server side solution is proposed against XSS attacks, this solution is not depends on web application provider. Cross site scripting mitigation mode reduces the XSS alert prompt. The authors reduce the amount of information leakage in browser side. Mainly the XSS attacks are based on injecting the malicious java script code in web pages that is why filtering of web pages are necessary. A server side solution allows easy integration of filter (Java Script Filter) in java based application. A novel approach Dynamic Hash Generation[16] makes the cookies useless for the attacker. This approach is easy to implement on web server without any changes required on web browser. In this technique, a hash value is calculated at the server side for the name attribute of cookies and then this sent to the browser. This hash value is used by the web server to authenticate the user at the browser side. In this experiment, the version 0 cookie is used. The purposed technique does not affect the performance of client side web browser and there is no single point of failure. It is a server side solution; it affects the performance of whole system. It takes time to generate the hash value at the server side. This technique is not work with version 1 cookie because it adds an extra attribute, Port Number. Major disadvantage of this solution is that it does not intercept the HTTP and SSL connection.

The code and data is separated from the web pages. deDacota [17]statically rewrites the existing application to separate the code and data. The static analysis is used to find out all inline java script code in web pages. In static analysis it is not easy to find out all HTML output. Some benign developer computes the HTML output statically. There is a second order problem, dynamic inline script. deDacota provides the partial solution to this problem. It provides alert message for all dangerous instances of dynamic java script generation in web application and safely sanitizes these instances. A prototype of deDacota is implemented to analyze and rewrite ASP.Net web applications. The authors applied this tool on six open source web applications and found that all the XSS vulnerabilities are eliminated. The performance test is applied to check the functionality of web application and they found, there is no difference in page loading time of original and rewritten applications.

3.2.2 Client side prevention of XSS

In general XSS attacks are easy to execute, but difficult to prevent. A client side solution is not easy, because malicious java script code is difficult to identify. NOXES [18]was the first

client side solution to mitigate the cross site scripting attack. It was a Microsoft-Window based personal web firewall application. It acts as a web proxy. The links which contain HTML elements with src, href attribute, url and CSS (Cascading Style Sheet) file is extracted. All links are passed through the NOXES can either be blocked or allowed based on the current security policy. It allows the user to create rules for web requests. It track the links visited by the browser and automatically create the permanent filter rule based on the specific domain collected during the session. NOXES focuses on ensuring the confidentiality of sensitive data by analyzing the flow of data through the browser. Duraisamy [19]described web proxy to protect the information leakage from the user environment. Client side solution does not depend upon web application provider. This solution mitigates the cross site scripting attack and reduces the number of connection alert. Shalini [20]purposed a model that provides a client side solution which does not degrade the performance of application. It provides efficient security from the XSS attacks with optimized web browsing. The implementation is done using open source Mozilla Firefox 1.5 web browser. They compared the purposed browser with Firefox, Microsoft's Internet Explorer, Apple's safari web browser and other available web browser on some platform and environment. The test is performed using data collected from the white hat and black hat sites. This system successfully detects and removes a number of XSS attacks.

3.2.3 Client side as well as server side prevention of XSS

Static and dynamic analysis is used to identify the faulty sanitization procedure that can be bypassed by an attacker. A tool saner [12] was used for implementation. Experiment was done on various real world applications. They identified the novel vulnerability that stem from incorrect and incomplete sanitization. A static analysis tool [6] was used for detecting the web application vulnerability. Flow sensitivity, inter procedural and context sensitivity data flow analysis was used to discover vulnerable point in program. Static analysis provide false positive. If the number of false positive is large it means site is vulnerable to XSS attack, so they perform dynamic analysis. Goal of dynamic analysis was to examine the entire path from source to sink. Dynamic analysis was performed by checking the code with various input value which have different way of encoding, then try to understand which type of input be a cause of security violation. In Browser enforced embedded policy[21] modified the web application and embedded some policies. Policy contains a hook function that will run before execution of any script. Modified browser check each java script to security hook function. The policy was flexible in nature. In first policy they used whitelist in which hook function uses one way hash to script. When browser parses a script, check it with security hook function . If hook returns true then script is legitimate otherwise it will be rejected. Second policy was DOM sandboxing takes a blacklist. Modifications to browser and web application are not difiicult to perform but some browser can not support hook function. BEEP did not provide any guidance to trust on third party. According to our web application it suffers from scalability problem.

4. Conclusion

XSS attacks are easy to perform but difficult to prevent. Several approaches have been proposed to detect and prevent the XSS attacks. These solutions are either client side or server side that protects a web application against XSS attacks like web proxy firewall at application layer. Scanner best suited to identify the technical flaws but less capable to recognize business flaws. White box tool exactly find the vulnerability, why and how they are occurred but generate a large number of false positive. Generally solutions are depending on the service provider to aware of the XSS attack and take appropriate action to mitigate the threat. But existing approaches are not sufficient for traditional web applications, some of them are relying on end user for protection of key aspect of a service. In

this paper author discusses detection and prevention techniques of the XSS attacks with their merits and demerits.

References

[1] CERT. Advisory, "Malicious HTML tags embedded in client web request," 2000.

[2] ECMA-261, "ECMA Script Language Specification," 1999.

[3] OWASP, "DOM Based XSS - https://www.owasp.org/index.php/DOM_Based_XSS," 2013.

[4] Y. Minamide, "Static Approximation of Dynamically Generated Web Pages," in *WWW '05 Proceedings of the 14th international conference on World Wide*, New York, NY, USA, 2005.

[5] H. Vigna, "Detecting malicious JavaScript code in Mozilla," in *10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05)*, 2005.

[6] N. Jovanovic, C. Kruegel and E. Kirda, "Pixy: a static analysis tool for detecting Web application vulnerabilities," in *2006 IEEE Symposium on Security and Privacy (S&P'06)*, Berkeley/Oakland, CA, 2006.

[7] M. Johns, B. Engelmann and J. Posegga, "XSSDS: Server-Side Detection of Cross-Site Scripting Attacks," in *Computer Security Applications Conference, 2008. ACSAC 2008. Annual*, Anaheim, CA, 2008.

[8] Hansen, "Cross- Site Scripting Cheat Sheet," 05 05 2007. [Online]. Available: http://hacker.org/XSS.html.

[9] V. B. A and J. K. P, "Prediction of Cross Site Scripting Attack Using Machine Learning Algorithm," in *ICONIAAC*, Amritapuri India, 2014.

[10] Acunetix, 2008. [Online]. Available: http://www.acunetix.com.

[11] D. Scott and R. Sharp, "Abstracting application-level web security," in *WWW '02 Proceedings of the 11th international conference on World Wide Web*, ACM New York, NY, USA, 2002.

[12] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel and G. Vigna, "Paper Review: Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications.," in *Security and Privacy, 2008 IEEE symposium*, Oakland CA, 2008.

[13] B. Akhawe, F. Saxsena and S. Weinberge, "A Systematic Analysis od XSS Sanitization in Web Application Framework," in *ESORICS'11 Proceeding of the 16th European Conference on Research in Computer Security*, 2011.

[14] P. Saxena, D. Molnar and B. Livshits, "ScriptGard: Automatic Context-Sensitive Sanitization for Large-Scale Legacy Web Applications," in *CCS'11*, Chicago, Illinois, USA, 2011.

[15] A. Duraisamy, M. Sathiyamoorthy and S. Chandrasekar, "A Server Side Solution for Protecting of Web Application from Cross Siye Scripting Attack," *International Journal Of Innovative Technology and Exploring Engineering (IJITEE)*, vol. 2, no. 4, March 2013.

[16] S. Gupta, L. Sharma , . M. Gupta and S. Gupta, "Prevention of Cross-Site Scripting Vulnerabilities using Dynamic Hash Generation Technique on the Server Side," *International Journal of Advanced Computer Research* , vol. 2, no. 5, pp. Start Page- 49, 2012.

[17] A. Doupe, W. Cui and M. H. Jakubowski, "deDcota: Toward Preventing Server - Side XSS via Automatic Code and Data Seperation," in *CCS'11*, Berlin Germany, 2013.

[18] E. Kirda, N. Jovanovic, C. Kruegel and G. Vigna, "Client - Side Cross Site Scripting protection," 2009.

[19] Duraisamy, Kannan and Selvamani, "Protection of Web Application from Cross

Site Scripting Attack in Browser Side," *IJCSIS*, pp. 229-236, 2010.

[20] S. and U. , "Prevention of Cross Site Scriptig Attack (XSS) on Web Application In The Client Side," *IJCSI International Journal Of Computer Science Issue*, 2011.

[21] T. Jim, N. Swamy and M. Hicks, "Defeating Script Injection Attacks with Browser - Enforced Embedded Policies," in *Interntional World Wide Conference Committee (IW3C2)*, Canada, 2007.