Detection and prevention methodologies against Cross-Site Scripting vulnerabilities: A review

DESAI Purva N.
JJT University, Jhunjhunu, Rajasthan, India
purva.desai79@gmail.com

Dr. PANDYA Jagdish G.
JJT University, Jhunjhunu, Rajasthan, India
pandyajg@yahoo.com

Abstract

Specially, when Internet has become handy and most advance useful technology due to smart phone, on other side cyber attackers are becoming limitless threat for their personal interest to the entire world. Now-a-days, cross-site scripting is a worst menace of web attack from attacker which is operates at application layer. Researchers and industry experts say in web application, XSS is one of the top most vulnerability. In this menace injected malicious code executes on browser site which affects victims badly. This paper presents study of various forms of cross-site scripting attacks. The aim of the paper is to review the research undertaken to discuss various methodologies that prevent cross-site scripting attacks.

Keywords: Cross-site scripting (XSS), Stored XSS, Reflected XSS, DOM-Based XSS

1. Introduction

At present, Internet has become very popular. The services provided by Internet help the present human civilization to such an extent that life without internet seems to be impossible. More than thousands of web sites are uploaded on daily basis to provide variety of customized services that become key part of man’s day-to-day life.

Due to its omnipresence, Internet has started attracting hackers/attackers who keep looking for new techniques to create vulnerabilities in web application. The Cross-site Scripting (XSS) is the one of the top most vulnerability in the web application.

A cross-site scripting is a one kind of application-layer web attack in which attackers try perform malicious actions on trusted websites by injecting malicious scripts. According to Shalini and Usha[1], XSS is called “cross-site” because it involves interaction between two separate web sites to achieve attacker’s goal. XSS executes at browser side and harm the users.

XSS generally execute when the web page is loaded or associated event is occur. XSS can be generally embedded in JavaScript and HTML. But it can be also embedded in VBScript, ActiveX,
AJAX, ActionScript like Flash, or any other browser executable scripting language and mark-up language. XSS can be used for many reasons like: Take over user’s account, Spread worms, Trojan horse, Control access of browser, Phishing, Expose of the user’s session cookie, Redirect the user to some other page or site, Modify presentation of content, Bypass restrictions, Malware attacks & DoS attack, Fake advertisement, Click fraud, etc.

2. Types of XSS attacks

Earlier there are two types of XSS attacks were identified i.e. Stored XSS & reflected XSS. Third type of XSS called DOM-Based is introduced by Amit Klein in 2005. These types are defined as follow:

2.1 Stored XSS (AKA Persistent XSS)

As shown in figure 1, stored XSS usually occurs when user’s input contains XSS vectors and is stored in database on the target server. It may be injected through blog, forum, visitor log, bulletin board, comment field of feedback, etc. Whenever the victim opens that webpage through the browser, the XSS script vectors executes automatically and it may sends user’s confidential information to attacker.

2.2 Reflected XSS (AKA Non-Persistent XSS)

Reflected attacks are those where the injected script is reflected off the web server, e.g. in an error message, search result, or any other response that includes input sent to the server as part of the
request. Reflected attacks are delivered to victims via e-mail message, links or by some other web site.

When a victim is misled by clicking on a crafted link, submitting data into fake form, or browsing to a fake site, the injected code flows to the vulnerable web site, which reflects the attack back to the user’s browser. The browser then executes the code because it came from an "authorized" server.

![Figure 2: Reflected XSS (AKA Non-Persistent XSS)](image)

### 2.3 DOM-Based XSS

DOM XSS is type cross site scripting attack which arises due to improper handling of data related with DOM (Document Object Model) present in HTML page. Every HTML entity present in page can be accessed and modified by using DOM properties like document.referrer, document.url and document.location. Attacker can manipulate or access DOM properties to execute such type of attack. Here, the malicious script does not reach to the web server. It is executed at client site only. DOM based XSS attack occurs when user provided untrusted data is interpreted as JavaScript using methods such as eval(), document.write() or innerHTML.

![Fig 3: DOM-Based XSS](image)
3. Review of Literatures

As discussed earlier, due to omnipresence of Internet, Internet has started attracting hackers/attackers who keep looking for new techniques to create vulnerabilities in web application. According to researchers and industry experts (like CENZIC, CISCO, OWASP, WhiteHat Security), the Cross-Site Scripting (XSS) is the one of the top most vulnerability in the web application. So as solution, various past studies have been carried out. This review of literatures helps us to analyze past and present scenario of different trends and technologies work towards defending such kind of web vulnerabilities cause through cross-site scripting attacks which harm web application as well as its authorized user.

Kirda et al. (2006) [2] developed client-site firewall known as Noxes tool which works as application-level firewall. This approach depends on web traffic. This firewall blocks or allows web-site connection on the bases of filter rules. Rules contain list of allowable or blocked URLs which are user specific. It alert user if web request is not mentioned in filter rules. This method requires continuous user’s interaction to update filter rules of this firewall. This tool does not provide solution for stored XSS and DOM-based XSS attacks.

Jim et al. (2007) [3] presented Browser-Enforced Embedded Policies known as BEEP, expects to use modified web browser that checks execution attempts of all scripts and policies provided by server. Here, authors suggest two types of policies: 1) List of hashes are maintained which are white-listed by web application. 2) Highlights nodes in HTML source which contains user provided inputs, so the browser can determine the script’s position in DOM. The modified browser examines the outcome of JavaScript execution by relating it with policy file. If policy allows such JavaScript then it is executed otherwise it is blocked. This proposed approach requires modification in server software as well as in the client browser. Such modifications required lots of efforts with accuracy and consistency. This approach does not contain any mechanism that protect against reflected cross-site scripting attacks.

Johns et al. (2008) [4] developed passive detection system called XSSDS that works at server-site to detect cross-site scripting attacks. It follows two approaches which are based on cross-site scripting attack observation and web application. Here, reflected XSS attacks are identified by comparing request and response that have direct relationship between input and script injection. For this, it removes all encoded code found in parameters of input and in script then each input parameters are compared against DFA list. To detect stored XSS attacks, they maintain list of known scripts. Here all outgoing script is compared with this known script list. This method detects only listed known XSS attacks. This proposed method does not provide solution against DOM-based attack.

Wurzinger et al. (2009) [5] proposed server-site solution called secure web application proxy (SWAP) to detect and prevent cross-site scripting attacks. It works as reverse proxy that checks all HTML responses. It also modifies web browser to check contents of script. This proxy has JavaScript detection module which checks script is there or not by decoding encoded script code. It blocks HTML response of server until script is analyzed by JavaScript detection module. This approach does not check URLs which points to malicious site. Solution for DOM-based XSS is also not included here.

Galan et al. (2010) [6] developed system that scan web site automatically to detected stored XSS attack. In this system, web page parser agent finds input points of application that can be vulnerable. Script injector agent injects selected attack vectors at this point. Verificator agent inspects whole flow path of web application in order to identify success of attack. Report regarding scanning process is evaluated and stored. Here, proposed system only detects attacks using
predefine attack vector. It does not provide any kind of prevention solution against different categorized XSS attacks.

Stephen et al. (2011)[7] proposed algorithm called E-Guard that prevents against cross-site scripting attacks. Here authors maintain whitelist, black list and gray list. Whitelist and black list is used to find vulnerabilities if web site is known web site. If web site is new then source code of web page is compared against gray list. It scan web site and count number of black list as well as whitelist. If count of matches with white list is more than that of black list, it marks web site as white list. It only detects attacks which are based on stored XSS and reflected XSS. This algorithm does not contain prevention mechanism against different types of cross-site scripting attacks.

Choudhary and Dhore (2012) [8]developed two modules namely script detector and query detector to deal with code injection attacks. Script detector filter out all invalid tag from request using sanitizing process and encodes it before passes it to the server for further processing. Query detector maintains repository of attack vectors. User supplied data in request compared against this repository. This approach does not provide solution for DOM-based XSS attacks.

Doupe et al. (2013)[9] presented approach that automatically recreates application in which code and data is separated in new web pages. It also keeps application’s semantics as it is. Here, all JavaScript inline code is shifted to external files that browser will execute on Content Security Policy basis. DeDacota only protects against server-side XSS attacks. If attack vector is available in HTML attributes than it does not detected. This approach does not provide solution for DOM-based XSS attacks.

Meshram and Patil (2014)[10] proposed firewall that protects web application against application level attacks like SQL injection, XSS, buffer overflow, cookie poisoning. Authors proposed seven algorithms that monitor incoming and outgoing HTTP traffic and compare it with attack list stored in database. It blocks malicious traffic and alert the user. WebGladiator admin home page allows admin to create log of attack detected and reports repeatedly occurred attacks. Its algorithms work on pattern matching scheme and is use blacklisted attacks. It does not provide mechanism to detect reflected XSS attacks and DOM-based XSS attacks.

Mahapatra et al. (2015)[11] stated that by encoding/escaping string input, by validating untrusted HTML input and by disabling scripting in browser; one can avoid cross-site scripting attacks. Here, authors proposed web proxy that rewrite cookie dynamically with randomized value. When cookie is sent to browser, it sent as randomized value. When cookie is sent back to server, this randomized value is converted into original value. This web proxy also contains filter to prevent SQLI and XSS attacks. This filter use cleansing algorithm which is based on pattern detection algorithm.

4. Comparative Analysis of Various Reviewed Models

The following section provides table of comparative analysis of various reviewed models that depict which model works on which type of attacks and whether it only detects or also provides prevention mechanism against such kind of attacks.

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Author Name</th>
<th>Attacks Detection(D) and Prevention(P)</th>
<th>Operate As</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Stored XSS</td>
<td>Reflected XSS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D</td>
<td>P</td>
</tr>
<tr>
<td>Model</td>
<td>Year</td>
<td>Client Site Firewall</td>
<td>Browser and Web App. Modification</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------</td>
<td>----------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Noxes Tool</td>
<td>Kirda et al. (2006)</td>
<td>✓ ✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>BEEP</td>
<td>Jim et al. (2007)</td>
<td>✓ ✓ x x ✓</td>
<td></td>
</tr>
<tr>
<td>XSSDS</td>
<td>Johns et al. (2008)</td>
<td>✓ ✓ ✓ ✓ x</td>
<td></td>
</tr>
<tr>
<td>SWAP</td>
<td>Wurzinger et al. (2009)</td>
<td>✓ ✓ x x ✓</td>
<td></td>
</tr>
<tr>
<td>Multi–agent Scanner</td>
<td>Galan et al. (2010)</td>
<td>✓ x x x x</td>
<td></td>
</tr>
<tr>
<td>E-Guard</td>
<td>Stephen et al. (2011)</td>
<td>✓ x ✓ x x</td>
<td></td>
</tr>
<tr>
<td>CIDT</td>
<td>Choudhary and Dhore (2012)</td>
<td>✓ ✓ ✓ ✓ x</td>
<td></td>
</tr>
<tr>
<td>deDacota</td>
<td>Doupe et al. (2013)</td>
<td>✓ ✓ ✓ ✓ x</td>
<td></td>
</tr>
<tr>
<td>Web Gladiator</td>
<td>Meshram and Patil (2014)</td>
<td>✓ ✓ x x x</td>
<td></td>
</tr>
<tr>
<td>Dynamic Cookies Rewriting</td>
<td>Mahapatra et al. (2015)</td>
<td>✓ ✓ ✓ ✓ x</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Comparative Analysis of Various Reviewed Models

5. Conclusion

Cross-Site Scripting is the simplest way for an attacker to gain user’s confidential information. This paper summarizes previously defined methodology that detects and prevents cross-site scripting vulnerabilities. But still cross-site scripting attacks remain a big problem for web application because attacker finds loopholes to bypass security mechanism adopted by web developer. There is no single solution that can effectively mitigates cross-site scripting attacks. Thus further research to provide complete solution against such kind of web vulnerabilities is needed.

References


