Analysis of Browser level defense mechanisms to prevent Cross Site Scripting attacks

Haneet Kour
Department of Computer Science & IT, University of Jammu, J & K, India
Email: haneetkour9@gmail.com

ABSTRACT

Context: Web Technologies were primarily designed to cater the need of ubiquitousness but the security concern has been overlooked and such overlooks resulted in vulnerabilities which are being highly exploited by hackers in various ways to compromise security. When a vulnerability is blocked, the attacker traces out a different mechanism to exploit it. Cross site scripting (XSS) attack is also an exploitation of one of the vulnerabilities existing in the web applications.

Objective: To conduct a study on XSS attacks and to analyze the various defense mechanisms provided at browser level to protect the web applications from XSS attacks.

Method: In the study of XSS attacks, various experiments have been performed to trace out the vulnerabilities in the JavaScript functions, html tags and their various attributes leading to Cross Site Scripting attacks on the local host server (XAMPP) and then defense mechanisms against XSS attacks which are provided at browser level have been evaluated in both modern web browsers and mobile browsers.

Results: Browser level defense approaches can mitigate Reflected XSS vulnerabilities, but the Stored and DOM based XSS vulnerabilities successfully by pass the prevention mechanisms provided at browser level.

Conclusion: XSS attack is emerging as one of the top 10 web application vulnerabilities leading to security breach. Although, the browsers can protect the web applications against the said vulnerability up to some extent, yet more research is required to enhance the browser functionality to protect the users of the web application against XSS vulnerability.

Keywords: Cross Site Scripting, Cookie, Web Vulnerability, Web Browser, Mobile Browser.

1. INTRODUCTION

Web Technology has become *lingua-franca* for companies in software development that allows the design of pervasive applications. Thousands of web applications are developed and accessed by millions of users. Security of these websites is becoming an important concern to ensure the user's authentication and privacy. Gartner group has noted that almost 75 percent of attacks are tunneled through web applications. According to the Tower Group, nearly 26 percent of customers don't use online banking services due to security fears and 6 percent do not use due to privacy issues. Over 70% of organizations reported of having been compromised security by cyber attacks [1]. In June / July 2006, the e-payment web application PayPal had been exploited by the attackers and sensitive data (e.g. credit card numbers) of its members was stolen [2] [3]. Among all the categories of attacks exploiting vulnerabilities in the web applications leading to security breach, Cross Site Scripting attack is the most prevalent.

Cross Site Scripting attack (XSS) is a code injection attack performed to exploit the vulnerabilities existing in the web application by injecting html tag / JavaScript functions into the web page so that it

gets executed on the victim's browser to access to any sensitive victim's credentials (e.g. cookies, session IDs, etc.) when one visits the web page. By exploiting XSS vulnerabilities in the scripts (mainly JavaScript since it is highly used scripting language on the client side by the web developers), the attacker targets the organizations that hold large online communities of users (i.e. social networking sites, blogs and online news sites) or the organizations that rely on web technology to generate revenue (i.e. providers of online services, services that store personal or financial information such as online payment, banking services, etc.). According to Firehost, there is an increase of 160% in XSS attacks in the last quarter of 2012 and accounted for 57% of the recorded attacks [4]. According to Trustwave security research, over 6% of top 1000 websites had a successful XSS attack [5]. The time gap between identifying an XSS attack and resolving it is found to be crucial. As per the study of the Ponemon Institute on the Cost of Cyber Crime, the average time taken to resolve a cyber attack was 32 days with an average cost of $1,035,769 (i.e. $32,469 per day) for the participating sample of organizations [6].

1.1 Types of XSS attack

XSS attack involves the theft of the user's sensitive information and invoking malicious acts on the user's behalf. The attacker carries XSS attacks by following ways:

1.1.1 *Persistent XSS or Type 2:*

The Persistent or Stored XSS attack is executed when the malicious code submitted by the attacker is saved by the server in the web application repository, and is run in the web page accessed by the victim's browser. The attacker posts the malicious script along with the hyperlink to it into the blogs, comments, message boards, social sites etc. which will be invoked later on by the other users while surfing that particular web page. The following server side pseudo code is used to display the posts / comments inserted by the users into the blogs. This code gets inserted into the database of the website and it will be executed every time whenever the surfer visits the webpage.

```
$user-input = $_POST["comments"];
echo "<html><body>";
echo "The posted comment is:";
echo  $user-input;
echo "</body></html>";
```

This code provides a way to XSS attack as the victim can insert the malicious script *(<script> -- steal user credentials-- </script>*) into *comments* field and then response to the victim's browser becomes: "*<html><body> The posted comment is: <script>… </script></body></html>"*. This script gets executed in the victim's browser leading to security breach. The Figure 1 illustrates the steps involved in stored XSS attack.

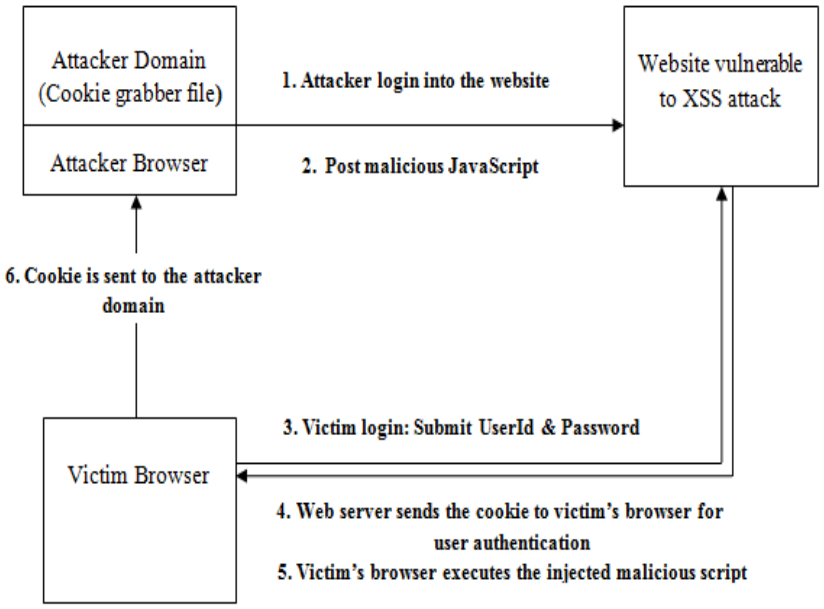

Figure 1: The architecture of Persistent XSS attack

A persistent XSS attack against Hotmail occurred on October 2001. In this attack, the remote attacker was allowed to steal .NET Passport identifiers of Hotmail's users by stealing their associated browser's cookies [7]. A persistent XSS attack against MySpace occurred on October 2005 and resulted in propagation of the worm Samy (that spread exponentially) across MySpace's user profiles [8].

1.1.2 *Reflected XSS or Type 1:*

Reflected or Non Persistent XSS attack is executed in websites when data submitted by the client is immediately processed by the server to generate the results that are then sent back to the browser on the client system. These vulnerabilities are generally found in search engines that return the input along with search results. The attacker uses standard means to deliver malicious XSS exploited URL to victim through e-mail, instant messenger applications, or search engines. This pseudo link ("*http://vulnerable-site.com/search.php?keyword=search-term*") redirects the user to the webpage showing searched term and the server side pseudo code is:

```
$searched-term = $_GET["keyword"];
echo "<html><body>" ;
echo "You searched for:" ;
echo  $searched-term;
echo "</body></html>";
```

The attacker crafts a malicious URL link (containing malicious JavaScript to redirect the victim's authentication details to attacker domain: *http://vulnerable-site.com/search.php?keyword="<script> …… </script>*") and sends this malicious link to the victim. By using social engineering techniques, he lures the victim to follow this malicious link. The server generates this response "*<html><body> You searched for: <script>……</script></body></html>"* to

the victim's browser which will execute the malicious script injected by the attacker. The Figure 2 illustrates the steps involved in reflected XSS attack [9].

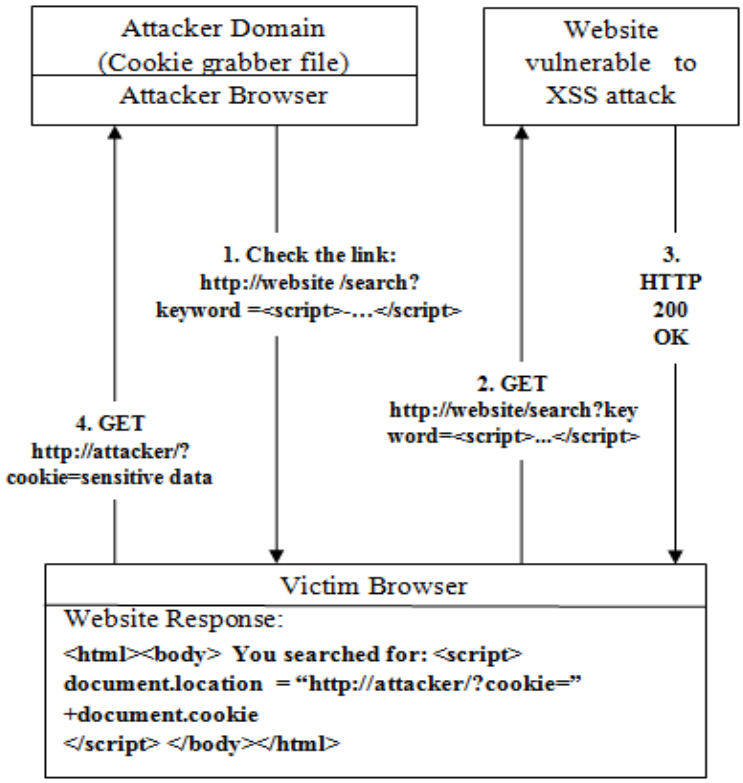

Figure 2: The architecture for Reflected XSS attack

Some reflected XSS vulnerabilities were traced out in the Google's web search engine on November 2005 and July 2006 [10] [11]. These vulnerabilities got fixed up in a reasonable short time but it questioned the integrity of the web application like the Google's web search engine.

1.1.3 *DOM-based XSS or Type 0:*

In this case, the vulnerability exists on the client-side code rather than on the server-side code. It is a case of reflected XSS where no malicious script inserted as part of the page, the only script that is automatically executed during page load is a legitimate part of the page i.e. legitimate JavaScript and careless usage of client-side data result in *XSS* conditions. DOM is abbreviated as Document Object Model and it is a platform and language-neutral interface which is using scripting or program to modify the content, update the data, structure and style of HTML and XML documents. Therefore, DOM-based XSS uses DOM's vulnerability to make the XSS come true. This XSS vulnerability is totally different from the persistent or non-persistent XSS attack and it does not inject malicious code into a page. So, it is the problem of the insecure DOM object which can be controlled by the client side in the web page or application. For this reason, hackers can make the attack payload execute in the DOM environment to attack the Victim side [12]. Consider the pseudo link (*http://vulnerable-site.com/search.php?keyword=search-term*) that redirects the user to the webpage showing searched item and the server side pseudo code is:

```
<?php
echo "<html>";
echo "You searched for :  <b id='demo'></b>";
echo "<script>";
echo " var Key = location.search.substring(9)";
echo "document.querySelector('#demo').innerHTML = Key" ;
echo "</script>";
echo "</html>";
?>
```

The attacker crafts a malicious URL link (*http://vulnerable-site.com/search.php?keyword= "<script> .... </script>*") and lures the victim to click on this link. The website receives the request, but does not include the malicious string in the response and the malicious string is not actually parsed by the victim's browser until the website's legitimate JavaScript gets executed. The Figure 3 illustrates the steps involved in DOM based XSS attack.

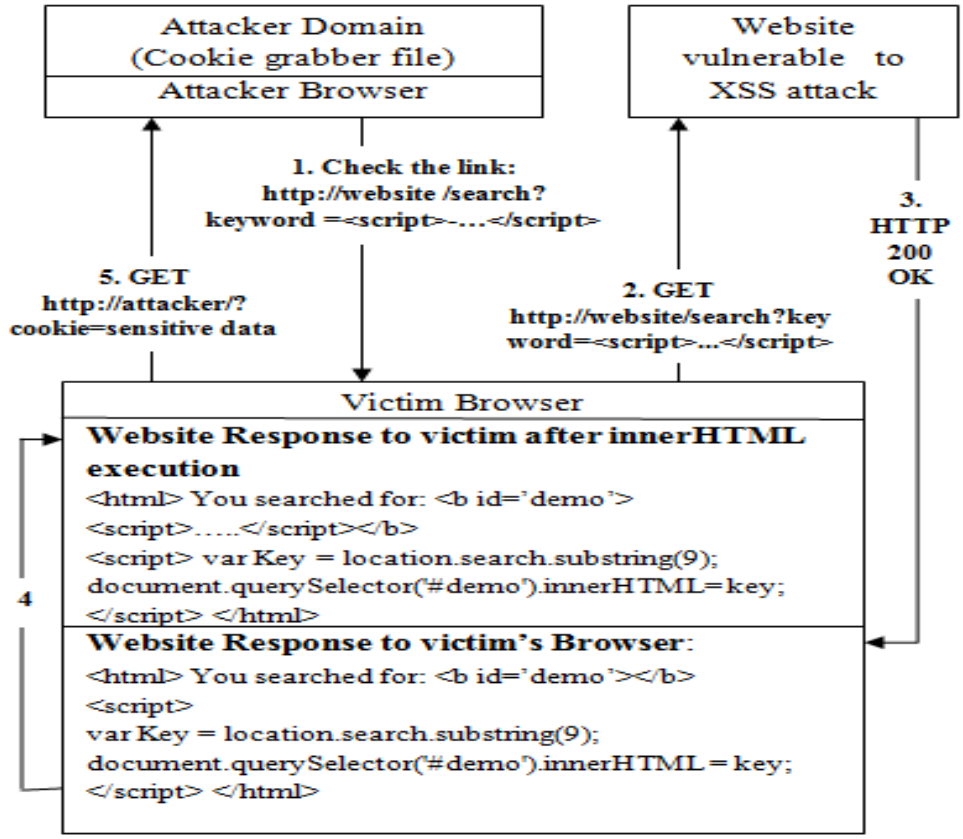

Figure 3: The architecture of DOM based XSS attack

2. AIMS AND OBJECTIVES

The objective of this paper is to study the various XSS attacks and to analyze the various defense mechanisms provided at browser level in web browsers and mobile browsers to protect the web applications from cross site scripting attacks.

3. EXPERIMENTAL SET UP

In this study, a website in *PHP* (say *http://localhost/Website/main.php*) has been developed to study *Stored* and *DOM* based *XSS* attacks and this website was hosted on the local host (*XAMPP* server). XSS vulnerabilities were also traced out in online platform (*http://testasp.vulnweb.com/search.asp*) to study *Reflected* XSS attack. The attacker domain (*http://attacker.com*) was also implemented on the virtual host in *XAMPP* server. The experiments to trace out possible *XSS* vulnerabilities (in the website that maintains user's authentication state by using *cookies)* have been performed on modern web browsers (*Google Chrome*51, *IE*10, *Opera*37, Safari5.1.7 and *Firefox*46.0.2). These experiments were carried out without implementing any prevention measures on the website in order to evaluate the defense mechanisms (provided in different browsers by default) against XSS vulnerabilities. This study was focused on both web browsers and mobile browsers (Google Android2.2.1 browser, Opera mini7.5.4, UC Browser 10.9.8, Firefox 47, Chrome 43, iPhone Safari). The attacker attempted to inject the following *JavaScript* code (that provides a hyperlink to redirect the victim's cookie to the attacker domain).

```
Malicious Script to be injected by attacker:
window.location="http://attacker.com?cookie="+document.cookie
```

Then, XSS vulnerabilities were also exploited in "*http://testasp.vulnweb.com/search.asp*" to check whether desktop and mobile browsers protect the users against XSS attack. Firstly, the original version of *Firefox* browser has been studied and then *it* was modified with *NoScript 2.9.0.11* Add-on to study its functionality against XSS vulnerabilities.

4. EVALUATION OF BROWSER LEVEL DEFENSE MECHANISMS AGAINST DIFFERENT XSS EXPLOITS

The attacker attempted to inject the malicious script using various html tags / JavaScript functions to execute it in the victim's browser to steal user's credentials to carry out XSS attack. The different browsers responses to the script injection are as under:

4.1 Stored XSS Attack

To trace out the browser defense mechanisms against Stored XSS vulnerabilities, the attacker injected the malicious script (<script>……</script>) into the *Add Comments* field (Figure 4) and these comments will be permanently stored in the back end database of the website (Figure 5). Whenever any Victim visits his *home page* (Figure 6), the malicious script will get executed leading to cookies stealth (Figure 7, 8). Stored XSS attack occurs in each web browser and even in *Firefox* extended with *NoScript* Add-on.

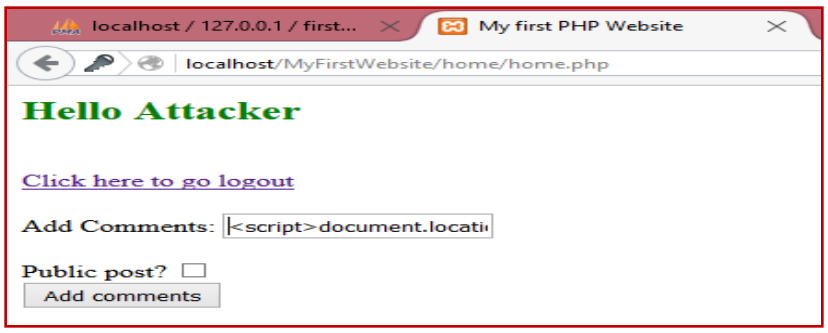

Figure 4: Attacker injects malicious script into Add Comments field

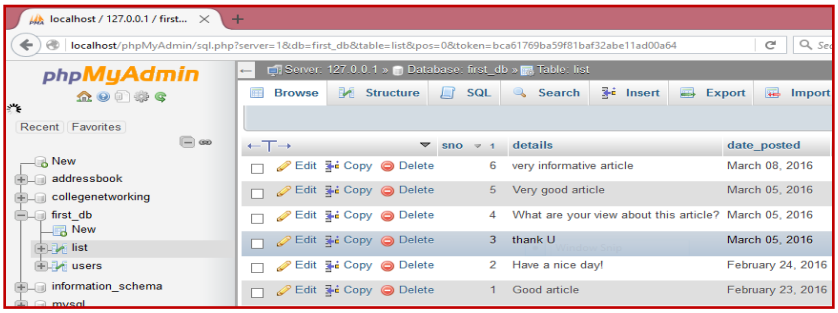

Figure 5: Database of the vulnerable website

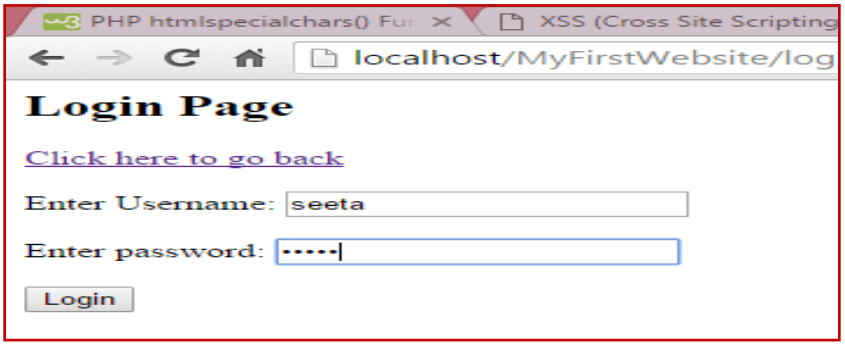

Figure 6: Victim logins into his home page

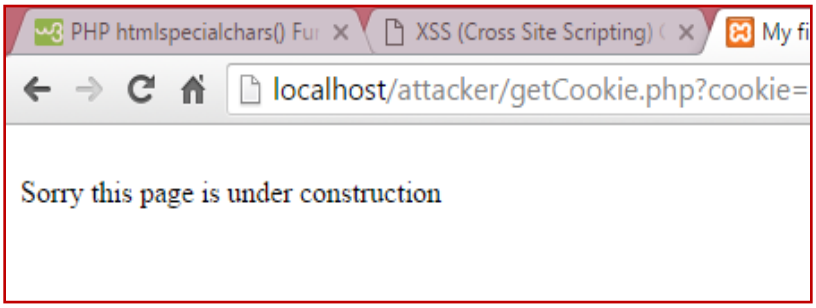

Figure 7: Malicious script gets executed and victim is redirected to attacker domain

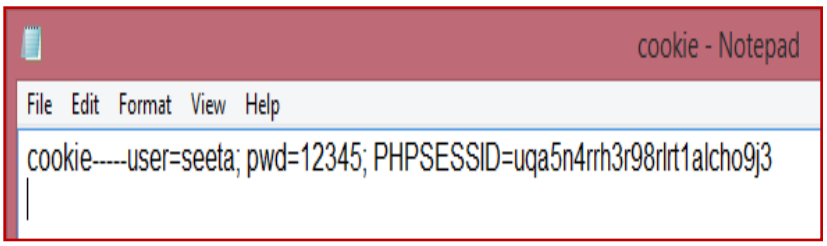

Figure 8: Victim's cookies stolen by the attacker

4.2 DOM based XSS Attack

To evaluate the browser defenses against DOM based XSS vulnerabilities, the attacker injected the malicious script *(<a href= # onclick=malicious script>)* into the *Search* field (Figure 9) and this searched term got embedded when intended script executed and it would be viewed later by the victim on surfing his *home page* (Figure 10). This XSS attack occurs in each desktop web browser and *Firefox* extended with *NoScript* Add-on.

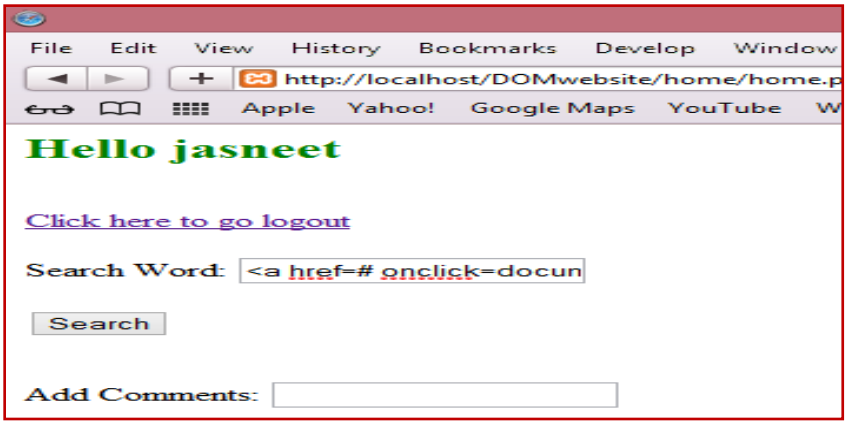

Figure 9: Attacker injects malicious script into Search Word field

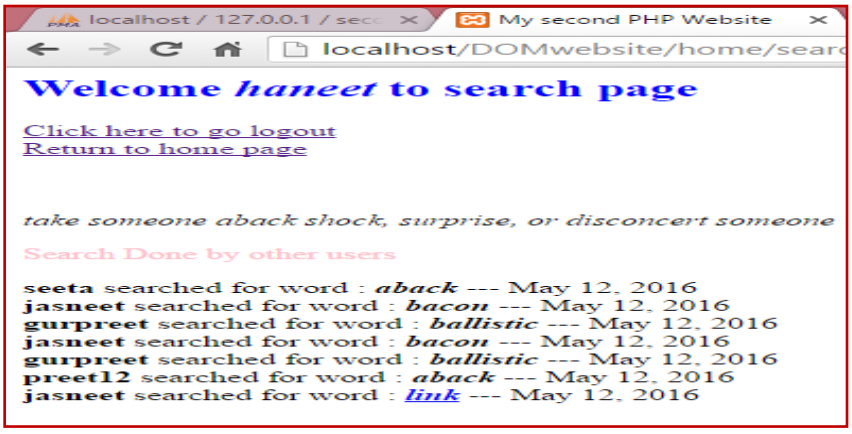

Figure 10: The malicious link gets embedded into the victim page leading to cookies redirection to attacker domain

4.3 Reflected XSS Attack

To evaluate the browser defense mechanisms against Reflected XSS vulnerabilities, the attacker injected the malicious script into the *Search* field (Figure 11) by crafting malicious URL *http://testasp.vulnweb.com/search.asp?tfSearch=“malicious-script”*.

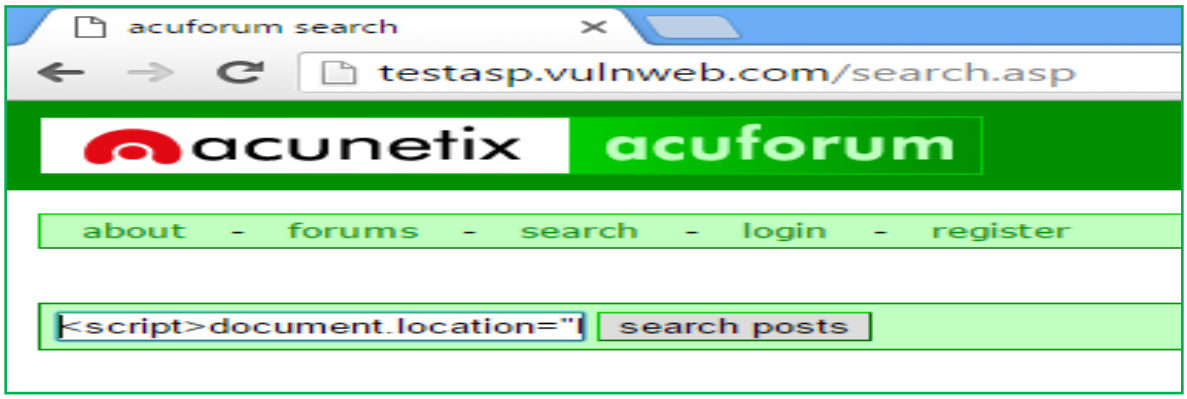

Figure 11: Malicious script injected to carry out Reflected XSS

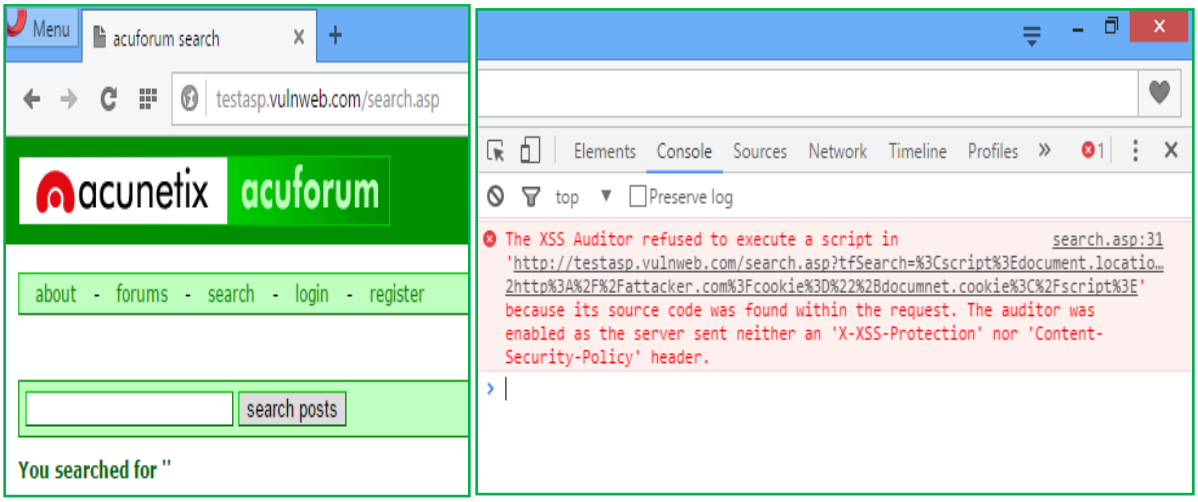

Figure 12: Malicious script gets blocked due to XSS Auditor in Chrome

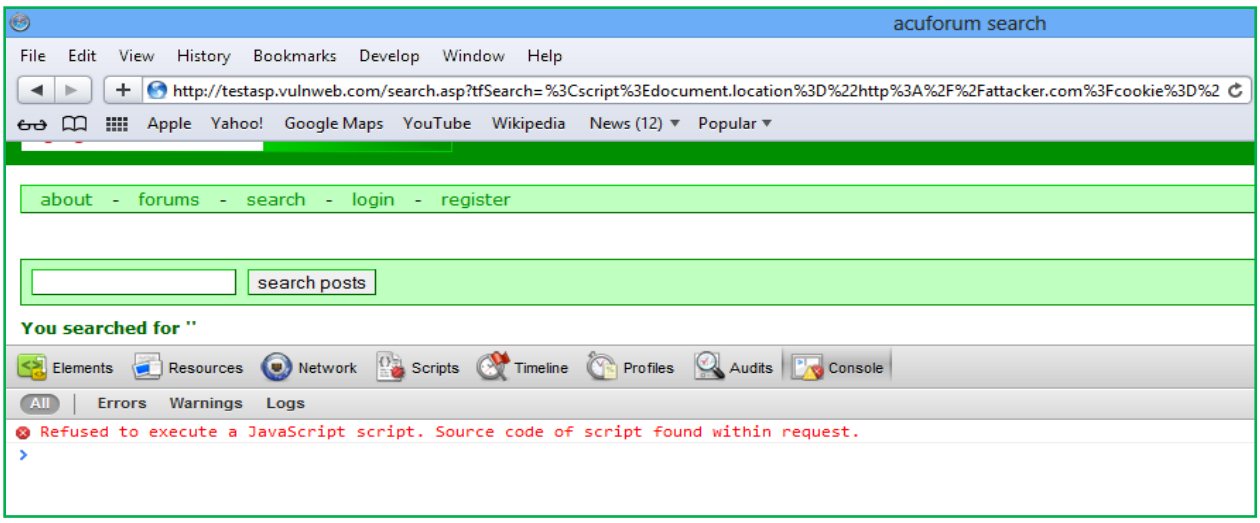

Figure 13: Malicious script execution is blocked due to XSS auditor in Safari

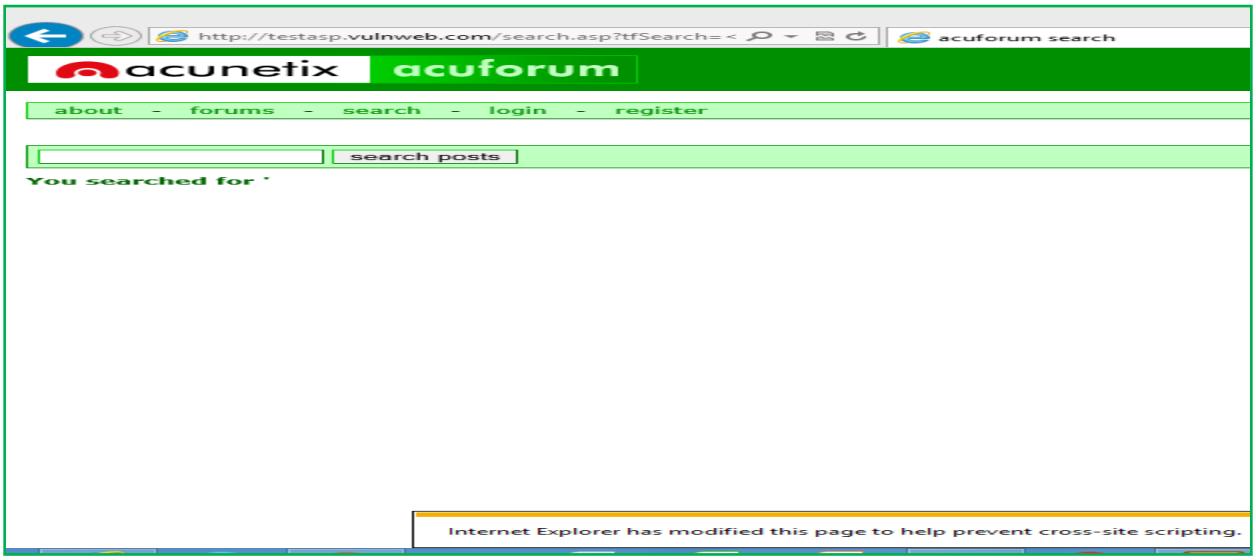

Figure 14: Malicious script gets blocked due to XSS filter in IE

Web Browser		*Mobile Browser*	
Chrome	Blocked it	Google Android	Attack successful
IE	Blocked it	UC Browser	Blocked it
Safari	Blocked it	Opera mini	Attack successful
Firefox	Attack successful	Firefox	Attack successful
Firefox with NoScript Add-on	Blocked it	iPhone Safari	Blocked it
Opera	Blocked it	Chrome	Blocked it

4.4 XMLHttpRequest()

The attacker injected *<script> XMLHttpRequest() object </script>* into *tfSearch* parameter in URL http://testasp.vulnweb.com/search.asp?tfSearch="value" and the browsers response to this URL is as under:

Web Browser		*Mobile Browser*	
Chrome	Blocked it	Google Android	Attack successful
IE	Blocked it and modified ( ) to ## and . to #	UC Browser	Blocked it
Safari	Blocked it	Opera mini	Attack successful
Firefox	Attack successful	Firefox	Attack successful
Firefox with NoScript Add-on	Blocked it	iPhone Safari	Blocked it
Opera	Blocked it	Chrome	Blocked it

4.5 Add Event Listener

The attacker injected *<script> add event listener code </script>* into *tfSearch* parameter in URL http://testasp.vulnweb.com/search.asp?tfSearch="value" and the browsers response to this URL is as under:

Web Browser		*Mobile Browser*	
Chrome	Blocked it	Google Android	Attack successful
IE	Blocked it and . changed to #	UC Browser	Blocked it
Safari	Blocked it	Opera mini	Attack successful
Firefox	Attack successful	Firefox	Attack successful
Firefox with NoScript Add-on	Blocked it	iPhone Safari	Blocked it
Opera	Blocked it	Chrome	Blocked it

4.6 <a> tag with JavaScript URI

The attacker injected *link* into *tfSearch* parameter in URL http://testasp.vulnweb.com/search.asp?tfSearch="value" and the browsers response to this URL is as under:

Web Browser		*Mobile Browser*	
Chrome	Blocked it	Google Android	Attack successful
IE	Blocked it	UC Browser	Blocked it
Safari	Blocked it	Opera mini	Attack successful
Firefox	Attack successful	Firefox	Attack successful
Firefox with NoScript Add-on	Blocked it	iPhone Safari	Blocked it
Opera	Blocked it	Chrome	Blocked it

4.7 <a> tag with Event attribute

The attacker injected *link* into *tfSearch* parameter in URL http://testasp.vulnweb.com/search.asp?tfSearch="value" and the browsers response to this URL is as under:

Web Browser		*Mobile Browser*	
Chrome	Blocked it	Google Android	Attack successful
IE	Blocked it	UC Browser	Blocked it
Safari	Blocked it	Opera mini	Attack successful
Firefox	Attack successful	Firefox	Attack successful
Firefox with NoScript Add-on	Blocked it	iPhone Safari	Blocked it
Opera	Blocked it	Chrome	Blocked it

4.8 Object tag

The attacker injected *<object data ="file.mp4" onload=malicious script></object>* into *tfSearch* parameter in URL http://testasp.vulnweb.com/search.asp? tfSearch="value" and the browsers response to this URL is as under:

Web Browser	
Chrome	Blocked it
IE	Blocked it modifies the <object> to <ob#ect> and onload is modified to #nload in source code
Safari	Blocked it
Firefox	Attack successful
Firefox with NoScript Add-on	Blocked it

4.9 IFrame tag with JavaScript URI

The attacker injected *<iframe src=javascript:malicious script></iframe>* into *tfSearch* parameter in URL http://testasp.vulnweb.com/search.asp?tfSearch="value" and the browser

responses to this URL is as under:

Web Browser		*Mobile Browser*	
Chrome	Blocked it	Google Android	Attack successful
IE	Blocked it	UC Browser	Blocked it
Safari	Blocked it	Opera mini	Blocked it
Firefox	Attack successful	Firefox	Attack successful
Firefox with NoScript Add-on	Blocked it	iPhone Safari	Blocked it
Opera	Blocked it	Chrome	Blocked it

4.10 IMG tag with event attribute

The attacker *injected* <*img src=# onclick=malicious script*> into *tfSearch* parameter in URL http://testasp.vulnweb.com/search.asp?tfSearch="value" and the browser responses to this URL is as under:

Web Browser		*Mobile Browser*	
Chrome	Blocked it	Google Android	Attack successful
IE	Blocked it and *onload* is changed to *#nload*	UC Browser	Blocked it
Safari	Blocked it	Opera mini	Attack successful
Firefox	Attack successful	Firefox	Attack successful
Firefox with NoScript Add-on	Blocked it	iPhone Safari	Blocked it
Opera	Blocked it	Chrome	Blocked it

5. RESULT AND ANALYSIS

After performing these experiments, it has been studied that *Chrome*, *Opera* and *Safari* web browsers protect the users against Reflected XSS vulnerabilities since these browsers contain "*XSS Auditor*" a built-in filter that checks if query parameters contain malicious JavaScript and if detection comes out to be positive, then the auditor is provoked to update the response to a non-executable state in the browser DOM. *IE* web browser also contains "*XSS filter*" to mitigate Reflected XSS vulnerabilities. On detecting the script in query parameters in *url*, IE displays the message "*Internet Explorer has modified the page to help prevent cross site scripting"*. It also modifies the source code of the web page to be displayed like *onload* attribute gets changed to *#nload*, *onmouseover* to *#nmouseover*, <*object*> tag to <*ob#ect*> tag, *()* to *##* in XMLHttpRequest object, *location* to *lo#ation* in event listeners and so on. But these browsers fail to block Stored and DOM based XSS attacks. *Firefox* web browser by default can't protect the users against XSS vulnerabilities. But *Firefox extended with NoScript Add-on* (that defends against XSS by filtering out suspicious parameter values) protects the user against reflected XSS

attack. These browsers can prevent whole script injection but fail to block partial script injection. Also *NoScript* suffers from the problems of false positive and complex policies.

These experiments were also performed on *mobile browsers* to evaluate their defensive functionality against XSS vulnerabilities. By performing these experiments, it has been evaluated that *Google Android*, *Firefox* and *Opera mini* mobile browsers don't offer any protection to the users against XSS attack. But *Opera mini* protect against *iframe* injection attack. *Chrome*, *iPhone Safari* and *UC browser* prevents against reflected XSS attack. The overall results of these experiments are shown in Table 1 for web browsers defense mechanisms against XSS and Table 2 for mobile browsers defense mechanisms against XSS.

Table 1: Result of evaluation of Desktop Browsers defense against XSS vulnerabilities

Type of XSS Attack	**Chrome 51**	**Opera 37**	**Firefox 46**	**IE 10**	**Safari 5.1.7**	**Firefox with NoScript2.9 Add-on**
Stored XSS	Yes	Yes	Yes	Yes	Yes	Yes
DOM based XSS	Yes	Yes	Yes	Yes	Yes	Yes
Reflected XSS	No	No	Yes	No	No	No
IFrame with javascript URI	No	No	Yes	No	No	No
<a> tag with event attribute	No	No	Yes	No	No	No
<a> tag with javascript URI	No	No	Yes	No	No	No
Object tag	No	No	Yes	No	No	No
Add Event Listener	No	No	Yes	No	No	No
XMLHttpRequest Object	No	No	Yes	No	No	No
IMG tag with event attribute	No	No	Yes	No	No	No
Yes : Attack occurred			No: Attack not occurred			

Table 2: Result of evaluation of Mobile Browsers defense against XSS vulnerabilities

Type of XSS Attack	**Google Android**	**UC Browser**	**iPhone Safari**	**Opera mini**	**Chrome**	**Firefox**
Reflected XSS	Yes	No	No	Yes	No	Yes
IFrame with javascript URI	Yes	No	No	No	No	Yes
<a> tag with event attribute	Yes	No	No	Yes	No	Yes
<a> tag with javascript URI	Yes	No	No	Yes	No	Yes
Add Event Listener	Yes	No	No	Yes	No	Yes
IMG tag with event attribute	Yes	No	No	Yes	No	Yes
Yes : Attack occurred			No: Attack not occurred			

6. CONCLUSION

XSS attack is emerging as a serious threat to the web application and its users. Various researchers have put their efforts to trace out and exploit the vulnerabilities of Cross Site Scripting attack in the web applications and on the basis of the result, they have proposed various types of prevention and protection mechanisms. They have proposed many new approaches or certain adjustments but a complete protection is still far off. Hackers are still able to exploit the vulnerabilities to carry out XSS in different ways. When a vulnerability is blocked, the attacker traces out another mechanism to exploit it. This requires a continuous watch over the new coming up technologies and test for the vulnerabilities. Also, web browsers and mobile browsers can protect the users against XSS attack up to some extent but more work needed to be done to enhance the defensive functionality of these browsers. Since users can access any web application from the desktop or mobile and he may be susceptible to reflected XSS attack when he opens the vulnerable site in his mobile browser. Thus more research is needed to implement defense functionality in mobile browsers to protect the users. Hence, more work should be done on the browser level to enhance security against the script injection. Also, the developers of these applications should adopt an efficient approach on the server side as well as client side to protect the users of the web application.

REFERENCES

[1] A Report by CyberEdge Group - 2015 Cyber Threat Defense Report. [Internet]: www.netiq.com/promo/security-management/2015-cyberthreat-defense-report.html. [10-Feb-2016].

[2] Mutton, P. PayPal, "Security Flaw allows Identity Theft", June 2006. [Internet]: http://news.netcraft.com/archives/2006/06/16/paypal_security_flaw_allows_identity_theft.html. [Feb 2016].

[3] Mutton, P. PayPal, "XSS Exploit available for two years", July 2006. [Internet]: http://news.netcraft.com/archives/2006/07/20/PayPal_xss_exploit_available_for_two_years.html. [Feb 2016].

[4] Secure cloud hosting company Firehost Superfecta Report, 30 July 2013. [Internet]. Available: http://www.firehost.com/company /newsroom/press-releases/firehost-report-suggests-commodity-cloud-providers-are-bolstering-botnet-agility. [Feb 2016].

[5] Ryan Barnett, "The web is vulnerable: XSS on the Battle Front", August 15, 2013. [Internet]. Available: http://blog.spiderlabs.com/2013/08/the-web-is-vulnerable-xss-on-the-battlefront-part1. html. [June 2016].

[6] Acunetix, "The ROI of protecting against XSS", 31 March 2014. [Internet]. Available: www.acunetix.com/blog/articles/return-on-investment-protecting-cross-site-scripting. [March 2016].

[7] Zero. Historic Lessons From Marc Slemko – Exploit number 3: Steal hotmail account. http://0x000000.com/index.php?i=270&bin=100001110.

[8] Wade Alcorn, "*XSS viruses: Cross-site scripting viruses and worms–a new attack vector*", Journal of Network Security, Elsevier, ISSN 1353-4858 Volume 2006 Issue 7, July 2006 pp 7-8.

[9] Joaquin G.A. and Guillermo N.A., "*Prevention of cross-site scripting attacks on current web applications*", OTM 2007, Lect. Notes Computer Science, vol. 4804, 2007, pp. 1770–1784.

[10] Y. Amit, "XSS vulnerabilities in Google.com", November 2005. [Internet]. Available: http://www.watchfire.com/securityzone/advisories/12-21-05.aspx.
[11] R. Hansen, "Cross Site Scripting Vulnerability in Google", July 2006. [Internet]: http://hackers.org/blog/20060704/cross-site-scripting-vulnerability-in-google/.
[12] Dom Based XSS. http://www.owasp.org/index.php/DOM_Based_XSS [www-document].