Analysis of mitigation techniques to prevent Cross Site Scripting attack in the web applications

Haneet Kour
Assistant Professor,
Department of Computer Science & Engineering,
MBSCET, Jammu, J&K, India
Email: haneetkour9@gmail.com

Abstract

The three tier architecture of the web has been developed to help the developers to create flexible web applications that are accessed by millions of users across the world. These web applications are developed by using various technologies like HTML, JavaScript, AJAX, XML etc. But the vulnerabilities at the design level in these technologies result in security compromise for the users. Thus, the security of these applications is becoming an important issue to ensure the user's authentication and privacy. Cross site scripting attack (XSS) is also an exploitation of these vulnerabilities (existing in the web applications) that result in theft of user's credentials. This paper studies the XSS attack and then analyses the various mitigation techniques to prevent XSS attacks.

Keywords— Cross Site Scripting, Cookie, Web Vulnerability, Mitigation.

INTRODUCTION

Cross Site Scripting (XSS) attack is emerging as one of the top web based security problems resulting in compromise of user's authentication and privacy. XSS refers to the script injection performed to exploit the vulnerabilities (existing at the design level in the web applications) 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 script, the attacker steals the user's sensitive information and invoking malicious acts on the user's behalf. The attacker generally 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.) [1]. The following server side pseudo code is used to display the comments posted by the users into the blogs.

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

This code gets stored into the backend database of the website and it will be executed every time on surfing this web page. But this code is vulnerable to XSS attack as it provides a way to the attacker to insert the malicious script *(<script> --steal user credentials-- </script>*) and then response generated by the server is: “*<html><body> The posted comment is: <script>… </script></body></html>*” and the malicious script gets executed in the victim’s browser leading to theft of victim’s credentials. The overview of XSS attack is presented in Fig. 1.

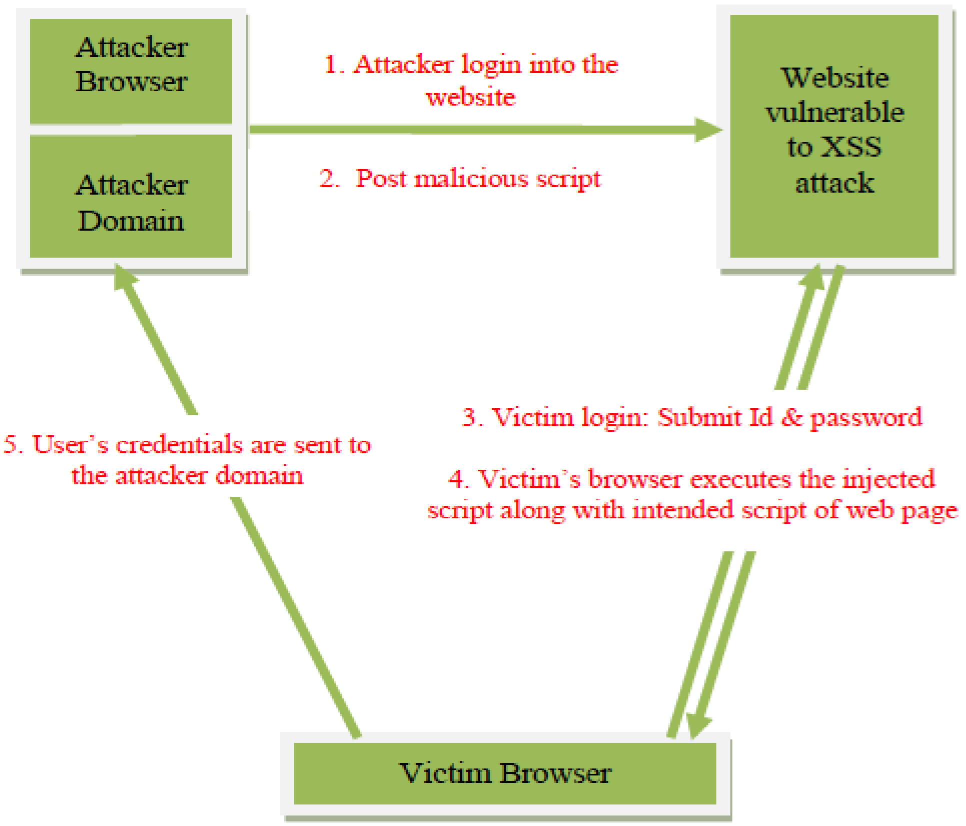

Attacker Browser
Attacker Domain
1. Attacker login into the website
2. Post malicious script
Website vulnerable to XSS attack
3. Victim login: Submit Id & password
4. Victim’s browser executes the injected script along with intended script of web page
5. User’s credentials are sent to the attacker domain
Victim Browser

Fig. 1. The overview of XSS attack

This architecture represents three main actors involved in XSS attack - Attacker Domain, Victim Domain and the Vulnerable Web Application. Firstly, the attacker traces out the vulnerabilities in the website that can be exploited by the JavaScript functions or attributes in html tags. Then, he injects malicious script into the web application by logging into the website or crafting malicious link and luring the victim to follow this link. Now, the victim logins into this vulnerable web application by submitting the Id and password. After authentication, the web server of the application would generate and transfer the cookie of that particular session to the victim's browser. The victim browser executes the malicious JavaScript code along with legitimate script of the web page leading to redirection of the victim's cookies to the attacker domain [2].

A. Types of XSS attack

The target of the attacker to steal user credentials (Cookies, Session Ids etc.) by carrying out script injection in the vulnerable web application where the victim visits. The attacker performs XSS attacks by following ways:

1) Stored XSS Attack: The 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. 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 [3]. 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 [4].

2) Reflected XSS Attack: Reflected 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. Some reflected XSS vulnerabilities were also traced out in the Google's web search engine on November 2005 and July 2006 [5] [6]. These vulnerabilities got fixed up in a reasonable short time.

AIMS AND OBJECTIVES

The objective of this paper is to study the XSS attack and to analyze the mitigation techniques to protect the web applications from cross site scripting vulnerabilities.

RELATED WORK

A lot of research has been done on script injection based web attacks in recent years and many researchers are continuing their study in this domain and various researchers have introduced different defensive techniques to prevent XSS vulnerabilities. These defense mechanisms are implemented either at the client side or the server side and some solutions integrate client and server approach. Some of these techniques are discussed as follows

David Scott and Richard Sharp [7] presented an application level firewall that needs correct identification and validation policies for each individual entry point to a web application to protect it by specifying what legal HTTP and HTML requests are. This approach suffers from high server response time.

O. Ismail *et al.* [8] devised an approach based on proxy mechanism that inspects the exchanged data between browser and web application's server to trace out the malicious requests that are reflected from the attacker to the victim domain. If request is found to be malicious, then characters contained within the request are encoded by the proxy, trying to avoid the success of the attack. This approach can detect reflected XSS attack.

Trevor Jim *et al.* [9] presented a mitigation technique known as BEEP (browser enforced embedded policies) against XSS that is implemented on the server side. In this approach, a website can embed a policy in its pages to specify which scripts are allowed to run. The browser, which knows exactly when it will run a script, can enforce this policy perfectly.

Gary Wassermann and Zhendong Su [10] devised an approach that detects Cross-Site Scripting Vulnerabilities statically for inspecting weak or missing input validation by combining the work on infected information flow with string analysis.

Yi Wang and *et al.* [11] introduced a static Stored XSS detection algorithm integrated with program slicing method to create the slices of web application that consists of threat injection and threat release, linked to possible Stored XSS for manual checking or other dynamic investigation.

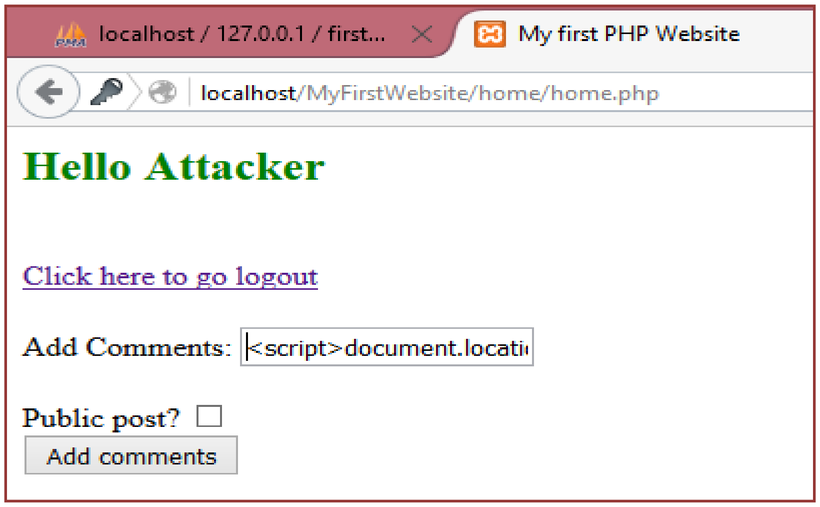
Shashank Gupta and B. B. Gupta [12] presented a security model called *Browser Dependent XSS Sanitizer* on the client-side web browser for mitigating the effect of XSS vulnerability. The authors used a three-step approach to eliminate the XSS attack without degrading much of the user's web browsing experience on various modern browsers.

EXPERIMENAL SETUP

In order to achieve the objectives of the study, a website in *PHP* was developed to study *XSS* attacks and this website was hosted on the local host (*XAMPP* server - *http://localhost/website/main.php*). The attacker domain (*http://attacker.com*) was also implemented on the virtual host in *XAMPP* server. Firstly no defense approach against XSS vulnerabilities was implemented in the website. The attacker attempted to execute the following *JavaScript* code on the victim's browser to carry out XSS.

```
window.location="http://attacker.com?cookie="+document.cookie
```

The execution of the above script resulted in redirection of the victim's cookie to the attacker domain.


localhost / 127.0.0.1 / first...
My first PHP Website
localhost/MyFirstWebsite/home/home.php

Hello Attacker

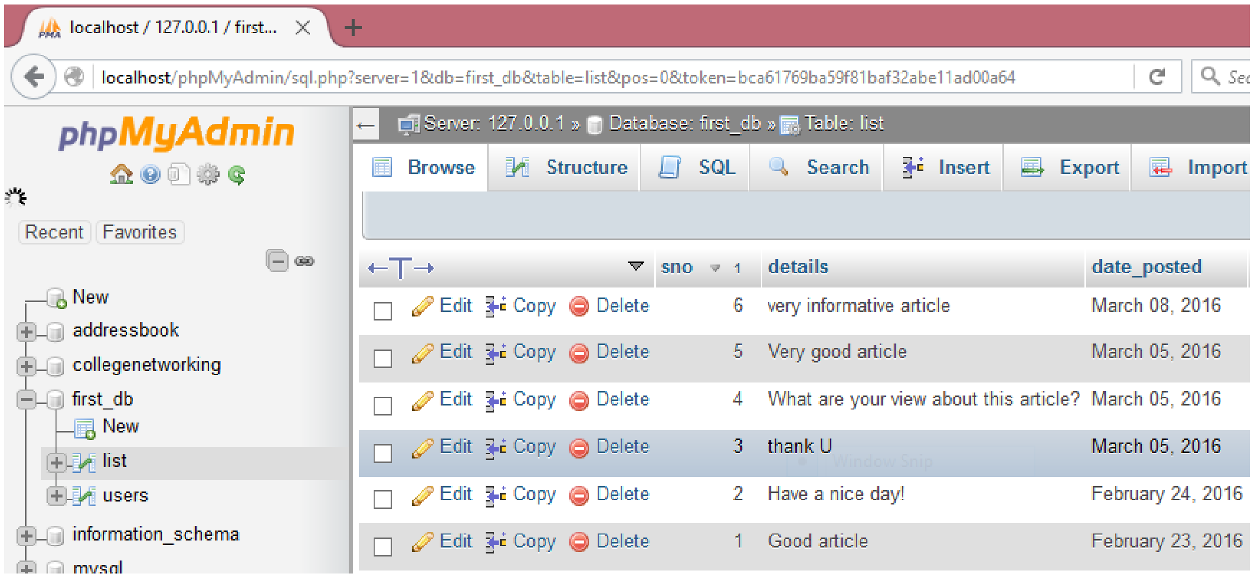
Click here to go logout

Add Comments: <script>document.locati

Public post?

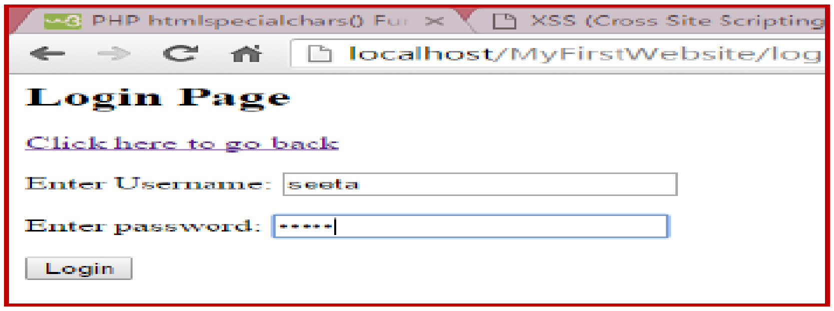
Add comments

Fig. 2. Home page of the attacker in the vulnerable website where he injects malicious script


localhost / 127.0.0.1 / first...
localhost/phpMyAdmin/sql.php?server=1&db=first_db&table=list&pos=0&token=bca61769ba59f81baf32abe11ad00a64
phpMyAdmin
Recent
Favorites
New
addressbook
collegenetworking
first_db
New
list
users
information_schema
mysql
Server: 127.0.0.1 » Database: first_db » Table: list
Browse
Structure
SQL
Search
Insert
Export
Import

	sno	details	date_posted
Edit Copy Delete	6	very informative article	March 08, 2016
Edit Copy Delete	5	Very good article	March 05, 2016
Edit Copy Delete	4	What are your view about this article?	March 05, 2016
Edit Copy Delete	3	thank U	March 05, 2016
Edit Copy Delete	2	Have a nice day!	February 24, 2016
Edit Copy Delete	1	Good article	February 23, 2016

Fig. 3. Malicious script stored into web repository to carry out XSS attack


PHP htmlspecialchars() Fu
XSS (Cross Site Scripting
localhost/MyFirstWebsite/log

Login Page

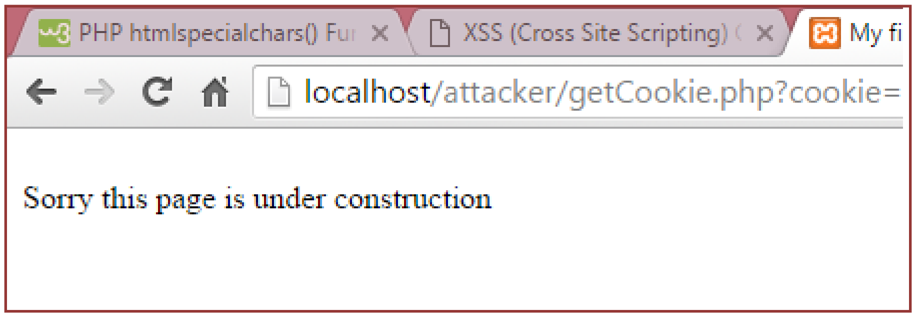
Click here to go back

Enter Username: seeta

Enter password: •••••

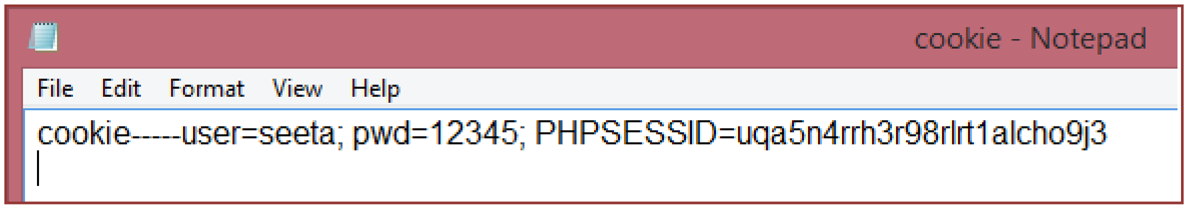
Login

Fig. 4. Victim logins into his home page


PHP htmlspecialchars() Fu
XSS (Cross Site Scripting)
My fi
localhost/attacker/getCookie.php?cookie=

Sorry this page is under construction

Fig. 5. Malicious script gets executed and victim is redirected to attacker domain



```
cookie-----user=seeta; pwd=12345; PHPSESSID=uqa5n4rrh3r98rlrt1alcho9j3
```

Fig. 6. Victim’s cookies stolen by the attacker

A. Preventing XSS attack

The main reason behind the XSS attack is the misinterpretation of the user input as a code rather than the data by the DOM. During the parsing of any web page in the browser, DOM parses the injected malicious script together with the intended script of the web page, thus resulting in script execution. Therefore, secure input handling is needed to interpret this input as data. In our study, following mitigation techniques were applied in the web application to prevent XSS Vulnerabilities:

1) Enoding: In this mitigation approach, the user input is first encoded and then it gets stored into the web repository. This approach mitigates XSS by escaping user input so that the browser interprets it only as data, not as code. It transforms special characters like < and > into &lt; and

&gt; respectively. Thus, the script does not get executed. In our study, *htmlentities(“user-input", flag)* function in php is used to implement encoding technique. This approach is implemented on the server side.

2) Filtering: In this approach, the user input is filtered to check whether it contains html tag or not. If it contains, then all the html tags will be removed from the input, only contents inside these tags will get stored into the backend database. In our study, it is implemented by using the function *filter_var(“user-input”, FILTER_SANITIZE_STRING)* in php to prevent the insertion of malicious code into the database of web application, thus mitigating XSS attack. It is also implemented on the server side.

3) Blacklisting: In this approach, the pattern for the possible malicious script (to carry out XSS) has been predefined. When the user enters the input, then it is matched with all predefined forbidden pattern to check whether input is valid or not. If input gets matched, then input will be blocked or sanitized to mitigate XSS vulnerabilities. In our study, *preg_match(“predefined forbidden pattern”, “user-input”)* function is used to trace out black listed input.

4) Whitelisting: It is the opposite of the blacklisting approach. In this approach, the pattern for the possible safe input has been predefined. When the user enters the input, then it is matched with all predefined allowed pattern to check whether input is valid or not. If input gets matched, then input will be passed to the backend database of the website without any transformation. In our study, *preg_match(“predefined allowed pattern”, “user-input”)* function is used to trace out white listed input.

5) Sandbox: In this approach, a sandbox environment is created on the web browser from where the victim surfs the web page. Even if the attacker becomes successful in executing malicious script in the victim’s browser, but the user credentials (authentication details, cookies etc.) will not be redirected to the attacker domain as the sandbox does not authorize the leakage of any information out of this protected environment. In our study, sandbox environment is created on IE web browser by using Content Security Policy as: *header("X-Content-Security-Policy: sandbox allow-forms allow-scripts allow-modals ");* it means the script gets executed but user’s authentication details will not be redirected to the attacker domain , thus bypassing XSS attack.

RESULT AND ANALYSIS

By performing the above experiments on the vulnerable website hosted on the local host, it has been found in these experiments that the attack was performed successfully by injecting malicious JavaScript in various ways. Then mitigation techniques were deployed to prevent XSS and these techniques successfully prevent XSS attack. These approaches are evaluated for their merits and demerits which are as under:

Merits:

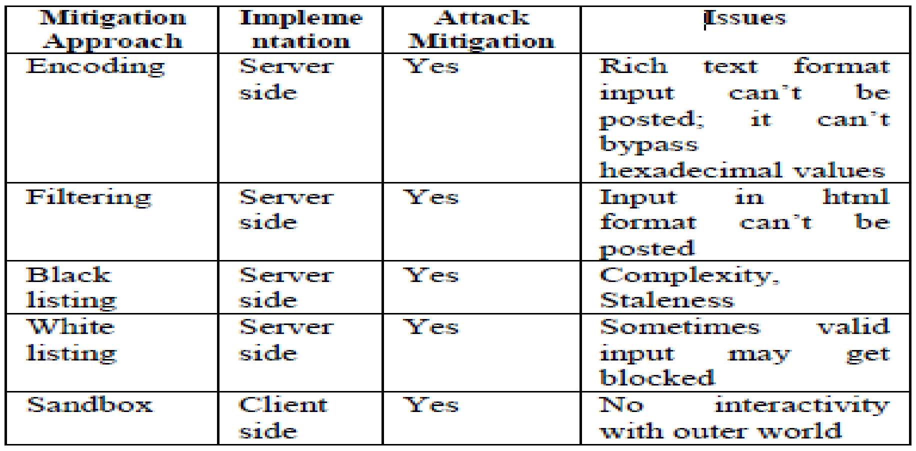
- These mitigation techniques successfully mitigate XSS attack risks.
- These techniques have no effect on the performance of the client's web browser.
- These techniques are compatible with modern browsers.

Demerits:

- By adopting encoding and filtering, users are not allowed to post their inputs in *html* format. They can post input only in data format.
- Although, blacklisting allows valid html input to get posted but the developers have to predefine the forbidden pattern for the malicious code. It causes overburden on the developer's side. Similar is the case with whitelisting approach.
- If the attacker inserts the malicious code that is not in the list of predefined forbidden expressions, then this code can get bypassed and it gets executed on the victim's browser.
- If any user inserts valid input but that is not in the list of predefined allowed expressions, then this valid input gets blocked by whitelisting approach.
- With sandboxing, the attack gets prevented, but the website can't interact with outer world.

The overall analysis of these mitigation techniques to prevent XSS attack is summarized in table 1.

TABLE I. ANALYSIS OF MITIGATION TECHNIQUES AGAINST XSS



Mitigation Approach	Impleme ntation	Attack Mitigation	Issues
Encoding	Server side	Yes	Rich text format input can't be posted; it can't bypass hexadecimal values
Filtering	Server side	Yes	Input in html format can't be posted
Black listing	Server side	Yes	Complexity, Staleness
White listing	Server side	Yes	Sometimes valid input may get blocked
Sandbox	Client side	Yes	No interactivity with outer world

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, 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] 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.

[2] Haneet Kour and Lalit Sen Sharma, "Tracing Out Cross Site Scripting Vulnerabilities in Modern Scripts", International Journal of Advanced Networking and Applications (IJANA), ISSN: 0975-0290 (Online) Volume 7 Issue 5, 2016, pp. 2862-2867.

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

[4] 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.

[5] Y. Amit, "XSS vulnerabilities in Google.com", November 2005. [Internet]. Available: http://www.watchfire.com/securityzone/ advisories/12-21-05.aspx.

[6] R. Hansen, "Cross Site Scripting Vulnerability in Google", July 2006. [Internet]: http://hackers.org/blog/20060704/cross-site-scripting-vulnerability-in-google/.

[7] David Scott and Richard Sharp. "Abstracting application-level web security", in WWW '02: Proceedings of the 11th international conference on World Wide Web, ACM, New York, USA, 2002, pp. 396-407.

[8] O. Ismail, M. Etoh, Y. Kadobayashi, and S. Yamaguchi, "A Proposal and Implementation of Automatic Detection/Collection System for Cross-Site Scripting Vulnerability", 18th Int. Conf. on

Advanced Information Networking and Applications 2004, pp. 145-151.

[9] Trevor Jim, Nikhil Swamy and Micheal Hicks, "Defeating Script Injection Attacks with Browser Enforced Embedded Policies", Proc. 16th International Conference on WWW ACM 2007, pp.601-610

[10] Gary Wassermann, Zhendong Su, "Static Detection of Cross-Site Scripting Vulnerabilities", ICSE "08: Proceedings of the 30th international conference on Software engineering, 2008, pp. 171-180.

[11] Yi Wang, Zhoujun Li, Tao Guo, “Program Slicing Stored XSS Bugs in Web Application”, Fifth International Symposium on Theoretical Aspects of Software Engineering, IEEE, 2011, pp. 191-194.

[12] Shashank Gupta and B.B. Gupta, “BDS: Browser Dependent XSS Sanitizer”, IGI-Global, Handbook of Research, Nonvember 2014, pp. 174-191.