Exploitation of Cross-Site Scripting (XSS) Vulnerability on Real World Web Applications and its Defense

Shashank Gupta
Lecturer in Department of Information Technology, Model Institute of Engineering and Technology, Jammu

Lalitsen Sharma
Associate Professor in Department of Computer Science and I.T., University of Jammu

ABSTRACT
Attacks on web applications are growing rapidly with the opening of new technologies, HTML tags and JavaScript functions. Cross-Site Scripting (XSS) vulnerabilities are being exploited by the attackers to steal web browser's resources (cookies, credentials etc.) by injecting the malicious JavaScript code on the victim's web applications. The existing techniques like filtering of tags and special characters, maintaining a list of vulnerable sites etc. cannot eliminate the XSS vulnerabilities completely. In this paper, initially we have tried out the experiments on the exploitation of XSS vulnerabilities using local host server (i.e. XAMPP). After this, we have investigated for the XSS vulnerabilities on social networking sites (like Facebook, Orkut, Blogs, Twitter etc.) and tried to exploit the same on blogs. Finally, on the basis of some analysis and results, we have discussed a novel technique of mitigating this XSS vulnerability by introducing a Sandbox environment on the web browser.

Keywords
Keywords are your own designated keywords which can be used for easy location of the manuscript using any search engines.

1. INTRODUCTION
Now-a-days Cross-Site Scripting (XSS) attack is a common vulnerability which is being exploited in web applications through the injection of HTML tags and malicious Java Script code. A weak input validation on the web application causes the stealing of cookies from the victim's web browser. Cookies are the most general way to identify and authenticate the users and it is being supported by almost all the web browsers [1]. Therefore, they are an attractive target for the attackers. Nearly all the web browsers support cookies and therefore allow a greater flexibility that how user sessions are maintained by the web applications. If an attacker can steal the valid cookies of a victim's session, then the attacker can hijack the victim's session. Cross-Site scripting continuously leads the most wide spread web application vulnerabilities lists (e.g. OWASP [2], WhiteHat Website Security Statistics Report [3]). Recent survey has shown that almost 67% of websites are vulnerable to XSS attacks [3]. XSS are broadly classified as two main attacks which are Persistent and Non-Persistent Attacks [4] [5].

Persistent attack (also called as stored attack) holes exist when an attacker post the malicious code on the vulnerable web application's repository. As a result, if the stored malicious code gets executed by the victim's browser, then stored attack gets exploited on the victim's web browser. Secondly non-persistent attack (also called as reflected attack) means that the vulnerable malicious code is not persistently stored on a web server but it is immediately displayed by the vulnerable web application back to the victim's browser. If so, then the malicious code gets executed on the victim's browser and finally the victim has to compromise its browser's resources (e.g. cookies).

The rest of the paper is organized as follows: Section 2 describes the background and related work on XSS attacks. Section 3 describes the architecture of exploitation of XSS attack. Section 4 describes the proposed technique of exploiting the XSS attack on Local Host Server and Blogs. Section 5 discusses the mitigation technique of XSS technique. Section 6 concludes the proposed work and discusses some ideas of future work.

We ask that authors follow some simple guidelines. In essence, we ask you to make your paper look exactly like this document. The easiest way to do this is simply to download the template, and replace the content with your own material.

2. BACKGROUND AND RELATED WORK
The related work has been surveyed focussing on some issues related to XSS attacks. The survey has been divided into three categories namely Exploitation, Detection and Prevention of XSS attacks.

2.1 Exploitation of XSS Vulnerability
Recently, researchers have shown some basic ways to demonstrate how XSS attacks can be used to control and modify the functionality of a web page. Various types of platforms (like Web Goat [6], Acunetix [7]) are available online to test or exploit some vulnerabilities of XSS attack. Web Goat is a deliberately insecure application maintained by Open Web Application Security Project (OWASP) [2]. On the other hand, Acunetix test website offers the platform to a user who wants to exploit the vulnerabilities of XSS attack. It is a way of limiting security testing to only systems that you own, or have permission to work with.

2.2 Detection of XSS Vulnerability
In static detection of XSS, testing is generally performed by source code analysis. On the other hand, in dynamic testing of XSS, known attacks are executed against web applications. Recently, researchers have proposed various detection techniques to discover the XSS Attacks. In [8], a Webmail XSS fuzzer called L-WMxD (Lexical based Webmail XSS Discoverer), which works on a lexical based mutation engine which is an active defence system to discover XSS before the Webmail application is online for service. The researchers have run L-WMxD on over 26 real-world Webmail applications and found vulnerabilities in 21 Webmail services, including some of the most widely used Yahoo-Mail. In [9], a static analysis for finding XSS vulnerabilities has been put forward that directly addresses weak input validation. This approach combines work on tainted information flow with

string analysis. Pixy [10] is a tool that performs data flow analysis on PHP code to detect reflected XSS vulnerabilities. Various prototype tools which are based on Pixy have been implemented by the researchers and test on the real world PHP programs. Similar approaches have been adopted by commercial products like AppScan [11], Acunetix [7], Nessus[12] and so on.

2.3 Prevention of XSS Vulnerability

Cross-site Scripting (XSS), the top most vulnerability in the web applications, demands an efficient approach on the server side as well as client side to protect the users of the web application. In [13], an application-level firewall is suggested, which is located on a security gateway between client and server and which applies all the security relevant checks and transformations. Some server side prevention approaches require the collaboration of web browsers. One such example is BEEP (Browser-Enforced Embedded Policies) [14], a mechanism that modifies the browser so that it cannot execute the malicious scripts. Security policies dictate what the server sends to BEEP-enabled-browsers. Apart from this, the researchers developed the WebSSARI (Web Security via Static Analysis and Runtime Inspection) tool [15], which per-forms type-based static analysis to identify potentially vulnerable code sections and instrument them with runtime guards.

On the client side, researchers have developed the Noxes [16] which acts as a personal firewall that allows or blocks connections to websites on the basis of filter rules, which are generally user-specified URL white-list and blacklist websites. When the browser sends a HTTP request to an unknown website, Noxes immediately alerts the client, who chooses to permit or deny the connection, and remembers the client's action for future use. Another client side approach is presented in [17], which aims to identify the information leakage using tainting of input data in the browser. In [18], a mechanism for detecting malicious java script is proposed. The system consists of a browser-embedded script auditing component, and IDS that processes the audit logs and compares them to signatures of known malicious behavior or attacks.

Several server-side countermeasures do exists, but such techniques have not been universally applied because of their deployment overhead. On the other hand, existing client side solutions degrade the performance of client's system resulting in poor web surfing experience. The necessity to install updates or additional components on each user's web browser or workstation also degrade the performance of client side solutions.

3. ARCHITECTURE OF EXPLOITING XSS VULNERABILITY

In this paper, we have exploited the vulnerabilities of XSS attack on local host Server (XAMPP) and then we have tried to exploit the same on Blogs by stealing the cookies of victim's session. Lastly, we have discussed a defensive technique which generally bye-pass the XSS attack by introducing a sandbox environment on the web browser. Web applications frequently use cookies for retaining an authentication state between users and web applications. These cookies are usually sent to the users by the web applications after the users have been successfully authenticated. Every consequent request that contains the legitimate cookies will be automatically approved by the web applications without any further authentication. The cookies are used to both identify and authenticate the users; therefore they are an interesting target for potential attackers. The following figure 1 is an architecture which shows the sequence of steps of exploiting the XSS vulnerability by a malicious attacker.

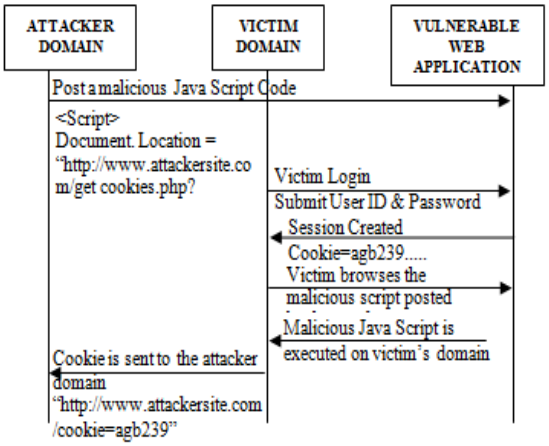

Fig 1: Architecture of Exploiting the XSS Vulnerability

The above architecture contains three useful commodities i.e. Attacker Domain, Victim Domain and Vulnerable Web Application. Here are some sequences of steps which will explain the above architecture of exploiting the XSS attack:-

- Firstly the attacker has found that the corresponding web application is vulnerable to Cross-Site Scripting Attack. After this, attacker will post a malicious Java Script Code on the Vulnerable Web Application whose function is to steal cookies of the victim's account session.
- Secondly, the victim logs into the vulnerable web application by giving the user-id and password. As a result, the web server of web application will generate and transfer the cookie of that particular session to victim's web browser.
- In the third step, the victim browses the malicious Java Script Code and gets executed on its browser.
- In the fourth step, the Java Script Interpreter of the victim's browser gets invoked and transfers the cookies of the victim's session to the attacker's domain.
- Now lastly, these cookies will be utilized by the attacker to get into the account of victim.

In this way, XSS attack gets exploited on the attacker's domain.

4. PROPOSED TECHNIQUE OF EXPLOITING THE XSS ATTACK

One of the most common techniques requires that the malicious code is stored into the repository of victim's web application which can be later executed by the victim's browser. The following figure is one of the malicious Java Script codes which can be posted on the victim's web application repository. This malicious script provides a hyperlink which will redirect the victim's cookie to the address (site) specified in the *document. Location*

```
<a onclick
document.location='http://localhost/xss-
attacker/getcookies.php?cookie='+escape(document.cookie);
href="#">click here </a> to know about XSS attack.
```

Fig 2: Malicious Java Script Code for Stealing the Cookies

The following are the details of this malicious Java Script code:

- *On Click Event*: This event executes the Java Script function, which is embedded inside it.
- *Document. Location*: This function will store the address or URL where the actual file has to be transferred.
- *Get Cookies.PHP*: It is a cookie grabber file which will store on the attacker site and steal the cookies of the victim's domain.
- *Escape (document.cookie)*: Escape is a function call which will pass the argument i.e. document. Cookie. Document is an object and Cookie is a file in Java Script which will capture the current cookies of any particular session.

The *get cookies.PHP* file as shown in the fig.2 is a cookie stealer file which is loaded on the attacker site and transfers the current cookies to a blank *cookie.txt* file which is also loaded on the attacker site. Following is the vulnerable PHP code of a cookie stealer file.

```
1. <?php
2. $cookie = '';
3. foreach($_GET as $key => $value)
4. {
5. $cookie .= $key.'='.$value.'---';
6. }
7. $date = date ("d/m/Y h:i:s A");
8. $user_agent = $_SERVER['HTTP_USER_AGENT'];
9. $file = fopen('cookies.txt', 'a');
10. fwrite($file,"$date : $cookie \n");
11. fclose($file);
12. echo '<b>sorry this page is under construction</b><br>
</br> Please click <a href "http://www.google.com/">
here</a> to go back to previous menu'
?>
```

Fig.3. Vulnerable PHP Code

In this paper, we have used both these vulnerable snippets as shown in the figures 2 and 3 for the exploitation of XSS attack on local host server and real time Google Blogs.

4.1 Exploitation of XSS Vulnerability on Local Host Server

Firstly, we have tried to exploit the vulnerabilities of Cross-Site Scripting attack on XAMPP Server. On XAMPP Server, we have made two separate domains of attacker and victim. The attacker domain has stored the cookie grabber file and blank cookie.txt file. So, when the victim click on the malicious link posted by the attacker, the cookie grabber file gets executed on the victim's domain and victim's cookie will get transfer to the blank cookie.txt file in which user-id, password and session-id of victim's account is stored clearly. In this way XSS attack will get exploited on the XAMPP server. Following fig. 4 shows some sequence of steps in the form of flowchart which gives the detailed idea of exploiting the XSS vulnerability on XAMPP server.

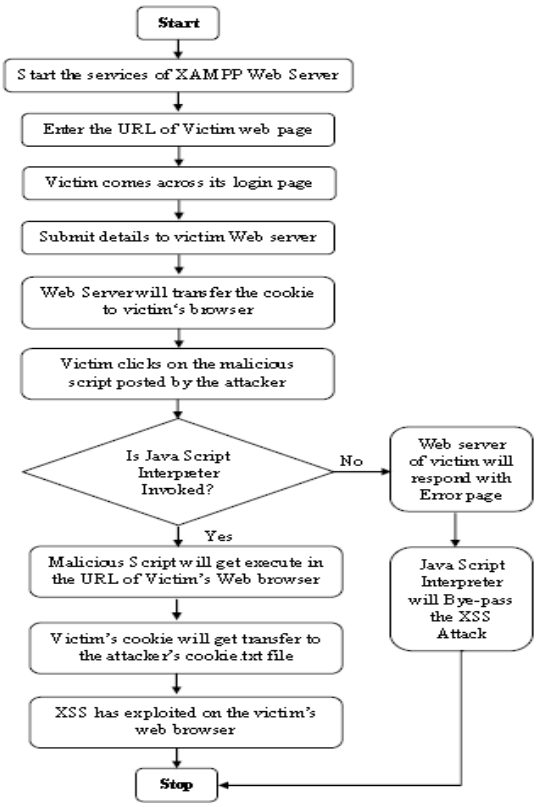

Fig.4. Flow Chart of Exploiting the XSS Attack on Local Host Server

4.2 Exploitation of XSS Vulnerability on Blogs

In real time scenario, we have tried to exploit the vulnerability of XSS on the Blogs. For this purpose, we require following three things:

- Attacker's Account on the Blog
- Victim's Account on the Blog
- Attacker's Web Hosting Site (http://www.shashankgupta.0fees.net)

On attacker's Web Hosting Site, we have uploaded the cookie grabber file and blank cookie.txt file. If the victim has clicked on the malicious script which is loaded in its Blog account, then the cookie of the victim's session account will get transfer to the attacker's web hosting site. Following are some sequence of steps of exploiting the XSS vulnerability on Blogs:-

Step 1: Attacker enters the URL of Blog (http://www.blogger.com) and submits its user-id and password to web server. Now the attacker will post the malicious java script code into its follower's account and simply logs out from its blogger account.

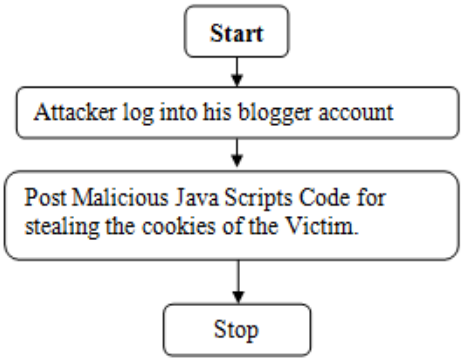

Fig.5. Flow Chart for Posting Malicious Script on Victim's Blog Account

Step 2: Now the attacker simply enters the URL of its domain and simply uploads the cookie grabber file code in PHP and blank cookie. Text file. The attacker also changes the read and writes permission of both these files.

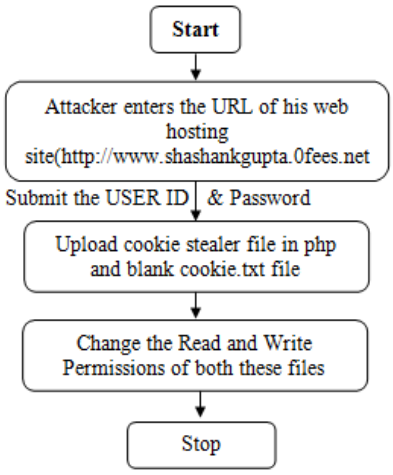

Fig.6. Flow Chart for Uploading the Cookie Stealer file

Step 3: In this step, the victim enters into its blogger account by giving the user-id and password to the web server. The web server will generate and transfer the corresponding cookie to the victim's web browser repository. After this, the victim executes the malicious script (posted by the attacker on its account) on its web browser which in turn will transfer the cookie of victim's session to the blank text file in the attacker's domain. This XSS attack will get exploit only if the java script interpreter gets invoked on the victim's web browser. The following figure 7 shows some sequence of steps of exploiting the XSS attack on victim's Blog account.

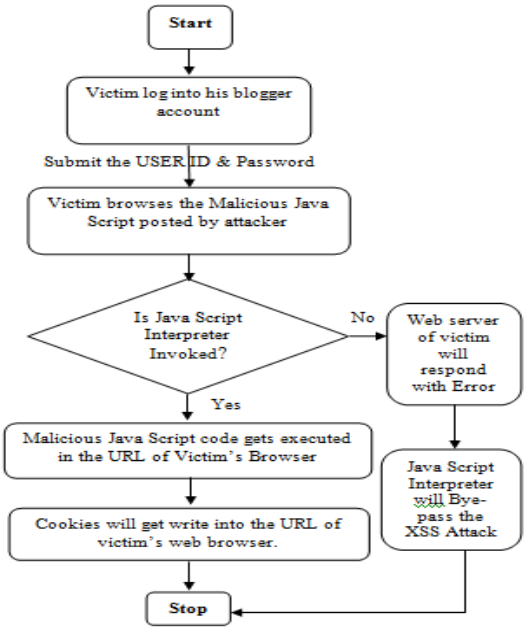

Fig.7. Flow Chart for Exploitation of XSS Attack on Victim's Blog

Step 4: : Now the attacker open his web hosting site and checks whether any victim clicked has clicked on that malicious link. The attacker will simply check its blank cookie.txt file that whether any value has got append on it or not. If yes, the attacker will simply utilize this information to get into the victim's blogger account. Here is the detailed flow chart explaining the whole scenario as shown in fig. 8.

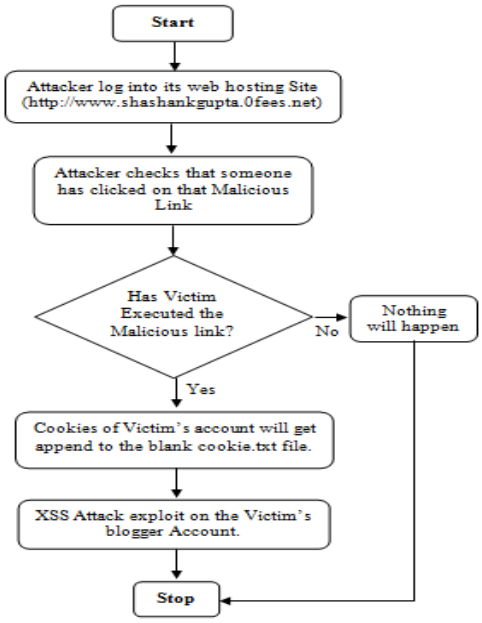

Fig.8. Flow Chart for Grabbing the Victim's Cookie File

5. Cross-Site Scripting Defense

The goal of XSS attack is to break the Same Origin Policy [30] of web browsers which guarantees that a file or a script loaded from a given site 'X' is not allowed from reading or modifying those browser's resources belonging to site 'Y'.

The aim of the technique regarding XSS defence is to not only address the attacks based on java script code embedded into the HTML documents but also attacks against other web application's technologies such as Flash, ActiveX and so on. Generally, XSS attacks are related to the stealing of victim's cookies. So, we are going to discuss an idea for the protection of cookies against XSS attack by introducing a sandbox environment on the web browser

Sandbox is a restricted environment which runs our programs in an isolated space, which prevents them from making permanent changes to other programs and data in the computer. The following are some benefits of sandboxing technique:-

- Secure Web Browsing: Running our Web browser under the protection of Sandbox means that all malicious software downloaded by the browser is trapped in the sandbox and can be discarded trivially.
- Enhanced Privacy: Browsing history, cookies, and cached temporary files collected while Web browsing stay in the sandbox and don't leak into Windows.
- Secure E-mail: Viruses and other malicious software that might be hiding in your email can't break out of the sandbox and can't infect your real system.
- Windows Stays Lean: It prevents wear-and-tear in Windows by installing software into an isolated sandbox.

The idea of protection of cookies can be achieved by introducing a sandbox environment on the web browser which is as shown in the figure 9.

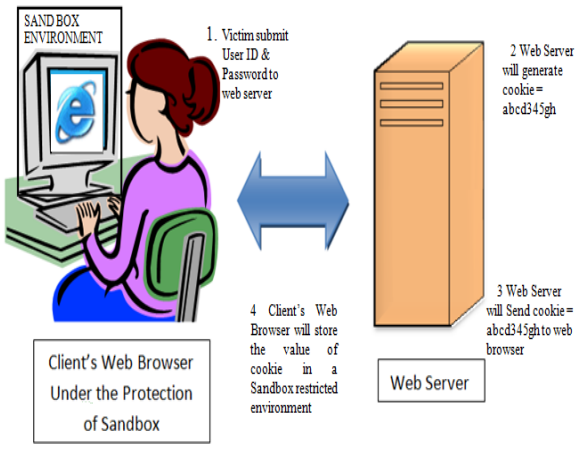

Fig.9. Cross-Site Scripting Mitigation Technique

Client's web browser under the protection of a sandbox submits the user-id and password to a web server. Web server will generate the cookie and send this cookie to client's web browser which is sandbox protected. Now this cookie value will neither leak into the windows nor it can be grabbed by any attacker. On the other hand, sandbox allows the execution of malicious script on the client's web browser but it cannot give the authority to simply leak the cookie out of this protected environment. So it can be concluded that sandbox environment will simply bye-pass the XSS attack.

6. CONCLUSION AND FUTURE WORK

Based on the proposed methodology and its consequent results, we have exploited the XSS intrusion by means of HTTP and cross-platform properties to steal and misuse the user's private information (cookies, session id's etc.). We have concluded that in order to bye-pass the vulnerabilities of XSS attack, the subsequent three precautions should be followed:

Condition 1: We should be aware of vulnerable website applications. There are lots of vulnerable web applications on the internet. As long as there are functions intended for editing HTML contents, it is likely to exploit the XSS attack. We can test the web applications by executing various types of experiments. We immediately need to put in some sentence into <BODY></BODY> tag. For instance <SCRIPT> alert ("XSS Attack") </SCRIPT>. The subsequently next step is to make use of web browser to surf this page. If our window jumps the "XSS Attack", it means that we can be attacked.

Condition 2: We should make use of only those web applications in which proper input validation has been done. A weak input validation cannot bye-pass the XSS Attack.

Condition 3: We should maintain a blacklist of vulnerable websites which are vulnerable to XSS attack. Apart from this, we should also be aware of latest web technologies like advanced java script functions etc.

In the future work, we would like to build up an investigation for web browsers to find out the set of strings that can cause their JavaScript interpreter to be invoked. Our own strategy does not cover a few exploits particular to browsers other than Firefox. Because of the complication of layout engine code, we expect that such a testing will involve some communication with the user.

7. REFERENCES

[1] D. Kristol, "HTTP State Management Mechanism" in Internet Society, 2000. http:// www.ietf.org/rfc/rfc2965.txt

[2] Open Web Application Security Project: https://www.owasp.org/index.php/Top_10

[3] White Hat Security. Website Security StatisticsReporthttp://www.whitehatsec.com/home/ resource/stats.html, 2008.

[4] J. Garcia-Alfaro and G. Navarro-Arribas, "Prevention of Cross-Site Scripting Attacks on Current Web Applications," Available: http://hacks-galore.org/guille/pubs/is-otm-07.pdf

[5] S. Saha, "Consideration Points: Detecting Cross-Site Scripting," (IJCSIS) International Journal of Computer Science and Information Security,Vol. 4, No. 1 & 2, 2009.

[6] Brian Blankenship, Introduction to Cross-Site Scripting using WebGoat, The OWASP LiveCD Education Project, 2005.

[7] Acunetix, Vulnerability Scanner "http:// www.acunetix.com/vulnerability_scanner".

[8] Zhushou Tang, Haojin Zhu, Zhenfu Cao, Shuai Zhao, L-WMxD: Lexical Based Webmail XSS Discoverer, IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2011, pp. 976-981.

[9] Gary Wassermann, Zhendong Su, Static Detection of Cross-Site Scripting Vulnerabilities, ACM/IEEE 30th International Conference on Software Engineering (ICSE), 2008, pp. 171-180.

[10] N. Jovanovic, C. Kruegel and E. Kirda, Precise alias analysis for static detection of web application vulnerabilities. In: ACMSIGPLAN Workshop on Programming languages and Analysis for Security, Ottawa, Canada, 2006.

[11] AppScan,http://www01.ibm.com/software/awdtools/appscan/.

[12] Nessus, http://www.nessus.org/.

[13] D. Scott and R. Sharp. Abstracting Application-level Web Security. In 11th World Wide Web Conference, 2002.

[14] M. T. Louw and V. N. Venkatakrishnan, "Blueprint: Robust Prevention of Cross-Site Scripting Attacks for Existing Browsers", Proc. 30th IEEE Symp. Security and Privacy (SP 09), IEEE CS, 2009, pp. 331-346.

[15] W. Halfond, A. Orso, and P. Manolios, "WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation", IEEE Trans. Software Eng., Jan. 2008, pp. 65-81.

[16] E. Kirda et al., "Client-Side Cross-Site Scripting Protection," Computers & Security, Oct. 2009, pp. 592-604.

[17] P. Vogt, F. Nentwich, N. Jovanovic, C. Kruegel, E. Kirda, and G. Vigna. Cross site scripting prevention with dynamic data tainting and static analysis. In 14th Annual Network and Distributed System Security Symposium (NDSS), 2007.

[18] O. Hallaraker and G. Vigna, Detecting Malicious JavaScript Code in Mozilla. In Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), 2005.