**Cross-Site Scripting (XSS) Worms in Online Social Network (OSN): Classification and Defensive Mechanisms**

Pooja Chaudhary Department of Computer Engineering

National Institute of Technology

Kurukshetra, Haryana, INDIA [pooja.ch04@gmail.com](mailto:pooja.ch04@gmail.com)

**Abstract –The propagation of XSS worms on the social networking sites like Twitter, LinkedIn, Facebook, etc. has observed exponential growth in modern era of Web 2.0 technology. According to recent survey, 43% of web applications are vulnerable to XSS worms. Such unbearable growth of XSS worms has raised some serious security and privacy concerns in OSN. This article discusses a detailed classification of XSS attacks and presents the recent occurrences of XSS attacks on numerous platforms of OSN-based web applications. Numerous existing XSS defensive solutions on OSN have been discussed in order to identify their main contributions and existing performance issues. We present the unique security challenges and issues that exist in the recent state-of-art techniques and based on this, we recommend the further scope.**

***Keywords – Cross-Site Scripting (XSS) Worms, HTML, HTTP, Java Script Code Injection Attacks, Online Social Network (OSN),***

1. INTRODUCTION

Web 2.0 enables user to interact and collaborate with each other in virtual community through the data generation on the social media. This leads to the abrupt increase in the popularity of the OSN site. OSN is virtual place that facilitate communication. Today many OSNs have tens of millions of registered users. Most popular and largest OSN is Facebook with over more than 1 billion active users [18]. Other famous OSNs are Google+ has approximately 235 million active users [19]; Twitter secures more than 200 million active users and LinkedIn covers approximately 160 million active users. Social networks are the online virtual network where user can find new friends, develop their friends’ circles, can share their photos, videos, text, and many more interested item. Its popularity is rising because user spend high amount of time to up-to-date their information by interacting with others users. Social networking is not limited to informal use but can be used for formal purposes like education, job finding, etc. The large amount of information (personal/professional) is stored by user on these networks, so hackers are more attracted toward social networks. Attackers can make use this available information for its benefits. There are different types of attacks on OSN sites like phishing, Clickjacking, identity clone, Social bots, malwares, XSS attack, etc.

XSS attack is the one of the most treacherous attack on the OSN server [11-13]. Approximately 80% of the today’s Web applications over the WWW are uncovered to the XSS attack. In this attack, adversary attempts to lure the victim either by clicking on the malicious link or visits the infected Web page so that malicious script injected by the attacker gets executed at the browser side. Attacker performs this activity to gain access to user’s private information like username and password. Numerous recent client-side and server-side XSS defensive techniques have been proposed by various researchers for detecting and alleviating the effect of XSS worms from real world web applications. However, these existing frameworks are not capable enough to detect as well as mitigate the dissemination of XSS worms from OSN-based Web applications. In addition to this, existing defensive techniques suffer from high false positive and negative rate. In this paper, we present some of the statistics of the recent occurrences of XSS attacks on OSN and a detailed classification of XSS worms. Numerous existing XSS defensive solutions have been discusses in this article with the key goal to identify the main contributions and the existing performance issues. Finally, we have recommended some of the future research directions corresponding to the identified research gaps.

The rest of the outline of this article is structured as follows: Section II illustrates the detailed overview on XSS worms in OSN with the taxonomy of XSS attacks. Detailed study of recent XSS defensive methodologies of OSN platforms is presented in Section III. Further research directions are recommended in Section IV and finally section V will concludes the article.

1. BACKGROUND ON XSS WORMS IN OSN

XSS attack was introduced in the year 2000 in CERT. Improper input validation is the root cause of the XSS attack. It occurs in those systems where user provides data at one interface and it is displayed on some other’s browsers are vulnerable to the XSS attack [14-17]. For example, forums, blogging Web sites, OSN sites, etc.

1. *Recent Victims of XSS Worms*

2131

978-9-3805-4421-2/16/$31.00 *§*c 2016 IEEE

This subsection highlights the incidences of the XSS attack on the OSN Web application. Table I highlights the statistics of such incidences.

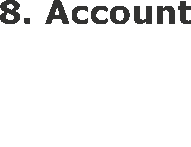


TABLE I. STATISTICS OF XSS ATTACKS ON OSN

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Target WebSite** | **Month, Year** | **Consequence** |
| 1. | UK Parliament Web Site | March, 2014 | Disinformation |
| 2 | Yahoo Web Site | January, 2013 | Account Hijacking |
| 3 | Hotmail Web Site | June, 2011 | Session Hijacking |
| 4 | Twitter Web Site | September, 2010 | Worm |
| 5 | Orkut Web Site | December, 2007 | Worm |

As it is reflected by the table I, that XSS worms have been found in the popular web sites, especially OSN platform due to its attractive features for sharing of data. The Yahoo mail users have been infected by the exploitation of DOM-based XSS worms. Consequently, either the users have to click on the maliciously crafted link or receive spam messages. Hotmail web services have been infested by the XSS vulnerability in which users are compromised by simply previewing the malicious e-mail messages. Twitter, the famous OSN site, contains a XSS flaw which displays a pop-up window that demands user’s login credentials. Orkut, popular OSN site, is also hitted by the persistent XSS worm. In this, attacker inject malicious scripts into his profile and persons, whosoever, visit the profile of attacker became infected. It has infected approximately 650,000 Orkut users.

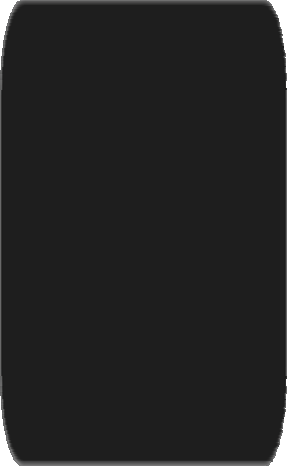
1. *Classification of XSS Worms*

XSS attack enables an attacker to vaccinate client-side malicious JavaScript into the source code of the Web pages visited by the benign users. When this script is rendered by the browser it is executed with the same permissions as the benign script code. Thus, it gets processed by the browser resulting into the theft of the personal information and leads to successfully launch of XSS attack. It is a type of the implemented at the application layer of the network hierarchy. It targets those scripts which get executed at the client side instead at the server side. It arises due to security loopholes in the advanced technologies to develop the Web site like HTML, XML, JavaScript, AJAX and so on. It is activated by the modification of the client side scripts in the Web application in a way that can be anticipated by the attacker. It impact magnitude can be measured by the presence of the sensitive information at the client side and theft of this data. Fig. 1 describes the scenario of simple XSS attack.

Fig. 1. Simple scenario depicting XSS attack

XSS attack can be categorized into 3 types:

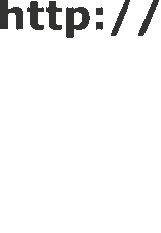
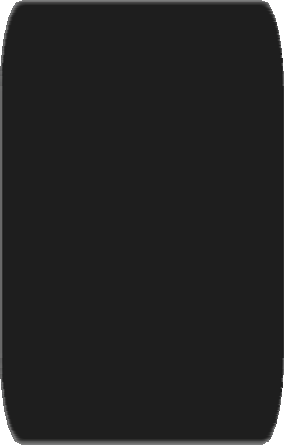
*Persistent/Stored XSS Attack:* It is the most dangerous attack because illicit script code resides at the server side permanently. In this attack, adversary inserts the malicious script code into the server i.e. in the database. So it persists there for a long period of time. When victim visits the Web site and request for the infected page then response page will also include the script provided by the attacker. This code is perceived by the browser and leads to harmful effects. Fig. 2 shows the malicious script code used inserted in the source code of the application.



**<script><input class="inputbox" type="text" name="email" size="40" value="**[**aaa@aa.com**](mailto:aaa@aa.com)**"/> document.location="**[**http://www.malicious.site/**](http://www.malicious.site/) **collectcookie.php?"+document.cookie </script>**

Fig. 2. Script code to initiate persistent XSS attack

*Non-Persistent/Reflected XSS Attack:* In this, attacker craft the URL containing the malicious script code and send it to the user. When user clicks this link then a request is sent to the server. Server will respond with an error message as the requested resource is not found in the server database. Malicious code is reflected back in this response to the client side and rendered by the browser. Fig. 3 illustrates the crafted malicious URL.



*2016 International Conference on Computing for Sustainable Global Development (INDIACom)*

Fig. 3. URL crafted by the attacker to initiate reflected XSS attack

*DOM-Based XSS Attack:* Browser parses the HTML document to produce the DOM tree. It is a tool through which browser interprets the document. This type of attack mainly cause due to the inappropriate handling of the data affect the DOM.

DOM properties like document.location and document.url are used by the attacker to manipulate the DOM objects to perform maliciously. It is a client side attack server is not conscious about this attack. Basically it exploits the innerHTML property to inject code.

*2016 International Conference on Computing for Sustainable Global Development (INDIACom)*

1. DEFENSIVE XSS METHODOLOGIES ON OSN

This section discusses the recent XSS defensive techniques on the OSN-based web application with the key goal to identify the main contributions of their work and existing performance issues. Table II highlights the strengths and limitations of existing defensive techniques of XSS attacks. Samuel et al. [1] proposed a novel method which restricts the XSS attack by applying the context sensitive sanitization on the Web Templating frameworks. This will ensure the protection of the Web application from the XSS attack by construction. Its two main components are: Context Type Qualifier and CSAS engine (Context Sensitive Auto Sanitization) engine. Context type qualifier qualifies the variables in the templates. For every untrusted variable it records the context in which it is interpreted and attach the qualifier as Safe or Unsafe on the basis of whether that variable is sanitized or not.

Louw et al. [2] provides robust protection from the XSS attacks in spite of having irregular browser parsing behavior. The existing browsers are not able to discriminate between the malicious and benign JavaScript code effectively due to presence of parsing quirks. Thus, blueprint allows the Web application to effectively generate a structural illustration of the unreliable HTML content called “blueprint” which is devoid of XSS attack. Basically it bridges the gap between the understanding of the Web application of the HTML generated Web content and the understanding of the Web browser has of the similar content. For legitimate JavaScript content this difference is not countable but this may create security related problems in the case of the untrusted JavaScript content. It makes browser independent of the generation of the DOM tree which will contain the Script node used to inject the XSS payload. It safely constructs the HTML tree at the server side and sends it to the browser in the encoded format.

Bisht et al. [3] proposed a server residing solution (i.e. XSS- Guard) to protect against the XSS worms by determining a collection of scripts that may be present in the generated we page in response to a HTTP request. It provides a method to check the collected scripts and remove those scripts that are not present in the HTML document generated by the web application. To achieve this, it generates an impression of the web page that is supposed to include both legal and illegal scripts. Any deviation between originally generated we page and shadow web page identify XSS attack. The main aim of the XSS-Guard is to evade malicious script data to become a part of the HTTP response web page. This technique aims to

discover any malicious script data that may go unobserved during the input filtering method.

Balzarotti et al. [4] proposed a technique that utilizes the static and dynamic analysis to determine the incorrect sanitization methods that can be circumvented by the attacker. Their methodology proposed two balancing methods to model the sanitization method. The first method is founded on static analysis models which states that how a web application modifies its input values with the help of exact modeling of string manipulation routines. While, the second technique presents a dynamic analysis method which uses a bottom-up approach and tries to recreate the source code used by the web application and then run this recreated code with the malicious input values in order to determine the defective sanitization routines. Also the authors implemented their methodology in a tool known as Saner, and test it on real time web applications. The key objective of this technique is to analyze the working of sanitization procedures to report the web application vulnerabilities like XSS attack and SQLI.

Cao et al. [5] proposed a technique (i.e. Path Cutter) that works by restricting the two main steps in the proliferation of the XSS worm: 1) access to DOM, at client side, to different views. 2) Illegitimate HTTP request to the OSN server. Path Cutter is a server side approach which attempts to reduce the impact of the XSS worm by constraining its dissemination path. However, it is not able to protect each and every view from being infected from the XSS attack vector. Saxena et al. [6] proposed a technique that utilizes the randomization of the application source code and tracing of the untrusted variables at runtime, in the JavaScript content present in the HTML code. it is done at both sides i.e. server and client. XSS attempts to modify the content of the Web application’s responded Web page to the client to successfully inject the malicious code. Thus, this technique aims to preserve the content’s integrity by constraining the XSS worm not to change the content of the Web page.

Likarish et al. [7] proposed a technique that works in three steps: 1) data collection, in this phase scripts are collected and stored into the feature database. Benign scripts are collected and then malicious scripts are integrated with them into the database. This phase is also considered as the training phase to build up a robust database of scripts. 2) feature extraction, in this phase features in the Web page are extracted to use them to identify that this page is infected or not.3) classifier, in this phase various classifier like Naïve Bayes, Support Vector Machine (SVM), ADTree, and RIPPER. Then their performance is evaluated. Classifiers use the extracted features and database to detect the infected Web page.

Johns et al. [8] keeps track of the used JavaScript code in the Web application and identify the variance in the application’s JavaScript set. This variance id used to identify the change in the Web application’s source code or is an indicator of the XSS attack. It is a passive technique which is

TABLE II. DETAILS OF RECENT XSS DEFENSIVE TECHNIQUES

|  |  |  |
| --- | --- | --- |
| **Techniques** | **Strengths** | **Limitations** |
| Samuel et al. [1] | It has low performance overhead. It protects the Web application from XSS attack by construction through applying context aware sanitization in the templates of the Web application | Context determination of the untrusted variable in Web templates at some places is not effectively done due to parsing quirks of the browser |
| Blueprint [2] | It guarantees the script less construction of the HTML tree at the browser side despite of having browser parsing quirks. It ensures that the Web application and browser have identical understanding of the HTML generated content | It incurs performance overhead. It requires modifications at both the sides i.e. client and server side. It does not protect against the non-scripting attacks. It is difficult to apply as changes the way application generates the HTML content for browser. |
| XSS-GUARD [3] | It allows the Web application to use rich HTML content in its Web pages. It attempts to detect the presence of scripts by observing the behavior of infected and benign HTTP response. | Use of this technique disturbs the browser’s parsers to deduce unsafe HTML code which may be susceptible to XSS attacks based on the use of browser parse quirks. |
| Saner [4] | This method utilizes both static and dynamic analysis approaches to identify the existence of the illicit JavaScript script code into the source code of the Web application. | Deployment cost of this technique is high due to the use of dynamic components. The new dynamic string used for the XSS Attack is not detected by it. |
| PathCutter [5] | It minimizes the propagation path of the XSS worm by constraining it to a particular view. | This technique does not provide protection against drive-by- download attack, phishing attack and not protect each view from being infected. |
| DSI [6] | It enforces the integrity constraint on the untrusted content in the Web application to defend against the injection of the malicious JavaScript code. | It is not effective against the detection of the DOM based XSS attack. It requires modifications at client side and server side as well. |
| Likarish et. al. [7] | It utilizes the features to detect the JavaScript obfuscation which is used for XSS attack. It has high detection rate and less false alarm rate. | There are some benign scripts for which Web application use obfuscation to send it to the user. So, classifier in this case does not work correctly. |
| XSSDS [8] | It possesses the capability of detecting the injected malicious script code by calculating the deviation between HTTP request and HTTP response web page. | Due to the absence of recent malicious script in the database false positive rate increases while detecting the stored XSS attack. |
| Pelizzi et al. [9] | It does the approximate string matching to detect the partial script injection to prevent against the reflected XSS attack. | Partial matching of malicious string may become the cause of increase in the false negative rates. Initially it has high false positive rate. |
| Noncespaces [10] | It use randomization method on the XML namespace prefixes of tags in each document. It is highly unpredictable so attacker cannot guess it to inject the code. It basically removes all the difficulties occur during sanitization process. | It is not effective to provide protection against the malicious JavaScript code from the remote Web site. |

*2016 International Conference on Computing for Sustainable Global Development (INDIACom)*

implemented at the server side. It is a robust technique which provides protection against traditional server side XSS attack. It identifies the reflected XSS attack by assessing the irregularity between the request’s parameters and response’s parameters produced by the Web application and sanitized by the filters. It recognizes the occurrence of the persistent XSS attack by collecting the script in the Web application at the server side and then measuring the deviation between these and injected scripts. If any script is not found in the recorded scripts it is considered as the initiation of the XSS attack. Pelizzi et al. [9] proposed XSSFilt, which performs the partial script matching i.e. modification of the existing script to insert the malicious code. This will enable this technique to handle the Web applications that executes the application specific sanitization. It defines some policy to detect the XSS attack either via the injection of the entire scripts or through the insertion of the malicious parameters in to the benign scripts. Firstly, request is send to the server and response is send back to the client where it is received by the *init* method of XSSFilt to parse the URL to detect the reflected XSS attack. Gundy et

al. [10] proposed a method that allows the Web client to discriminate between the legitimate and illicit JavaScript code to thwart XSS attack. This method utilizes the randomization of the XML prefixes of the tags embed in each document of the Web application. This randomization is highly unpredictable so that attacker cannot predict this prefix to inject the malicious script code. This randomization is done to fulfill two purposes: 1) this helps in the identification of the untrusted data so that client enforces the security policy to be safe. 2) This prevents unwanted distortion of the DOM tree by the untrusted content given by the attacker. To achieve this goal, it divides the Web application at the server side into different trust classes. Then some policy decides, at the client side, the privileges that a trust class can have. It requires collaboration from client and server side both. Server classifies each element and attribute in the XHTML document into trust classes. And also transfers the policy which defines the classification criteria of the trust class so that at client side it can be decided what a trust class can perform. In addition to this, Table III highlights the evaluation of

Cross-Site Scripting (XSS) Worms in Online Social Network (OSN): Taxonomy and Defensive Mechanisms

TABLE II. COMPARISON OF XSS DEFENSIVE TECHNIQUES BASED ON WELL DEFINED PARAMETERS

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Parameters Techniques** | **Implementation location** | **Persistent XSS attack Detection** | **Non- persistent XSS attack Detection** | **DOM based XSS attack Detection** | **Invigilation type** | **Modification location** | |
| **Client- Side** | **Server Side** |
| **Samuel et al. [1]** | Server | Yes | Yes | Yes | Active | No | Yes |
| **Blueprint [2]** | Server | Yes | Yes | No | Active | Yes | Yes |
| **XSSGUARD [3]** | Server or proxy | Yes | Yes | Yes | Active | No | Yes |
| **Saner [4]** | Server | Yes | Yes | Yes | Passive | No | Yes |
| **PathCutter [5]** | Server or proxy | Yes | Yes | Yes | Active | No | Yes |
| **DSI [6]** | Client and server | Yes | Yes | No | Active | Yes | Yes |
| **Likarish et al. [7]** | Client | Yes | Yes | Yes | Passive | No | No |
| **XSSDS [8]** | Client | Yes | Yes | Yes | Passive | No | No |
| **Pelizzi et al. [9]** | Client | Yes | Yes | Yes | Passive | Yes | No |
| **Noncespaces [10]** | Server | Yes | Yes | No | Active | No | Yes |

existing XSS defensive techniques based on some well-defined parameters as their comparative study. It is clearly reflected from the Table III that most of the existing techniques does not protect against DOM-based XSS attack.

*2016 International Conference on Computing for Sustainable Global Development (INDIACom)*

1. DISCUSSION AND FUTURE SCOPE

XSS attack is an assault against the Web application especially OSN sites, where an adversary introduces the illicit scripting code into the Web application code. It aims to steal the sensitive information of the user like cookie information. Recent existing defensive solutions of XSS attacks on OSN needs the implementation of infrastructure of web application and therefore bring the runtime performance overhead. No proper adequate mechanism is introduced to distinguish among benevolent JavaScript code and malevolent JavaScript code in the recent state-of-art techniques. Therefore, a non-acceptable rate of false positives and false negatives is observed in the present defensive solutions of XSS attacks. Various existing literature on XSS defensive techniques restrict the web crawler to test the code of web applications for deriving the possible user-injection points and the derived specifications of web applications. In addition to this, Numerous OSN web applications perform the sanitization in their JavaScript code in a context insensitive manner. The necessity of sanitization practices fluctuate in different environments.

Keeping in view, all these performance issues encountered in all these existing state-of-art techniques, a novel robust XSS defensive for OSN web applications is required that must overcome all such issues and meet the following requirements:

x Technique must inject the sanitizers in the JavaScript code in a context-sensitive manner.

x Crawling mechanism should be optimized enough that detects all the vulnerable injection points of OSN-based

web applications.

x All the four exploitation techniques of XSS worms must be easily detected by the technique.

x Defensive methodology must be strong enough to diminish the rate of false positives and false negatives.

1. CONCLUSION

The enormous utilization of services of OSN-based Web applications (like Facebook, Twitter, LinkedIn, etc.) without any concern about the propagation of XSS worms can lead the online user to become a victim of XSS attack. XSS worms are injected in the form of post on OSN web applications for hijacking the online user’s accounts. This paper presents the detailed classification of XSS worms and discusses some of the statistics of recent incidences on XSS attacks. In addition to this, we discussed the performance issues in the existing XSS defensive techniques and pointed out some further research directions with the key goal to eliminate the effect of XSS worms from OSN-based web applications.

REFERENCES

[1] Samuel, Mike, Prateek Saxena, and Dawn Song. "Context-sensitive auto- sanitization in web templating languages using type qualifiers." In Proceedings of the 18th ACM conference on Computer and communications security, pp. 587-600. ACM, 2011.

[2] Louw, Mike Ter, and V. N. Venkatakrishnan. "Blueprint: Robust prevention of cross-site scripting attacks for existing browsers." In Security and Privacy, 2009 30th IEEE Symposium on, pp. 331-346. IEEE, 2009.

[3] Bisht, Prithvi, and V. N. Venkatakrishnan. "XSS-GUARD: precise dynamic prevention of cross-site scripting attacks." In Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 23-43. Springer Berlin Heidelberg, 2008.

[4] Balzarotti, Davide, Marco Cova, Vika Felmetsger, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. "Saner: Composing static and dynamic analysis to validate sanitization in web applications." In Security and Privacy, 2008. SP 2008. IEEE Symposium on, pp. 387-401. IEEE, 2008.

[5] Cao, Yinzhi, Vinod Yegneswaran, Phillip A. Porras, and Yan Chen. "PathCutter: Severing the Self-Propagation Path of XSS JavaScript Worms in Social Web Networks." In NDSS. 2012.

[6] Nadji, Yacin, Prateek Saxena, and Dawn Song. "Document Structure Integrity: A Robust Basis for Cross-site Scripting Defense." In NDSS. 2009.

[7] Likarish, Peter, Eunjin EJ Jung, and Insoon Jo. "Obfuscated malicious javascript detection using classification techniques." In Malicious and

Unwanted Software (MALWARE), 2009 4th International Conference on, pp. 47-54. IEEE, 2009.

[8] Johns, Martin, Björn Engelmann, and Joachim Posegga. "Xssds: Server- side detection of cross-site scripting attacks." In Computer Security Applications Conference, 2008. ACSAC 2008. Annual, pp. 335-344.

IEEE, 2008.

[9] Pelizzi, Riccardo, and R. Sekar. "Protection, usability and improvements in reflected XSS filters." In ASIACCS, p. 5. 2012.

[10] Van Gundy, Matthew, and Hao Chen. "Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart Cross- Site Scripting Attacks." In NDSS. 2009.

[11] Shashank Gupta and B.B.Gupta, “XSS-SAFE: A Server-Side Approach to Detect and Mitigate Cross-Site Scripting (XSS) Attacks in JavaScript Code”, Arabian Journal for Science and Engineering, Springer, pp. 1-24, 2015.

[12] Shashank Gupta, B.B.Gupta, “Cross-Site Scripting (XSS) Attacks and Defense Mechanisms: Classification and State-of-Art”, International Journal of System Assurance Engineering and Management, Springer, 2015.

[13] B.B.Gupta, Shashank Gupta, S.Gangwar, M.Kumar, P.K. Meena “Cross- Site Scripting (XSS) Abuse and Defense: Exploitation on Several Testing Bed Environments and its Defense”, Special Issue of Secured Communication in Wireless and Wired Networks in Journal of Information Privacy and Security, Taylor & Francis Online, Vol. 11, Issue 2, pp. 118-136, 2015.

[14] Shashank Gupta, B.B. Gupta, “BDS: Browser Dependent XSS Sanitizer”, Book on Cloud-Based Databases with Biometric Applications, IGI-Global’s Advances in Information Security, Privacy, and Ethics (AISPE) series, pp. 174-191, USA, 2014.

[15] Shashank Gupta, Lalitsen Sharma et al. “Prevention of cross-site scripting vulnerabilities using dynamic hash generation technique on the server side”. International journal of advanced computer research (IJACR), 2012, 49-54.

[16] Shashank Gupta, Lalitsen Sharma “Exploitation of Cross-site Scripting (XSS) vulnerability on Real World Web Applications and its Defense” International journal of computer applications (IJCA), pp. 28- 33, 2012.

[17] Shashank Gupta, B.B. Gupta, “PHP-Sensor: A Prototype Method to Discover Workflow Violation and XSS Vulnerabilities in PHP web applications”, in 12th ACM International Conference on Computing Frontiers (CF ’15), Ischia, Italy, 2015.

[18] Facebook, Facebook Reports Fourth Quarter and Full year 2013 Results, accessed Jan. 14, 2014. [Online]. Available: <http://investor.fb.com/releasedetail.cfm?ReleaseID=821954>

[19] Google+, accessed Jan. 14, 2014. [Online]. Available: https://plus.google.com/