Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 85 (2016) 198 – 205

Procedia Computer Science

International Conference on Computational Modeling and Security (CMS 2016)

CSSXC: Context-Sensitive Sanitization Framework for Web Applications against XSS Vulnerabilities in Cloud Environments

Shashank Gupta[#], B. B. Gupta[*]

Department of Computer Engineering, NIT Kurukshetra, India

Abstract

This paper presents a context-sensitive sanitization based XSS defensive framework for the cloud environment. It discovers all the hidden injection points in HTML5-based web applications deployed on the platforms of cloud and sanitizes the XSS attack payloads injected in such points in a context sensitive manner. The identification of such injection points permits our technique to retrieve each possible web page of application, allowing a wider exploration and accelerating the process of applying the sanitizers on the untrusted variables of web application. The XSS attack mitigation capability of our framework was evaluated on web applications deployed for the cloud users in the cloud environment. The experimental results reveal that this technique detects the XSS attack payloads with minimum rate of false negatives and less runtime overhead.

Peer-review under responsibility of the Organizing Committee of CMS 2016

*Keywords:*Cloud Computing; Cross-Site Scripting (XSS) attacks; JavaScript Code Injection Attacks; HTML5; Cloud Security.

1. Introduction

Currently, cloud computing is apparently considered to be the utmost outlook technologies because of its high flexibility as well as its cost effectiveness (Almorsy, 2010). Instead of referring the outdated Internet settings for constructing an expensive setup, nowadays numerous IT enterprises utilize the capabilities of cloud techniques (Tiwari, 2015). Therefore, several IT organizations install the setup of their web applications in the infrastructures of cloud. However, it is clearly known that the cloud settings are installed on the backbone of Internet. Therefore, numerous web application vulnerabilities in the conventional Internet infrastructures also exist in the backgrounds of cloud-based environments. XSS vulnerability is considered to be one of topmost web application vulnerability (Gupta, 2015a) and have now turned out to be a critical security concern, as web applications are facing this problem from the time when XSS first time discovered in the year 2000.

E-mail address: gupta.brij@gmail.com [*](Tel. +91-1744-233488), mr.shashank.gupta@ieee.org[#]

Peer-review under responsibility of the Organizing Committee of CMS 2016
doi:10.1016/j.procs.2016.05.211

This attack generally occurs due to the injection of malicious scripts in the vulnerable injection points of web application (Gupta, 2014).Fig. 1 highlights the abstract view of XSS attack. XSS falls third in the list according to the statistics of OWASP Top 10 2013 (OWASP, 2013). Recently, numerous approaches have been proposed to detect and mitigate the effect of XSS vulnerabilities from real world web applications. XSS Auditor (Bates, 2010) is a filter that realizes equally extraordinary performance as well as high accuracy via jamming scripts following the HTML parsing and prior to execution. Noncespaces (Gundy, 2009) is an end-to-end mechanism that facilitates web browsers to differentiate between benign and malicious content to apply the techniques from Instruction Set Randomization (ISR) for thwarting the exploitation of XSS vulnerabilities. XSS-Guard (Bisht, 2008) is a server-side solution for defending against the XSS attacks by discovering the collection of scripts that a web application intends to create for any HTML web request. BLUEPRINT (Louw, 2009) is a server-side solution for thwarting XSS attacks where the web application transfers two replicas of output HTML document to a web browser for detecting any deviation, one with user inputs and other with legitimate values. (Saxena, 2010) proposed a hybrid and dynamic analysis methodology, which examine the JavaScript-based web applications for the discovery of input validation vulnerabilities. This technique was executed in a tool known as FLAX, a taint enriched black-box fuzzer, which is capable of finding the client side validation bugs in Java script programs. However, the testing of FLAX has not focused on the complexity of sanitization errors, which still remain in the client-side Java script code. (Livshits, 2013) proposed an automated technique of sanitizer placement by statically analyzing the stream of infected data in the program. However, this technique presumes every possible source, sinks and sanitizers to be identified in advance, and as a result suffers from non-tolerable runtime overhead.

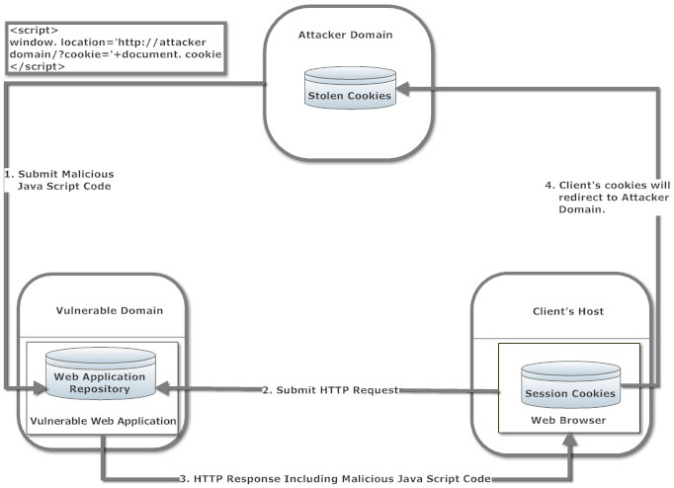

Fig. 1. Abstract View of XSS Attacks

With the escalating boom in the field of cloud computing, the investigation on the security of web applications deployed in the cloud environment turn out to be a significant challenge. XSS attacks are turned out to be plague for the modern HTML5-based web applications (Gupta, 2012). Therefore, the defense of XSS attack has now become a pre-requisite for the web applications deployed on the platforms of cloud computing. In addition to this, the existing XSS defensive solutions are inappropriate for the cloud platforms and cannot be easily integrated in the cloud environments. As such solutions demand countless modifications in web browsers and in the source code of web applications. In addition to this, existing XSS defensive solutions do not provide real time protection (Gupta, 2016).

To address these issues, this paper presents a robust XSS defensive framework for the HTML5-based web applications deployed in the cloud environment. It discovers all the hidden injection points in the cloud-based web applications and performs context-sensitive sanitization on the XSS attack payloads injected in such points for mitigating the effect of XSS vulnerabilities from these web applications. This framework does not need any alterations required on the web browser and in the source code of web applications. The testing and evaluation of our work was done on real world web applications deployed in the cloud environment. The observed results

demonstrate that our framework has a high rate of precision and almost negligible impact on the performance of web applications deployed in the cloud platforms.

2. Background and Motivation

XSS attack is the utmost harmful and widespread security issue affecting web applications. A successful cross site scripting attack can result in stern security violations for not only the user but also for the web site. (Gupta, 2012a), (Gupta, 2015b), (Gupta, 2015c), (Gupta, 2012b).This attack is accelerating exponentially on the cloud environments. Moreover, the effect of XSS attacks has been seen globally by the entire cloud service providers. The main motivation of our work is based on the following key challenges in the existing cloud environments:

- Fragile input validation mechanisms deployed in the web applications of cloud environments.
- Lack of existing XSS defensive frameworks on the platforms of HTML5-based web applications.
- Integration of existing XSS defensive solutions on the cloud platform demand modifications at the web browser and web server.
- Absence of context-sensitive sanitization in the existing XSS sanitization-based solutions.
- Existing cloud platforms avoid the inclusion of legitimate HTML and JavaScript code in the web applications installed in their environments.
- High rate of false positives encountered in several existing XSS defensive solutions.

Based on these key issues, this paper presents a novel XSS defensive framework integrated on the cloud platforms for the real world HTML5-based web applications. This novel framework detects the vulnerable injection points in the web applications deployed in the cloud environments. Our technique sanitizes the injected XSS attack vectors in a context-sensitive manner. The evaluation of our proposed framework was done on a tested suite of real world HTML5-based web applications deployed in the cloud environment and mitigates the effect of XSS attack. Our framework can be easily integrated in the existing cloud environments and does not need any modifications on the web browser and web server. The next section illustrates the detailed description of our proposed framework.

3. Proposed Framework

This paper presents a robust XSS defensive framework that detects and mitigates the XSS vulnerabilities in the web applications deployed in the cloud environment. The framework is completely based on the discovery of hidden vulnerable injection points in the web applications. In addition to this, it detects the malicious XSS payloads by referring the blacklist of JS attack vectors from the freely available XSS repository (RSnake, 2008). Finally, it performs an automated placement of sanitizers in a context-sensitive manner on the untrusted variables of XSS attack payloads for mitigating the effect of XSS vulnerabilities from the web applications deployed on the cloud platform. Fig. 2 depicts the overall sketch of proposed framework. This framework comprises of three entities namely Cloud User, Web Application Server (WAS) and Malicious JavaScript (JS) Detection Server, which are as follows:-

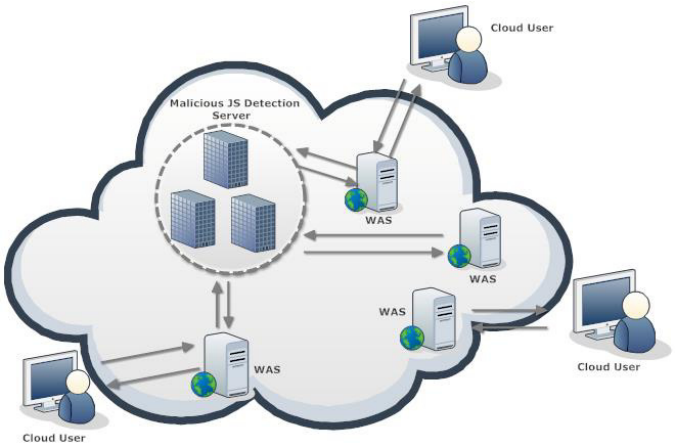

Fig. 2. Overall Sketch of Proposed XSS Defensive Framework

1) Cloud User: The cloud user can be a simple web application user, web application admin or a cloud platform admin. Cloud user can access the services of web application deployed in the cloud environment by directly interacting with the Web Application Server (WAS).
2) Web Application Server (WAS): This server has one main sub-module: Injection Point Locator (IPL). The key goal of IPL is to detect the hidden vulnerable injection points in the HTML5-based web applications deployed in the settings of cloud platforms.
3) Malicious JavaScript (JS) Detection Server: This server has also one sub-module: Sanitization Routine Injector (SRI). This server will access the JavaScript attack vectors from the freely available XSS attack repositories (RSnake, 2008) and the SRI will perform the automated injection of sanitizers in a context-sensitive way on these untrusted variables of malicious XSS payloads.

3.1. Extraction Unit

This segment extracts info like form parameters, forms, links and recognizes the functions (e.g. GET or POST). This unit seizes all possible HTTP requests and decodes them. It also extracts all the possible web links based on the extracted requests and by referring the possible injection points stored in an analytical repository. Link addresses could be saved in the data repository so that such addresses can be retrieved in future. In addition to this, this unit follows the fixed set of steps in the form of well-designed algorithm for discovering the injection points in the web application. This segment also maintains the statistics of collected XSS attack vectors in the well-defined reports. Fig. 3 highlights the detailed algorithm for the extraction of web links and Injection Points (IPs) of web application.

Algorithm: Web Link and Injection Points (IPs) Extraction

Link Extraction	1.	Start
	2.	Generate 'N' web addresses (W_1, W_2, W_3, --------- W_N) of all the blacklisted websites.
	3.	Initialize a threshold value $x \rightarrow 0$ for crawling all positions of web pages up to maximum value of x.
	4.	Declare an array 'URL_Log' for including the list of crawled web pages.
	5.	Store all possible extracted web addresses in the URL_Log file in first iteration (i.e. x = 0)
	6.	Capture all the web pages (WP_1, WP_2, WP_3, -------- WP_N) from the extracted web addresses.
	7.	Retrieve all the JavaScript tags, attributes and capture all the possible links.
	8.	Load the database to incorporate all the extracted links.
IPs Extraction	9.	Declare an array 'IP_Log' for incorporating the list of crawled injection points.
	10.	Obtain all web pages from the extracted web addresses i.e. (WP_1, WP_2, WP_3, -------- WP_N).
	11.	Retrieve all the JavaScript tags, attributes while analyzing the list of extracted web pages and capture all the possible links.
	12.	Store all the extracted injection points (I_1, I_2, I_3, --------- I_N) in the IP_Log.
	13.	End

Fig. 3.Algorithm for Extraction of Web Links and IPs

3.2. Identification Unit

This unit explores all the extracted links of the web application, which are being verified for classifying all possible injection points. The main aim of this unit is to find out what XSS attack vector are considered to be more suitable for injection at all extracted injection points of an explicit web form. In addition to this, we have also referred several vulnerable JavaScript functions which can be injected at the extracted vulnerable injection points by an attacker. Table 1 highlights some of the details regarding vulnerable JavaScript functions.

3.3. Sanitization Unit

In this unit, the extracted XSS attack vector templates will be explored for some malicious categories of JS code (For e.g. event handlers, data URI, etc.). Here, the blocks of templates of XSS attack vectors will be sanitized in a context-sensitive manner by an automated technique of placement of sanitizers by SRI in the source code of web

applications. Here, we have developed a procedure of sanitizing the extracted templates of XSS attack vectors by executing the algorithm of sanitization of XSS attack vectors as shown in the Fig. 4. In order to perform the automated placement of sanitizers by SRI, our framework ensures that that every variable outputs associated with a web page are properly encoded/encrypted before being reverted towards the web browser. Our framework has sanitized the XSS attack vectors by using the &# organization followed by some characters. Table 2 highlights some of the list of escape codes.

Algorithm: Sanitization of XSS Attack Payloads

1. Start
2. Initialize an array Sanitize_Log that includes all the list of sanitized XSS attack payloads.
3. Extract all the possible JavaScript attack vectors (JS_1, JS_2, JS_3, -------- JS_N) from the extracted templates of web application.
4. Extract the possible untrusted variables (V_1, V_2, V_3, ------- V_N) that require sanitization from the extracted set of (JS_1, JS_2, JS_3, -------- JS_N).
5. Inject the applicable set of sanitizers (S_1, S_2, S_3, -------S_N) and perform the replacement of extracted untrusted variables (V_1, V_2, V_3, ------- V_N) with these sanitizers.
6. Include all these sanitized set of strings (S_1, S_2, S_3, -------S_N) in the array Sanitize_Log.
7. Finally, inject these sanitized strings (S_1, S_2, S_3, -------S_N) in the extracted IPs (I_1, I_2, I_3, --------- I_N) of templates of web application.
8. Generate the reports and transmit it to the cloud user.
9. End

Fig. 4. Algorithm for Sanitization of XSS Attack Vectors

Table 1. Malicious JavaScript Functions

Method Type	Method Category	Possible Threats
getCookie() setCookie()	Cookie Access	Modification of Cookie Value
String.indexOf() String.charAt() String.spilt() String.fromCharCode() String.charCodeAt()	String Functions	Hide intension, via encryption
window.setInterval() window.setTimeout() eval()	Dynamic Code Execution	Dynamic Code Construction
document.writeln() document.write() element.appendChild() element.innerHTML	DOM Function	Inject imperceptible iframe, malicious script, etc.
location.assign() location.replace()	Modify Existing URL Address	Redirect to malicious URL
getAppName() getUserAgent()	Check Web Browser	Target Particular Web Browser

Table 2. List of Escape Codes

Display	Hexadecimal Code	Numerical Code
“	&#x22;	&#34;
#	&#x23;	&#35;
&	&#x26;	&#38;
‘	&#x27;	&#39;
(	&#x28;	&#40;
)	&#x29;	&#41;
/	&#x2F;	&#47;
;	&#x3B;	&#59;
<	&#x3C;	&#60;
>	&#x3E;	&#62;

3.4. Testing Unit

This unit is accountable for injecting and executing the sanitized XSS attack payload at every possible extracted injection point. Our framework recommends the utilization of a modified automated sanitization procedure, i.e., sanitizers are injected depending on the category of possible injection point, which is being tested. For e.g., suppose the injection point is present at the welcome web page, then only appropriate context-sensitive sanitizers related to welcome page would be utilized. The key motto of such method is to circumvent the execution of large quantity of sanitization tests.

4. Experimental Evaluation

The experiment background is simulated with the help of a normal desktop system, comprising 1.6 GHz Intel Dual Core processor, 2 GB DDR2 RAM and Windows 7 operating system. We have also utilized the VMware Workstation 7 for modifying three virtual desktop systems to act as a malicious JS detection server and including one desktop system to be act as a WAS. In addition to this, we have also utilized a tested suite of four real time web applications vulnerable to XSS attacks and deployed their settings on the cloud environment.

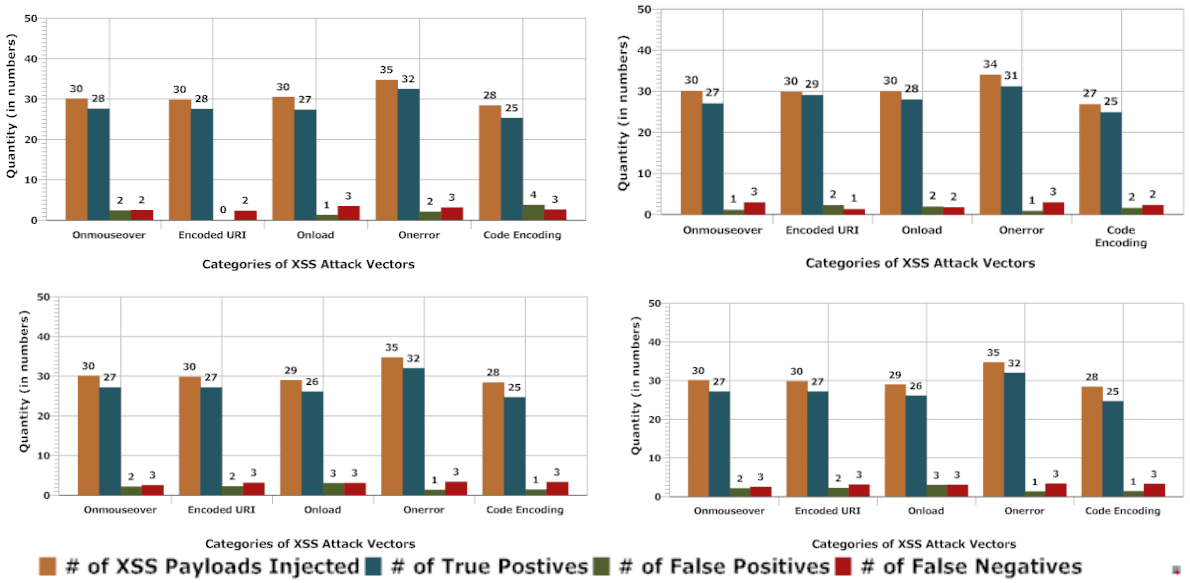

Fig. 5. (a) Observed Results of BlogIt; (b) Observed Results of OsCommerce; (c) Observed Results of Scarf; (d) Observed Results of Wackopicko;

Bloggit is a freely available web application comprising a web search engine. As this web application comprises of numerous input fields and therefore would be vulnerable to XSS attacks. We have not performed any modifications to this web application. Scarf is a conference administration scheme that is utilized for handling meetings, documents, operators and notes. This web application has also recognized with authentication violation vulnerability (CVE-2006-5909). OsCommerce is a freely available e-commerce and online store-management web application. It could be utilized on numerous web servers like MYSQL, WAMP etc. Wackopicko is an online photo sharing web application that allows online users to post their pictures, comment on and also can buy the pictures of other users, etc. This web application is primarily developed for analyzing the vulnerability scanners of web applications. It is vulnerable to wide variety of vulnerabilities, like XSS, SQL injection, etc.

We have also used the malicious XSS attack vectors from the XSS cheat sheet (RSnake, 2008) and tested on the extracted injection points of these web applications. The evaluation results reveal that the malicious JS detection server deployed in the cloud platform precisely injects the sanitizers on the untrusted variables of XSS attack vectors and mitigates the effect of XSS vulnerabilities from web applications deployed in the cloud platform. Fig. 5(a, b, c, d) highlights the detailed statistics of observed results on all the four platforms of web applications deployed on the cloud platforms. It is clearly reflected from these statistics that the proposed design is capable of detecting and mitigating the effect of XSS vulnerabilities with high rate of true positives and low rate of false positives as well as false negatives.

5. Performance Evaluation

Here, we discuss the performance evaluation of our proposed framework during its testing on real world web applications deployed in the settings of cloud platform. We have presented a detailed performance analysis of our proposed design by conducting a statistical analysis method (i.e. F-Measure). F-Measure generally analyzes the performance of system by calculating the harmonic mean of precision and recall. The analysis conducted reveals that our framework exhibits high performance as the observed value of F-Measures in all the platforms of web applications in 0.9. Table 3 highlights the detailed performance analysis of our approach.

$$Precision = \frac{True\,Positives}{True\,Positives + False\,Positives}$$

$$\text{Recall} = \frac{True\,Positives}{True\,Positives + False\,Negatives}$$

$$\text{F-Measure} = \frac{2(True\,Positives)}{2(True\,Positives) + False\,Negatives + False\,Positives}$$

Table 3. Performance Analysis of Framework by Calculating F-Measure

Web Applications	# of True Positives	# of False Positives	# of False Negatives	Precision	Recall	F-Measure
BlogIt	140	9	13	0.939	0.915	0.927
OsCommerce	140	8	11	0.945	0.927	0.936
Scarf	137	9	15	0.938	0.901	0.919
Wackopicko	136	10	9	0.931	0.937	0.934

7. CONCLUSION AND FUTURE WORK

This paper presents a novel and robust context-sensitive sanitization based XSS defensive framework for the HTML5-based web applications deployed in the cloud platform. This framework is capable of discovering all the vulnerable user injection points in a web application by executing the crawling session in all the extracted web pages. It performs the context-sensitive sanitization on the malicious XSS attack vector injected in the injection

points of a vulnerable web applications deployed in the cloud environment. This framework is easily integrated on the existing cloud platforms and was evaluated on tested suite of real world web application. The evaluation results reveal that our framework is capable of discovering malicious XSS attack payloads in the web applications deployed in the cloud environment with high rate of true positives and low rate of false negatives and false positives. We will try to test the capabilities of our framework on mobile web applications as well as web applications related to Online Social Networks (OSNs) as a part of our further work.

References

[1] M. Almorsy, J. Grundy, I. Mueller, "An analysis of the cloud computing security problem," In the proc. of the 2010 Asia Pacific Cloud Workshop, Colocated with APSEC2010, Australia, 2010.

[2] B.B.Gupta, S. 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, 2015a.

[3] 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.

[4] OWASP Top Ten Vulnerability Project 2013. Available at: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[5] D. Bates, A. Barth, and C. Jackson. Regular expressions considered harmful in client-side XSS filters. In Proceedings of the Conference on the World Wide Web, pages 91–100, 2010.

[6] M. V. Gundy and H. Chen. Noncespaces: Using randomization to enforce information flow tracking and thwart cross-site scripting attacks. In Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, Feb. 8-11, 2009

[7] P. Bisht and V. N. Venkatakrishnan. XSS-GUARD: precise dynamic prevention of cross-site scripting attacks. In Detection of Intrusions and Malware, and Vulnerability Assessment, 2008.

[8] M. Ter Louw and V. Venkatakrishnan. Blueprint: Precise browser-neutral prevention of cross-site scripting attacks. In Proceedings of the 30th IEEE Symposium on Security and Privacy, May 2009.

[9] P. Saxena, S. Hanna, P. Poosankam, and D. Song. FLAX: Systematic Discovery of Client-side Validation Vulnerabilities in Rich Web Applications. In NDSS, 2010.

[10] Livshits, Benjamin, and Stephen Chong. "Towards fully automatic placement of security sanitizers and declassifiers." ACM SIGPLAN Notices 48, no. 1, pp. 385-398, 2013.

[11] 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) , 60, pp.28-33, 2012a.

[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, 2015b.

[13] Gupta, Shashank, and B. B. Gupta. "PHP-sensor: a prototype method to discover workflow violation and XSS vulnerabilities in PHP web applications." Proceedings of the 12th ACM International Conference on Computing Frontiers. ACM, 2015.

[14] 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), pp. 49-54, 2012b.

[15] Rsnake. XSS Cheat Sheet. http://ha.ckers.org/xss.html, 2008

[16] Gupta, Shashank, and B. B. Gupta. "JS-SAN: defense mechanism for HTML5-based web applications against javascript code injection vulnerabilities." Security and Communication Networks (2016).

[17] Tiwari, Ashish, Manoj Kumar Sah, and Shashank Gupta. "Efficient Service Utilization in Cloud Computing Exploitation Victimization as Revised Rough Set Optimization Service Parameters." Procedia Computer Science 70 (2015): 610-617.