Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 125 (2018) 669–675

Procedia Computer Science

www.elsevier.com/locate/procedia

6th International Conference on Smart Computing and Communications, ICSCC 2017, 7-8 December 2017, Kurukshetra, India

Efficient yet Robust Elimination of XSS Attack Vectors from HTML5 Web Applications Hosted on OSN-Based Cloud Platforms

[1]Gurpreet Kaur, [2]Bhavika Pande, [3]Aayushi Bhardwaj, [4]Gargi Bhagat, [5]*Shashank Gupta

[1,2,3,4]*Department of Computer Science and Engineering, Jaypee Institute of Information Technology Noida, UP, 201304, India*
5 Department of Computer Science and Information Systems, BITS, Pilani, Rajasthan-333031*

Abstract

The authors suggested a cloud-hosted XSS defensive model that defends the cloud-hosted web applications against the injection of HTML5 attack vectors. The model is categorized into 2 phrases: namely HTML5 Feature Injection and HTML5 Feature Comparison. The earlier one basically involves calculating and storing the features of JavaScript code in the feature repository. The other one compares the features extracted in the offline and online mode. Any oddity results in consequent sanitization of the HTMl5 script code. We have developed our prototype on the environmental set-up of ICAN Cloud simulator and its settings were integrated by creating the infrastructure set-up of various virtual machines on this simulator. The HTML5 XSS attack vector detection proficiency of our scheme was tested on numerous cloud-hosted web applications installed on different virtual machines facilitating with the malicious intention of injection of attack vectors on regular intervals. Experimental results disclosed the facts that our proposed scheme is proficient enough to detect and eliminate the HTML5 attack vectors from the tested web applications with tolerable rate of False Negatives (FNs), False Positives (FPs) and lesser overhead.

Peer-review under responsibility of the scientific committee of the 6th International Conference on Smart Computing and Communications.

Keywords: XSS Attack, Cloud Computing, Online Social Network Security, HTML5 Web Applications, Context-Aware Sanitization

1. Introduction

Online Social Networks (OSN) are virtual places that facilitate communication. Today, many OSNs have tens of millions of active users. Most accepted and biggest OSN is Facebook with greater than 1 billion vigorous users. Social networking is not limited to informal use but is also used for formal purposes. The pool of information (personal/professional) is stored by user on these networks, so hackers are paying more attention toward these sites.

*Corresponding author Tel ; +91-1204195942
E-mail address: shashank.csit@gmail.com

1877-0509
Peer-review under responsibility of the scientific committee of the 6th International Conference on Smart Computing and Communications
10.1016/j.procs.2017.12.086

To offer users with enhanced services, the OSNs take help of the JavaScript or contemporary HTML5 programming language. The support for such language platforms provides fertile platform for XSS attacks. According to the recent statistics, XSS attack vector vulnerabilities are highly seen vulnerabilities in Web Applications. XSS vulnerabilities are present because of the incorrectly validated user input. Extenuating all possible XSS attacks is infeasible due to the size and intricacy of modern web application and the various ways that browsers call upon their JavaScript/HTML5 engines. Initially when XSS was discovered it was classified in 2 categories: Stored XSS and Non-Persistent XSS.A third category of XSS, i.e. DOM based XSS later was defined in 2005 [11-21].

XSS is often believed to be one of the most frequent client-side hacking skills.XSS can be described as one of the most alarming and recurrently found vulnerability affiliated to web applications. It is the most generally exploited attack to delineate the malicious code to the victim's account. However, the effects of XSS attacks have been witnessed collectively by the Worldwide Web. Security analyst have found such vulnerabilities in most widely used websites such as Google, Facebook, E-Bay etc. which arises an urgent need to take preventive measures against the XSS vulnerability. Cao et al. [1] presents PathCutter, a design to thwart the self-replicating attack vectors by hampering the DOM entrance to various views at the client side. Sharma et al. [2] presents an integrated approach for the prevention for XSS attacks in addition to SQL Injection attacks. The model works in two routines, namely safe mode and production mode environment. the implementation of the designed approach is done in PHP and gives more effective results in production web environment. Zhang et al. [3] presents implementation-flow understanding for JavaScript loaded in the web browser by developing a finite state automata (FSA) to model customer side behaviour of applications and deploying the system in proxy model. Most of the existing work was unable to integrate their infrastructural framework on the implicit desktop systems of cloud-hosted OSN [4-9]. Web developers are continuously finding very hard to integrate the settings of such XSS defensive frameworks implicit desktop systems of cloud computing. Keeping this thing in mind, in this article, the authors proposed a cloud-based HTML5 XSS attack vector defensive model that alleviates such attack vectors from the OSN hosted on implicit machines of cloud infrastructure settings. The authors utilized the capabilities of ICAN Cloud simulator for developing their prototype and embody its infrastructural settings on its virtual machines. The infrastructure settings of open source HTML5-based web applications were hosted on such virtual machines and the evaluation results disclosed that our cloud-based framework was able to neutralize the effect of such attack vectors from such applications. The next section illustrates the details regarding the proposed cloud-based XSS defensive framework.

2. Proposed Framework

The authors presented a novel HTML5 XSS attack vectors defensive framework that scans for such malicious attack vectors on HTML5 web applications hosted on implicit machines of cloud infrastructure settings. The proposed framework works in two routines i.e. HTML5 Feature Injection and HTML5 Feature Comparison. Fig. 1 exemplifies the detailed functioning of our design. Commencing with the feature injection phase, it includes parsing of the web application module by the web parser to which a corresponding DOM tree is generated by the DOM Generator. DOM Generator receives URL links as its input from the Parser. The features of the JavaScript code extracted from its corresponding DOM tree are estimated and injected in the original source code. Feature repository reserves the injected features along with the source code. Further, working with the feature comparison routine, HTTP request is forwarded to the web server to which a corresponding HTTP Response is received by the browser. The hidden injection points are rummaged by the HTML Parser and are provided to the JavaScript extractor to excerpt the required JavaScript code. This code is served as an input to 2 components: JavaScript Decoder and Feature computation component. Feature comparator receives the features of the JavaScript code computed by the Feature Computation component. Henceforth, these features are compared with those stored in the Feature Repository. If any malicious code is detected by the Similarity indicator or feature comparator, it is passed to the sanitizer. As a result, web browser receives a sanitized document free from XSS vulnerabilities or worms. The main modules of the designed framework include: DOM Generator, Feature Estimation and Injection,Context Locator, Feature Estimator, Similarity Indicator and the Sanitizer.

DOM Generator: The Document Object Model (DOM) generates a hierarchy arrangement having every node as an entity that represents a component of file. This module will parse all the web pages and extract all the nodes. The working of the DOM generator is explained in algorithm in Fig. 2.

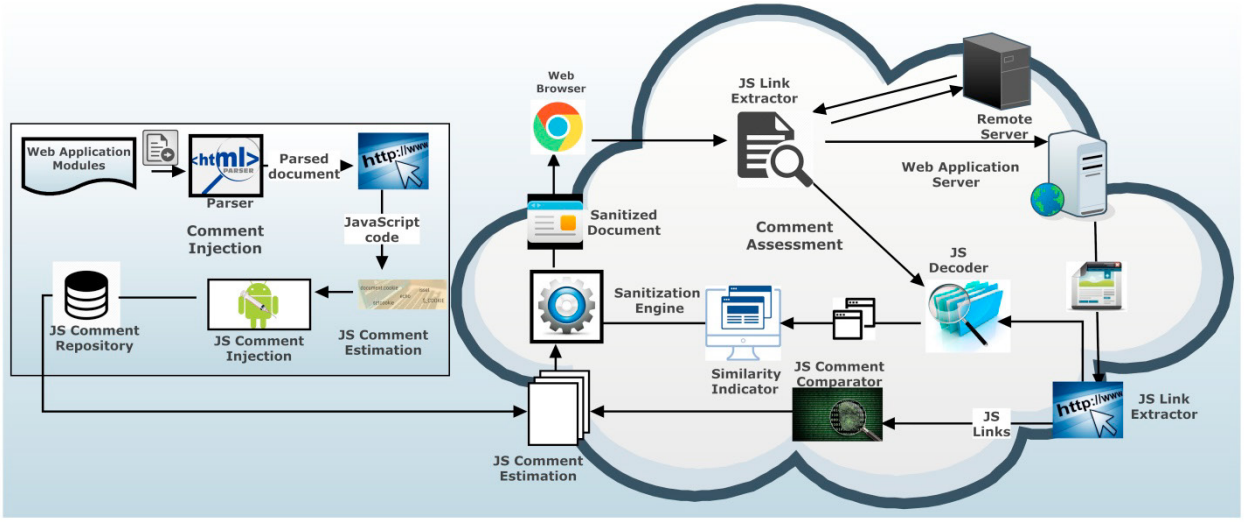

Fig.1. Detailed design of the proposed model

Input: Parsed document

Output: Set of DOM Trees

Start

Stack st $\Leftarrow \phi$

Tree tr $\Leftarrow \phi$

Tag_List $\Leftarrow \phi$

for each $web_p \in WP$

$Tag_{new} \Leftarrow Tag_{ext}(web_p)$; //Extracting tags from the web page

Tag_List $\Leftarrow Tag_{new} \bigcup$ Tag_List;

st.push(Tag_{new}); //Insert node to stack

t.insert_node(Tag_{new}, null); //Insert first node to tree

while stack is not empty **do**

data $\Leftarrow Tag_{ext}(web_p)$

If opening_Tag(data) **do**

len $\Leftarrow$ st.size();

Tag_List $\Leftarrow$ data $\bigcup$ Tag_List; //Store all opening tags

ptr $\Leftarrow$ crawl_DOM_tree(tr, len); //Crawl tree to get pointer for new node

tr.insert_node(data, ptr);

st.push(data);

End

else closing_Tag(data) **do**

st.pop();

End

End

End

Returntr;

Fig. 2. Algorithm for the generation of DOM Tree

Feature Estimator: This module accepts the HTTP Response in the H_RES_List parameter. It examines all the nodes of the graph G[N][T] and extracts the embedded JavaScript code. Feature estimation involves estimating feature of object under scanning and then injecting them as comments for further use in prevention of XSS attacks.

Context Locator: Here, the authors extort the user key context by proposed context-locator algorithm in Fig. 3. During the feature extraction process, pure HTML code statements are extracted.

URI Link Extractor: The algorithm will inspect for the existence of URI links in the HTTP Request. For each extracted parameter Pj and stores the URI links in Uri.

Feature Comparator: Feature Computation component computes the features of the JavaScript code and provides it to the feature comparator component. It compares the features provided by feature computation component and features stored in the feature repository. If similarity indicator or feature comparator detect any malicious code, it is passed to the sanitizer else the HTTP Response is forwarded to be displayed by the browser.

Sanitizer: It parses the response and extracts all the untrusted variables UDA in a variable TOK. For each UDA it will extract the context of the untrusted variables and pass it to SAN_APP that includes all the sanitization routines to be used in the sanitization. The sanitized output is stored in S_List which is now free from the XSS worms. Algorithm in Fig. 4 depicts the functioning of the sanitizer.

```
Context-Locator: Locating the context of the input
Input: A String STR containing the HTML code and segment context Type CXTS.
Output: Context of the input by the user.
TypeCXT=A variable holding the input of the user.
If(STR consists of a complete HTML tag)  then
    Return  CXTS;
else if( String STR begin with <&& end with= | = | =) then

if(STR begins with any kind of special tag)
Then
         if (STR consists of event handlers) then
TypeCXT = TypeCXTS+Event_Attr_Value+[DQ|SQ|NQ];
            Return TypeCXT;
            ELSE
TypeCXT = TypCXTS+STag_Attr_Value+(DQ|SQ|NQ);
             return TypeCXT;
end if
           ELSE
              // STR does not begin with a special tag
if (STR consists of event handlers) then
TypeCXT = TypeCXTB + Tag_CS_Attr + (DQ|SQ|NQ);
                 return TypeCxt;
else if (some pattern in string STR) in that case
TypeCXT = TypeCXTS + Tag_CS_Attr + (DQ|SQ|NQ);
return TypeCxt;
end if
end if
else if (IsStringSTR == <Non_sp_tag ) in that case
        .
TypeCXT = TypeCXTB + Tag_CSS_Attr_V al
                    return TypeCXT;
                    Else
                    return TypeCXTB;
end if
```

Fig. 3. Algorithm for locating the context of the Input

3. Prototype Implementation and Experimental Evaluation

The authors utilized the capabilities of ICAN Cloud simulator for the designing of their prototype model and integrated their infrastructural settings by creating different virtual machines of this simulator. Different tested freeware platforms of HTML5 web applications were also hosted on such virtual machines of ICAN Cloud simulator. An impenetrable perspective was taken to assess our approach towards shielding from cross site scripting attacks. Efficient and stable after-effects were detected as a consequence of exhaustive testing of the system. The liable JavaScript detection as well as prevention methodology of our framework is accessed on five virtual platforms of HTML5 web applications namely Humhub, Wordpress, Joomla, Drupal and Elgg. Installation of all these platforms was done using localhost server- XAMPP. After the installation, the next step involved searching the vulnerabilities in our websites with the help of XSS attack vector repositories. These repositories comprise of a number of script or payloads accountable for XSS attacks. Initially, the tester detects the input vectors by determining all the input fields. It includes remote or isolated inputs such as POST data, HTTP parameters/specifications, concealed values of form field etc. Secondly, it analyzes each input vector so as to detect probable vulnerabilities. Specially crafted input data is be used by the tester which triggers responses from the web browser that manifest the vulnerability.

Input: HTTP Response(H_RESI)
Output : Sanitized document

Start:
UDA $\Leftarrow \phi$
S $\Leftarrow \phi$
SAN_APP: Sanitized Application
UDA $\Leftarrow$ Extract(H_RESI)
While(TOK!=NULL)
If(TOK==UDA) then
Move(TOK to UDA_List)
for each(UDA $\in$ UDA_Lst)
CXT $\Leftarrow$ Extract(UDA)
S $\Leftarrow$ SAN_APP(UDA ,CXT)
S_List $\Leftarrow$ S $\cup$ S_List
End for each
End if
End while
End

Fig. 4. Context aware sanitization Algorithm

Finally, for each test input attempted, the tester analyses the results and determines if it represents a vulnerability that has a realistic impact on the web application's security. Event Handlers which determine the flow of execution of a program capitalize most of vulnerabilities of the Websites. Table 1 shows the overall results of our cloud-hosted framework during the detection of HTML5 XSS attack vectors in online mode. The cloud-hosted framework was developed on the ICAN Cloud simulatorby creating different virtual machines and hosted different HTML5 web applications on such machines. The framework defends against XSS worms by eliminating the constraint values that appear to be cautious. The XSS attack vectors have been taken from certain repositories found in the internet and injected on several injection points of the downloaded web applications. We have injected the five different categories of XSS attack vectors [10] (i.e. event handlers, character encoding tags, URL obfuscation, HTML entity elements and embedded character tags). Fig. 5 and Fig. 6 highlight the information of experiential outcomes on every HTML5 platforms installed on our cloud platform. A minimal rate of FPs as well as FNs emulated with the help of statistics proclaimed the efficient detection and mitigation of XSS vulnerabilities. The evaluation results after distribution of VCS on the platform of cloud reveals that our model precisely detects the propagation of reflected category of XSS worms by successfully comparing the two extracted script sets.

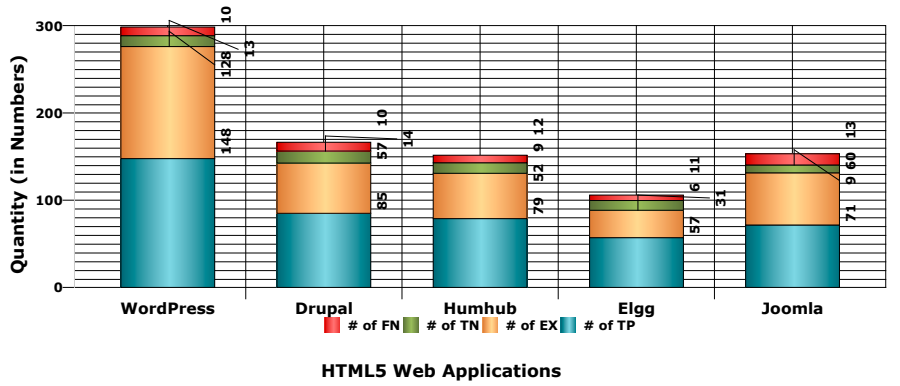

Fig. 5. Exploitation of HTML5 Attack Vectors on Web Applications

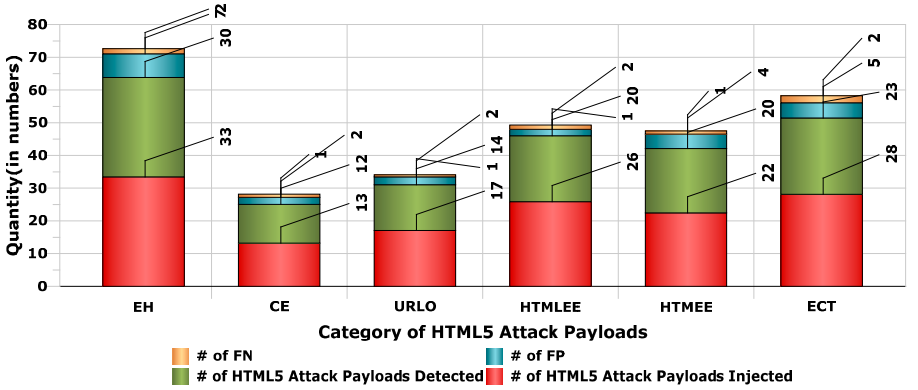

Fig. 6. Evaluation Results of our Cloud-Hosted Framework

Table 1. Statistics of Exploited Attack Vectors on Different HTML5 Web Applications

HTML5 Web Applications Quantity	Wordpress	Drupal	Humhub	Elgg	Joomla
Exploited XSS	147	84	78	57	71
# of TP	128	57	52	31	59
# of TN	12	13	12	11	9
# of FN	10	10	8	6	12

4. CONCLUSION AND FUTURE WORK

In this article, the authors presented a cloud-hosted framework that senses the HTML5 XSS attack vector injection vulnerabilities on the core web servers of web applications hosted on the implicit host systems of cloud-hosted infrastructure settings. Initially, the framework generates the sanitized HTML5 attack vector templates and computes the features of such script code as a part of static pre-processing on thevirtual machines of Cloud data centres. Such machines re-execute the feature computation process on the HTML5 script code present in the HTTP response. Any abnormality observed between these features and those computed statically will indicate the injection

of malicious HTML5 attack vectors on the web servers hosted on the implicit host systems of cloud-hosted infrastructure settings. Very low and acceptable rate of FPs and FNs were found during the HTML5 detection phase of the attack vectors on the web servers incorporated on the implicit host systems of ICAN Cloud simulator. For our future work, we would prefer to analyse the response time of our framework by completely integrating its settings on the online edge servers of Fog computing infrastructure without the intervention of Cloud data centres.

References

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

[2] Sharma, Pankaj, Rahul Johari, and S. S. Sarma. "Integrated approach to prevent SQL injection attack and reflected cross site scripting attack." International Journal of System Assurance Engineering and Management 3, no. 4 (2012): 343-351.

[3] Zhang, Qianjie, Hao Chen, and Jianhua Sun. "An execution-flow based method for detecting cross-site scripting attacks." In Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on, pp. 160-165. IEEE, 2010.

[4] Gupta BB, Gupta S, Chaudhary P. Enhancing the Browser-Side Context-Aware Sanitization of Suspicious HTML5 Code for Halting the DOM-Based XSS Vulnerabilities in Cloud. International Journal of Cloud Applications and Computing (IJCAC). 2017 Jan 1;7(1):1-31.

[5] Agten, Pieter, Steven Van Acker, YoranBrondsema, Phu H. Phung, LievenDesmet, and Frank Piessens. "JSand: complete client-side sandboxing of third-party JavaScript without browser modifications." In Proceedings of the 28th Annual Computer Security Applications Conference, pp. 1-10. ACM, 2012.

[6] Gupta S, Gupta BB. An Infrastructure-Based Framework for the Alleviation of JavaScript Worms from OSN in Mobile Cloud Platforms. InInternational Conference on Network and System Security 2016 Sep 28 (pp. 98-109). Springer International Publishing.

[7] Gupta S, Gupta BB. XSS-immune: a Google chrome extension-based XSS defensive framework for contemporary platforms of web applications. Security and Communication Networks. 2016 Nov 25;9(17):3966-86.

[8] Wang, Rui, XiaoqiJia, Qinlei Li, and Daojuan Zhang. "Improved N-gram approach for cross-site scripting detection in Online Social Network." In Science and Information Conference (SAI), 2015, pp. 1206-1212. IEEE, 2015.

[9] Gupta S, Gupta BB. XSS-secure as a service for the platforms of online social network-based multimedia web applications in cloud. Multimedia Tools and Applications. 2016:1-33.

[10] OWASP, XSS. "prevention Cheat-sheet."Available at: https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet.

[11] Gupta, Shashank, and B. B. Gupta. "Alleviating the proliferation of JavaScript worms from online social network in cloud platforms." In 2016 7th International Conference on Information and Communication Systems (ICICS), pp. 246-251. IEEE, 2016.

[12] Gupta S, Gupta BB. Smart XSS Attack Surveillance System for OSN in Virtualized Intelligence Network of Nodes of Fog Computing. International Journal of Web Services Research (IJWSR). 2017 Oct 1;14(4):1-32.

[13] Gupta S, Gupta BB. XSS-SAFE: a server-side approach to detect and mitigate cross-site scripting (XSS) attacks in JavaScript code. Arabian Journal for Science and Engineering. 2016 Mar 1;41(3):897-920.

[14] Gupta S, Gupta BB, Chaudhary P. Hunting for DOM-Based XSS vulnerabilities in mobile cloud-based online social network. Future Generation Computer Systems. 2017 Jun 12.

[15] Chaudhary P, Gupta S, Gupta BB, Chandra VS, Selvakumar S, Fire M, Goldschmidt R, Elovici Y, Gupta BB, Gupta S, Gangwar S. Auditing defense against XSS worms in online social network-based web applications. Handbook of Research on Modern Cryptographic Solutions for Computer and Cyber Security. 2016 May 16;36(5):216-45.

[16] Gupta S, Gupta BB. Detection, Avoidance, and Attack Pattern Mechanisms in Modern Web Application Vulnerabilities: Present and Future Challenges. International Journal of Cloud Applications and Computing (IJCAC). 2017 Jul 1;7(3):1-43.

[17] Chaudhary P, Gupta BB, Gupta S. Defending the OSN-Based Web Applications from XSS Attacks Using Dynamic JavaScript Code and Content Isolation. InQuality, IT and Business Operations 2018 (pp. 107-119). Springer, Singapore.

[18] Gupta S, Gupta BB. Automated discovery of javascript code injection attacks in PHP web applications. Procedia Computer Science. 2016 Jan 1;78:82-7.

[19] Gupta BB, Gupta S, Gangwar S, Kumar M, Meena PK. Cross-site scripting (XSS) abuse and defense: exploitation on several testing bed environments and its defense. Journal of Information Privacy and Security. 2015 Apr 3;11(2):118-36.

[20] Gupta S, Gupta BB. 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. 2014 Oct 31:174-91.

[21] Gupta S, Gupta BB. CSSXC: Context-sensitive Sanitization Framework for Web Applications against XSS Vulnerabilities in Cloud Environments. Procedia Computer Science. 2016 Dec 31;85:198-205.