Journal of Engineering and Applied Sciences 13 (16): 6616-6621, 2018
ISSN: 1816-949X
© Medwell Journals, 2018

Towards Removing Cross-Site Scripting Vulnerabilities from Mobile Web Applications

Isatou Hydara, Abu Bakar Md Sultan, Hazura Zulzalil and Novia Admodisastro
Department of Software Engineering and Information System,
Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM),
43400 Serdang, Selangor, Malaysia

Abstract: Cross-site scripting vulnerabilities are among the most common of security vulnerabilities found in web applications and more recently in mobile versions of web applications. They have caused many successful attacks on web applications on a daily basis including loss of financial and health information, exposure to malware and viruses and denial of service attacks. Cross-site scripting vulnerabilities are easy to exploit but difficult to mitigate. Most of the existing solutions to cross-site scripting vulnerabilities focus only on the desktop version of web application and there is hardly any focus on the mobile versions. Also, most solutions provided only focus on preventing attacks or detecting the vulnerabilities. Very few research works have addressed eliminating these vulnerabilities from the web applications source codes. In this study, we present our research in progress on the removal of detected cross-site scripting vulnerabilities in mobile versions of web applications. We have proposed an approach in a previous research to detect and remove cross-site scripting vulnerabilities in desktop web applications. We have enhanced that approach and are currently testing it for the removal of cross-site scripting vulnerabilities in mobile versions of web applications. Initial evaluations have indicated promising results. We believe this approach can help web application developers to eliminate cross-site scripting vulnerabilities in not only their desktop web applications but also in the mobile version ones.

Key words: Cross-site scripting, cross-site scripting vulnerability, software security, vulnerability removal, web application, application developers

INTRODUCTION

Recently, technology has become an important part of our lives and as it grows, we rely on it more and more to accomplish our daily transactions be it business, personal or otherwise (Hydara *et al*., 2015a). Businesses and organizations also, use web applications to provide many of their services, not only on desktop versions but also on mobile versions as more people use their mobile phones to access some of those services. However, as web applications become very important to the success of businesses and organizations their securities have increasingly become more complex (Fogie *et al*., 2007). Hence, more security issues have emerged due to the increasing number of security threats affecting web applications (Fogie *et al*., 2007).

Testing web applications for security has therefore, become a crucial issue to the software security industry as well as governments, businesses and organizations. Study of the major security threats in web applications has shown that XSS vulnerabilities are among the top ten vulnerabilities, as reported by the Open Web Application Security Project (OWASP) (Anonymous, 2016a).

Cross-Site Scripting (XSS) vulnerabilities (Anonymous, 2017, 2016a) are found in web applications source code and they can be exploited through XSS attacks when such vulnerable applications are deployed and running online. Injecting malicious scripts where these applications accept user inputs can result to serious security breaches such as cookie theft, account hijacking, manipulation of web content and theft of private information (Hydara *et al*., 2014).

XSS vulnerabilities were discovered in the 1990s in the early days of the world wide web. They are a type of injection problems that enable malicious code to be injected into trusted web applications (Anonymous, 2016a). This is a result of a failure during software development to validate inputs from the web application users. This lack of proper verification of the user inputs enables hackers to attack an application.

Corresponding Author: Isatou Hydara, Department of Software Engineering and Information System, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM), 43400 Serdang, Selangor, Malaysia

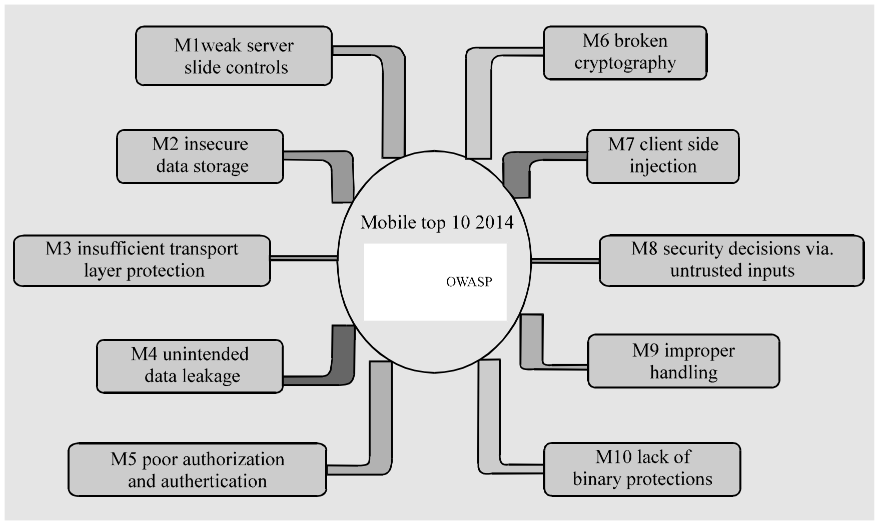
M1weak server slide controls
M2 insecure data storage
M3 insufficient transport layer protection
M4 unintended data leakage
M5 poor authorization and authertication
Mobile top 10 2014
OWASP
M6 broken cryptography
M7 client side injection
M8 security decisions via. untrusted inputs
M9 improper handling
M10 lack of binary protections

Fig. 1: Mobile top 10, adopted from (Anonymous, 2014)

Many research activities have been conducted to address problems related to XSS vulnerabilities since, their discovery. According to this systematic review (Hydara *et al*., 2015a), most of the existing approaches focused on preventing XSS attacks (Sharma *et al*., 2012; Sun and He, 2012; Gundy and Chen, 2012; Scholte *et al*., 2012) and detecting XSS vulnerabilities (Agosta *et al*., 2012; Huyam and El-Qawasmeh, 2012; Acker *et al*., 2012; Duchene *et al*., 2012) in web applications. Few research activities have addressed the removal of XSS vulnerabilities (Bathia *et al*., 2011; Shar and Tan, 2012) (Fig. 1).

There is lack of security in most web applications that are in used today including mobile web applications. According to this study (Javed and Schwenk, 2014), 81% of all the mobile applications they surveyed were vulnerable to XSS attacks. HTML5 is used by many mobile application developers and it is reported to contain XSS vulnerabilities (Chen *et al*., 2015). Moreover, XSS has been added to the Top 10 List of vulnerabilities in mobile applications (Anonymous, 2014) as shown in Fig. 1 as well as in HTML5 (Shah, 2012).

Figure 2 gives a high level view of XSS attack. Successful XSS attacks can result in serious security violations for both the web application and the user. An attacker can inject malicious codes into a web application's user input and if the input is not validated, the code can steal cookies, transfer private information, hijack a user's account, manipulate the web content, cause denial of service and many other malicious activities.

XSS attacks are of three types namely reflected, stored and DOM (Document Object Model) based

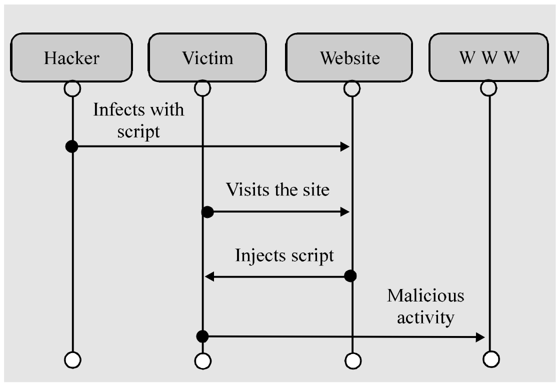

Fig. 2: High Level View of XSS, adopted from (Vonnegut, 2015)

(Anonymous, 2017, 2016a; Vonnegut, 2015). Reflected XSS is executed by the victim's browser and occurs when the victim provides input to the web site such as username and password. Stored XSS attacks store malicious scripts in databases, message forums, comments fields, etc. of the attacked server. The malicious script is executed by visiting users thereby passing their privileges to the attacker. Both reflected and stored XSS vulnerabilities can be found on either client side or server side codes. On the other hand, DOM-based XSS vulnerabilities are found on the client side. Attackers are able to collect sensitive or important information from the user's computer.

In this study, we propose an approach for the removal of XSS vulnerabilities in mobile web applications. This reseach is in progress and is built on a previous approach (Shar and Tan, 2012) and extends it by including all the three types of XSS as well as following all the 7 rules stated in the OWASP's XSS prevention rules (Anonymous, 2017, 2016c).

Literature review: Research works related to XSS in mobile versions of web application are not many as the focus to it has only started recently. A recent study (Javed and Schwenk 2014) has found that XSS vulnerabilities are present in 81% of the mobile versions of web applications they surveyed. This is alarming and shows the need for more research on XSS in mobile versions of web applications. They have also, developed an XSS filter which proved successful against well-known XSS vectors and has been added as support in WordPress and Drupal platforms. Their XSS detection rules have been added to the OWASP ModSecurity Core rule set (Anonymous, 2016c). Their approach, however, only focused on Reflected XSS.

HTML5 is used in the development of many mobile applications and has been found to contain XSS vulnerabilities. Hence, Chen *et al*. (2015) developed a tool called DroidCIA for the detection of injection attacks, including XSS in HTML based mobile applications. The tool registered a 99.21% accuracy rate with some false positives when evaluated with their dataset. Some limitations in their research included problems related to input string and dead code in the dataset and the research did not include removing XSS vulnerabilities.

Another research work on HTML5 was conducted by (Dong *et al*., 2014). They constructed a new vector repository for XSS by collecting and adding a set of XSS attack vectors related to HTML5 to the previously existing XSS vectors. They also, developed a dynamic tool for XSS vulnerability detection. Their proposed solution was tested on webmail systems and found to be effective in detecting XSS vulnerabilities related to HTML5. It was only focused on webmail services. A study by Mutchler *et al*. (2015) was also, conducted to detect vulnerabilities in Android-based mobile web applications. XSS vulnerabilities were among those discovered in the tested applications. Again, vulnerability removal was not part of these works.

Shar and Tan (2012) have proposed an approach for removing XSS vulnerabilities from web applications. Their approach is in two phases. The first uses static analysis to identify potential XSS vulnerabilities in the source code of tested applications. The second phase uses pattern matching techniques to provide an escaping mechanism to prevent any input values causing script execution, thereby eliminating XSS vulnerabilities. This approach, however, only focused on reflected and stored XSS and was only evaluated on desktop applications.

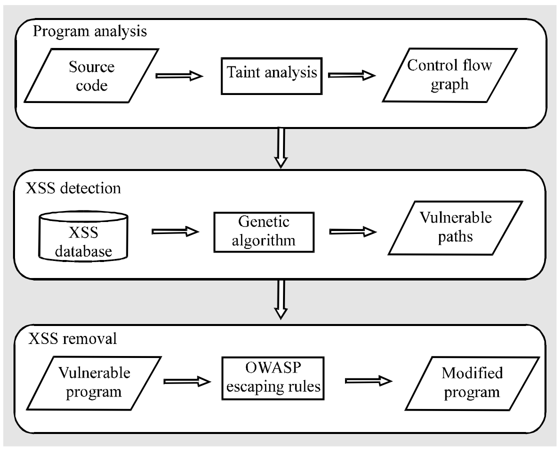

Fig. 3: Our XSS detection and removal approach

MATERIALS AND METHODS

Proposed approach: The solution being implemented in this research uses a Genetic Algorithm (GA) based approach in the detection and removal of XSS vulnerabilities in mobile versions of web applications. The approach, as shown in Fig. 3 is in three components. The first component involves converting the source codes of the applications to be tested to Control Flow Graphs (CFGs) using static analysis techniques where each node will represent a statement and each edge will represent the flow of data from node to node.

The second component focuses on detecting the vulnerabilities in the source codes using GA approach whiles the third component concentrates on their removal using the OWASP ESAPI security guidelines. In this research in progress, we only focus on the third component of the approach which is to remove detected vulnerabilities resulting from the application of the first two components. The details of the approach can be found in our previous research (Hydara *et al*., 2014).

The third component of our approach has been successfully, implemented on desktop web applications (Hydara *et al*., 2015b). It can be used to secure web applications containing XSS vulnerabilities. After XSS vulnerabilities are detected in the source code of applications, the removal process can be carried out. The OWASP's Enterprise Security API (ESAPI) security mechanisms (Anonymous, 2016b) are followed as a guideline to remove the detected XSS vulnerabilities. The lines of code where the XSS vulnerabilities are located are identified. Then, we determine which of the ESAPI escaping rules can be applied to replace those lines of code without compromising their functionality. Finally, we generate the secure codes of the escaping statements and put them in place of the vulnerable statements, using the OWASP XSS prevention rules discussed in the next subsection.

RESULTS AND DISCUSSION

The OWASP's XSS prevention rules: XSS attacks occur when special characters such as '•', '••', '¡', '<' are injected into user inputs. Typically when a web program references user inputs in its HTML outputs it expects that client-side browsers treat those inputs as only data. However, the malicious injected characters cause these browsers to interpret them as code. Therefore, an XSS exploit occurs by illegally changing the data context to code context. Anonymous (2016d, 2017)has established XSS prevention rules (Table 1) to follow to ensure that any user input referenced in an HTML output is only treated as data.

The rules ensure that the appropriate escaping mechanisms are used on data inputs based on the HTML context the vulnerable code is referenced. They help to ensure injected especial characters are not executed thereby disabling XSS vulnerabilities. In addition, using these rules poses no negative effects on the referenced data even if the HTML output is not vulnerable to XSS. The XSS prevention rules for reflected and stored XSS are summarized in Table 1 while those for DOM-Based XSS are summarized in Table 2. For more details on these rules please refer to Anonymous (2016d, 217).

There is a fundamental difference between reflected and stored XSS when compared to DOM based XSS. reflected and stored XSS are server side execution issues while DOM based XSS is a client (browser) side execution issue. However, all of this code originates from the server, therefore, it is the application developer or owner's responsibility to make it secure from XSS, regardless of the type of XSS flaw it is Anonymous (2017). There are 5 sub rules under this rule.

The ESAPI API: OWASP has released the Enterprise Security API (ESAPI) (Anonymous, 2016b) to the security community which is a tool that can be used to enforce the XSS prevention rules discussed in the previous section in vulnerable web applications. ESAPI helps to enforce security in both developed and developing web applications. It is available in different programming languages such as Java, NET and PHP. In our approach, we make use of the ESAPI implemented for Java, since, our test subjects were developed in Java. ESAPI provides many security mechanisms including authentication, validation, encoding, encryption, security wrappers, filters and access control to mitigate various

Table 1: Prevention rules for reflected and stored XSS

Prevention rules	Descriptions
Rule#0	Do not reference user inputs in any other cases except the ones defined in Rule#1-Rule#7
Rule#1	Use HTML entity escaping for the untrusted data referenced in an HTML element
Rule#2	Use HTML attribute escaping for the untrusted data referenced as a value of a typical HTML attribute such as name and value
Rule#3	Use java script escaping for the untrusted data referenced as a quoted data value in a Java script block or an event handler
Rule#4	Use CSS escaping for the untrusted data referenced as a value of a property in a CSS style
Rule#5	Use URL escaping for the untrusted data referenced as a HTTP GET parameter value in a URL
Rule#6	Use a specific sanitizer for tainted data that contains HTML encoding such data is difficult since all the tags in the input can be broken. Therefore a library that can parse and clean HTML formatted text is needed. An example is the OWASP Java HTML Sanitizer (Anonymous, 2015)
Rule#7	Prevent DOM-Based XSS. The only rule focusing mainly on DOM based XSS. It has five sub-rules under it

Table 2: Prevention rules for DOM based XSS

Prevention rules	Descriptions
Rule#7-1	HTML escape then JavaScript escape before inserting untrusted data into HTML subcontext within the execution context
Rule#7-2	JavaScript escape before inserting untrusted data into HTML attribute subcontext within the execution context
Rule#7-3	Careful when inserting untrusted data into the event handler and JavaScript code subcontexts within an execution context
Rule#7-4	JavaScript escape before inserting untrusted data into the CSS attribute subcontext within the execution context
Rule#7-5	URL escape then JavaScript escape before inserting untrusted data into URL attribute subcontext within the execution context

web security issues. Hence, it is easier for developers to implement any of these mechanisms in their applications with the same API.

ESAPI needs to be installed into an application for its security controls to be used. Details on its installation and configuration procedures can be found in the documentation downloadable from: code.google.com/p/owasp-esapi-java/. The ESAPI Java documentation for the escaping or encoding APIs can be downloaded from: owasp-esapijava.googlecode. com/svn/ trunk_ doc/latest/org/owasp/esapi/Encoder.html. After it is properly installed, ESAPI could be used to secure data that was not validated in an application. It helps to encode the data before it is referenced in an HTML output.

The XSS prevention rules to enforce and the corresponding HTML context determine which appropriate encoding API is to be used. For example, Rule #1 applies when the context is HTML element. Therefore, the following escaping API is used: <div>ESAPI. encoder().encodeForHTML(untrusted-data)</div>(Shar and Tan 2012).

Removing XSS vulnerabilities: This approach has two steps. The first step is to locate the lines of code vulnerable to XSS attacks and identify the HTML context in which the vulnerable code is referenced. This step also, identifies which escaping mechanism to use according to the HTML context and the appropriate XSS prevention rule. The second step is the source code replacement for the vulnerable codes. Then, the ESAPI escaping rules are applied to replace those vulnerable lines of code without compromising their functionality. The details of the algorithms used in these steps are provided by Shar and Tan (2012). The removal of all XSS vulnerabilities detected in the tested programs can be done using these algorithms. Since, only the vulnerable codes are escaped there is not much modification done in the programs that are tested.

Evaluation: The above approach has been implemented in a prototype and we are currently evaluating it for its effectiveness in removing XSS vulnerabilities in mobile versions of web applications. The development of the tool was implemented with the Eclipse IDE using the Java programming language. The OWASP's XSS prevention rules and the ESAPI API are integrated into the tool.

Preliminary evaluation results on some mobile applications are promising and show the feasibility of our approach in removing XSS vulnerabilities in mobile versions of web applications. We expect our tool to be able to remove all XSS vulnerability types in mobile applications as we did previously in desktop web applications. We will continue testing the approach with similar dataset as Javed and Schwenk (2013) and Dong *et al.* (2014) and compare our results with theirs. We expect our results to be an improvement to their solutions. We will also, look for more dataset to increase the overage of our evaluation

CONCLUSION

This study presented a research in progress approach for XSS removal using the OWASP XSS prevention rules as well as the ESAPI security API. The proposed approach is an enhancement based on our previously proposed approach for desktop web applications. The approach can remove detected XSS vulnerabilities in web applications. Cross-site scripting is a major security problem for web applications and it is now affecting mobile web applications as well. It can lead to account or web site hijacking, loss of private information and denial of service, all of which victimize web application users.

Preliminary evaluation results have shown promising results. Next on this progressive work, we will fully evaluate and validate the proposed approach. A prototype tool has been developed to automate this process. Preliminary evaluation indicated promising results. We will continue to test the approach on real world mobile web applications. We expect our approach to help mobile application developers to be able to detect XSS vulnerabilities in their web applications. This will benefit the applications users by protecting them from XSS attacks.

ACKNOWLEDGEMENT

We acknowledge that this research received support from the Fundamental Research Grant Scheme FRGS/1/2015/ICT01/UPM/02/12 awarded by Malaysian Ministry of Higher Education to the Faculty of Computer Science and Information Technology at Universiti Putra Malaysia.

REFERENCES

Acker, S.V., N. Nikiforakis, L. Desmet, W. Joosen and F. Piessens, 2012. FlashOver: Automated discovery of cross-site scripting vulnerabilities in rich internet applications. Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, May 2-4, 2012, ACM, Seoul, Korea, ISBN:978-1-4503-1648-4, pp: 12-13.

Agosta, G., A. Barenghi, A. Parata and G. Pelosi, 2012. Automated security analysis of dynamic web applications through symbolic code execution. Proceedings of the 9th International Conference on Information Technology New Generations (ITNG'12), April 16-18, 2012, IEEE, Las Vegas, Nevada, ISBN:978-1-4673-0798-7, pp: 189-194.

Anonymous, 2014. Mobile top 10 2014-m7. OWASP, Maryland, USA. https://www.owasp.org/index.php/Mobile_Top_10_2014-M7

Anonymous, 2015. OWASP Java HTML sanitizer project. OWASP, Maryland, USA. https://www.owasp.org/index.php/OWASP_Java_HTML_Sanitizer_Project

Anonymous, 2016a. Category: OWASP enterprise security API. OWASP, Maryland, USA. https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API

Anonymous, 2016b. Cross-Site Scripting (XSS). OWASP, Maryland, USA. https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

Anonymous, 2016c. OWASP modsecurity Core Rule Set (CRS). OWASP, Maryland, USA. https://modsecurity.org/crs/

Anonymous, 2017. CWE-79: Improper neutralization of input during web page generation (Cross-site Scripting). Continental Wrestling Entertainment, Jalandhar district, India. http://cwe.mitre.org/data/definitions/79.html

Anonymous, 2017. DOM based XSS prevention cheat sheet. OWASP, Maryland, USA. https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet

Bathia, P., B.R. Beerelli and M.A. Laverdiere, 2011. Assisting programmers resolving vulnerabilities in Java web applications. Proceedings of the 1st International Conference on Computer Science and Information Technology (CCSIT'11), January 2-4, 2011, Springer, Bangalore, India, ISBN: 978-3-642-17880-1, pp: 268-279.

Chen, Y.L., H.M. Lee, A.B. Jeng and T.E. Wei, 2015. Droidcia: A novel detection method of code injection attacks on html5-based mobile apps. Proceedings of the 2015 IEEE International Conference on Trustcom/BigDataSE/ISPA Vol. 1, August 20-22, 2015, IEEE, Helsinki, Finland, ISBN: 978-1-4673-7951-9, pp: 1014-1021.

Dong, G., Y. Zhang, X. Wang, P. Wang and L. Liu, 2014. Detecting cross site scripting vulnerabilities introduced by HTML5. Proceedings of the 11th International Joint Conference on Computer Science and Software Engineering (JCSSE), May 14-16, 2014, IEEE, Beijing, China,ISBN:978-1-4799-5822-1, pp: 319-323.

Duchene, F., R. Groz, S. Rawat and J.L. Richier, 2012. XSS vulnerability detection using model inference assisted evolutionary fuzzing. Proceedings of the 2012 IEEE 5th International Conference on Software Testing, Verification and Validation (ICST'12), April 17-21, 2012, IEEE, Montreal, Canada,ISBN: 978-1-4577-1906-6, pp: 815-817.

Fogie, S., J. Grossman, R. Hansen, A. Rager and P.D. Petkov, 2007. XSS Attacks: Cross Site Scripting Exploits and Defense. Syngress, Boston, Massachusetts, ISBN-13: 978-1-59749-154-9, Pages: 464.

Gundy, M.V. and H. Chen, 2012. Noncespaces: Using randomization to defeat cross-site scripting attacks. Comput. Secur., 31: 612-628.

Huyam, A.A. and E. El-Qawasmeh, 2012. Discovering security vulnerabilities and leaks in ASP: NET websites. Proceedings of the 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec'12), June 26-28, 2012, IEEE, Kuala Lumpur, Malaysia, ISBN: 978-1-4673-1425-1, pp: 329-333.

Hydara, I., A.B.M. Sultan, H. Zulzalil and N. Admodisastro, 2014. An approach for cross-site scripting detection and removal based on genetic algorithms. Proceedings of the 9th International Conference on Software Engineering Advances ICSEA, October 12-16, 2014, IARIA, Nice, France, ISBN: 978-1-61208-367-4, pp: 227-232.

Hydara, I., A.B.M. Sultan, H. Zulzalil and N. Admodisastro, 2015b. Current state of research on cross-site scripting (XSS)-A systematic literature review. Inform. Software Technol., 58: 170-186.

Hydara, I., A.B.M. Sultan, H. Zulzalil and N. Admodisastro, 2015a. Removing cross-site scripting vulnerabilities from web applications using the OWASP ESAPI security guidelines. Indian J. Sci. Technol., 8: 1-5.

Javed, A. and J. Schwenk, 2013. Towards elimination of cross-site scripting on mobile versions of web applications. Proceedings of the 14th International Workshop on Information Security Applications (WISA'13), August 19-21, 2013, Springer, Jeju Island, Korea, ISBN: 978-3-319-05148-2, pp: 103-123.

Mutchler, P., A. Doupe, J. Mitchell, C. Kruegel and G. Vigna, 2015. A Large-scale study of mobile web app security. Proceedings of the 2015 International Workshop on Mobile Security Technologies (MoST'15), May 21, 2015, Fairmont San Jose, San Jose, California, USA., pp: 1-11.

Scholte, T., W. Robertson, D. Balzarotti and E. Kirda, 2012. Preventing input validation vulnerabilities in web applications through automated type analysis. Proceedinsg of the 2012 IEEE 36th Annual International Conference on Computer Software and Applications, July 16-20, 2012, IEEE, Izmir, Turkey, ISBN: 978-1-4673-1990-4, pp: 233-243.

Shah, S., 2012. HTML5 top 10 threats-stealth attacks and silent exploits. BlackHat, Las Vegas, Nevada, USA. https://media.blackhat.com/bh-us-12/Briefings/Shah/BH_US_12_Shah_Silent_Exploits_WP.pdfShar, L.K. and H.B.K.Tan, 2012. Automated removal of crosssite scripting vulnerabilities in web applications. Inform. Software Technol., 54: 467-478.

Shar, L.K. and H.B.K.Tan, 2012. Automated removal of cross site scripting vulnerabilities in web applications. Inform. Software Technol., 54: 467-478.

Sharma, P., R. Johari and S.S. Sarma, 2012. Integrated approach to prevent SQL injection attack and reflected cross site scripting attack. Intl. J. Syst. Assur. Eng. Manage., 3: 343-351.

Sun, Y. and D. He, 2012. Model checking for the defense against cross-site scripting attacks. Proceedings of the 2012 International Conference on Computer Science and Service System (CSSS'12), August 11-13, 2012, IEEE, Nanjing, China, ISBN: 978-1-4673-0721-5, pp: 2161-2164.

Vonnegut, S., 2015. XSS: The definitive guide to cross-site scripting prevention. Checkmarx, Nuremberg, Germany.https://www. checkmarx.com/2015/04/14/xss-the-definitive-guide-to-cross-site-scripting-prevention/