International Journal of Engineering & Technology, 7 (4.1) (2018) 18-21

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Towards Cross-site Scripting Vulnerability Detection in Mobile Web Applications

Isatou Hydara*, Abu Bakar Md Sultan, Hazura Zulzalil, Novia Admodisastro

Dept. of Software Engineering and Information System, Faculty of Computer Science and Information Technology
Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia
**Corresponding author E-mail:ishahydara@gmail.com*

Abstract

Cross-site scripting vulnerabilities are among the top ten security vulnerabilities affecting web applications for the past decade and mobile version web applications more recently. They can cause serious problems for web users such as loss of personal information to web attackers, including financial and health information, denial of service attacks, and exposure to malware and viruses. Most of the proposed solutions focused only on the Desktop versions of web applications and overlooked the mobile versions. Increasing use of mobile phones to access web applications increases the threat of cross-site scripting attacks on mobile phones. This paper presents work in progress on detecting cross-site scripting vulnerabilities in mobile versions of web applications. It proposes an enhanced genetic algorithm-based approach that detects cross-site scripting vulnerabilities in mobile versions of web applications. This approach has been used in our previous work and successfully detected the said vulnerabilities in Desktop web applications. It has been enhanced and is currently being tested in mobile versions of web applications. Preliminary results have indicated success in the mobile versions of web applications also. This approach will enable web developers find cross-site scripting vulnerabilities in the mobile versions of their web applications before their release.

Keywords: *Cross-site scripting; cross-site scripting vulnerability; software security; software testing; vulnerability detection*.

1. Introduction

In today's era, technology has become an important aspect of people's lives throughout the world. As this technology advances, we are relying more on the Internet to accomplish our daily transactions through Desktop and Mobile applications. Many businesses and organizations also rely heavily on the Internet through Desktop and mobile applications when providing access to many of their services.

Although web applications continue to be very vital to the successful and fast delivery of services by business and government entities, their security is mostly an afterthought. Hence, many security issues have been highlighted and discussed in recent times due to the many increasing security threats affecting web applications. Also, most of these applications can now be accessed through mobile phones, which increases their security risks even further.

Cross-site scripting (XSS) vulnerabilities [1, 2] are a type of injection vulnerabilities existing in web applications that lack proper encoding for user inputs. Web applications with this type of vulnerability can easily be exploited with online XSS attacks resulting to serious damages affecting users. There is lack of security in most web applications that are running online and mobile web applications are no exception. It is reported by [3] that 81% of all the mobile web applications they surveyed were vulnerable to XSS security attacks. HTML5, which is used by many mobile application developers, is reported to contain XSS vulnerabilities [4]. XSS has been added among the top 10 vulnerabilities in Mobile applications [5] as shown in Figure 1, as well as in HTML5 [6].

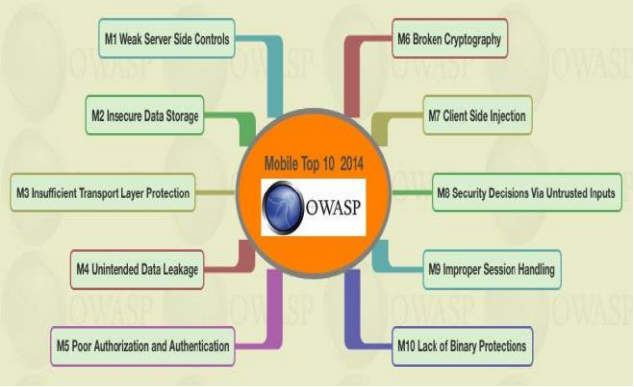

Fig. 1: Mobile Top 10, adopted from [5]

XSS vulnerabilities have been existed since the 1990s. This was in the early days of the World Wide Web. They are one of the injection vulnerabilities that allow malicious code to be injected into and executed by trusted web applications [1]. This happens as a result of failure during software development in validating user inputs in web application. Therefore, any web application that does not properly verify its users' inputs, make it easy for hackers to attack the application through XSS.

Figure 2 gives a high level view of XSS attack. A successful XSS attack can lead to serious security breaches affecting web applications and users mostly. When attackers inject malicious codes into a web application's user input that is not validated, they

Copyright © 2018 Authors. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

can inject scripts to steal cookies, steal users' private information, hijack users' accounts, manipulate some web contents, or cause denial of services attacks, and many other security breaches.

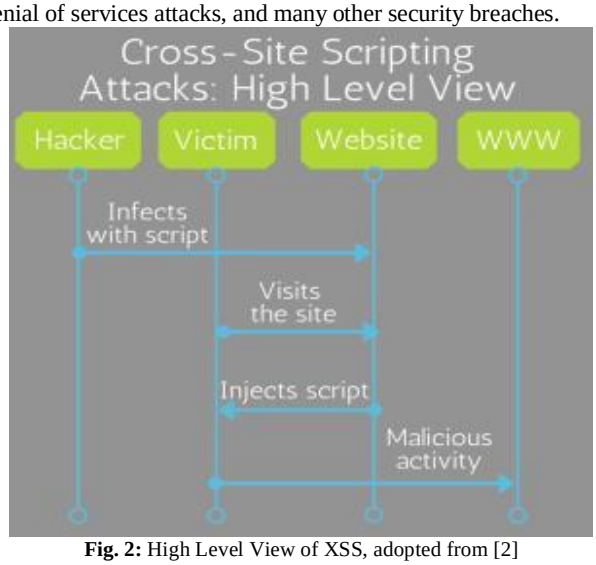

Fig. 2: High Level View of XSS, adopted from [2]

There are three types of XSS vulnerabilities referred to as reflected, stored and DOM-based [1, 2] that lead to XSS attacks. Reflected XSS attacks occur when the application references invalidated data from HTTP requests and sends feedback to the user immediately such as when username and password are given to login to an application. Stored XSS attacks can be stored in the web application's databases where information is saved, message forums, and comment fields. The malicious code is executed every time users open it thereby releasing their privileges to the attacker. Differently to the other two types of XSS, DOM-based XSS attacks are executed in the browser. This type of XSS vulnerability affects users more as it can steal private and sensitive information from the users' computers.

In this paper, we discuss a work in progress on an enhanced genetic algorithm-based approach for detecting XSS vulnerabilities in mobile web applications. This work is in progress and is built on our previous approach [7] on the detection of XSS vulnerabilities in desktop web application. It extends the approach to include mobile web applications as well as all the three types of XSS [8], [9]. The rest of the paper is thus organized: In Section 2, we discuss research studies related to our research work and in Section 3 we give an overview of our proposed solution to the problem. Section 4 describes the evaluation of the proposed approach and the concluding part is in Section 5.

2. Related Work

Research works related to XSS in mobile versions of web application are not many as the focus to it has only started recently. In their research [3], they have 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 [10]. Their approach, however, only focused on Reflected XSS.

As stated in the introduction section, HTML5 is used in the development of many mobile applications and has been found to contain XSS vulnerabilities. Hence, [4] 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 work included problems related to input string and dead code in the dataset.

Another research work on HTML5 was conducted by [11]. 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 [12] was also conducted to detect vulnerabilities in Android–based mobile web applications. XSS vulnerabilities were among those discovered in the tested applications.

Kaur et al. [13] and Wang et al. [14] have also proposed defensive frameworks for detecting DOM-based XSS on cloud-hosted and Desktop web applications, respectively. The work by [13] is also based on web applications built on HTML5, and hence detects HTML5 XSS attack vectors. The framework was successful in detecting and eliminating XSS attacks with some false positives and negatives, and it only focused on DOM-based XSS. With [14], the framework uses taint tracking technique in detecting XSS attacks by collecting and analyzing URL information, doing taint tracking analysis, and automatically verifying vulnerabilities. It also focused only on DOM-based XSS.

Shar and Tan [15] have proposed a method for detecting and removing XSS vulnerabilities in web applications. Their method is in two stages. The first uses static analysis technique to detect potential XSS vulnerabilities in the code of the tested applications. The second phase uses pattern matching techniques to provide an escaping mechanism to prevent any input values to be executed as script, thereby eliminating XSS vulnerabilities. This solution, however, only focused on reflected and stored XSS and was only evaluated on Desktop applications.

3. The Proposed Approach

The solution being implemented in this work employs a genetic algorithm (GA)-based solution to detect XSS vulnerabilities in mobile versions of web applications. The approach, as shown in Figure 3, has three parts, namely static analysis, vulnerability detection and vulnerability removal. In the first part, the conversion of the application source codes to Control Flow Graphs (CFGs) is carried out. Static analysis techniques are used in this part, whereby every node represents a statement and every edge represents the flow of data from one node to another. The second part conducts the detection of the vulnerabilities in the applications using GA, and the third part involves the vulnerabilities removal. In this work in progress, we only focus on the first two components of the approach. The details of the approach can be found in our previous work [7].

This approach has been successfully implemented on Desktop web applications [16]. In the approach we formulated the detection of XSS vulnerabilities as a search optimization problem in evolutionary computing. GAs have been used successfully to generate a smaller number of test cases in software testing to discover as many problems as possible in web application source codes [17]. Another research by [18] also used Genetic Algorithms to detect XSS vulnerabilities in web applications. Similarly, we can use a GA-based approach to mitigate similar XSS vulnerabilities in mobile versions of web applications.

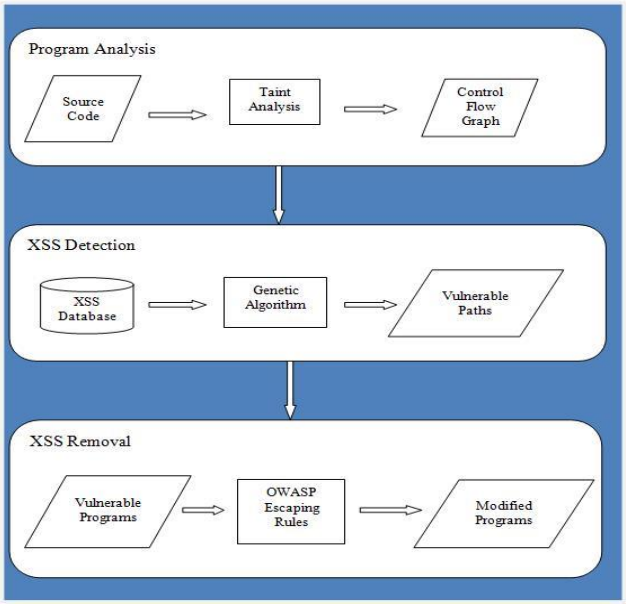

Fig. 3: Approach for XSS Detection and Removal

3.1. Taint Analysis

Taint analysis technique is a well known White Box testing technique used to follow the tainted or untainted statuses of variables flowing in an application and to determine whether a data input submitted and processed without proper validation [15, 19, 20]. In the detection of XSS vulnerabilities, the tainted variables refer to the inputs in a web application that are from users or databases, as well as print statements appending strings into web pages.

In order to completely test the source codes of web applications, the White Box testing coverage criteria should be used. These criteria are known as statement coverage, branch coverage, or path coverage [18]. The path coverage criterion is selected in this case as it encompasses the previous two and provides more coverage. However, it should be noted that it is quite unlikely to be able to cover all the paths of the source code in software security testing. It is important, therefore, to select an identified subset of the paths that are of interest to us. These paths are the vulnerable ones that when executed will reveal existing XSS vulnerabilities in the source code. They are, therefore, the paths where inputs are executed without any validations.

3.2. Enhanced Genetic Algorithm

Genetic Algorithms (GAs) are classified as a type of Evolutionary Algorithms (EAs). EAs are defined as metaheuristic optimization algorithms that are inspired by biology and are based on natural population [21]. GAs use natural evolution processes to find the best solution to a search problem in a particular search space. Such processes include mutation, crossover, natural selection, and survival of the fittest [22]. However, GAs have differences with respect to other EAs in that they use crossover operations also known as recombination and also use binary coding in bits or bit-strings in representing populations [22].

GAs have been and are still being used to solve problems in many computer science fields including software testing [23, 24], intrusion detections in network security [25, 26] as well as many other similar areas. We believe GA techniques could help in detecting XSS vulnerabilities in mobile web applications. For this part, techniques similar to those using test case generations with GAs for software testing can be used.

GAs have been used and proven as great solutions to many application engineering problems, since being discovered. They have been successfully used in software security testing [23, 24] as well as in intrusion detection systems [25, 26]. Thus we have used it detecting XSS security vulnerabilities in mobile web applications.

4. Evaluation

Implementation of the proposed approach has been carried out in a prototype tool and we are currently evaluating it for its effectiveness in detecting XSS vulnerabilities in mobile versions of web applications. The development of the tool was done using the Eclipse IDE and Java as the Programming Language and relevant libraries. The JGAP (Java Genetic Algorithm Package) tool [27] was also used in the Eclipse IDE as a library to enable us to use its GA operators in our implementation.

Preliminary evaluation results on some mobile applications are promising and show the feasibility of our approach in detecting XSS vulnerabilities in mobile versions of web applications. In the future, we will continue testing our approach with similar dataset as [3] and [11] 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.

5. Conclusion

In this paper, we presented our work in progress on XSS detection in mobile applications using an enhanced genetic algorithm-based approach. XSS is a well known security problem for web applications and it is becoming common in the mobile versions. Successful attacks can lead to very serious consequences thereby affecting website users the most. The proposed approach is an enhancement based on our previous approach for Desktop web applications. It employs better and improved GA operators in detecting XSS vulnerabilities and it also included all the three types of XSS vulnerabilities. As this is a progressive work, we will continue with the full evaluation and validation the proposed approach. Preliminary evaluation with our prototype tool has indicated promising results. The approach will be validated also 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

[1] OWASP, Cross-site Scripting (XSS) - OWASP, *OWASP Foundation*, 2016. Available online: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS). Accessed: January 26, 2017.

[2] S. Vonnegut, XSS: The Definitive Guide to Cross-Site Scripting Prevention, *Checkmarx.com*, 2015. Available online: https://www.checkmarx.com/2015/04/14/xss-the-definitive-guide-to-cross-site-scripting-prevention/. Accessed: January 26, 2017.

[3] A. Javed and J. Schwenk (2014), Towards Elimination of Cross-Site Scripting on Mobile Versions of Web Applications, in *14th WISA: International Workshop on Information Security Applications*, vol. LNCS 8267, pp. 103–123.

[4] Y. L. Chen, 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, in *Proceedings - 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2015*, vol. 1, pp. 1014–1021.

[5] OWASP, Mobile Top 10 2014-M7 - OWASP, *OWASP Foundation*,

2014. Available online: https://www.owasp.org/index.php/Mobile_Top_10_2014-M7. Accessed: January 26, 2017.
[6] S. Shah (2012), HTML5 Top 10 Threats - Stealth Attacks and Silent Exploits, in *BlackHat USA 2012*, pp. 1–21.
[7] I. Hydara, A. B. M. Sultan, H. Zulzalil, and N. Admodisastro (2014), An Approach for Cross-Site Scripting Detection and Removal Based on Genetic Algorithms, in *ICSEA 2014 : The Ninth International Conference on Software Engineering Advances*, no. November 2014, pp. 227–232.
[8] OWASP, XSS (Cross Site Scripting) Prevention Cheat Sheet, *OWASP Foundation*, 2016. Available online: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet. Accessed: January 26, 2017.
[9] OWASP, DOM based XSS Prevention Cheat Sheet, *OWASP Foundation*, 2017. Available online: https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet. Accessed: October 9, 2017.
[10] OWASP, OWASP ModSecurity Core Rule Set (CRS), *OWASP Foundation*, 2016. Available online: https://modsecurity.org/crs/. Accessed: January 26, 2017].
[11] G. Dong, X. Wang, P. Wang, and L. Liu (2014), Detecting Cross Site Scripting Vulnerabilities Introduced by HTML5, in *11th International Joint Conference on Computer Science and Software Engineering*, pp. 319–323.
[12] P. Mutchler, A. Doupe, J. Mitchell, C. Kruegel, and G. Vigna (2015), A Large-Scale Study of Mobile Web App Security, *Mob. Secur. Technol. 2015*.
[13] G. Kaur, B. Pande, A. Bhardwaj, G. Bhagat, and S. Gupta (2018), Efficient yet Robust Elimination of XSS Attack Vectors from HTML5 Web Applications Hosted on OSN-Based Cloud Platforms, *Procedia Comput. Sci.*, vol. 125, pp. 669–675.
[14] R. Wang, G. Xu, X. Zeng, X. Li, and Z. Feng (2017), TT-XSS: A novel taint tracking based dynamic detection framework for DOM Cross-Site Scripting, *J. Parallel Distrib. Comput.*, pp. 4–10.
[15] L. K. Shar and H. B. K. Tan (2012), Automated removal of cross site scripting vulnerabilities in web applications, *Inf. Softw. Technol.*, vol. 54, no. 5, pp. 467–478.
[16] I. Hydara, A. B. Sultan, H. Zulzalil, and N. Admodisastro (2015), Cross-Site Scripting Detection Based on an Enhanced Genetic Algorithm, *Indian J. Sci. Technol. ISSN*, vol. 8, no. 30, pp. 1–7.
[17] A. Avancini and M. Ceccato (2013), Circe: A grammar-based oracle for testing Cross-site scripting in web applications, in *Proceedings - Working Conference on Reverse Engineering, WCRE*, pp. 262–271.
[18] M. A. Ahmed and F. Ali (2016), Multiple-path testing for cross site scripting using genetic algorithms, *J. Syst. Archit.*, vol. 64, pp. 50–62.
[19] A. Avancini and M. Ceccato (2010), Towards Security Testing with Taint Analysis and Genetic Algorithms, in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*, no. Section 5, pp. 65–71.
[20] B. Shuai, M. Li, H. Li, Q. Zhang, and C. Tang (2013), Software vulnerability detection using genetic algorithm and dynamic taint analysis, *2013 3rd Int. Conf. Consum. Electron. Commun. Networks*, pp. 589–593.
[21] T. Weise, *Global Optimization Algorithms – Theory and Application* –, 2nd Ed. 2009. pp. 1-820.
[22] S. H. Aljahdali, A. S. Ghiduk, and M. El-Telbany (2010, The limitations of genetic algorithms in software testing, *ACS/IEEE Int. Conf. Comput. Syst. Appl. - AICCSA 2010*, pp. 1–7.
[23] A. Rathore (2011), Application of Genetic Algorithm and Tabu Search in Software Testing, in *Proceedings of the Fourth Annual ACM Bangalore Conference*, pp. 1–4.
[24] P. R. Srivastava and T. Kim (2009), Application of Genetic Algorithm in Software Testing, *Intenational J. Softw. Eng. Its Appl.*, vol. 3, no. 4, pp. 87–96.
[25] Z. Banković, D. Stepanović, S. Bojanić, and O. Nieto-Taladriz (2007), Improving network security using genetic algorithm approach, *Comput. Electr. Eng.*, vol. 33, no. 5–6, pp. 438–451.
[26] A. B. . A. Al Islam, M. A. Azad, M. K. Alam, and M. S. Alam (2007), Security Attack Detection using Genetic Algorithm (GA) in Policy Based Network, *2007 Int. Conf. Inf. Commun. Technol.*, pp. 341–347.
[27] JGAP, JGAP: Java Genetic Algorithms Package, 2016. Available online: http://jgap.sourceforge.net/. Accessed: December 29, 2016.