Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 72 (2015) 112 – 121

Procedia Computer Science

The Third Information Systems International Conference

A Method for Web Application Vulnerabilities Detection by Using Boyer-Moore String Matching Algorithm

Ain Zubaidah Mohd Saleh[a,]a*, Nur Amizah Rozali[a,]b*, Alya Geogiana Buja[a,b,]c*, Kamarularifin Abd. Jalil[a], Fakariah Hani Mohd Ali[a], Teh Faradilla Abdul Rahman[a,c]

[a] *Universiti Teknologi MARA, Shah Alam 40000, Selangor*
[b] *Universiti Teknologi MARA Kampus Jasin, Merlimau 77300, Melaka*
[c] *Universiti Teknologi MARA Kampus Puncak Alam, Kuala Selangor 42300, Selangor*

Abstract

Internet have become a great medium of communication as it is free, supportive, entertaining and easily for reachable to millions of people today. The usage of Internet among people become higher day by day, thus also increase the number of web application. Nevertheless, most of the web application exists have some vulnerability as there are some irresponsible people known as hacker that able to interrupt the peace of it. Some of well-known web application vulnerabilities are SQL Injection, Buffer Overflow, Cross Site Scripting and Cross Site request Forgery. In order to overcome this vulnerabilities, it is important to detect first the problem before prevent it. At present, there are a lot of web application vulnerabilities scanner that have been proposed by researcher for detecting web application vulnerabilities such as Acunetix WVS by Acunetix, Netsparker by Mavituna Security, w3af by w3af.org and Firefuzzer. However, these scanners have some limitation such as higher false negative although some of it has no false positive. Therefore, this paper proposed a technique aim to solve these issues by developing a detection method for detect the web application vulnerabilities by using Boyer-Moore String Matching Algorithm. Numerous experiments have been conducted in order to evaluate the performance. The result shows that proposed method has performed well in terms of the ability to accurately detect vulnerabilities based on false negative and have no false positive with low processing time.

Peer-review under responsibility of organizing committee of Information Systems International Conference (ISICO2015)
Keywords: Web Application Vulnerabilities, SQL Injection, Buffer Overflow, Cross-Site Scripting, Cross-Site Request Forgery;

* Corresponding author. Tel.: +0-006-013-4248134
E-mail address: ainzubaidahmsaleh@gmail.com
* Corresponding author. Tel.: +0-006-013-4775977
E-mail address: nuramizahrozali@ymail.com
* Corresponding author. Tel.: +0-006-014-8852844.
E-mail address: geogiana@melaka.uitm.edu.my.

Peer-review under responsibility of organizing committee of Information Systems International Conference (ISICO2015)
doi:10.1016/j.procs.2015.12.111

1. Introduction

As Internet become more and more significant, there are some individuals identified as hackers that have the ability to interrupt the peace of consuming Internet. For that reason, network security is required. Network security emphasis on securing networks from any violence or exploit especially from hackers and typically handles by network administrator on each organization that applies security policy [1]. Thus, to ensure the three main goals of security which are integrity, availability and confidentiality is guarantee, network security become the main role to some kind of vulnerabilities in web application. Vulnerabilities can be referring as the flaws where attacker can take benefit by exploiting it to gain unauthorized access to their target. There are four most common web application vulnerabilities that exist in a web application are SQL Injection, Buffer Overflow, Cross Site Scripting and Cross Site Request Forgery.

SQL Injection is a violence in which the attacker inserts SQL commands into form or parameter values. It abuses the use of SQL query in the application [2]. SQL Injection has become a predominant type of attacks that target web applications. The Open Web Application Security Project (OWASP) ranks it on top among the Top-10 security threats [3]. Buffer Overflow is an activity that can make the memory allocated to a certain application become massive [4]. For example, an application supposing a five-digit postcode therefore the programmer only allocates enough memory for the perimeter. If an attacker enters more than five digits for example hundreds of digit, the application will end up using more memory than what it should. As of September 2010, 12 of the 20 most severe vulnerabilities ranked by US-CERT were Buffer Overflow related [5].

Cross-Site Scripting (XSS) is a strike that tries the use of JavaScript in a web and by using JavaScript, the grouped information can be gotten as JavaScript can get to private information [6]. Cross-Site Request Forgery (CSRF) assault strengths a logged-on victim's program should send a pre-authenticated demand should a vulnerable web application, which at that point constrains those victim's programs to perform a dangerous activity of the profit of the assailant [7]. According to Application Vulnerability Trends Report 2014, CENZIC found about 25% of the total, the Cross-Site Scripting was the most frequently found vulnerabilities in applications tested in 2013 [8]. According National Vulnerabilities Database (NVD), in the year 2014 the number of Cross-Site Request Forgery detected are 254 vulnerabilities and the number of these vulnerabilities will increase more in the next year [9].

There are a lot of web application vulnerabilities detection scanners existing in Internet. Either it free source or need to buy, there are more or less problems faced by these tools. The common problem meet by some of the scanner are false positive and false negatives. A false positive is when there is an error whereby a web application tested for is mistakenly found the vulnerabilities which actually there is none. Meanwhile, false negatives are the scanner does not found any vulnerability in a web application and telling user that the web is secure. However, actually the web application may have some vulnerability [10]. Thus, by proposed a method for detecting the vulnerabilities in web application by implement Boyer-Moore String Matching Algorithm, it will help the web application administrator to take a look and always standby in secure mode to avoid and secure mode for avoiding any attacks from the attacker.

2. Related Works

This section discussed on four subjects. First subject is about some previous works regarding on web application vulnerabilities which are on SQL Injection, Buffer Overflow, Cross Site Scripting and Cross Site Request Forgery. The second subject is on web application vulnerabilities scanner. There are some related works that have been done by some of the researcher out there regarding the web application vulnerability scanner. Some of them just focus on a certain type of web application vulnerability but some able to detect many type of vulnerability in one project. The last subject is on Boyer-Moore Algorithm.

Acunetix used the AcuSensor Technology can consequently check vulnerabilities found through a discovery, filtering, strategies by performing extra test amid the executions of the application's source code and an Acunetix output to give a close to 0% false positive rate when AcuSensor is utilized [11]. Netsparker Web Application Security Scanner is definitely not hard to use and its exceptional acknowledgement and safe abuse routines license it to be dead exact in reporting, subsequently it is disclosures [10]. w3af is an open-source web application security scanner. The project provides a vulnerability scanner and exploitation tool for Web applications. This cross-platform tool is available in all of the popular operating systems such as Microsoft Windows, Linux and Mac OS and is written in the Python programming language [12]. Firefuzzer is a penetration testing tool. The aim of the fuzzer is to discover unknown vulnerabilities in web applications. The FireFuzzer application would be executed from the Command Prompt. It has two major modules which are Buffer Overflow and Cross Site Scripting [13].

In Boyer-Moore String Matching Algorithm, Ganga [34] proposed an algorithm which is a phrase based document retrieval to retrieve the similar documents by combining two exiting algorithm suffix tree, index data structure and Boyer-Moore String Matching Algorithm is applied to check the presence of pattern for example the input phrase in order and without order to retrieve the similar documents. Kumar and Veerendranath [14] have implemented the Boyer-Moore String Matching Algorithm in data mining for Network Intrusion Detection (IDS). Another deployment of Boyer-Moore String Matching is in library book information on mobile application [15]. Other than that, the Boyer-Moore String Matching Algorithm is also applied in Network Security Audit System [16]. The Boyer Moore String Matching Algorithm also implemented in assertion-based software testing which is a powerful tool for automatic run-time detection of software errors during testing, debugging and maintenance [17]. In this research paper, the Boyer-Moore use for scanning the URL from right to left in order to detect the vulnerabilities.

3. Boyer-Moore String Matching Algorithm

The most well-known single pattern matching algorithm is the Boyer–Moore (BM) String Matching Algorithm. It compares the characters of the pattern with the characters of the text from right to left by using two heuristics called as the bad-character shift and the good-suffix shift [18]. Boyer-Moore Algorithm has two heuristic phases. First: bad character shift, which starts the comparison between the pattern "P" and the text "T" from the right to the left, and in case of mismatching, the algorithm will shift forward to an "M" character (the pattern length). Second heuristic is good suffix shift, which starts a comparison from the right to the left, and in case of matching, then the algorithm move to the next character in the text "T" with the next character in the pattern "P", until get matching with all string characters; but in case of mismatching, the algorithm is move to the next occurrence that was matched before [19]. Boyer-Moore String Matching Algorithm have been choose to detect the web application Overflow vulnerabilities as it fulfill the desired criteria which are able to scan from right to left and scan character by character in one pattern.

4. Methodology

Research methodology describes the methods and procedure that have applied in order to complete this paper and there are six phases involved. The first phase is information gathering in order to come out with the research requirements. For this paper, the literature review focused on the computer security, web application vulnerabilities, vulnerabilities scanning and string matching technique. Due to the vulnerabilities, detection model which able to recognize the type of vulnerabilities is needed. In this paper, only four type common vulnerabilities have been focused which are SQL Injection, Buffer Overflow, Cross Site Scripting and Cross Site Request Forgery. String matching technique has been selected as an algorithm to detect the vulnerabilities.

Once these requirements are identified, the next phase is data collection. In order to ensure that the proposed method works, several experiments have been carried out by using several URL of web application and it's collected and extracted from different sources. 20 URL have been collected for each SQL Injection, Cross Site Scripting and Cross Site Request Forgery which 10 URL for testing the false positive while another 10 URL is for testing false negative. For Buffer Overflow, only 10 URL collected as it is difficult to find URL that satisfies the requirement needed. The third phase is to determine the algorithm requirement for the method and to select the suitable detection technique. Based on the requirements, Boyer-Moore String Matching Algorithm has been selected to be used in the proposed method. This is because Boyer-Moore will scan character by character from right to left until the match is found.

Phase four identify the process of detecting web application vulnerabilities. This phase specifically focuses on how to perform the process of detecting the web application vulnerabilities. The modules are SQL Injection, Buffer Overflow, Cross Site Scripting and Cross Site Request Forgery. The process of proposed method worked can be referring in Appendix A. The next phase is constructing the algorithm for detecting the web application vulnerabilities. During this phase, the algorithms for detecting the web application vulnerabilities have been constructed. There are three steps for development of algorithm which are identify the problem, define the algorithm process and construct the algorithm. The problem for this paper is the web application vulnerabilities. After the problem has been identified, it has been formulated and as a result, algorithms have been developed. The last phase is evaluation. To evaluate this method, a series of experiments have been conducted. The metric evaluations for the experiment have been identified which are based on accuracy and efficiency.

5. Result and Discussion

5.1. Accuracy

Throughout this paper, the accuracy can be defined measures as how detection method able to detect either there is vulnerable or not based on given condition which is the rate of false positive and false negative. A false positive is considered as a drawback of a detection method. Minor weaknesses of any detection method can lead to false positives. False positives are equally dangerous as false negatives [20]. The accuracy of the detection method also has been compared with other commercial tools which are Acunetix and Netsparker for SQL Injection, Cross Site Scripting and Cross Site Request Forgery. While for Buffer Overflow, w3af and Firefuzzer have been chosen to be used to compare the accuracy. The False Positive and False Negative measurements are as follows:

$$Accuracy_{FP} = (Total\ Number\ of\ FP\ Instances/Total\ Number\ of\ Benign\ Instances) * 100 \quad (1)$$

Where;

$Accuracy_{FP}$	=	False Positive Rate
Total Number of FP Instances	=	Number of tests of URL that lead to False Positive.
Total Number of Benign Instances	=	Total overall number of tests

$$Accuracy_{FN} = (Total\ Number\ of\ FN\ Instances/Total\ Number\ of\ Benign\ Instances) * 100 \quad (2)$$

Where;

$Accuracy_{FN}$	=	False Negative Rate
Total Number of FN Instances	=	Number of tests of URL that lead to False Negative.
Total Number of Benign Instances	=	Total overall number of tests

Figure (c) shows that the percentage of accuracy of false negative of the proposed approach, proposed method, Acunetix and Netsparker for Cross Site Scripting are achieving 100% accuracy. This means that, all these three web application vulnerability scanners have zero false negative error. Meanwhile, the percentage of accuracy of false positive of the proposed method, Acunetix and Netsparker is achieving 100% accuracy as shown. Therefore, this result shows that all these three web application vulnerability scanners have zero false positive. Figure (d) shows that the percentage of accuracy of false negative of the proposed method, Acunetix and Netsparker is achieving 100% accuracy. This means that, all these three web application scanners have zero false negative error. Meanwhile, the percentage of accuracy of false positive of the proposed method, Acunetix and Netsparker is achieving 100% accuracy as shown in Figure (d). Therefore, this result shows that all these three web application vulnerability scanners have zero false positive.

Figure (a) shows the result of the SQL Injection accuracy based on the study from 20 URLs. The accuracy of the proposed method is examined based on false negative and false positive of the vulnerabilities. Out of 20 URLs collected, the false negative is about 10% which is lower than Acunetix and Netsparker tool. Acunetix have highest false negative which is about 30% for overall testing vulnerable URL while Netsparker is about 20% of false negative. In terms of false positive, which is not vulnerable URL, all the method is free for false positive. This result highlights one of the most important advantages of proposed method which is able to detect the SQL Injection in web pages dynamically. Figure (b) shows the result of the Buffer Overflow accuracy based on the study from 10 URLs. Out of 10 URLs collected, proposed method have lowest false positive which is 0% compared with another tools which are 10% for w3af and 20% for Firefuzzer. Difference with SQL Injection detection, for false negative, there are no data to test as most of web application nowadays are free with Buffer Overflow. Buffer Overflow basically happened in C/C++ and assembly language as this language provide no built-in protection against accessing or overwriting data in any part of memory and do not automatically check that data written to any array. Figure (a), (b), (c) and (d) can refer in Appendix B.

5.2. Processing Time

In this section, the efficiency is evaluated as the time taken for all modules of web application vulnerabilities to complete the scan and detect the vulnerabilities. The time is measured in seconds. The formula of the processing time in research are defined as end time (time taken for each modules detect vulnerabilities) minus by start time (after the detection process completed).

Based on the result in Figure (a), the processing time for all 20 URL for SQL Injection detection are less than 300 seconds. The low processing time for some of the URL is caused by the different point of injections in each webpage. The smaller number of different point of injections, which is the URL for number 2, 4, 5, 7 and so on, require only 120 seconds compared to the URL for 6 and 9 which require more than 150 seconds. While for Buffer Overflow detection in Figure (b), the processing times for all 10 URL are less than 70 seconds. The low processing time for some of the URL is caused by the length of input string in each webpage.

Based on the result in Figure (c), the processing times for all 20 URL for XSS module are less than 500 seconds. The low processing time for this testing is URL 10, 14, 15 and 20 which is 0 seconds. Meanwhile, in the Figure (c) shows the result of processing time for CRSF module. The processing times for all 20 URL for CSRF module are less than 75 seconds. The low processing time for CSRF module testing is at URL 1, 3, 5, 7, 13, 14, 15 and 19 which is 0 seconds. In the Figure (c), we can see that the URL 2 have higher processing time and this situation happen because there are several factors that affecting the processing time of web application vulnerability scanner. So the situation of URL number two is affected by the factor of size and complexity of the URL itself. Figure (a), (b), (c) and (d) can refer in Appendix C.

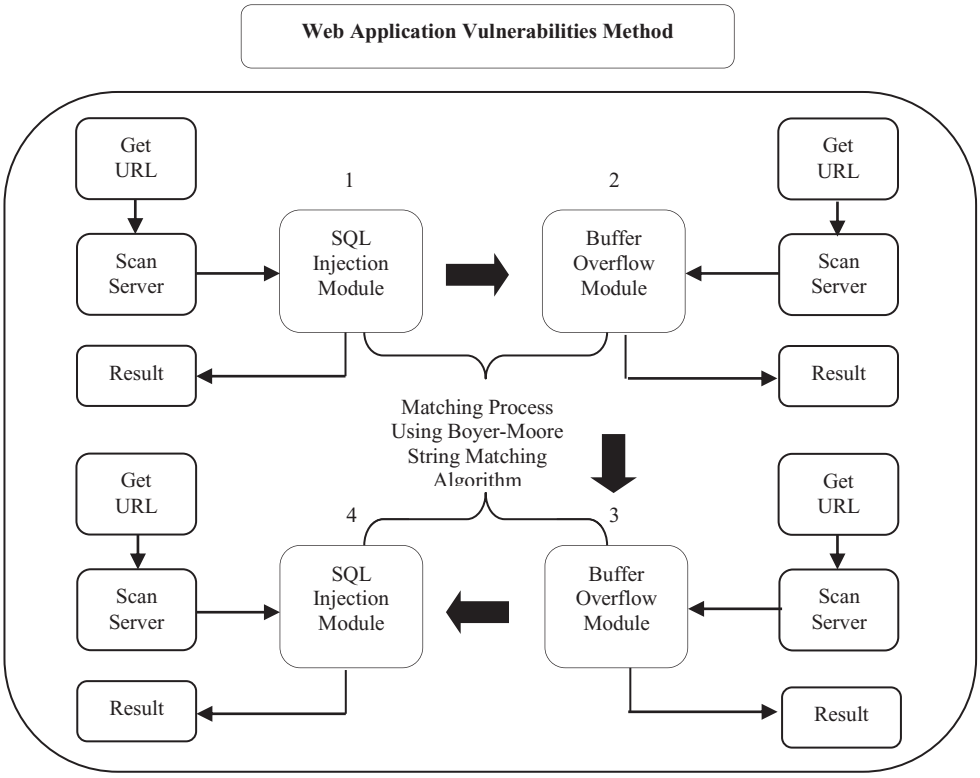
6. Conclusion

This paper proposed a method for Web Application Vulnerabilities detection. Proposed method uses Uniform Resource Locator (URL) as input for Web Application Vulnerabilities detection. Two sets experiments have been conducted to evaluate the performance of proposed method based on the accuracy (false positive and false negative) and efficiency (processing time). The results show that proposed method yields low false negative and false positive for all detection. For SQL Injection, the number of different point injection for each web pages tested can affect the processing time, where the higher number of different point of injection, the higher the processing time. For Buffer Overflow, the length of input string has become the main factor of increasing of processing time. For Cross Site Scripting and Cross Site request Forgery, the size and complexity of the URL itself become the main factor the processing time become high. In addition, the proposed method can help the web application developer or administrator to take any further action to secure their application from being hacked or attacked by the unethical person outside the network by exploiting and compromising the vulnerabilities of the web application towards SQL Injection, Buffer Overflow, Cross Site Scripting and Cross Site Request Forgery attack. At this moment, there is a limitation of the method proposed in this dissertation in order to detect the vulnerabilities. The proposed method is unable to detect the SQL Injection vulnerabilities when the number of different point of injection is large. Meanwhile for the Cross Site Scripting and Cross Site Request Forgery, when the length of URL is long so the scanner will take long time to complete their scanning.. This method also only can detect the web pages not overall web application. For further works, this paper suggests that the proposed method should be combined with the hybrid string matching in order to further strengthen the detection capabilities.

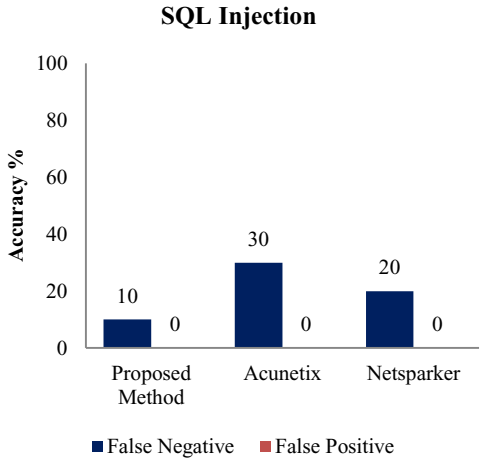
Acknowledgements

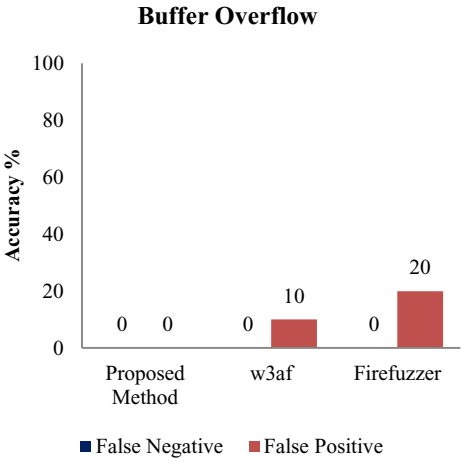
The authors would like to thank University Teknologi MARA for supporting this work. This work was supported by University Teknologi MARA, Malaysia and Ministry of Higher Education of Malaysia under Research Acculturation Grant (RAGS) 2013.

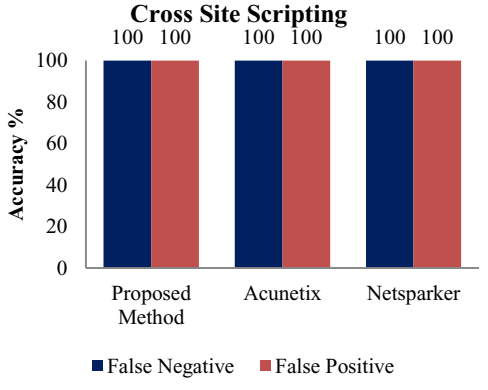
Appendix A. Process of proposed method for web application vulnerabilities detection

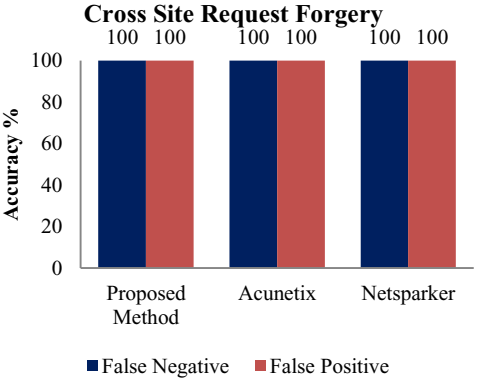


Appendix B. Accuracy for SQL injection, Buffer Overflow, Cross-Site Scripting and Cross-Site Request Forgery

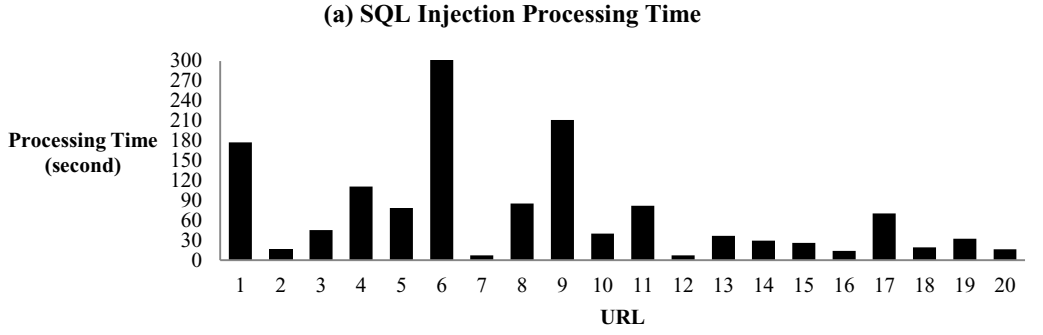


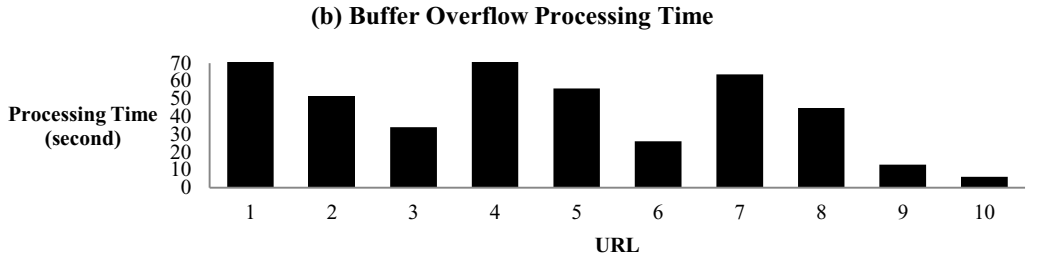


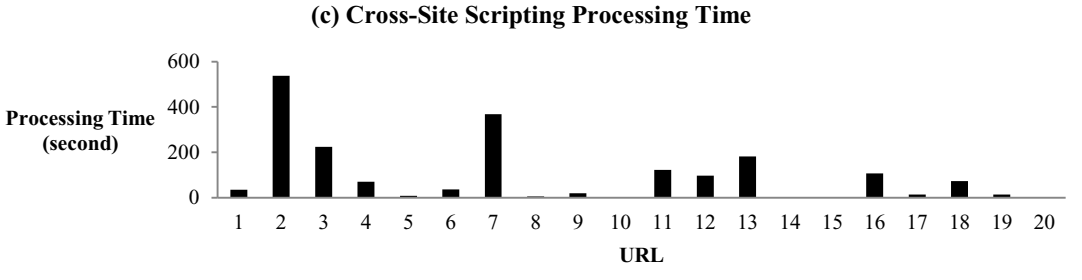


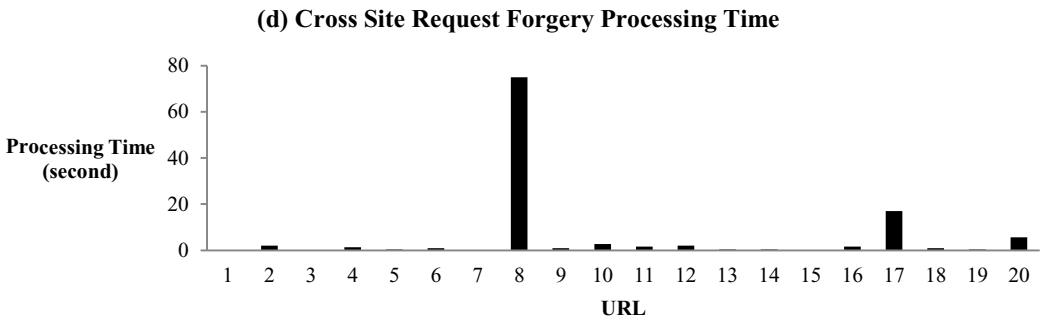


Appendix C. Processing Time for SQL injection, Buffer Overflow, Cross-Site Scripting and Cross-Site Request Forgery









References

[1] Brinkley, D. L., & Schell, R. R. (1995). Concepts and terminology for computer security. Information Security: An integrated collection of essays, 40-97.

[2] William G.J. Halfond, Jeremy Viegas and Alessandro Orso, "A Classification of SQL Injection Attacks and Countermeasures," *College of Computing Georgia Institute of Technology IEEE*, 2006.

[3] OWASP: Top 10 Security Threats 2013. Retrieved online: https://www.owasp.org/index.php/Top_10_2013-A1-Injection (2013)

[4] Wilander, J., & Kamkar, M. (2003, February). A Comparison of Publicly Available Tools for Dynamic Buffer Overflow Prevention. In NDSS (Vol. 3, pp. 149-162)

[5] US-CERT. Vulnerability notes database. Retrieved online: www.kb.cert.org/vuls.

[6] Manico, J., (2012), Future of XSS Defense

[7] Kuchiwale, S. L., & Lolge, S. A Survey on Website Attacks Detection And Prevention.

[8]CENZIC- Application Vulnerability Trends Report 2014. Retrieved online: https://www.trustwave.com/Resources/Library/Documents/Cenzic-Application-Vulnerability-Trends-2014/

[9]National Vulnerability Database. Retrieved from: https://web.nvd.nist.gov/view/vuln/statisticsresults?adv_search=true&cves=on&cwe_id=CWE-79.

[10] Abela, R. (2015). "Netsparker 4.1 Release – New Security Checks and Improvements". Retrieved online: https://www.netsparker.com/blog/releases/netsparker-4-1-new-security-checks/

[11]Naudi, T. (2015). "Keeping your Website Secure just got Easier with Acunetix 10". Retrieved online: ://www.acunetix.com/blog/news/keeping-yourwebsite-secure-just-got-easier-with-acunetix-10/

[12] w3af. Retrieved online: w3af.org

[13] Firefuzzer. Retrieved online: https://code.google.com/p/firefuzzer/

[14] Ganga, B. (2014). Phrase Based Document Retrieving by Combining Suffix Tree index data structure and Boyer-Moore faster string searching algorithm. *International Journal of Advancements in Research & Technology, ISSN*, 2278-7763.

[15] Kumar, K. V., & Veerendranath. B. (2014). Data Mining Model for Network Intrusion Detection Using Boyer-Moore Algorithm. *International Journal of Advanced Research in Computer Science & Technology (IJARCST 2014)(pp. 66 – 69),* ISSN : 2347 – 8446

[16] Basari, A. S. H., Yusop, N., Hussin, B., & Shibghatullah, A. S. (2014). Hybrid of Boyer Moore and Rule based System for Mobile Library Book Information.*International Journal of Computer Applications*, *90*.

[17] Wang, X., ZhuGe, B., & Wang, W. (2010, May). A BM algorithm oriented on Network Security Audit System. In *e-Business and Information System Security (EBISS), 2010 2nd International Conference on* (pp. 1-4). IEEE.

[18] Boyer, R. S., & Moore, J. S. (1977). A fast string searching algorithm.*Communications of the ACM*, *20*(10), 762-772.

[19] Hnaif, A. A. (2015). A New Platform NIDS Based On WEMA.

[20] Alakeel, A. M. (2015). Using Fuzzy Logic Techniques for Assertion-Based Software Testing Metrics. *The Scientific World Journal*, *2015*.