International Journal of Engineering & Technology, 7 (4.31) (2018) 40-45

SPC

International Journal of Engineering & Technology

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

Research paper

Comparing Web Vulnerability Scanners with a New Method for SQL Injection Vulnerabilities Detection and Removal EPSQLiFix

Kabir Umar[1*], Abu Bakar Sultan[2], Hazura Zulzalil[3], Novia Admodisastro[4], Mohd Taufik Abdullah[5]

[1]Faculty of Computer Science and Information Technology, Bayero University Kano, Gwarzo Road, Kano, Nigeria
[2]Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang, Selangor, Malaysia
**Corresponding author E-mail: ukabir.se@buk.edu.ng*

Abstract

Web vulnerabilities have become a major threat to the security of information and services accessible via the internet. Dynamic analysis based Web Vulnerability Scanners (WVS) have been employed to facilitate detection of vulnerabilities, though, such scanners could not remove the detected vulnerabilities. Empirical evidences show that some existing static analysis techniques targeted both detection and removal of vulnerabilities. However, these techniques are not adequately effective – they report considerably large number of false positives and do not achieve fully automatic vulnerabilities removal. Although, clear understanding of the workflow of WVSs is very essential in designing more improved scanners, current literature does not provide a comprehensive presentation on workflow of WVSs. Thus, this paper presents thorough description of generic WVS through synthesis and aggregation of knowledge. In addition, the paper presents overview of an Evolutionary Programming (EP) based static analysis method for automatic detection and removal of vulnerabilities called EPSQLiFix. Lastly, the paper compares the workflow of WVSs to that of EPSQLiFix method.

Keywords: *SQL Injection, Reachability Analysis, Vulnerability Detection and Removal*.

1. Introduction

In today's world of increasing dependence on Internet based applications and services, web vulnerabilities have become a major threat to information security [1]. Thus, it is highly desirable to conduct adequate testing of web applications for detection and removal of vulnerabilities at development stage, before deployment [2, 3]. Although Web Vulnerability Scanners (WVSs) have been employed by developers to facilities testing for vulnerabilities detection [4], unfortunately, WVSa are generally incapable of correcting (i.e., removing) vulnerabilities after detection. Consequently, modification of source code for vulnerabilities removal is predominantly done manually by the developer, thus allowing room for human errors and limitations [3, 5-7].

Clear understanding of workflow of WVSs is very essential for revealing strengths and weaknesses of existing WVSs, and also, for designing more improved scanners.

This paper presents comprehensive description of generic WVS through synthesis and aggregation of knowledge reported in several empirical studies. In addition, the paper presents overview of an Evolutionary Programming (EP) based static analysis method for automatic detection and removal of SQLIVs, called EPSQLiFix, and highlights how the workflow of the method compares to that of WVSs. The remaining of this paper is arranged as follows. Section II presents review on existing WVSs, and highlight vulnerability detection approaches followed by WVSs. Section III presents generic workflow of WVS. Section IV gives overview of proposed method for detection and removal of SQLIVs, EPSQLiFi. Section V presents comparative analysis between the proposed method and WVS. Finally, conclusion is presented in Section VI.

2. Existing Web Vulnerabilities Scanners

In general, Web Vulnerability Scanners (WVSs) perform black-box testing for detection of vulnerabilities, where an application is tested by analysis of its behavior at runtime [1, 2, 8 - 15]. These scanners provide mechanisms for analyzing response of web application in order to reveal vulnerabilities. [1, 2]. Scanners do provide report about detected security vulnerabilities to the developer [1, 8, 9, 16], though, code modifications for vulnerabilities removal is mostly done manually by developers. This is liable to human errors and limitations [3, 6, 7, 17].

Several WVSs are available from industry, research communities and open-source communities. Some scanners target detection of wide range of web vulnerabilities, for example, AppScan from IBM and WebInspect from HP are capable of detecting wide range of vulnerabilities including SQL injection, Cross-site scripting, Buffer overflow, OS Command injection, and XPath injection. Other scanners are designed to focus on detection of specific type of vulnerability, for example, Wasapy web scanner targets SQLIV [1]. Table 1. shows some existing web vulnerability scanners that are commercial scanners, open-source scanners, or reported in research articles. The table also shows capacity of scanners at detection and removal of SQLIVs. Existing WVSs follow two major approaches for vulnerabilities detection, namely. Pattern matching approach, and HTML Page Similarity approach.

2.1. Pattern Matching Approach

The idea behind this approach is to send HTTP request containing imbedded keywords (i.e., patterns) to a web application. Obviously, if the application did not sanitize (or validate) the request, then it would returns corresponding HTTP response page containing the imbedded keywords [8, 9]. Another possibility is that such application might return HTTP response containing SQL error message generated by database server [16]. In both scenarios, when an attack request is sent to an application, then presence of imbedded keywords or SQL error patterns in corresponding response page, could serve as evidence that the application is vulnerable [1]. For detection of SQLIVs, techniques that follow this approach send specially crafted attacks to web application. Thereafter, responses are collected and analyzed to check for specified keywords-pattern or error-pattern [1]. Existing WVSs that follow this approach include W3af (http://w3af.sourceforge.net), Wapiti (http://wapiti.sourceforge.net), Secubat [16], and the scanners proposed in [8, 9]. Error pattern matching strategy was adopted in W3af and SecuBat [16]. In W3af, the sqli module sends three requests based on the SQL injection d'z"0 (or d%2Cz%220 encoded in ASCII), and analyze the three corresponding responses. An application is reported as vulnerable if SQL error messages is found in any response. Similarly, SecuBat uses single quote character to craft and launch attacks targeting injection points, and then analyze responses for patterns that matches known database error

Table 1: Some Existing Web Vulnerabilities Scanners

Company/ Reference	Tool Name	SQLIVs Detection	SQLIVs Removal
*Commercial Scanners			
Acunetix	WVS	Yes	No
HP	WebInspect	Yes	No
IBM	AppScan	Yes	No
N-Stalker	QA Edition	Yes	No
Qualys	QualysGuard	Yes	No
Cenzic	HailStorm	Yes	No
PortSwigger	Burp Suite (1.6.18)	Yes	No
NTObjectives	NTOSpider	Yes	No
MileScan	ParasPro	Yes	No
	Powerfuzzer	Yes	No
NetSparker	NetSparker	Yes	No
*Open Source Scanners			
Nicolas Surribas	Wapiti (2.3.0)	Yes	No
Michal Zalewski	Skipfish	Yes	No
Andres Riancho	W3Af	Yes	No
Marcin Kozlowski	Powerfuzzer	Yes	No
David Byrne	Grendel-Scan	Yes	No
Reported in Research Article			
[1]	Wasapy	Yes	No
[16]	SecuBat	Yes	No
[8]	Not mentioned	Yes	No
[9]	Not mentioned	Yes	No
[18]	WAVES	Yes	No
[21]	BIOFUZZ	Yes	No

*Source: Deepa & Thilagam [2]

messages. Keywords pattern matching strategy was adopted in scanners proposed by Patil, et al. [8], and Chen & Wu [9]. In [9], the technique use rules set to craft and launch attacks targeting injection points, and then analyze response for patterns that matches imbedded keywords.

2.2. HTML Page Similarity Approach

The idea behind this approach is that when many diverse HTTP requests are sent to web application, then it is possible to classify the response pages into distinct clusters based on pages similarities. The principle is based on three basic assumptions as reported in [1].

1. Execution and rejection pages are dissimilar,
2. Requests which generate rejection pages are easy to build (e.g., through generation of random or syntactically invalid requests), and
3. It is not easy to build successful injection attacks that generate execution pages.

For detection of SQLIVs, scanner crafts and launches several attacks to web application. Responses are then analyzed and categorized into clusters based on textual similarity distance. Finally, SQLIVs are detected by analyzing attributes of the clusters [1]. Scanners such as Wasapy, and Skipfish (http://code.google.com/p/skipfish) follow this approach. Wasapy uses Levenshtein textual similarity distance measure to group responses into three clusters. Skipfish evaluates similarity distance based on frequency of words in the response pages[1]. In addition, Huang, et al., [18] proposed WAVES scanner based on similarity approach. However, their algorithm incorporated error pattern matching strategy as first step that guide classification, and then employs similarity approach to address uncertainty which arises when an injection does not generate error message.

2.3. Discussion

The various WVSs presented above have achieved considerable results in testing of web application for detection of vulnerabilities. However, thorough literature investigation reveals their common weaknesses such as partial coverage of vulnerabilities detection, reporting of false negatives and false positives [12, 15]. Coverage becomes an issue where the crawling stage of scanner is unable to find all web pages [9]. In scanners that use attack patterns [8] or rules sets [9] to craft attacks, the accuracy and effectiveness of vulnerability detection relies heavily on the quality and adequacy of attack patterns or rules used. Techniques based on error pattern matching could face uncertainty of vulnerability detection where an injection does not generate error message [1]. Additionally, error message found in HTML response page may not necessarily have come from database server. The aforementioned scenarios could possibly lead to false negative or false positive. Techniques that follow similarity approach requires wide coverage of different types of response pages that could possibly be generated by an application. Thus, emphasizing the need to craft and launch large number of attacks. Unfortunately, most existing scanners following this approach generate too few requests. For example, Skipfish and the scanner proposed in [8] use only 3 requests.

Generally WVSs conduct black-box testing, often called penetration testing, where an application is tested dynamically by subjecting it to attacks, and its responses analyzed. The actual code of the application is not accessed during the testing [18]. Consequently, this kind of testing cannot support actual code modification for vulnerabilities removal. The weaknesses of WVSs highlighted above and the need to support code modification for automating vulnerabilities removal, serves as motivation for finding white-box testing approach, specifically, static source code analysis, very attractive. In white-box testing approach, the actual code of an application is accessed, and analyzed for vulnerabilities detection, and could be modified for vulnerabilities removal.

3. Generic Workflow of Web Vulnerability Scanner

In general, detection of vulnerabilities using an automatic WVS involves generation of HTTP request, sending requests to web application, and analysis of response [11]. Several research works analyzed performance of existing WVSs, and consequently, revealed that the workflow of most existing WVSs consist of three steps as listed below [12, 13, 19, 20].

1. Crawl: Navigates URL of target web application to discover available web forms and injection points.
2. Attack: Crafts and launches several attacks targeting all discovered injection points.
3. Analyze: Analyze responses returned by web application, so as to ascertain vulnerabilities

Further investigation of related empirical studies listed in Table 2. reveals that the above three-steps workflow of "Crawl, Attack, and Analyze" does not cover some important steps involved in the working of a typical WVS. Hence, for the purpose of clarity and completeness, generic workflow of WVS can be refined into a

Table 2: Vulnerabilities Detection Approaches of Some WVSs

Reference	Product/Tool name	SQLIV Detection Approach
[8]	Not mentioned	Pattern Matching & HTML page similarity
[18]	WAVES	Pattern Matching & HTML page similarity
[9]	Not mentioned	Pattern Matching
[1]	Wasapy	HTML Page Similarity
[16]	SecuBat	Pattern Matching
[17]	BIOFUZZ	Pattern Matching

seven steps process. This is because, the working of a typical WVS begins with collection of target URL as input, and ends with generation of output report. Tthese kinds of important steps need to be reflected in the generic workflow of WVS. Consequently, we propose seven steps workflow as listed below.

1. Get initial target URL
2. Crawl website to discover injection points
3. Craft attacks targeting injection points
4. Launch attacks
5. Collect responses
6. Analyze responses
7. Report vulnerabilities found

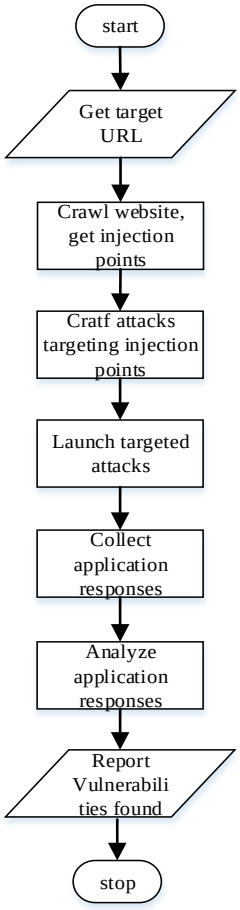

Fig 1: Generic Workflow of Web Vulnerability Scanner.

The seven steps workflow is diagrammatically depicted using flowchart of Fig. 1. The flowchart shows chain of processing activities performed by WVS during testing of web application for detection of vulnerabilities. Further literature investigation has shown some important facts about WVSs.

First, the initial steps (i.e., steps i, ii and iii) are for information gathering and generation of inputs for subsequent steps. Scanner collects input of target url. Then it crawls website to gather information about injection points. The information is used to craft attacks. Most existing scanners use site crawling for information gathering, though, few scanners use combinatorial crawling for improving website coverage [1]. Existing scanners employ different strategies for crafting attacks targeting injection points. For example, [1, 8, 16] uses attack patterns, [9] uses rules set, whereas [21] uses GP.

Second, the middle steps (i.e., steps iv, v and vi) are for attacks launching, response analysis and vulnerabilities detection. To launch attacks, some scanner use replay strategy [21], whereas others send http requests containing malicious inputs [9,18]. In response analysis, scanners follow a number of approaches, including pattern matching [9,15], HTML page similarity [1] and combination of pattern matching and HTML page similarity [8].

Third, the last step (i.e., step vii), is for providing vulnerabilities detection report to developer or tester. The report guides developer as to the vulnerabilities found, and possibly names of files and corresponding location where vulnerability are found.

4. Proposed EPSQLiFix Method for Detection and Removal of Web Vulnerability

Generally, black-box WVSs fail at removal of vulnerabilities, because vulnerability removal requires modifying source code. Unfortunately, to secure vulnerable web application, vulnerabilities need to be removed by actual code modification. Empirical evidences reveal that static analysis techniques are capable of addressing SQLIVs detection and removal. However, existing detection techniques do not consider grammatical linkages among program statements into detection process, as such, they report many false positives [25, 26]. On the other hand, existing removal techniques do not perform fully automatic vulnerabilities removal. Instead, they only generate fix, and allows developer to manually apply the auto-generated fix to the vulnerable source code for SQLIVs removal [5, 17]. Apparently, such techniques left the very important, yet very challenging task of SQLIVs removal predominantly manual, regardless of the fact that manual bug fixing is prone to errors and human limitations [27, 28]. These issues suggest the need for new method for automatic detection and removal of SQL injection vulnerabilities for web application.

Consequently, this section presents an overview of the proposed Evolutionary Programming (EP) based static analysis method for automatic detection and removal of SQLIVs, called EPSQLiFix. The method consists of four components, namely, grammar rules extractor component, EP search component, SQLIVs detector component, and SQLIVs remover component. The processes performed by the components, and sequence of interactions between them is diagrammatically represented using activity diagram of Fig. 2, and briefly explained below.

As shown in Fig. 2, the working of EPSQLiFix starts in a grammar rules extractor component. The component uses string analysis to recognize all declaration and assignment statements from source code, and then extracts Context Free Grammar (CFG) production rules for each statements. The extracted CFG production rules serve as input to EP search component which performs EP reachability search to evolve candidates, which are represented as productions sequences, for finding reachability from SS (Sensitive Sink) statements to AEP (Application's Entry Point) statements. Optimal solutions are collected and analyzed by SQLIVs detector component. The component performs grammar reachability analysis for detection of vulnerabilities. The SQLIVs remover compo-

nent implements EP search variation operations that performs source code modification for SQLIVs removal. Finally, EPSQLiFix produce modified version of web application containing insertions of auto generated data validation for fixing SQLIVs.

5. Comparison of EPSQLiFix Method and Web Vulnerability Scanners

This section discuss the underlying techniques of vulnerabilities detection and removal employed by Web Vulnerabilities Scanner (WVS) in comparison with the underlying techniques behind the proposed method, EPSQLiFix.

The key goal is to highlight differences between the two. Flowchart diagram showing generic workflow of WVS, and flowchart diagram showing workflow of the proposed method are shown side-by-side in Fig. 3. For ease of comparison, we map steps of the flowcharts into three stages, namely: 1) preparatory stage, 2) Vulnerabilities detection and removal stage, and 3) reporting stage. These three stages are shown in Fig. 2 by applying different colours to flowchart steps belonging to each stage (See figure legend). The comparative analysis is presented according to these three stages.

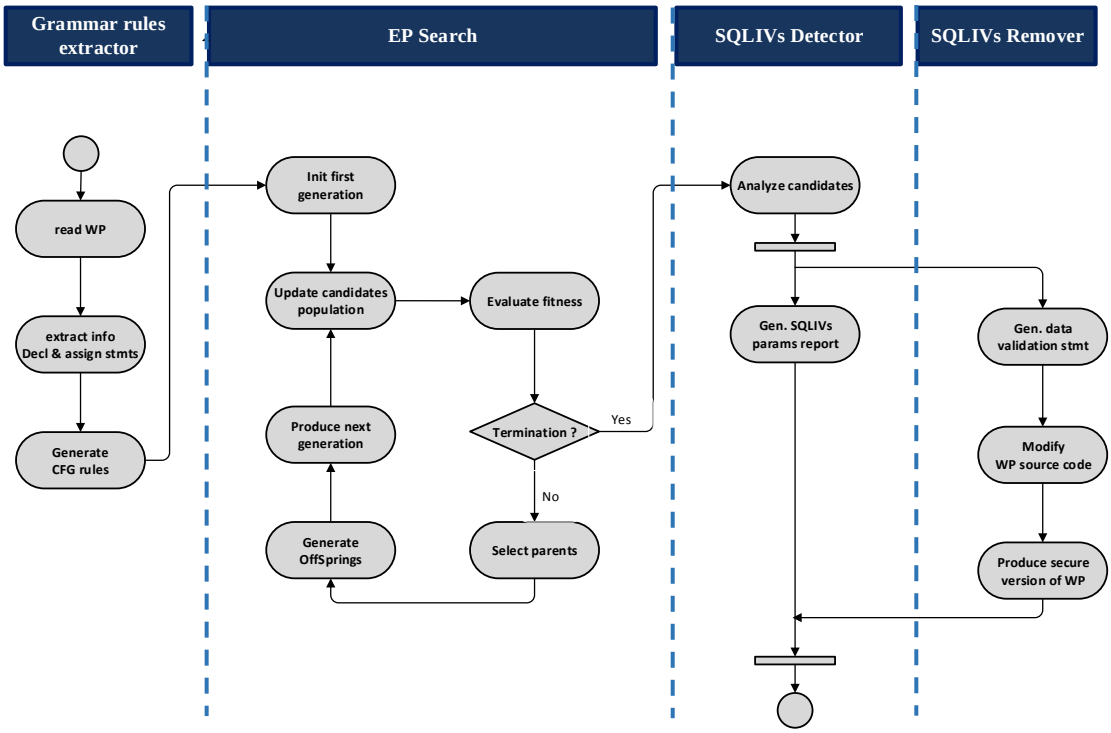

Fig 2: Activity Diagram for EPSQLiFix

5.1. Preparatory Stage

In both WVS and the EPSQLiFix method, the preparatory stage is basically for information gathering and getting inputs for tasks of stage two. The strategy followed by WVS is very different from that followed by EPSQLiFix. The working of WVS, generally, begins with input of initial url, and then crawling of website, starting from initial url, to discover other webpages of application [2, 8, 9]. Some WVSs use spider crawling strategy [9], whereas others use combinatorial site crawling strategy [1]. Webpages are downloaded and analyzed to find forms and injection points [9]. The information gathered during crawling is used to craft attacks which target all discovered injection points. WVSs suffer problem of partial coverage of web site during crawling, which happens when crawler could not find some pages. This problem do affects vulnerability detection accuracy and lead to false negatives [9, 12, 15]. Use of inadequate attack patterns for crafting targeted attacks is another problem associated with WVSs. This could compromise vulnerabilities exploitation, and reduces vulnerability detection coverage.

In contrast, the preparatory stage of EPSQLifix begins with input of source code files of web application. Since all source code files are provided as input, then application coverage is guaranteed. EPSQLiFix uses parsing to extract grammar rules from source code of application. The grammar extraction is guided by source language grammar, therefore, no any attack patterns or rules set is required.

5.2. Vulnerabilities Detection and Removal Stage

This stage involves applying techniques for detecting vulnerabilities, as well as techniques for code modification aimed at vulnerabilities removal. In WVSs, this stage begins by launching targeted attacks. Thereafter, responses are analyzed for vulnerabilities. Most scanners launch attacks by sending http request containing malicious input [1, 2, 8, 9, 16], although few scanners use replay strategy [21]. In WVSs, responses are mostly collected using http proxy [1, 8, 9], or through database logger [21]. The responses are analyzed to verify vulnerabilities. The analysis approaches used in WVSs include error pattern matching [9], keywords pattern matching [8], HTML page similarity [1], and combination of error pattern matching and HTML page similarity [18]. Regardless of the response analysis approach, existing WVSs suffer noticeable weaknesses. For example, scanners based on error pattern matching face uncertainties about presence of vulnerabilities where an injection did not result in error message. Scanners based on HTML page similarity requires large number of attacks. As black-box testing tools, WVSs do not support code modification, and hence cannot achieve vulnerabilities removal.

In contrast, the vulnerabilities detection and removal stage of EPSQLiFix begins by finding grammar reachability productions sequences that shows data flow from Sensitive Sink statement to Application's Entry Point statement. The reachability productions sequences are then analyzed using grammar reachability analysis for detection of vulnerabilities. In EPSQLiFix, vulnerability is

found when no data validation function is applied along reachability path. It should be acknowledged that grammar reachability analysis has been successfully applied in static analysis detection of vulnerabilities [22, 23]. The vulnerabilities detection of EPSQLiFix is achieved by statically analyzing reachability paths for inclusion of data validation function, and does not require launching of attacks. Moreover, EPSQLiFix identifies parts of source code that contains vulnerabilities, and automatically generate appropriate data validation statements that could fix the vulnerabilities. EPSQLiFix employs Evolutionary Programming (EP) search operator to automate code modification which insert auto-generated data validation statements into source code, and consequently, produce modified and secure version of application.

5.3. Reporting Stage

This stage involves generation of report to developer or tester. Most scanners produce an on-screen vulnerabilities report at the end of the testing process [24]. The report gives information about application being tested, vulnerabilities detected, and names of vulnerable parameters. In contrast, EPSQLiFix also provides an on-screen vulnerabilities report at the end of testing process. However, report of EPSQLiFix provides information about both detection and removal of vulnerabilities. It contains information about application being tested, vulnerabilities found, vulnerable parameters and their corresponding locations, and name/location of folder containing modified version of application.

6. Conclusion

The paper presented comprehensive description on generic workflow of Web Vulnerability Scanner comprising seven steps of activities. The workflow is derived through synthesis and aggregation of knowledge. An Evolutionary Programming based static analysis method for automated detection and removal of SQLIVs, called EPSQLiFix, was proposed. The difference between WVSs and EPSQLiFix was highlighted. The proposed method is currently implemented in a software tool based on static source code analysis, and is capable of removing SQLIVs through source code modification using EP mutation operation. Details about EPSQLiFix method is in the process of being published.

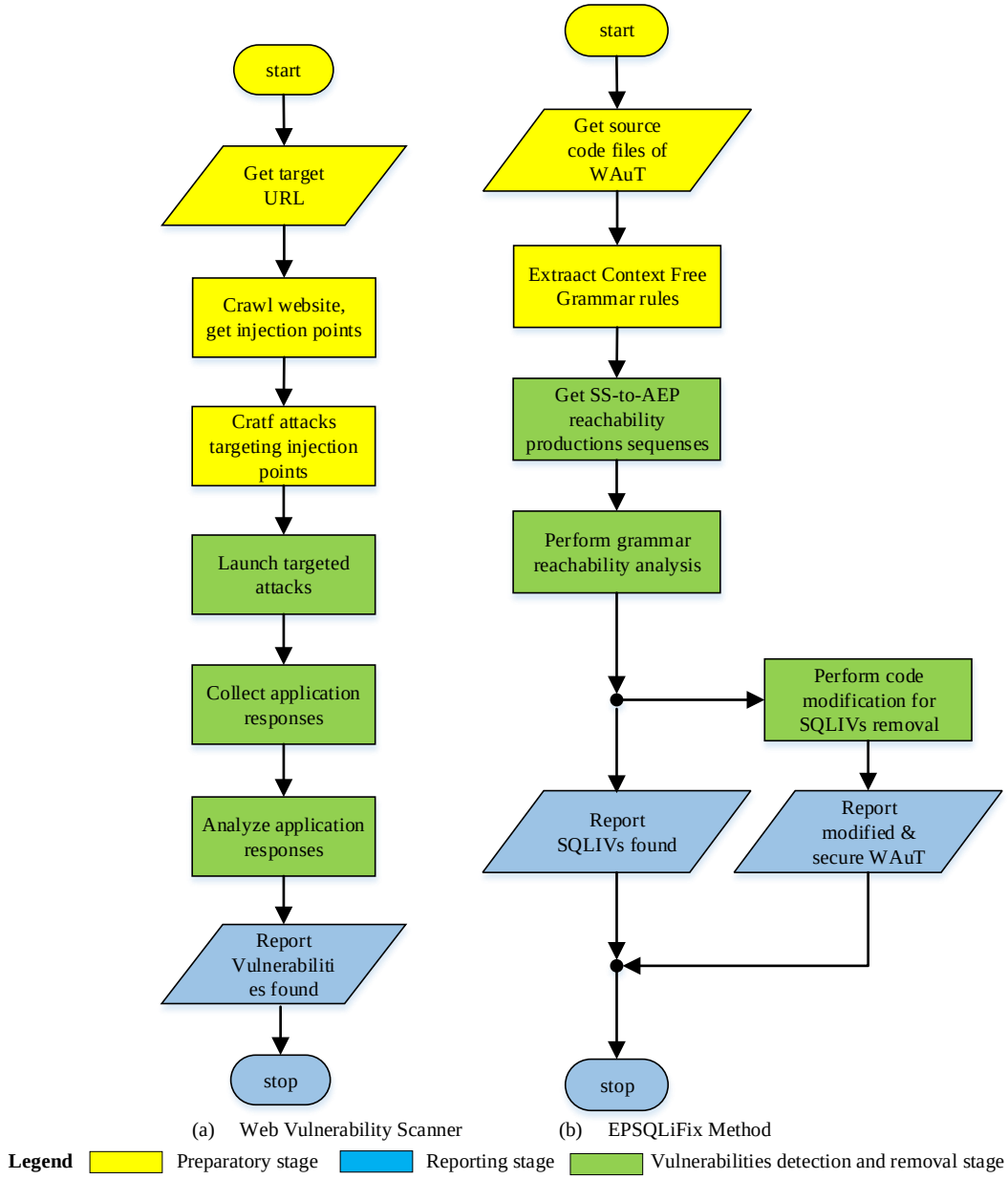

Fig 3: Flowchart Comparison between WVS and EPSQLiFix

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 Education to the Faculty of Computer Science and Information Technology at Universiti Putra Malaysia.

Reference

[1] Arcuri A (2011), Evolutionary repair of faulty software, *Journal of Applied Soft Computing, 11*, 4, 3494–3514. DOI: 10.1016/j.asoc.2011.01.023.

[2] Arcuri A (2008), On the automation of fixing software bugs, *In Proceedings of the 30th International Conference on Software Engineering*. Leipzig, Germany: ACM, pp. 1003-1006, DOI: 10.1145/1370175.1370223

[3] Halfond W & Orso A (2005), AMNESIA: Analysis and monitoring for neutralizing sql-injection attacks, In Pro*ceedings of the 20th International Conference on ASE*. Long Beach, CA: IEEE/ACM. pp.174-183

[4] Medeiros I, Neves NF & Correia M (2014), Automatic detection and correction of web application vulnerabilities using data mining to predict false positives, *Proceedings of the 23rd International Conference on World Wide Web*. New York: IEEE, 63-74, DOI: 10.1145/2566486.2568024.

[5] Kaur D & Parminder K (2017), SQLI Attacks: Current State and Mitigation in SDLC, *Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications*. Springer, Singapore.

[6] Sun F, Xu L & Su Z (2011), Static Detection of Access Control Vulnerabilities in Web Applications, *USENIX Security Symposium*.

[7] Minamide Y (2005), Static approximation of dynamically generated web pages, *Proceedings of the 14th international conference on World Wide Web*. ACM.

[8] Thomé J, Gorla A & Zeller A (2014), Search-based security testing of web applications, *Proceedings of the 7th International Workshop on Search-Based Software Testing*. ACM.

[9] Doupé A, Cova M & Vigna G (2010), Why Johnny can't pentest: An analysis of black-box web vulnerability scanners, *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, Berlin, Heidelberg.

[10] Bau J, Bursztein E, Gupta D & Mitchell J (2012), Vulnerability factors in new web applications: Audit tools, developer selection & languages, *Stanford, Tech. Rep*.

[11] Huang YW, Huang SK, Lin TP & Tsai CH (2002), Web application security assessment by fault injection and behavior monitoring, *Proceedings of the 12th international conference on World Wide Web*. ACM, pp. 148-159.

[12] Thomas S, Williams & Xie T (2009), On automated prepared statement generation to remove SQL injection vulnerabilities, *Information and Software Technology* 51.3, 589-598.

[13] Kals S, Kirda E, Kruegel C & Jovanovic N (2006), Secubat: a web vulnerability scanner, *Proceedings of the 15th international conference on World Wide Web*. ACM.

[14] Vieira M, Antunes N & Madeira H (2009), Using web security scanners to detect vulnerabilities in web services, *International Conference on Dependable Systems & Networks,DSN'09. IEEE/IFIP*. IEEE.

[15] Jose S, Priyadarshini K & Abirami K (2016), An Analysis of Black-Box Web Application Vulnerability Scanners in SQLi Detection, *Proceedings of the International Conference on Soft Computing Systems*. Springer, New Delhi, pp. 177-185.

[16] Bau J, Bursztein E, Gupta D & Mitchell J (2010), State of the art: Automated black-box web application vulnerability testing, *IEEE Symposium on Security and Privacy (SP)*.

[17] Fonseca J, Vieira M & Madeira H (2007), Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks, *13th Pacific Rim International Symposium on Dependable Computing,PRDC2007*, IEEE.

[18] Pałka D, Zachara M & Wójcik K (2016), Evolutionary scanner of web application vulnerabilities, *International Conference on Computer Networks*. Springer, Cham.

[19] Saleh AZM, Rozali NA, Buja AG, Jalil KA, Ali FHM & Rahman TFA (2015), A method for web application vulnerabilities detection by using boyer-moore string matching algorithm, *Procedia Computer Science* 72, 112-121.

[20] Chen JM & Wu CL (2010), An automated vulnerability scanner for injection attack based on injection point, *International Computer Symposium (ICS)*, IEEE.

[21] Patil S, Marathe N & Padiya P (2016), Design of efficient web vulnerability scanner, *Proceedings of International Conference on Inventive Computation Technologies (ICICT)*, 2. IEEE.

[22] Trivedi SH (2012), Software testing techniques, *International Journal of Advanced Research in Computer Science and Software Engineering* 2.10.

[23] Al-Khashab E, Al-Anzi FS & Salman AA (2011), PSIAQOP: preventing SQL injection attacks based on query optimization process, *Proceedings of the Second Kuwait Conference on e-Services and e-Systems*. ACM.

[24] Thomas S & Williams L (2007), Using automated fix generation to secure SQL statements, *Proceedings of the Third International Workshop on Software Engineering for Secure Systems*. IEEE Computer Society.

[25] Salas MIP & Eliane M (2015), A black-box approach to detect vulnerabilities in web services using penetration testing. *IEEE Latin America Transactions* 13, 3, 707-712.

[26] Deepa G & Thilagam PS (2016), Securing web applications from injection and logic vulnerabilities: Approaches and challenges. *Information and Software Technology* 74, 160-180.

[27] Irena J (2006), Software testing methods and techniques. *Journal of the IPSI BgD Transactions on Internet Research* 30.

[28] Akrout R, Alata E, Kaaniche M & Nicomette V (2014), An automated black box approach for web vulnerability identification and attack scenario generation. *Journal of the Brazilian Computer Society* 20.1: 4.