SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2017; **00**:1–16
DOI: 10.1002/sec

RESEARCH ARTICLE

Performance-Based Comparative Assessment of Open Source Web Vulnerability Scanners

Mansour Alsaleh[1]* Noura Alomar[2], Monirah Alshreef[2], Abdulrahman Alarifi[1], AbdulMalik Al-Salman[2]

[1]King Abdulaziz City for Science and Technology, Riyadh, KSA
[2]College of Computer and Information Sciences, King Saud University, Riyadh, KSA

ABSTRACT

The proliferation of malicious content on the web and the rapidly growing demand for defending web-based systems motivated quality assurance practitioners to use cost-effective web penetration testing tools. Although the performance of leading commercial web vulnerability scanners has been benchmarked and validated, the effectiveness of open source dynamic testing tools has gained little attention in the literature. The high cost of commercial web vulnerability detection tools and the lack of sound evaluation of open source scanners contributed to a sharp decline in both static and dynamic testing of web content. This paper experimentally evaluates the features and the performance of a set of popular open source web vulnerability detection tools. By analyzing the crawling capabilities of the chosen scanners and identifying their performance shortcomings, we aim at providing a baseline for aiding designers of open source scanners in building effective tools. Several trusted security benchmarks were utilized to perform a quantitative comparative performance assessment of 4 open source web vulnerability scanners. As a case study, 140 unique web-based applications were also scanned using some selected scanners from those included in the comparative evaluation. While the observed vulnerability detection accuracy of the majority of the evaluated scanners is high, our findings identify some fundamental limitations in the crawling capabilities of these scanners. The scanners included in the conducted case study agreed that 12.86% of the scanned web-based systems were vulnerable to Structured Query Language (SQL) injection attacks whereas Cross-Site Scripting (XSS) vulnerabilities were discovered in 11.43%. After demonstrating the ways in which different performance properties of one scanner might correlate with each other and highlighting the inconsistencies between the results reported by different scanners, we emphasize on the importance of addressing the problem from a software engineering perspective and we provide recommendations for helping developers to improve their tools' capabilities. Copyright © 2017 John Wiley & Sons, Ltd.

KEYWORDS

Internet security; Web application scanner; Web vulnerability; Web security scanner; Dynamic security testing

***Correspondence**

Mansour Alsaleh, KACST. E-mail: maalsaleh@kacst.edu.sa

1. INTRODUCTION

The multi-layered architectures of web-based systems and their sophisticated interactions with different types of sub-systems increase the number of security flaws that can be exploited by attackers. As emphasized by the Open Web Security Application Project (OWASP) [1], attackers might follow various paths through the digital infrastructure of an organization to find security weaknesses that might cause severe consequences. According to Symantec [2], over 6500 web vulnerabilities were reported in 2014 and twenty percent of them were expected to have critical consequences. Kaspersky also reported that more than 93 million malicious URLs were detected in the first quarter of 2015 [3]. By 2018, web server vulnerabilities are also expected to be ranked as the first in the list of the most serious cybersecurity attacks [4].

Dynamic security testing and static code analysis are of the main approaches that are used for detecting web vulnerabilities. Software developers use web vulnerability scanners to automate the process of examining the security of their web-based applications and conduct large-scale security analyses on many different web applications [5]. While automated static code analysis techniques are more effective for detecting security defects (e.g., SQL vulnerabilities) [6, 7], the adoption of these techniques

Prepared using secauth.cls [Version: 2010/06/28 v2.00]

is limited compared to the widespread application of automated dynamic security testing tools. One of the reasons that contributed to limiting the adoption of static code analysis by software developers is that it cannot be performed without accessing the source code of the evaluated web-based application, which might not always be accessible [5].

There are many commercial tool-based web vulnerability detection approaches for identifying web security defects. IBM Security AppScan [8], HP webInspect [9], Acunetix Web Vulnerability Scanner [10] are all examples of commercial dynamic web vulnerability detection tools. Although these scanners are considered state-of-the-art and were proved to be effective in identifying and analyzing security vulnerabilities, their utilization is limited to the developers who can afford their high prices [11–14]. In addition, a large percentage of the web content is developed by either individual developers or small and medium-sized enterprises that do not follow a secure software development life cycle and thus pay more attention to delivering the functionality of their systems than dealing with system security complexities. Such web applications and systems are vulnerable to various attacks compromising their web servers' code integrity [15]. Due to the high impact of the propagation of security attacks on the web, there is a high demand for facilitating the vulnerability detection activities by providing effective, efficient and easy to use web vulnerability scanners. As an attempt to fulfill this demand, our study focuses on evaluating the performance of open source web vulnerability scanners and determining whether their cost-effectiveness negatively correlates with their detection capability and accuracy.

In this paper, we conduct a comprehensive evaluation of the performance of a set of open source dynamic web application security testing tools. These tools work by crawling or scanning all web pages within a domain and automating the security testing process by randomly simulating attacking scenarios [16]. Although web security scanners might be effective in identifying security flaws and malicious content, previous studies proved that there are inconsistencies between the results reported by different scanners [17–19]. Further, these tools vary in their performance characteristics, types of detected vulnerabilities and crawler coverage [17–19]. This in turn decreases the level of trustworthiness in web vulnerability scanners and increases the demand for conducting experiments that quantitatively evaluate their quality and accuracy.

Our study started by reviewing some of the currently available open source dynamic security testing tools and choosing four of them based on a predefined selection criteria. After performing a comparative evaluation of the performance of the four chosen scanners, two scanners were chosen to be included in a case study which utilized these tools to detect the security flaws in 140 unique web-based applications. Wapiti, Skipfish and two versions of Arachni open source scanners were studied in our comparative assessment whereas the experimental evaluation performed in our case study included Wapiti 2.3.0 and Arachni 0.4.3 only.

Of the four evaluated scanners, our results show that Skipfish 2.1 [20] is the fastest. In terms of crawler coverage; however, Arachni 1.0.2 [21] ranks first with a coverage percentage of 94%. We utilized the benchmark provided by the Web Application Vulnerability Scanner Evaluation Project (WAVSEP) [22] to evaluate the accuracy of the four scanners in question. Our findings reveal that the best SQL and XSS vulnerability detection rates were obtained by Arachni 1.0.2 which detected 100% of the SQL test cases and 87.67% of the XSS test cases. After scanning a sample of 140 websites using Arachni 0.4.3 and Wapiti 2.3.0 [23], we found that both scanners agreed that at least 21% of the assessed websites had high severity security vulnerabilities. Arachni 0.4.3 and Wapiti 2.3.0 also found that 12.86% of the evaluated websites were vulnerable to SQL attacks whereas 11.43% of them had XSS vulnerabilities. Given that the results of our comparative evaluation did not show significant performance differences (e.g., scanning speed) among the scanners while the results of the conducted case study revealed high level of disagreement between the reports generated by different scanners, we conclude that the inconsistencies between the reports generated by different scanners might not necessarily correlate with their performance properties. Further research is therefore needed to investigate the factors that could degrade the effectiveness of web vulnerability detection scanners and thus lead them to report inconsistent results.

Contributions. We frame the contributions of our work as follows.

1. Comparative Evaluation of Open Source Web Vulnerability Scanners. After reviewing currently available open source web vulnerability scanners, a comparative evaluation of the security features as well as the performance of four web vulnerability detection tools is conducted. Two versions of Arachni scanner as well as Wapiti and Skipfish scanners are evaluated in terms of speed, crawler coverage, detection accuracy and extra supported features. The conducted evaluation is based on rigorous performance indicators as well as standard security benchmarks.

2. Case Study on the Security State of Web-Based Systems. As a second step for evaluating the performance of web vulnerability detection tools, a comprehensive web vulnerability assessment on 140 web-based applications is conducted. Two selected tools were utilized to identify the types, quantities and degrees of severity of security vulnerabilities that exist in the evaluated web content. The results reported by these scanners are then analyzed, aggregated, compared and linked to the results obtained from our

DOI: 10.1002/sec

scanners' comparative evaluation experiment to clearly understand the detection accuracy of these tools in various scenarios.

3. DESIGN GUIDELINES. We present some recommendations for helping developers of web vulnerabilities scanners to improve their tools' capabilities.

Organization. We first describe the steps followed to review, compare and select open source web vulnerability scanners in Section 2. We then present the results of the conducted comparative evaluation in Section 3. In Section 4, we use two web vulnerability scanners to detect the security vulnerabilities in a large collected dataset. Further discussion, design guidelines for tool developers and the related literature are then presented in Section 5, Section 6 and Section 7, respectively. We present the limitations and and future work in Section 8 and conclude in Section 9.

2. WEB VULNERABILITY SCANNERS: EVALUATION APPROACH

We followed a structured research methodology that focuses on addressing the problem from multiple perspectives. We started by reviewing some popular open source web vulnerability scanning tools based on a predefined criteria. We then randomly chose a set of web-based applications and scanned them using some of the surveyed open source scanners. Three of them were then selected to be included in our detailed open source web vulnerability scanners comparative evaluation. Fig. 1 summarizes the web vulnerability scanners and security benchmarks utilized throughout each phase of our evaluation approach. The following subsections further explain the decisions taken throughout the phases of the comparative assessment.

2.1. Web Vulnerability Scanners Selection

The first phase of our study involved conducting an initial survey of the most popular open-source web vulnerability assessment tools. Six scanners were evaluated against a predefined criteria that considers the performance properties of the scanners as well as their crawler coverage and types of web vulnerabilities they can identify (see Table I). The open source web vulnerability assessment scanners that were included in our survey are Skipfish, Arachni, Wapiti, IronWASP, Vega [24] and w3af [25].

For each scanner, we investigated the provided security scanning and visualization features, the supported operating systems, and the scanning scope. In order to evaluate these scanners and include the most effective ones in our comparative evaluation, we chose the best four scanners and tested them on some randomly selected websites. These scanners are Wapiti, Arachni, Skipfish and IronWASP. In the testing phase, Wapiti, Arachni and Skipfish were installed on Ubuntu 12.04 whereas IronWASP [26] was installed on Windows 8.

Based on the scanning results obtained in the testing phase, two web vulnerability scanners were chosen to be included in the conducted comparative assessment: Wapiti 2.3.0 and Arachni 0.4.3. In fact, after comparing the results reported by the four scanners, we found inconsistencies between the types and quantities of security vulnerabilities reported by IronWASP and the other three scanners. Given this and the limited reporting features of IronWASP, we did exclude IronWASP scanner. Skipfish was also not included in the security evaluation conducted in our case study. However, we included Skipfish in our comparative evaluation of web vulnerability detection tools in order to compare its speed with those of Wapiti and Arachni scanners.

2.2. Web Vulnerability Scanners Comparison

As a pre-step to evaluating the security of a sample of 140 distinct web-based applications, we conducted a comparative evaluation of four web vulnerability detection tools which are: Arachni 0.4.3, Arachni 1.0.2, Wapiti 2.3.0 and Skipfish 2.10. The aim of this evaluation was to assess the effectiveness of these scanners and decide whether their reported results will accurately reflect the security state of the websites examined in our case study. We evaluated the scanning speed of each tool, their crawler depth and their vulnerability detection accuracy. Our evaluation criteria also considered the extra features supported by each scanner (e.g., reporting and visualization services) and the types of web vulnerabilities that can be discovered by each scanner. Table I summarizes the criteria that was defined to evaluate the chosen web scanners, which covers the essential performance characteristics that allowed us to base our evaluation on rigorous standards and benchmarks.

Three vulnerability test websites were considered for evaluating the speed of each scanner: Web Scanner Test Site [27] and two Acunetix [28] sites. These test websites were fully scanned using the selected four web vulnerability scanners. We then compared the scanning times of the scanners and ranked them accordingly. Similarly, Altoro Mutual [29] and WAVSEP [22] were utilized for assessing the accuracy of the scanners. We also used the Web Input Vector Extractor Teaser (WIVET) [30] benchmarking project to examine the crawling capabilities of the evaluated scanners. To assess the accuracy of each scanner, we calculated: (1) the True Positive Rates (TPRs); (2) the True Negative Rates (TNRs); (3) the False Positive Rates (FPRs); (4) the False Negative Rate (FNRs); (5) the Positive Predictive Values (PPVs); (6) the Negative Predictive Values (NPVs); (7) and the False Omission Rates (FORs) [31, 32]. We also measured the vulnerability detection accuracy of the evaluated scanners and their associated f-measures.

DOI: 10.1002/sec

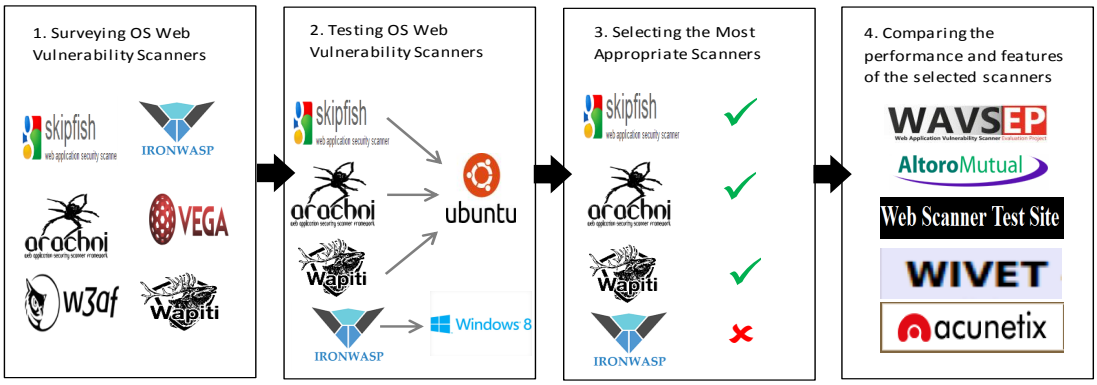

Figure 1. Web Vulnerability Scanners' Comparison: Research Approach

Scanners' Selection Criteria			Scanning speed Visualization features Scanning scope Export file formats Supported operating systems Consistency with other scanners Supported programming languages Availability of web-based GUI
Scanners' Evaluation Criteria	Performance	Quantitative measures	True Positive Rate (TPR) True Negative Rate (TNR) False Positive Rate (FPR) False Negative Rate (FNR) Positive Predictive Values (PPV) Negative Predictive Values(NPV) False Omission Rate (FOR) Accuracy F-measure
			Scanning speed Crawler Coverage Vulnerability detection accuracy
	Features		Visualization features Reporting features Ease of configuration Types of vulnerabilities that can be detected

Table I. Web Vulnerability Scanners' Comparison: Scanners' Selection and Evaluation Criteria

Scanner	Version	webscantest (HH:MM:SS)	testphp (HH:MM:SS)	testaspnet (HH:MM:SS)	Rank (speed)	Crawler coverage	Rank (Crawler coverage)
Arachni	0.4.3	01:23:22	00:22:10	01:05:33	2	19%	4
Arachni	1.0.2	12:11:13	00:41:34	01:00:00	3	94%	1
Wapiti	2.3.0	06:20:00	01:30:00	03:30:00	4	44%	3
Skipfish	2.1	00:11:06	00:27:40	00:11:30	1	48%	2

Table II. The speed and crawler depth results obtained after scanning the Web Scanner Test Site and the two Acunetix sites

Scanner	Version	WAVSEP test cases				IBM AppScan Test Site (Altoro Mutual) test cases		
		SQL (135)	SQL FP (10)	XSS (73)	XSS FP (7)	SQL (13)	Blind SQL (1)	XSS (10)
Arachni	0.4.3	134	2	47	0	9	1	8
Arachni	1.0.2	135	0	64	0	2	0	4
Wapiti	2.3.0	131	2	45	3	5	0	7
Skipfish	2.1	104	0	63	0	1	0	3

Table III. WAVSEP and Altoro Mutual: Test Cases Detected by Each Scanner

Security Comm. Networks 2017; **00**:1–16
DOI: 10.1002/sec

3. SCANNERS' COMPARATIVE EVALUATION: RESULTS AND ANALYSIS

After using Arachni 0.4.3, Arachni 1.0.2, Wapiti 2.3.0 and Skipfish 2.10 to scan the chosen test websites, we observed the scanning behaviors of the evaluated scanners and quantitatively compared their speed, detection accuracy and crawler coverage. We then ranked these scanners accordingly and chose Arachni 0.4.3 and Wapiti 2.3.0 to be included in our case study (as explained in Section 4). In this section, we present and discuss the results of our web vulnerability scanners comparative assessment.

In general, our results show that the speed of the evaluated scanners was relatively acceptable. However, in terms of crawler coverage, the obtained WIVET scores for the four scanners show that 75% of them covered less than 50% of the crawled web-based systems. By considering the detection accuracy of the evaluated scanners, we found variations between the scanning results obtained from Altoro Murual and WAVSEP. For example, all the scanners detected at least 77% of WAVSEP's SQL test cases whereas 75% of them detected at most 38% of Altoro Mutual's SQL test cases. However, this might be due to differences between the security properties examined by WAVSEP's and Altoro Mutual's test cases. We also observed variations in number and types of web vulnerabilities that each scanner can detect. However, all the chosen web scanners were able to discover basic security defects including database injection attacks, XSS vulnerabilities and misconfiguration attacks. Further, no difficulties were faced in understanding the functionality of the evaluated scanners, as all of them provide sufficient documentation in their corresponding websites. The following subsections detail the results of the conducted comparative assessment.

3.1. Speed

After scanning three test sites and recording the elapsed scanning times (see Section 2.2), we observed that Skipfish took at most 28 minutes to scan a test website whereas Wapiti took at least an hour and a half to perform a full vulnerability scan. Similarly, the vulnerability assessments conducted using Arachni scanner consumed at least 23 minutes. Thus, Skipfish is considered the fastest scanner whereas Wapiti is the slowest one in detecting web based vulnerabilities. Table II shows the scanning times for each evaluated scanner.

3.2. Crawler Coverage

We based our evaluation of the crawler coverage of the evaluated scanners on WIVET scores. The best crawling coverage percentage was 94% which was obtained by Arachni 1.0.2. The crawlers of the remaining three scanners covered at most 48% of the scanned web applications. As shown in Table II, WIVET results show that Skipfish 2.1, Wapiti 2.3.0 and Arachni 0.4.3 covered 48%, 44% and 19% of the crawled web applications, respectively.

By linking the crawling coverage with the speed of the evaluated scanners, note that the excellent crawling ability of Arachni 1.0.2 could had negatively affected its speed. After comparing the crawler coverage of Arachni 0.4.3 and Arachni 1.0.2, we noted that the newer release was slower due to the significant improvement of its crawler. However, this does not necessarily imply that the advanced crawling capability of a given web vulnerability scanner is correlated with its poor performance. For example, although Skipfish was the fastest scanner, the results show that the coverage of its crawler was better than the coverage of both Wapiti 2.3.0 and Arachni 0.4.3.

3.3. Accuracy

We scanned IBM's AppScan test site (Altoro Mutual) to evaluate the level of accuracy of the scanning results reported by the chosen tools. We also used 225 of the test cases included in the Web Application Vulnerability Scanner Evaluation Project (WAVSEP) to clearly understand the accuracy weaknesses of the four evaluated scanners (see Table III). Altoro Mutual site helped us to test our scanners against 13 SQL test cases, one blind SQL test case and ten XSS test cases. Similarly, we scanned the cases presented by WAVSEP benchmark which included 135 SQL true positive cases, 10 SQL false positive cases, 73 XSS true positive cases and 7 XSS false positive cases. Based on WAVSEP benchmark, our results show that the four scanners were able to detect at least 104 SQL injection true positives and 45 XSS true positive cases. However, the majority of Altoro Mutual's SQL cases were not detected by Wapiti 2.0.3, Arachni 1.0.2 and Skipfish 2.1. Similarly, all the evaluated scanners were able to report at least 62% of WAVSEP's XSS true positive test cases whereas 50% of the evaluated scanners detected at most 40% Altoro Mutual XSS vulnerabilities. For WAVSEP's SQL and XSS false positive test cases, the four scanners reported at most 20% SQL cases and 43% XSS cases. Table III details the number of cases detected by each scanner and Section 5 discusses these observations.

3.4. Additional Features

To comprehensively evaluate the chosen web vulnerability scanners, we manually experimented with all the features offered by these tools and utilized the extra information documented in their corresponding websites. While we did not conduct a user study, we did not observe any notable problems with the graphical user interfaces of these scanners as we found them user friendly, easy to use and understandable. Further, the tools provided live feedback which helped us to easily identify the vulnerable URLs and understand the types of detected vulnerabilities and their degrees of severity. Furthermore, we were satisfied with the vulnerability reporting features of these tools as all of them provided reports that included detailed statistics and

Scanner	Version	web-based Interface	Features	Live Feedback	Detailed Reports	User Friendliness	Configuration
Arachni	0.4.3	Yes	More than 10 active and 20 passive checks	Yes	Yes	Easy	Flexible
Arachni	1.0.2	Yes	More than 10 active and 20 passive checks	Yes	Yes	Easy	Flexible
Wapiti	2.3.0	No	10 modules for 9 vulnerability types	Yes	Yes	Easy	Basic
Skipfish	2.1	No	Different scanning modes and dictionaries	No	Yes	Easy	Basic

Table IV. Vulnerability Scanners' Comparative Assessment: Extra Features

Scanner	Version	TPR	TNR	FPR	FNR	PPV	NPV	FOR	Accuracy	F-measure
Arachni	0.4.3	99.26%	80%	20%	0.74%	98.53%	88.89%	11.11%	97.93%	98.89%
Arachni	1.0.2	100%	100%	0%	0%	100%	100%	0%	100%	100%
Wapiti	2.3.0	97.04%	80%	20%	2.96%	98.50%	66.67%	33.33%	95.86%	97.76%
Skipfish	2.1	77.04%	100%	0%	22.96%	100%	24.39%	75.61%	78.62%	87.03%

Table V. Scanners' Detection Accuracy of SQL Attacks

Scanner	Version	TPR	TNR	FPR	FNR	PPV	NPV	FOR	Accuracy	F-measure
Arachni	0.4.3	64.38%	100%	0%	35.62%	100%	21.21%	78.79%	67.50%	78.33%
Arachni	1.0.2	87.67%	100%	0%	12.33%	100%	43.75%	56.25%	88.75%	93.44%
Wapiti	2.3.0	61.64%	57.14%	42.86%	38.36%	93.75%	12.50%	87.50%	61.25%	74.38%
Skipfish	2.1	86.30%	100%	0%	13.70%	100%	41.18%	58.82%	87.50%	92.65%

Table VI. Scanners' Detection Accuracy of XSS Attacks

explanations about the detected security vulnerabilities. Further, Arachni vulnerability assessment scanners offer extra features that allowed us to identify redundant URLs, classify the scanned URLs according to their security levels and modify the crawling scope to include or exclude some web pages.

The four scanners varied in the types and number of security checks that can be performed. For example, Arachni 0.4.3 and Arachni 1.0.2 have many plugins that can be used for performing more than 30 active and passive vulnerability checks for a wide range of web-based vulnerabilities. Similarly, nine different types of web vulnerabilities can be checked using Wapiti 2.3.0 scanner including SQL and XSS injection attacks, file disclosure attacks, and XPath attacks. In addition, Skipfish has an extra feature that helps in improving the crawling coverage and including unlinked web pages in the vulnerability scanning process. This feature performs brute-force based security checks and allows specifying the crawling scope as minimal, medium or complete. Table IV summarizes some of our scanners' additional features.

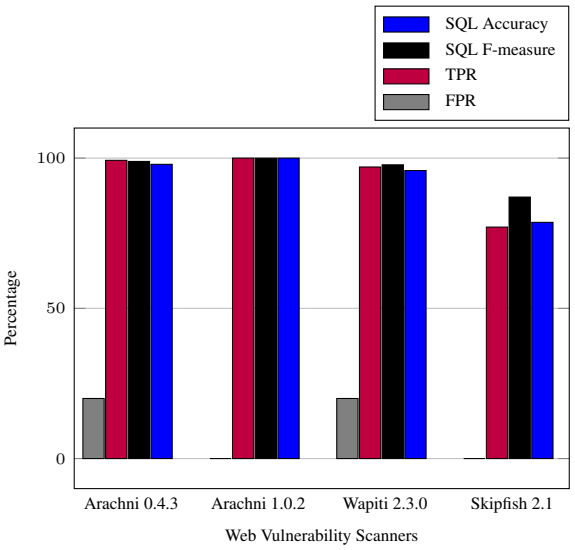

Figure 2. Scanners' Performance: Detection of SQL Vulnerabilities

3.5. Quantitative Measures

Based on WAVSEP test cases, nine metrics were calculated to review the effectiveness of the four scanners in detecting SQL and XSS attacks. For each scanner, the true positive and true negative rates were measured to determine whether the results reported by the evaluated scanners truly reflect the security state of the scanned websites. We also measured the rates of false positives and false negatives reported by the scanners as well as the positive predictive and negative predictive values for each scanner. To clearly understand the detection accuracy of SQL and XSS attacks, the rate of true positives and true negatives to the total number of test cases and the f-measures were calculated for each scanner (see Section 2.2 and Table I). The results of our analysis are presented in Tables V and VI.

Based on the calculated f-measures, the detection accuracy of SQL vulnerabilities was over 97% in Arachni 0.4.3 and Arachni 1.0.2 (see Fig. 2). Although Skipfish's SQL detection accuracy was the lowest, it has a false positive rate of 0%, a false negative rate of 23% and a true negative rate of 100%. Furthermore, by considering

DOI: 10.1002/sec

the accuracy of each test which is measured through dividing the sum of true positives and true negatives by the total number of test cases, the rates of Arachni and Wapiti scanners were higher than 95%. For SQL vulnerability detection, the accuracy measures of Arachni 1.0.2 were the best relative to the remaining three scanners. Further, Skipfish has a true positive rate of 77%, which was the minimum observed TPR. For SQL attacks, the positive predictive values of all the scanners were very high ranging from 98.50% to 100%. Also, the false negative rates of three of the evaluated scanners were less than 3%. After ranking the four scanners according to their SQL vulnerability detection rates, Arachni 1.0.2 took the first position whereas Arachni 0.4.3, Wapiti 2.3.0 and Skipfish 2.1 took the second, third and fourth positions, respectively.

The scanners' performance related to their effectiveness in detecting XSS vulnerabilities was also evaluated. Although Skipfish's accuracy rates for detecting SQL vulnerabilities were the lowest (see Fig. 2), our results show that its detection rate for XSS vulnerabilities was very high in comparison with the values recorded for Wapiti 2.3.0 and Arachni 0.4.3. For all the evaluated scanners, we also recorded a false positive rate of 0% except for Wapiti which had a FPR of 42.86%. Similarly, for all the four scanners except Wapiti, we also obtained true negative rates and positive predictive values of 100%. The lowest true positive rates, f-measures and accuracy values related to detecting XSS vulnerabilities were recorded for Wapiti 2.3.0 and Arachni 0.4.3 (see Fig. 3). While the highest FNR related to SQL vulnerability detection was recorded for Skipfish (see Table V), the number of XSS false negatives recorded for Skipfish was lower than the ones calculated for Arachni 0.4.3 and Wapiti 2.3.0. Similar to our observations for SQL vulnerability detection, the best accuracy measures related to the detection of XSS attacks were the ones recorded for Arachni 1.0.2. However, in contrast to our observations for SQL attacks, the results show that Skipfish 2.1 was more accurate than Wapiti 2.3.0 and Arachni 0.4.3 in detecting XSS vulnerabilities.

We observed that the newer version of Arachni was the most effective one in detecting SQL and XSS vulnerabilities. As presented in Tables V and VI, Arachni 1.0.2 has the highest true positive rate, true negative rate, accuracy value and f-measure. In most of the cases, we did not note significant differences in performance among all the evaluated scanners. Generally, the overall performance of the four scanners was good as the lowest obtained f-measure was 74.38% and the highest FNR was 38.36%. Further, the recorded true positive rates for most of the evaluated scanners are above 86% whereas the worst FPR is 42.86%. Based on the results obtained after scanning IBM AppScan test site (see Table III), the highest number of detected test cases was scored by Arachni. During our evaluation, scanning IBM AppScan using Arachni 1.0.2 unexpectedly stopped at the end of the experiment. The results shown in Table III are therefore according to the number of vulnerabilities detected before experiencing this technical problem. We also note that it was challenging to determine if the calculated false negative rates were actual false negatives or were resulted from some crawler coverage issues.

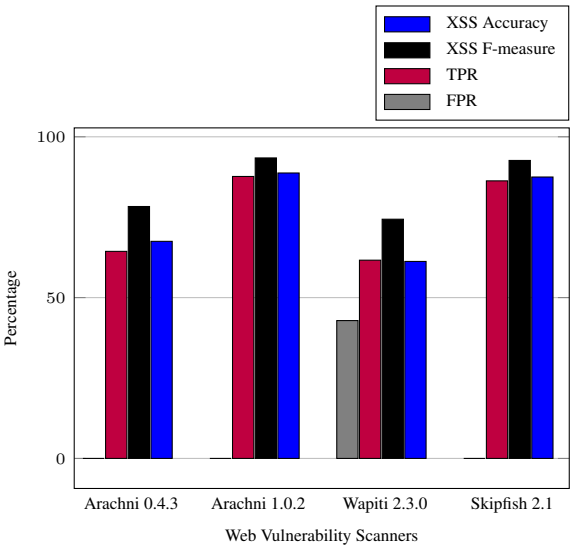

Figure 3. Scanners' Performance: Detection of XSS Vulnerabilities

4. CASE STUDY: EVALUATING THE SECURITY STATE OF PART OF THE WEB CONTENT

We followed our performance evaluation of the scanners (see Section 3) with an evaluation of the overall security state of part of the web content as a case study. Two scanners were chosen for this purpose: Wapiti 2.3.0 and Arachni 0.4.3. First, we collected a sample of 140 unique URLs to popular web-based systems and used the chosen scanners to assess the security state of the websites in question (the collected URL dataset is available in [33]). Based on the quantitative analysis of the security scan reports generated by Wapiti 2.3.0 and Arachni 0.4.3, a security holistic view of part of the web content was then analyzed. In the following sections, we describe the conducted security scans, analyze the generated scan reports and compare the results obtained from the two chosen scanners.

4.1. Dataset and Methodology

In this section, we describe the methodology followed to collect, scan and analyze a sample of unique 140 URLs. Fig. 4 summarizes the steps followed in our experiment. The following sections detail the decisions taken throughout each step in this case study and present the results of the conducted security vulnerabilities assessment.

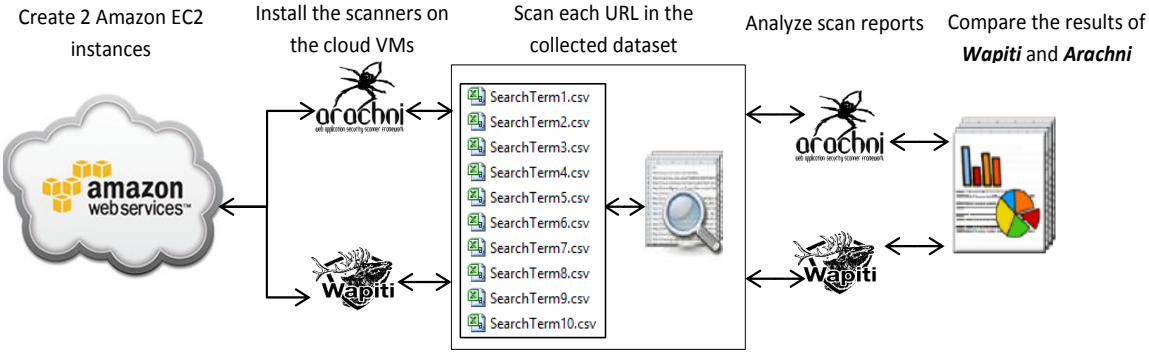

Figure 4. Arabic Websites Security Evaluation: The Followed Approach

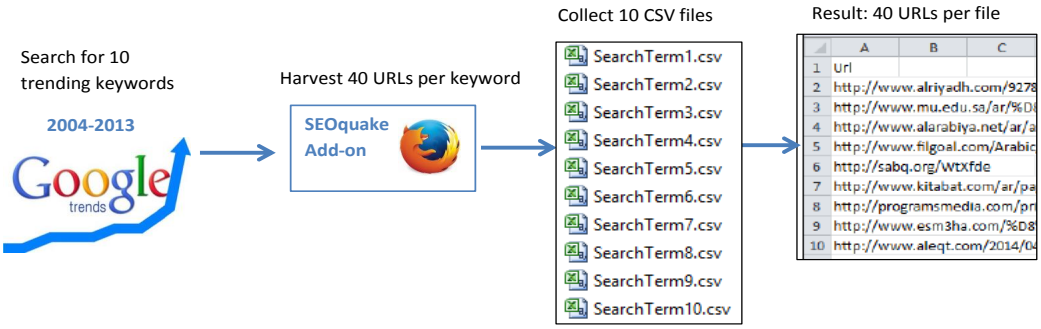

Figure 5. Collecting the Sample URLs

4.1.1. Dataset Collection

One of the main objectives of this case study is to evaluate the overall security state of a part of the Web content. The sample of the examined URLs includes different websites that vary in size, complexity and topics. We used Google Trends [34] to select trending search terms from the period of 2004 to 2013. This was followed by performing Google searches on the trending keywords and using SEOquake [35] Firefox extension to harvest the first thirty URLs obtained from each search (the collected URL dataset is available in [33]). Fig. 5 describes the followed steps to collect our dataset.

4.1.2. Methodology

To scan the collected URLs, two virtual machines were installed on Amazon EC2 [36] cloud-based service. The two instances were accessed using PuTTY [37] client and TightVNC [38] remote desktop controller. This was followed by installing Wapiti and Arachni web vulnerability scanners on the cloud instances and running the security vulnerabilities scans for each URL in our dataset. Note that Arachni was utilized to detect a wider range of security defects relative to Wapiti which was restricted to detecting SQL injection and XSS attacks.

We then analyzed and compared the results generated by each scanner to accurately understand the overall security state of part of the web content. Our analysis is based on the overall performance of each scanner as discussed in Section 3. For each website, we recorded the types of detected vulnerabilities by each scanner and their degree of severity. We were therefore able to identify the most common vulnerabilities in the examined URLs, rank them according to their degree of severity and then obtain an idea about the general security state of the evaluated URLs.

4.2. Detected Vulnerabilities

The findings of the conducted experiments and the level of agreement between the web vulnerability scanners evaluated in this study are discussed in this section.

4.2.1. Arachni

After scanning 140 unique URLs using Arachni, we found that 98% of them had security vulnerabilities. 60% of the vulnerable web servers included high-severity vulnerabilities and 96% had informational vulnerabilities. Arachni reports also show that 72% of the vulnerable web servers included low-severity vulnerabilities and 65% had medium-severity vulnerabilities.

Based on Arachni results, our sample had seven high-severity, four medium-severity, three low-severity and three informational web vulnerability types. Table VII summarizes the types, numbers and severity ratings of the web vulnerabilities that were detected by Arachni and ranks them according to their frequency of occurrence. According to Arachni, around 64% of the web server with high-severity vulnerabilities were also vulnerable to

Severity	Vulnerability	# of Vulnerable URLs
High	CSRF	53
	XSS	30
	SQL	21
	Source Code Disclosure	6
	Backdoor File	5
	File Inclusion	5
	X-Forward-For	1
Medium	Common Directory	68
	Unencrypted Passwords	43
	Backup Files	9
	Unvalidated Redirect	2
Low	Common Sensitive Files	77
	Password Autocomplete	42
	Directory Listing	4
Informational	Interesting Response	123
	Insecure Cookie	90
	Email Disclosure	45

Table VII. Arachni's Reported Vulnerabilities: A Summary

Vulnerability	Severity	Highest Number
Common Directory	Medium	54
XSS	High	33
SQL	High	29
Backup File	Medium	27
CSRF	High	26
PW Auto-complete	Low	25
Interesting Response	Informational	25
Backdoor File	High	24
Source Code Disclosure	High	18
File Inclusion	High	10
Common Sensitive Files	Low	10
Email Disclosure	Informational	8
Insecure Cookie	Informational	7
Unencrypted Password	Medium	5
Directory Listing	Low	1
X-Forward-For	High	1

Table VIII. Arachni: Highest Number of Occurrence of each Vulnerability Type

Vulnerability	Number of Vulnerable URLs	Highest Number
XSS	64	600
SQL	39	234

Table IX. Wapiti: Detection of SQL and XSS vulnerabilities

cross-site request forgery (CSRF) attack. Further, cross-site scripting vulnerabilities (XSS) were the second most frequently observed security issues in our sample (36% of the scanned URLs that had high-severity vulnerabilities). Arachni also reported that 15% of the sample were vulnerable to SQL injection attacks. Also, 4% of the evaluated websites had backdoor files and file inclusion vulnerabilities. Source code disclosure vulnerabilities were also observed with a percentage of 7% of our URLs that had high-severity vulnerabilities. The most spread high-severity vulnerabilities in the evaluated web-based applications were CSRF, XSS and SQL injection attacks.

As presented in Table VII, 90 of the scanned web servers were vulnerable to four types of medium-severity vulnerabilities. Among medium-severe vulnerabilities, at least 75% were vulnerable to the attacks related to the use of common directories on web servers. Arachni's results also showed that at least 30% of our evaluated websites did not implement proper encryption mechanisms (e.g., users' credentials were sent without encryption). Arachni also reported that 6% had stored their backup files in the same web servers. Furthermore, unvalidated redirect security defects were observed in 2% of the scanned URLs with medium-severity vulnerabilities. In short, 65% of the evaluated websites had medium-severity vulnerabilities that could have been caused by incorrect software development practices.

Three types of low-severity web vulnerabilities were detected by Arachni in at least 71% of the evaluated URLs. In at least 77% of these vulnerable web servers, Arachni was able to locate some files (e.g., robot.txt, sitemap.xml and config.php) that might reveal sensitive information about the web servers in question. For instance, attackers might use such files to view the names of some hidden directories or disclose information about the configurations of the respective servers. In 30% of the sample, Arachni was also able to find some form fields with password auto-completion option enabled. Similarly, Arachni reports show that a directory listing option was permitted in 4% of the URLs that included low-severity vulnerabilities.

Informational vulnerabilities were found in 95% of our sample in which more than 92% were related to receiving "Not Found" error message (i.e., 404 HTTP responses) from the requested web servers. Arachni also reported that over 67% of the evaluated URLs had informational vulnerabilities related to sending data using unencrypted cookies. In 32% of the collected URLs, Arachni was also able to discover some email disclosure vulnerabilities in which some email addresses that were written in some scripts or code comments were accessible by the scanner.

This experiment enabled us to identify the most spread web vulnerabilities and rank them accordingly. Without considering the severity degrees of the detected vulnerabilities, our results show that the average number of web vulnerabilities found in one web server was 21. For each web vulnerability type, we recorded the highest number of vulnerabilities found in one web server. For example, the highest number of SQL injection, XSS and CSRF vulnerabilities found in a website was 29, 33 and 26, respectively. For each vulnerability type, Table VIII ranks the number of web vulnerabilities detected in one web server based on their frequency of occurrence.

4.2.2. Wapiti

We used Wapiti to detect two types of web vulnerabilities: SQL injection and XSS attacks (see Fig. 7 for a sample scan result). Based on Wapiti results, 86 out

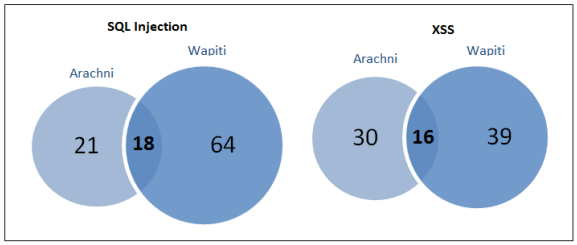

Figure 6. Wapiti vs Arachni: Detection of Vulnerable URLs

Figure 7. Wapiti Web Vulnerability Scanner: Sample scan result

of our collected 140 URLs had either SQL injection or XSS vulnerabilities. Around 92% of the 86 vulnerable web servers had high-severity SQL injection or XSS vulnerabilities. Wapiti also reported that at least 19% of our collected sample were vulnerable to low-severity risks. Wapiti's scan results also indicated that at least 81% of the URLs that had high-severity vulnerabilities were vulnerable to SQL injection attacks. Similarly, Wapiti detected high severity cross-side scripting vulnerabilities in 28% of the evaluated websites. The highest numbers of SQL injection and cross-side scripting vulnerabilities discovered in one web-based application were 600 and 234, respectively. Additionally, Wapiti found that the highest number of HTTP internal server error responses received from one web server is 759. Table IX summarizes the number of URLs that were reported as vulnerable to high severity XSS or SQL attacks as well as the highest numbers of SQL or XSS vulnerabilities that were found in one web server.

4.2.3. Arachni and Wapiti: A Comparison

After scanning the same sample using Arachni and Wapiti, we compared their results and quantified the number of URLs that were reported by the two scanners as vulnerable to SQL injection or XSS attacks. Based on the reports generated by Arachni, 15% of our collected sample were vulnerable to SQL injection attacks. However, the number of URLs that were reported as vulnerable to SQL injection attacks by Wapiti scanner was at least two times higher than the Arachni's one. According to Wapiti, around 46% had SQL injection vulnerabilities. Both scanners agreed that 12.86% of the evaluated sample was vulnerable to SQL injection attacks. On the other hand, the findings of our comparison also showed that 11.43% of the scanned URLs were reported as vulnerable to cross-side scripting attacks by both scanners. For XSS vulnerabilities, there was a little difference between those reported positive by the two scanners (i.e., Wapiti reported 27.86% relative to 21.43% reported by Arachni's). Both Wapiti and Arachni agreed that at least 12.86% were vulnerable to SQL injection attacks whereas 11.43% had XSS vulnerabilities (see Fig. 6).

5. FURTHER DISCUSSION

We combined the results of our first experiment, which compared the performance of four scanners based on objective measures, with the findings of the second experiment which evaluated the performance of two scanners on a large set of selected URLs. Before conducting our case study and based on the results obtained from our scanners' comparative evaluation, we expected Arachni to report less number of security vulnerabilities compared to Wapiti, as the crawler coverage of Arachni was 19% whereas Wapiti's crawler coverage was 44%. In our case study, we observed that the number of SQL and XSS vulnerabilities reported by Wapiti was higher than the results reported by Arachni. As shown in Table VII, Arachni's reports showed that 15% of our sample was vulnerable to high severity SQL attacks and 21% of the collected URLs were vulnerable to high-severity XSS attacks. According to Wapiti, around 46% of the URLs examined in our case study were vulnerable to SQL attacks and XSS vulnerabilities existed in 28% of our sample. However, this might not necessarily correlate with Wapiti's crawler coverage capabilities as our observations also showed that Wapiti had the highest FPR for XSS and SQL attacks (as shown in Tables V and VI). Additionally, the results showed that Arachni's vulnerability detection rates were higher than Wapiti's ones. Hence, because Arachni's crawler coverage was the lowest, there is a possibility that many web pages were not reached by Arachni's crawler.

Although there were little differences in f-measures and detection accuracy rates for SQL and XSS vulnerabilities between Wapiti and Arachni, we observed that Wapiti had higher false negative rates and lower true negative rates for both SQL and XSS attacks (see Tables V and VI). Thus, although Wapiti's crawler coverage rate was higher than Arachni's one, there is a possibility that Wapiti had not detected some vulnerable URLs that were examined in the conducted case study. For XSS attacks, we also observed that Wapiti 2.3.0 and Arachni 0.4.3 had the lowest true positive rates compared to Skipfish 2.1 and Arachni 1.0.2. The percentage of URLs that were vulnerable to XSS attacks might therefore be higher than those reported by the two scanners included in our case study. This finding might also be supported by the number of WAVSEP XSS true positive test cases that were detected by Arachni 0.4.3

DOI: 10.1002/sec

and Wapiti 2.3.0 (see Table III). Based on the comparative evaluation, both scanners had detected at most 47 XSS cases out of WAVSEP's 73 true positive XSS test cases. Further, although both scanners had high true positive rates for SQL vulnerabilities, Wapiti had detected only 5 out of 13 Altoro Mutual's SQL true positive cases whereas the number of cases detected by Arachni 0.4.3 was 9 (see Table III). Therefore, the number of URLs that were vulnerable to SQL attacks might also be higher than the numbers reported by Wapiti and Arachni.

We observed notable differences between the scan results of Wapiti and Arachni. Although Arachni's reports showed that 21 of the examined web servers had SQL vulnerabilities, Wapiti's scan results indicated that 64 of our 140 URLs were vulnerable to SQL attacks. However, only 18 URLs were reported as vulnerable to SQL attacks by the two scanners. For XSS vulnerabilities, we note that the two scanners agreed that 16 web servers were vulnerable although each scanner flagged at least 30 URLs as vulnerable to XSS attacks (see Fig. 6). Although the results of our scanners' comparative assessment did not reveal notable differences between the performance of the four evaluated scanners, there was a high level of disagreement between the scan results of Wapiti and Arachni. This suggests that web vulnerability detection scanners have limitations that might be related to their crawling capabilities, detection accuracy, or other implementation details that need to be researched. Our observations also show that the slow performance of the studied scanners might not necessarily imply that the respective scanner has a good crawling coverage. We observed that Wapiti's long scanning time was resulted from scanning web pages containing similar or duplicate content. This means that the there was a considerable increase in the scanning time that was resulted from sending a high number of unnecessary HTTP requests and repeatedly filling many form elements without guaranteeing the coverage of all web pages belonging to the websites in question.

By observing the highest numbers of XSS and SQL injection vulnerabilities that existed in one website, we also note variations between the scan results reported by Arachni and Wapiti. Although Arachni's reports showed that the highest number of SQL injection vulnerabilities that were found in one website was 29 (as shown in Table VIII), Wapiti's scan results indicated that 600 SQL vulnerabilities were detected in one website. Similarly, Wapiti found 234 XSS vulnerabilities in one website whereas the highest number of XSS vulnerabilities detected in one website by Arachni was 33. There is therefore considerable differences between the scan results of Wapiti and Arachni although our quantitative comparative performance assessment did not reveal notable differences between the detection accuracy rates of these scanners. This variation can be either explained by the crawler coverage which is presented in Table II or by the false alarm rates of Wapiti and Arachni.

Our comparative assessment results showed that Wapiti's observed false alarm rate for XSS vulnerabilities was 42.86% whereas no false positives were recorded for Arachni 0.4.3 (see Table VI). For SQL injection vulnerabilities, our observations also showed that a false positive rate of 20% was observed for the two scanners; however, our case study showed that there was a notable difference between the number of SQL vulnerabilities detected by the two scanners. Because Arachni's crawler coverage percentage was poor, there is a possibility that Arachni'e crawler did not scan all the pages that were scanned by Wapiti. However, this does not imply that Wapiti's results were more accurate since we observed that Wapiti had repeatedly scanned a considerable number of URLs containing similar web content.

6. GUIDELINES AND RECOMMENDATIONS

Our experimental findings highlighted limitations related to scanners' crawling capabilities, reporting and visualization features, efficiency and vulnerability detection accuracy. In this section, we note some areas that should be addressed by developers in order to increase the effectiveness of their tools.

We recommend system designers to utilize the available security benchmarks and test sites (e.g., the ones shown in Fig. 1) to evaluate the performance of their open source vulnerability detection solutions. This could contribute to improving the capabilities of open source scanners as it would allow comparing the quantities and types of vulnerabilities that were detected by open source scanners with those discovered by well-known proprietary scanners (e.g., IBM Security AppScan [8]). We also recommend tool developers to improve the crawling capabilities of their tools by ensuring that all the clickable elements in a crawled web page are identified and thus all the web pages that belong to the examined domains are traversed. For increasing the coverage of attack classes checked by web vulnerability scanners, tool developers could also enhance the comprehensiveness of their tests by validating all possible combinations of inputs to web forms while ensuring that the efficiency of their tools is not compromised. The elimination of duplicate links before starting the vulnerability scanning process is also recommended for improving the efficiency of vulnerability detection.

Designers of open source web vulnerability scanners are also recommended to improve the accuracy of the security verification checks incorporated in their tools to simplify the task of manually checking whether the vulnerabilities reported by these scanners are false positives or could actually be exploited by attackers [39]. That is, reporting web pages as vulnerable to exploitable attacks when in fact they do not exist increases the effort spent by penetration testers on validating the accuracy of the results reported by

these tools. For instance, web vulnerability scanners might sometimes report security vulnerabilities related to having cookies that are not flagged as secure when in fact they store non-sensitive data (e.g., user preferences) that is of no value to attackers. Platform designers could therefore address this problem by identifying the characteristics of non-sensitive cookies to avoid reporting them as potential security vulnerabilities. Web vulnerability scanners might also sometimes report vulnerabilities relating to locating server files without verifying whether the content of these files could actually disclose hidden information about the configurations of the web servers in question or not. We also recommend improving the reporting mechanisms implemented in open source scanners by presenting the user with more details about detected vulnerabilities and highlighting potential false positives. Reports generated by penetration testing tools should also include more guidance to draw the testers' attention to the factors that could characterize false positives and provide remediation advice on how to address detected vulnerabilities. Penetration testers should also be made aware of the circumstances that could cause a scanner to not detect certain vulnerabilities or misinterpret the observed behavior of the scanned website and report a vulnerability that does not exist.

From a usability viewpoint, tool designers are recommended to implement web-based interfaces of their open source scanners to allow web developers to access these scanners and run their scans easily. They are also recommended to improve the vulnerability reporting and visualization features of these tools to ensure that users are able to monitor the scanning processes in real time. Users should also be provided with various options allowing them to customize and control the scope of their scans [40].

With the emergence of new client and server side technologies, the developers of open source vulnerability scanners should continuously update their solutions to provide support to a variety of web platforms and protocols. A significant attention should be given to resolving the compatibility and interoperability issues that prevent software developers from scanning their web-based systems. Although the inclusion of all attack classes in one database is challenging, tool developers should ensure that their vulnerability databases are regularly updated to include newly discovered attacks. In order for a vulnerability scanning process to be successful and comprehensive, all applications states should also be thoroughly examined to ensure that all complex attack scenarios are handled by the corresponding web vulnerability scanner [5].

7. RELATED WORK

Prior work addressed web security vulnerabilities by either researching the ways to improve dynamic testing techniques [41], proposing or evaluating static analysis testing mechanisms [42–45] or using these testing approaches to evaluate the security of web-based systems [46–49]. The majority of research studies that evaluated the performance of web vulnerability scanners suggested that they have limited crawling capabilities and high false positive rates. The findings reported in this paper show that there is a low level of agreement between the results reported by different open source web vulnerability scanners. In the following subsection, we summarize the literature related dynamic testing tools and the detection of web vulnerabilities.

7.1. Web Security Scanners: Detection Effectiveness

Evaluating the security of web-based applications and detecting the security vulnerabilities in these multi-sourced systems is a complex problem that can be addressed in many different ways [50–52]. With the increasing complexity of web-based systems and the emergence of new development technologies that vary in their support for security, the demand for tools that automatically detect security vulnerabilities is growing [53–55]. Penetration testing is one of the widely used techniques that is applied to detect security vulnerabilities in a cost-effective way and without requiring technical expertise [6, 53, 56, 57].

In spite of the advantages of tool-based web penetration testing approaches, previous research attempts proved that web vulnerability scanners have poor performance and found that they are often unable to detect many security vulnerabilities [6, 53, 58, 59]. The limited discoverability capability is also one of the drawbacks of web vulnerability scanners (i.e., security vulnerability cannot be detected until the crawler of the corresponding scanner reaches the URL in question) [5]. Although the results generated by these scanners are mostly regarded by software developers as comprehensive and accurate [56], prior work found that these scanners only detect a subset of the security vulnerabilities in the evaluated systems and give a significant number of false alarms [53, 56].

Prior work has also addressed the performance of web security vulnerability scanners by either evaluating the detection effectiveness of a set of scanners [18, 19, 32, 58] or developing techniques that can be incorporated into these tools to increase their detection accuracy [5, 60, 61]. Other research attempts focused on comparing the effectiveness of dynamic testing with other security testing approaches (e.g., static testing or manual code review) [62]. Vieira et al. used some commercial web vulnerability detection tools for quantifying the number of security vulnerabilities in publicly accessible web services and observed a high false positive rate [17]. Consistent with our findings, the experimental results presented in [17] highlight limitations in the vulnerability detection and crawling capabilities of dynamic testing tools and show that there are major differences between the types and quantities of security vulnerabilities detected by different scanners.

DOI: 10.1002/sec

Their findings also suggested that there are variations between the results reported by different web vulnerability scanners. Similarly, three commercial web vulnerability detection scanners were evaluated in [18] by injecting SQL and XSS vulnerabilities into web-based systems and testing the scanners' effectiveness in detecting these security flaws. Their results furthermore suggest that these scanners leave some vulnerabilities undetected and produce inconsistent results. Although the observations reported in [17, 18] show that the percentages of false positives were high, the FPRs of 75% of the evaluated scanners in this study were less than or equal to 20%. This implies that the inconsistencies between the scan results of different scanners might not necessarily be linked to their poor performance.

Doupe et al. [32] also highlighted the limitations in the crawling capabilities of web vulnerability scanners after evaluating eight commercial and three open source scanners. Although some open source scanners were included in the evaluation conducted in [32], their comparative assessment is based on scanning one web-based system that was specifically developed for the purpose of their study. In our study, we evaluate the selected scanners based on a systematic approach that considers scanning a dataset that includes more than 140 distinct web-based systems. Clearly, there is a lack of studies that thoroughly analyze the effectiveness of open source web vulnerability scanners based on well-known security benchmarks and standards.

A number of researchers have focused on identifying the root causes of web vulnerabilities and proposed solutions to mitigate the effects of web security attacks [63–66]. For instance, as an attempt to explore the correlations between security vulnerabilities and software faults, Fonseca et al. analyzed a large collection of security patches and found that the majority of SQL injection and XSS attacks were not caused by software bugs [63]. Soska et al. addressed the maliciousness of web content in a different way by implementing a classification system that predicts whether a given web server will be compromised in the future [64]. In this study, we focused on comparing the effectiveness of two open source web vulnerability scanners in detecting SQL injection and XSS attacks. Our results show that there is a high level of disagreement between the results reported by the two scanners which implies that there is a demand for improving the efficiency, effectiveness and detection accuracy of these scanners.

8. LIMITATIONS AND FUTURE WORK

While our web vulnerability scanners' comparative evaluation included three scanners that are among the widely used open source web vulnerabilities detection tools, our results might not be generalized to all other similar open source tools. However, our evaluation approach can be applied to any set of open source or commercial web vulnerability scanners. Although the dataset collected for our case study included 140 URLs only, the approach followed to collect, scan, analyze and evaluate the security state of these web applications can be applied to other web-based systems as well. The sample examined in our case study had only 140 URLs which were collected based on a predefined approach that allowed us to collect a sample of websites that vary in their size, complexity and development technologies. Our case study can therefore be repeated to include a larger URL dataset. In our case study, we compared the results of the evaluated scanners based on their detection of SQL and XSS vulnerabilities only. Future work might consider other types of web vulnerabilities (e.g. CSRF and Source Code Disclosure). Future research efforts might also utilize the methodology followed in this paper to evaluate commercial web vulnerability scanners. Other research efforts might also be directed toward investigating the level of agreement between open source and commercial web vulnerability scanners. Our experiment could also be applied on a large set of open source web vulnerability scanners to further understand the factors that might contribute to improving the performance of these scanners. In future work, special attention should also be given to thoroughly investigating the effect of false positives on the quantities and types of security vulnerabilities reported by the scanners. For example, repeating the tests multiple times could reveal inconsistencies in the scanners' reports and thus help researchers in drawing more accurate conclusions on the detection accuracy of the scanners.

9. CONCLUDING REMARKS

The widespread adoption of web vulnerability scanners and the differences in the functionality provided by these tool-based vulnerability detection approaches increase the demand for testing their detection effectiveness. Although there are many previously conducted research studies that addressed the performance characteristics of web vulnerability detection tools by either quantifying the number of false alarms or measuring the corresponding crawler coverage, the scope of the majority of these studies is limited to commercial tools.

Despite the advantages of dynamic testing approaches in detecting software vulnerabilities at any stage in the software development process, the literature lacks studies that comprehensively and systematically evaluate the performance of open source web vulnerability scanners based on sound measures. The main objectives of this study is to assess the performance of open source scanners from multiple perspectives and to examine whether the cost-effectiveness of these tools negatively correlates with their detection capability. We expect the results of this research work to guide tool developers in enhancing the software processes followed while designing these tools, which in turn is expected to encourage software engineers

to effectively utilize web vulnerability detection scanners during and after releasing their software products.

The results of our comparative evaluation of a set of open source scanners highlighted variations in the effectiveness of security vulnerability detection and indicated that there are correlations between different performance properties of these scanners (e.g., scanning speed and crawler coverage, crawler coverage and number of detected vulnerabilities). There was a considerable variance both on types and numbers of detected web vulnerabilities among the examined tools.

We followed our comparative assessment with a case study in which we evaluate the level of agreement between the results reported by two open source web vulnerability scanners. These scanners were utilized to examine the web security vulnerabilities in 140 URLs and quantify the number of low, medium and high degree security vulnerabilities that existed in the evaluated URLs. By considering the fact that the examined scanners agreed that only 18 URLs were vulnerable to SQL injection attacks, this might suggest that the inconsistencies between the scan results of different tools might had been caused by factors other than performance differences. This finding might also be supported by the fact that although our scanners' comparative assessment did not show significant performance differences among the evaluated scanners, we observed a high level of disagreement between the scan results. Therefore, additional research should be directed toward identifying the factors that lead different widely-known software verification tools to report inconsistent results.

ACKNOWLEDGMENTS

This project was supported by NSTIP strategic technologies program number (11- INF1855-02) in the Kingdom of Saudi Arabia. Also, we thank the anonymous reviewers for their comments which helped improve this paper to its present form. We also thank Monirah Alshreef for preparing the URL dataset and conducting the experiments reported in the present study. This work was supported in part by KACST.

COMPETING INTEREST

The authors declare that there is no conflict of interest regarding the publication of this paper.

REFERENCES

1. OWASP: The Ten Most Critical Web Application Security Risks. Technical report, The Open Web Application Security Project, 2013.
2. Internet Security Threat Report. Technical report, Symantec, 2015.
3. Garnaeva Maria, Chebyshev Victor, Makrushin Denis, and Ivanov Anton. IT threat evolution in Q1 2015. Technical report, Kaspersky, 2015.
4. Gartner. The Next Three Years in Security Threats. http://www.gartner.com/smarterwithgartner/the-next-three-years-in-security-threats/, 2015. [Online; accessed: 14-August-2015].
5. Adam Doupé, Ludovico Cavedon, Christopher Kruegel, and Giovanni Vigna. Enemy of the State: A State-Aware Black-Box Web Vulnerability Scanner. In *USENIX Security Symposium*, pages 523–538, 2012.
6. Nuno Antunes and Marco Vieira. Comparing the effectiveness of penetration testing and static code analysis on the detection of SQL injection vulnerabilities in web services. In *15th IEEE Pacific Rim International Symposium on Dependable Computing*, pages 301–306. IEEE, 2009.
7. Frank Elberzhager, Jürgen Münch, and Vi Tran Ngoc Nha. A systematic mapping study on the combination of static and dynamic quality assurance techniques. *Information and Software Technology*, 54(1):1–15, 2012.
8. IBM AppScan. http://www-03.ibm.com/software/products/en/appscan, 2015. [Online; accessed: 15-September-2015].
9. HP Web Inspect. http://www8.hp.com/us/en/software-solutions/webinspect-dynamic-analysis-dast/, 2015. [Online; accessed: 15-September-2015].
10. Acunetix Web Vulnerability Scanner. http://www.acunetix.com/vulnerability-scanner/, 2015. [Online; accessed: 15-September-2015].
11. Kinnaird McQuade. Open Source Web Vulnerability Scanners: The Cost Effective Choice? In *Proceedings of the Conference for Information Systems Applied Research*, volume 2167, page 1508, 2014.
12. Shay Chen. SECTOOL Market: Price and Feature Comparison of Web Application Scanners. http://goo.gl/ZUKaK8, 2015. [Online; accessed: 17-November-2015].
13. David Geer. Are companies actually using secure development life cycles? *Computer*, (6):12–16, 2010.
14. Jim Witschey. Secure development tool adoption in open-source. In *Proceedings of the 2013 companion publication for conference on Systems, programming, & applications: software for humanity*, pages 105–106. ACM, 2013.
15. Giuseppe A Di Lucca and Anna Rita Fasolino. Testing Web-based applications: The state of the art and future trends. *Information and Software Technology*, 48(12):1172–1186, 2006.
16. Elizabeth Fong and Vadim Okun. Web application scanners: definitions and functions. In *40th Annual Hawaii International Conference on System Sciences*. IEEE, 2007.

DOI: 10.1002/sec

17. Marco Vieira, Nuno Antunes, and Henrique Madeira. Using web security scanners to detect vulnerabilities in web services. In *IEEE/IFIP International Conference on Dependable Systems & Networks*, pages 566–571. IEEE, 2009.
18. Jose Fonseca, Marco Vieira, and Henrique Madeira. Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks. In *13th Pacific Rim International Symposium on Dependable Computing*, pages 365–372. IEEE, 2007.
19. Larry Suto. Analyzing the effectiveness and coverage of web application security scanners. *San Francisco, October*, 2007.
20. Skipfish scanner. https://code.google.com/p/skipfish/, 2012. [Online; accessed: 26-August-2015].
21. Arachni Web Application Security Scanner Frameworkork. http://www.arachni-scanner.com/, 2015. [Online; accessed: 26-August-2015].
22. The Web Application Vulnerability Scanners Evaluation Project. https://code.google.com/p/wavsep/, 2013. [Online; accessed: 27-August-2015].
23. Wapiti- web application vulnerability scanner. http://wapiti.sourceforge.net/, 2014. [Online; accessed: 27-August-2015].
24. Vega Vulnerability Scanner. https://subgraph.com/vega/, 2014. [Online; accessed: 28-August-2015].
25. w3af: Open Source Web Application Security Scanner. http://w3af.org/, 2013. [Online; accessed: 28-August-2015].
26. IronWASP. https://ironwasp.org/, 2014. [Online; accessed: 28-August-2015].
27. Web Scanner Test Site. http://webscantest.com/, 2015. [Online; accessed: 29-August-2015].
28. Test Website for Acunetix Web Vulnerability Scanner. http://testaspnet.vulnweb.com/, 2005. [Online; accessed: 29-August-2015].
29. AltoroMutual. http://demo.testfire.net/, 2015. [Online; accessed: 29-August-2015].
30. Web Input Vector Extractor Teaser. https://github.com/bedirhan/wivet, 2014. [Online; accessed: 29-August-2015].
31. Nuno Antunes and Marco Vieira. Benchmarking vulnerability detection tools for web services. In *IEEE International Conference on Web Services (ICWS)*, pages 203–210. IEEE, 2010.
32. Adam Doupé, Marco Cova, and Giovanni Vigna. Why Johnny can't pentest: An analysis of black-box web vulnerability scanners. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 111–131. Springer, 2010.
33. Performance-Based Comparative Assessment of Open Source Web Vulnerability Scanners [List of Scanned Websites]. https://goo.gl/XjlFMl, 2016. [Online; accessed: 20-March-2017].
34. GoogleTrends. https://www.google.com/trends/, 2015. [Online; accessed: 30-August-2015].
35. SEOquake Firefox Extension. http://www.seoquake.com/, 2012. [Online; accessed: 30-August-2015].
36. Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/, 2015. [Online; accessed: 30-August-2015].
37. PuTTY. http://www.putty.org/, 2015. [Online; accessed: 30-August-2015].
38. TightVNC Software. http://www.tightvnc.com/, 2014. [Online; accessed: 30-August-2015].
39. Andrew Austin and Laurie Williams. One technique is not enough: A comparison of vulnerability discovery techniques. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 97–106. IEEE, 2011.
40. Hui-zhong Shi, Bo Chen, and Ling Yu. Analysis of Web security comprehensive evaluation tools. In *Second International Conference on Networks Security Wireless Communications and Trusted Computing (NSWCTC)*, volume 1, pages 285–289. IEEE, 2010.
41. Nor Fatimah Awang and Azizah Abd Manaf. Detecting Vulnerabilities in Web Applications Using Automated Black Box and Manual Penetration Testing. In *Advances in Security of Information and Communication Networks*, pages 230–239. Springer, 2013.
42. V Benjamin Livshits and Monica S Lam. Finding Security Vulnerabilities in Java Applications with Static Analysis. In *Usenix Security*, pages 18–18, 2005.
43. Yichen Xie and Alex Aiken. Static Detection of Security Vulnerabilities in Scripting Languages. In *USENIX Security*, volume 6, pages 179–192, 2006.
44. Katerina Goseva-Popstojanova and Andrei Perhinschi. On the capability of static code analysis to detect security vulnerabilities. *Information and Software Technology*, 68:18–33, 2015.
45. Gabriel Díaz and Juan Ramón Bermejo. Static analysis of source code security: Assessment of tools against SAMATE tests. *Information and Software Technology*, 55(8):1462–1476, 2013.
46. Abdulrahman Alarifi, Mansour Alsaleh, and Abdul-Malik Al-Salman. Security analysis of top visited Arabic web sites. In *15th International Conference on Advanced Communication Technology (ICACT)*, pages 173–178. IEEE, 2013.
47. Mansour Alsaleh and Abdulrahman Alarifi. Analysis of Web Spam for Non-English Content: Toward More Effective Language-Based Classifiers. *PloS one*, 11(11):e0164383, 2016.
48. Abdulrahman Alarifi and Mansour Alsaleh. Web spam: A study of the page language effect on the spam detection features. In *11th International Conference on Machine Learning and Applications (ICMLA)*, volume 2, pages 216–221. IEEE, 2012.

49. Abdulrahman Alarifi, Mansour Alsaleh, Abdulmalik Al-Salman, Abdulmajeed Alswayed, and Ahmad Alkhaledi. Google penguin: Evasion in non-english languages and a new classifier. In *12th International Conference on Machine Learning and Applications (ICMLA)*, volume 2, pages 274–280. IEEE, 2013.
50. Martin R Stytz and Sheila B Banks. Dynamic software security testing. *IEEE security & privacy*, (3):77–79, 2006.
51. Mansour Alsaleh, Abdullah Alqahtani, Abdulrahman Alarifi, and AbdulMalik Al-Salman. Visualizing PHPIDS log files for better understanding of web server attacks. In *Proceedings of the Tenth Workshop on Visualization for Cyber Security*, pages 1–8. ACM, 2013.
52. Mansour Alsaleh, Abdulrahman Alarifi, Abdullah Alqahtani, and AbdulMalik Al-Salman. Visualizing web server attacks: patterns in PHPIDS logs. *Security and Communication Networks*, 8(11):1991–2003, 2015.
53. Mark Curphey and Rudolph Arawo. Web application security assessment tools. *Security & Privacy, IEEE*, 4(4):32–41, 2006.
54. Ziyad Alshaikh, Abdulrahman Alarifi, and Mansour Alsaleh. Christopher Alexander's fifteen properties: Toward developing evaluation metrics for security visualizations. In *IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 295–300. IEEE, 2013.
55. Noura Alomar, Mansour Alsaleh, and Abdulrahman Alarifi. Social Authentication Applications, Attacks, Defense Strategies and Future Research Directions: A Systematic Review. *IEEE Communications Surveys & Tutorials*, 2017.
56. Brad Arkin, Scott Stender, and Gary McGraw. Software penetration testing. *IEEE Security & Privacy*, (1):84–87, 2005.
57. Jason Bau, Elie Bursztein, Divij Gupta, and John Mitchell. State of the art: Automated black-box web application vulnerability testing. In *IEEE Symposium on Security and Privacy (SP)*, pages 332–345. IEEE, 2010.
58. Natasa Suteva, Dragi Zlatkovski, and Aleksandra Mileva. Evaluation and Testing of Several Free/Open Source Web Vulnerability Scanners. pages 221–224, 2013.
59. Andrew Austin, Casper Holmgreen, and Laurie Williams. A comparison of the efficiency and effectiveness of vulnerability discovery techniques. *Information and Software Technology*, 55(7):1279–1288, 2013.
60. Viktoria Felmetsger, Ludovico Cavedon, Christopher Kruegel, and Giovanni Vigna. Toward automated detection of logic vulnerabilities in web applications. In *USENIX Security Symposium*, pages 143–160, 2010.
61. Yao-Wen Huang, Chung-Hung Tsai, Tsung-Po Lin, Shih-Kun Huang, DT Lee, and Sy-Yen Kuo. A testing framework for Web application security assessment. *Computer Networks*, 48(5):739–761, 2005.
62. Matthew Finifter and David Wagner. Exploring the relationship between Web application development tools and security. In *USENIX conference on Web application development*, 2011.
63. José Fonseca and Marco Vieira. Mapping software faults with web security vulnerabilities. In *IEEE International Conference on Dependable Systems and Networks With FTCS and DCC*, pages 257–266. IEEE, 2008.
64. Kyle Soska and Nicolas Christin. Automatically detecting vulnerable websites before they turn malicious. In *Proceedings of USENIX Security*, 2014.
65. R Sekar. An Efficient Black-box Technique for Defeating Web Application Attacks. In *NDSS*, 2009.
66. Abdulrahman Alarifi, Mansour Alsaleh, and Noura Alomar. A model for evaluating the security and usability of e-banking platforms. *Computing*, pages 1–17, 2017.

DOI: 10.1002/sec