# An empirical comparison of commercial and open-source web vulnerability scanners

Richard Amankwah, Jinfu Chen, Patrick Kwaku Kudjo, Dave Towey

RESEARCH ARTICLE

# An empirical comparison of commercial and open-source web vulnerability scanners

**Richard Amankwah[1] | Jinfu Chen[1] | Patrick Kwaku Kudjo[2] | Dave Towey[3]**

[1]School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, China
[2]Department of Information Technology Studies University of Professional Studies, Accra Ghana
[3]School of Computer Science, University of Nottingham Ningbo China, Ningbo, Zhejiang, China

**Correspondence Author**
Jinfu Chen, School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, China.
Email: jinfuchen@ujs.edu.cn

**Funding information**
This work is partly supported by the National Natural Science Foundation of China (NSFC grant numbers: U1836116, 61762040 and 61872167), the project of Jiangsu provincial Six Talent Peaks (Grant numbers: XYDXXJS-016), and the Graduate Research Innovation Project of Jiangsu Province (Grant numbers: KYCX17_1807).

**SUMMARY**
Web vulnerability scanners (WVSs) are tools that can detect security vulnerabilities in web services. Although both commercial and open-source WVSs exist, their vulnerability detection capability and performance vary. In this paper, we report on a comparative study to determine the vulnerability detection capabilities of eight WVSs (both open and commercial) using two vulnerable web applications: WebGoat and Damn vulnerable web application (DVWA). The eight WVSs studied were: Acunetix; HP WebInspect; IBM AppScan; OWASP ZAP; Skipfish; Arachni; Vega; and Iron WASP. The performance was evaluated using multiple evaluation metrics: precision; recall; Youden index; OWASP web benchmark evaluation (WBE); and the web application security scanner evaluation criteria (WASSEC). The experimental results show that, while the commercial scanners are effective in detecting security vulnerabilities, some open-source scanners (such as ZAP and Skipfish) can also be effective. In summary, this study recommends improving the vulnerability detection capabilities of both the open-source and commercial scanners to enhance code coverage and the detection rate, and to reduce the number of false-positives.

**KEYWORDS**
commercial scanners, open-source scanners, software vulnerability, vulnerable web application, detection capability.

## 1 | INTRODUCTION

The economic importance of web applications in multiple domains, including banking [1], transportation [2], manufacturing [3], business [4], and education [5], has increased the need for a mechanism to control and improve their quality. The extensive, almost ubiquitous, use of web applications has also resulted in an equally dramatic increase in attacks [6]. These attacks normally target weaknesses, flaws, and errors, (commonly referred to as security vulnerabilities) that may cause an explicit failure to protect the confidentiality, integrity, and availability of the application [7]. Examples of attacks include: command injection [8]; buffer overflow [9],[10]; data or path manipulation [11]; access control [12]; session hijacking [13]; and cookie poisoning [6],[14]. When the attacks succeed, they can result in data breaches and have other serious security implications.

In an attempt to improve both vulnerability detection and the general quality of web applications, several web vulnerability scanners (WVSs) have been developed and studied, including: the web application attack and audit framework (W3af) [15]; OWASP zed attack proxy (OWASP ZAP) [16]; Skipfish [17]; Arachni [18]; Vega, [19]; Stalker [20]; and IronWASP [21]. Seng et al. [22] defined WVSs as tools used to test and detect common security breaches in web applications. The National Institute of Standards and Technology (NIST) reported varied vulnerability detection capability among the WVSs [23], findings supported by later studies [24],[25],[26]. A key question regarding both commercial and open-source WVSs is: Which MVS is most suited for detecting a particular class of security vulnerability, doing so with high detection and low false-positive rates? Previous studies have attempted to answer this, with Fonseca et al. [20] and Suto [27], for example, performing comparative studies of various open-source and commercial WVSs. Antunes and Vieira [28] investigated the vulnerability detection capabilities of three WVSs (IPT-WS, SIGN-WS and RAD-WS), assessing their effectiveness based on coverage and false-positives, and finding that they could effectively detect the topmost web vulnerabilities, such as SQL injection and cross-site scripting (XSS). Makino and Kleve [25] examined the vulnerability detection capability of two open-source scanners, OWASP ZAP and Skipfish, using the damn vulnerable web application (DVWA) and web application vulnerability scanner project [29],[30]: Their experimental results showed ZAP to be superior to Skipfish.

Although there are several comparative studies on WVSs, the focus has mainly been on commercial scanners, with few studies empirically examining the effectiveness of open-source tools. To address this, following a similar procedure to that of Makino and Kleve [25], this study examines the vulnerability detection capabilities of both the commercial scanners Acunetix [22], HP Webinspect [19], IBM Appscan [31], and the open-source scanners OWASP Zed Attack Proxy (OWASP ZAP) [16], Skipfish [32], Arachni, Vega [33] and Iron WASP [34]. This choice of WVSs was partly motivated by software vendor interest in these specific tools (including reported skepticism over their detection capabilities, in terms of their false positive, false negative and coverage [35]), but also due to their apparent wide usage and regular updates [36]. In addition, vendors need to be well-informed of the effectiveness of the tools (both open-source and commercial) to enable appropriate evaluation and informed choices. This comparative study of the detection capabilities of the tools (both open-source and commercial) will support vendors' selection of the most appropriate WVS.

To the best of our knowledge, no other study has empirically analyzed these scanners against the DVWA [37] and WebGoat tools [38], using our selected evaluation metrics (precision; recall; Youden index; OWASP web benchmark evaluation (WBE); and the web application security scanner evaluation criteria (WASSEC)) [39],[40]. This study makes the following contributions:

- An extensive experiment evaluating the vulnerability detection effectiveness of eight commercial and open-source WVSs is reported on.
- The functionality of the commercial and open-source WVSs is studied and compared.
- A number of possible measures to improve the commercial and open-source WVSs are suggested.

The rest of this paper is structured as follows: Section 2 presents the background of the study and some previous related research. The methodology and experimental setup are given in Section 3. The experimental results are presented in Section 4. Section 5 presents a detailed discussion of the results. Section 6 examines the threats to validity of the study, and, finally, the conclusion and recommendations are presented in Section 7.

## 2 | BACKGROUND AND RELATED WORK

This section presents the background to the study and an overview of some related work. It includes a description of the evolution of web applications, web vulnerability scanners (WVSs), and the various security vulnerabilities in web applications. There is also a summary of recent research into the evaluation of web scanners.

The web application security consortium (WASC) [41] defines a web application as "a software application executed by a web server, which responds to dynamic web page requests over HTTP." According to Paulson [42], the turning point in web application development was the introduction of **A**synchronous JavaScript and **X**ML (AJAX), a technique for creating better, faster, and more interactive web applications, which helped transition the old concept of static web pages into a method for deploying interactive web applications. The common gateway interface (CGI) became the first standard environment used to generate dynamic web pages, with the use of CGI for website processing becoming known as web applications [43]. The introduction of CGI led to the appearance of other web application development tools such as PHP, Perl, Java Server Pages (JSP), JavaScript, and VBScript [43]. Figure 1 shows the evolution of web applications.

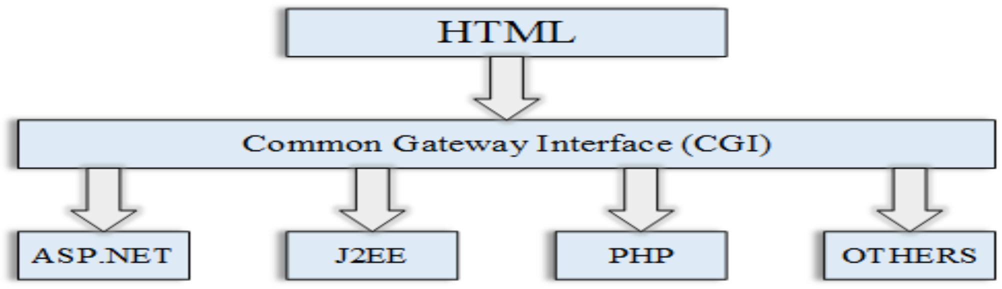

**FIGURE 1** Web application evolution

A web application typically includes a client, a web server, an application server (sometimes several), and a persistent database server, often with a firewall placed between the client and the webserver/application. Figure 2 depicts a simplified web application framework.

A WVS performs penetration testing by going through its web pages without executing the program. Most MVSs have three main components: one for crawling, one for attacking, and one for analysis [44]. The crawling component identifies the input and related pages of the web application based on its uniform resource locator (URL). The attacking component breaks down information discovered from the various webpages for each input vector and vulnerability type, and then sends the content to the webserver. The analysis component evaluates and interprets the responses from the server to determine if the attacks were successful or not. Techniques for testing web applications for vulnerabilities can be categorized as either white or black box testing [45]. White box testing is often used to analyze the application's source code (manually or using a code analysis tool); Black box testing, also known as penetration testing, executes the application to detect and locate security vulnerabilities [46]. Ashcan [47], Web King [48], Web Inspect [49], and Topsider [50] are some of the most widely applied commercial web application scanners.

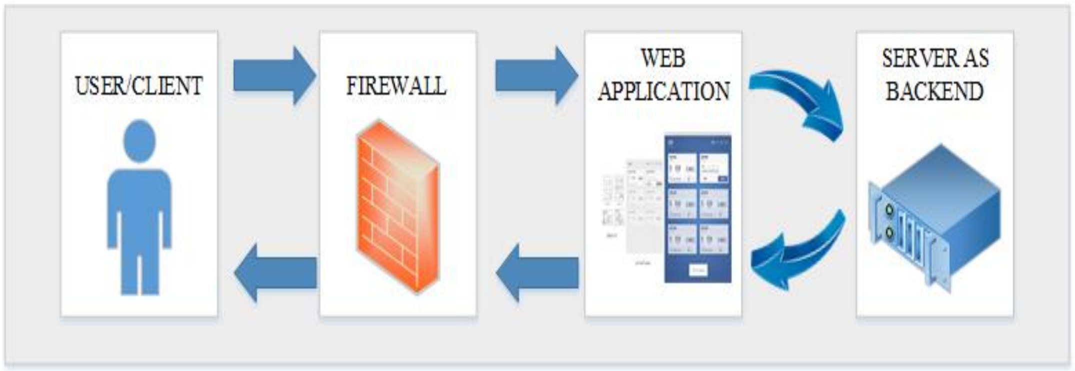

**FIGURE 2** Simplified view of a web application framework

Since its creation in 1997, the National Vulnerability Database (NVD) [51] has published information about more than 43,000 software vulnerabilities affecting more than 17,000 software applications [52]. Previous studies have successfully used vulnerabilities in this database to validate the vulnerability detection capabilities of their models [53], [54], [55]. Our study also used vulnerabilities presented in this database, as well as the vulnerabilities in DVWA and WebGoat. Table 1 presents a summary of the studied vulnerability types. There has been growing interest in research evaluating WVS. For example, Vieira et al. [56] evaluated the flaw detection capability of four commercial WVSs (Webinspect, Appscan, WSDigger, and Wsfuzzer): They conducted an experiment using 300 well-known web applications, finding that the selected scanners generated false positives between 35% and 40% of the time. Parvez et al. [26] later conducted a comparative study of three other WVSs (Acunetix, Appscan, and ZAP), with results indicating an improved detection rate.

Alsaleh et al. [57] examined four open-source scanners, finding similar detection rates for all four. More recently, Sagar et al. [7] evaluated the vulnerability detection capability of three other open-source WVSs (w3af, Skipfish, and OWASP ZAP) on the damn vulnerable web application (DVWA), concluding that OWASP ZAP performed better than the other scanning tools. An examination of these related studies reveals that most evaluated the effectiveness of commercial scanners or open-source scanners, but not both. Most studies focused only on SQL injection and cross-site scripting. Finally, none of the studies examined and compared the WVS performance based on both DVWA and OWASP WebGoat, using all the metrics used in our study.

**TABLE 1** Web application vulnerability types

| Vulnerability type | Abbr. | Vulnerability description |
|---|---|---|
| **Denial of Service** | DOS | Event or action that reduces or prevents the function of a user's target resource or application [58]. |
| **Code Execution** | CMD Exec | A situation where an attacker capitalizes on the weakness of a web application injects and executes a malicious server script on the targeted application to gain access to authorized resources [59]. |
| **Buffer Overflow** | BO | When an attacker exploits a vulnerability to exceed the memory buffer size and copy data from the adjacent memory location to make changes to the application [60]. |
| **Authentication Flaws** | AF | When an attacker gains access to a user's data through an exposed password [61]. These types of weaknesses can allow an attacker to either capture or bypass the authentication methods that are used by a web application. |
| **Cross-Site Scripting** | XSS | When an attacker gains access to a user's web application privileges by injecting malicious JavaScript code into the user's web browser [44]. |
| **Cross-Site-Request Forgery** | CSRF | When the attacker sends an unauthenticated HTTP request to a user's browser intending to send information (such as the user's session cookie and other relevant information) to a web application [62]. |
| **SQL injection (blind)** | BSQLi | When an attacker has access to security details (error details) which developers have hidden. The attack uses a sequence of SQL statements to snip the hidden details to perform malicious activities [63]. |
| **File inclusion** | FI | This error is caused when an application builds a path to executable code using an attacker-controlled variable in a way that allows the attacker to control which file is executed at run time [64]. |
| **Reflected Cross-site scripting** | RXSS | When an attacker supplies code (using different dynamic programming languages such as ActiveX, Flash, JavaScript, or Java) to the web browser of a user through viewed pages [65]. |
| **SQL Injection** | SQL | When an attacker inserts unvalidated input into the database of the web application to compromise its expected use [66]. |
| **Access Control Flaws** | ACF | It is an unintended access decision caused by misconfigured rules, policies, or algorithms within an access control system [67]. |

## 3 | EMPIRICAL STUDY

Our empirical study first identified the most widely-used and applied open-source and commercial WVSs, according to criteria from the Web Application Security Consortium [68]. We scanned the two benchmark web applications (WebGoat and DVWA) for vulnerabilities by configuring the browser and the selected WVSs for vulnerability detection. The detection results for each scanner were analyzed, and the performances were compared using the target metrics (precision, recall, Youden index, WBE and WASSEC).

### 3.1 | Research questions

Most commercial WVSs have automated crawlers and scanners, simplifying the vulnerability detection process. Open-source scanners, in contrast, typically do not have automated crawlers and scanners, and require human intervention, including to configure the tool as a proxy server. Because of this, it may be expected that commercial scanners would outperform the open-source scanners. Therefore, our first research question addresses the effectiveness of the commercial and open-source WVSs:

- **RQ1** - How do commercial WVSs compare with open-source WVSs, in terms of detection capability, for all vulnerability types in web applications?

Similar to the motivation behind RQ1, it may also seem more likely that open-source WVSs would generate more false-positive results than commercial WVSs. Automated crawlers in commercial WVSs can more efficiently crawl all parts of a web application than the manual crawling of open-source WVSs. This leads to the second research question:

- **RQ2** - How well do commercial WVSs compare with open-source WVS in terms of the number of false-positives generated?

Penetration testing is an important issue in cybersecurity, which partly explains the large number of WVSs developed. Typical questions asked by stakeholders lead to the third research question(s):

- **RQ3** - Which WVS is the most effective for vulnerability detection?

### 3.2 | Experimental setup

The experimental activity was divided into three steps: pre-experimental activities, experimental activities, and post-experimental activities. In the first stage, we conducted a detailed analysis of the eight WVSs to generate the workload (i.e., an idea of the actual work going to be performed). This was followed by the selection and detection of vulnerabilities in the respective vulnerable web applications.

The last stage involved the analysis and performance evaluation of the WVSs against the target metrics. The experiment was conducted on a workstation with an Intel(R) Core (TM) i5-6500 CPU at 3.20GHz, 4 GB of RAM, running Windows 7 Ultimate.

### 3.3 | Vulnerable web applications

To test our approach, we used two vulnerable web application programs: DVWA and WebGoat. Both DVWA and WebGoat consist of the OWASP TOP 10 security vulnerabilities. DVWA has a friendly user interface that allows developers, teachers, and students to explore and analyze web service security. It consists of multiple vulnerabilities, including command execution; cross-site request forgery; insecure captcha; file inclusion; SQL injection (standard and blind); reflected cross-site scripting (RXSS); and stored cross-site scripting (XSS) [25]. WebGoat is an open-source OWASP application created to help developers and experts examine the detection capability of WVS tools. The vulnerability types in WebGoat include: access control flaws; ajax security issues; authentication flaws; buffer overflows; poor code quality problems; concurrency cross-site scripting; bypass error handling flaws; injection flaws; denial of service; insecure communication; insecure configuration; insecure storage; malicious execution; parameter tampering; and session management flaws [69]. These vulnerabilities which we intend to detect in DVWA and WebGoat are intentionally injected based on the OWASP TOP 10 vulnerability in our study. These main web application vulnerability types in DVWA and WebGoat are shown in Table 1.

### 3.4 | WVSs under-study

Although there are several distributed network scanners with complex architecture, Makino and Kleve [25] reported that WVS architecture generally includes four modules: scan engine; scan database; report module; and user interface. The scan engine identifies security vulnerabilities with respect to its installed plug-ins and compares the outcome with known vulnerabilities. The scan database stores detailed information about various vulnerabilities. The report module presents a scan result with recommended solutions for developers and security administrators. The user interface provides a visual platform that can be graphical or command-driven, or both, for users to interact with the WVS. Our study examined eight WVSs, both commercial and open-source, all of which have a graphical user interface and run under Windows OS:

- Acunetix [70] WVS is a commercial security web tool that scans web applications to detect exploitable vulnerabilities. It scans for cross-site scripting, SQL Injections and other types of vulnerabilities in web applications. Additionally, the tool uses a multi-threaded fast approach to crawl through a series of web pages without breaks and produces various forms of compliance and technical reports.
- WebInspect [71] is an automated commercial web application security testing tool that identifies known and unknown vulnerabilities, including parameter injection; cross-site scripting; and directory traversal in web applications.
- AppScan [22] is a commercial secure web that finds and resolves known vulnerabilities in web applications.
- ZAP [25] is an open-source WVS with a user-friendly interface used for penetration testing. It can be used by people with different software security abilities.
- Skipfish [72] is an open-source web application security reconnaissance tool. It provides an interactive sitemap for the targeted site by carrying out a recursive crawl and dictionary-based probes. The resulting map is then annotated with the output from several active security checks. The final report generated by the tool is meant to serve as a foundation for professional web application security assessments.
- Arachni [21] is an effective and user-friendly open-source WVS, written in Ruby. It is very fast at scanning, and offers different user interfaces. It also provides a customized, command-driven input, and its output is in the form of HTML.
- IronWASP [73] (iron web application advanced security testing platform) is an advanced open-source web application security testing platform that comes in various external libraries such as IronPython, IronRuby, JSON, and .NET.
- Vega [74] is an automated open-source WVS for detecting SQL and other vulnerability types.

These scanners were selected for the study based on the comparison criteria proposed by the Web Application Security Consortium, Web Application Security Scanner Evaluation [75] and a study conducted by Suteva et al. [34] on the most popular open-source vulnerability scanners.

### 3.5 | Performance metrics

Similar to previous studies [25],[76], we compared the performance of the eight WVSs using five evaluation metrics: precision; recall; Youden index; OWASP web benchmark evaluation (WBE); and the web application security scanner evaluation criteria (WASSEC). Table 2 summarizes the notation and abbreviations [77].

**TABLE 2** Confusion matrix

| Metrics | Description |
|---|---|
| **True Positive (TP)** | Correctly detected vulnerability. |
| **False Positive (FP)** | Vulnerabilities incorrectly classified as vulnerabilities. |
| **True Negative (TN)** | No vulnerabilities present, and the tool confirms by not detecting any. |
| **False Negative (FN)** | The tool does not identify a vulnerability that is actually present. |

#### 3.5.1 | Precision

OWASP [78] defined precision as the percentage of correctly detected vulnerabilities as a proportion of all reported vulnerabilities (including those incorrectly labeled). The formula for this metric is given in Eq. 1. High precision values indicate a high detection accuracy of actual vulnerabilities.

$$\text{Precision}=\frac{TP}{TP+FP} \tag{1}$$

#### 3.5.2 | Recall

Recall [79] is the number of correctly detected vulnerabilities represented as a proportion of all the known vulnerabilities (including those that should have been detected by the tool but were not). The formula for the recall is given in Eq. 2.

$$\text{Recall}=\frac{TP}{TP+FN} \tag{2}$$

#### 3.5.3 | OWASP WBE

The OWASP benchmark project proposed a system for evaluating the effectiveness of static analysis tools called the WBE result interpretation guide [78]. The guide is a visual representation of a tool's detection performance based on fall-out (false positive) and recall rates. As shown in Figure 3, the line extending from the point (0%, 0%) to (100%, 100%) is the "guessing line", with the bug detection TP rate equal to the FP rate: performance on this line indicates the same performance as random selection. A plot of a tool's FP rate against its TP rate that is located in the top right corner indicates that the tool reported everything as vulnerabilities; location in the bottom left corner means that the tool recorded no vulnerabilities. The top left corner is the ideal location, indicating the best detection accuracy.

#### 3.5.4 | Youden index

The Youden index [80] was proposed to evaluate the performance of analytical (diagnostic) tests. It outputs values in the range [-1, 1], where a value of 1 (perfect detection) indicates detection of all vulnerabilities with no false positives; -1 indicates only false positives, and no true positives (no actual vulnerabilities detected); and a Youden index of 0 means the tool recorded the same result for a web application with vulnerabilities and without vulnerabilities — an invalid result. Eq. 3 shows the formula for calculating the Youden index.

$$J=\frac{TP}{TP+FN}+\frac{TN}{TN+FP}-1 \tag{3}$$

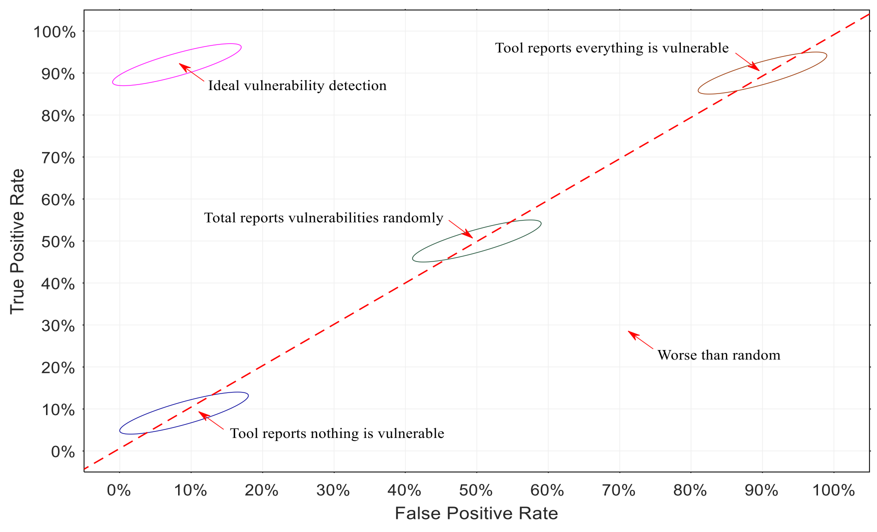

**FIGURE 3** OWASP WBE interpretation

### 3.5.5 | Web application security scanner evaluation criteria (WASSEC)

WASSEC [81] is comprised of six evaluation criteria/metrics that can help developers assess WVS detection capability. Table 3 presents the six WASSEC metrics.

**TABLE 3** WASSEC metrics

| Metric | Area of coverage |
|---|---|
| Protocol Support | Get, post, cookie, header, secret, pname, custom, proxy, gzip, eflate, ssl. |
| Session Management | Custom cookie, custom, header, logout, detection, exclude, log-out, exclude, url, exclude, param. |
| Testing | Sqli, bsqli, ssjsi, rxss, pxss, dxss, jsonh, lfi, rfi, cmdexec, upload, redirect, crlfi, ldapi, xpaphi, mxi, ssi, formati, codei, xmli, eli, buffero, integero, codedisc, backupf, padding, authb, prive, xxe, session, fixation, csrf, ados. |
| Parsing | Xml, xmlatt, xmltag, json, netenc, amf, javaser, netser, wcf, wcf-bin, websock, dwr, url file. |
| **Authentication** | Basic, digest, ntlm, ntlmv2, kerberos, form, cert, captcha. |
| Crawling | Manual crawl, html crawler, ajax crawler, flash crawler, applet crawler, silver-light crawler, wsdl crawler, rest crawler, field autofill, smart autofill, anti csrf support, viewstate support. |

## 4 | RESULTS

### 4.1 | Detection rates

Figure 4 presents the scanners' true-positive scores for the seven vulnerabilities in DVWA (BSQLi, CMDExec, CSRF, RXSS, SXSS, FI, and SQLi). As can be seen from the figure, while all scanners detected some CMDExec, RXSS, SXSS, and SQLi vulnerabilities, there was considerable variation in performance. For the RXSS vulnerabilities, for example, OWASP ZAP discovered 19; Acunetix and WebInspect detected five; Arachni detected four; Vega and AppScan detected three, and Skipfish and

IronWasp detected one. (The remaining results can be obtained from Figure 4.) The variation in detection rates could be attributed to how individual scanners are developed for specific vulnerability classes, with licensing also appearing to have an influence: The free edition of Acunetix, for example, was only able to detect XSS vulnerabilities. Furthermore, the detection capabilities of the scanners also vary from one web application to another.

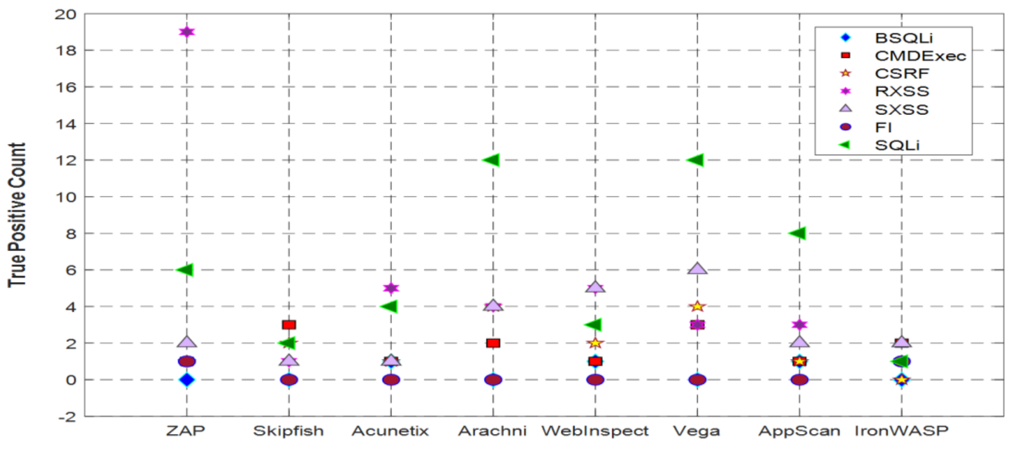

**FIGURE 4** Vulnerability detection capability (true positive count) in DVWA

Figure 5 shows the true-positive scores for the nine vulnerabilities in WebGoat (DoS, CMDExec, BF, XSS, CQ, BP, ACF, AF, and BSQLi,). Apart from IronWasp (which only detected two vulnerabilities), all tools were able to detect multiple vulnerabilities. Although no tool was able to detect all WebGoat vulnerabilities, the individual WVS performances are a clear indication that the tools were developed differently, leading to different strengths and weaknesses. The WebGoat vulnerabilities most detected were XSS and SQLi. Overall, the results give an indication of the commonalities and complementary strengths among the WVSs.

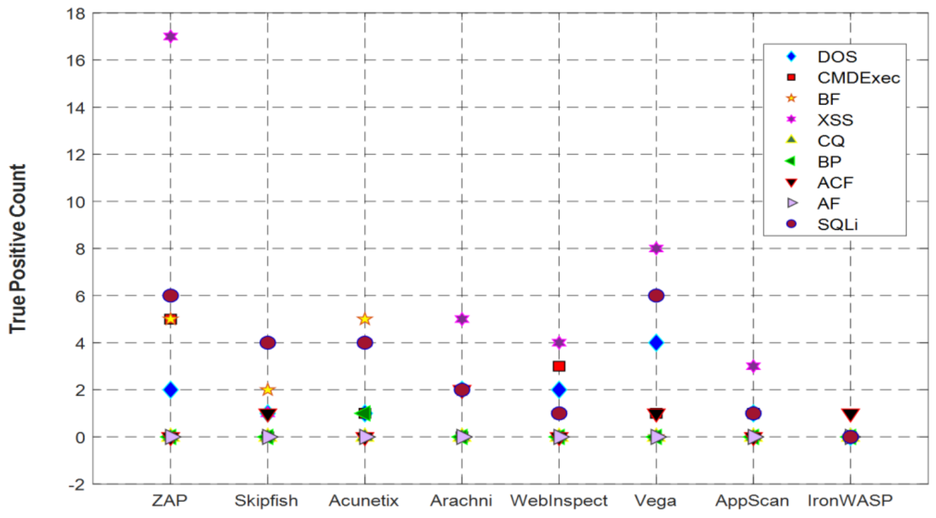

**FIGURE 5** Vulnerability detection capability (true positive count) in WebGoat

## 4.2 | Scanning time

We also evaluated the efficiency of the scanners based on the time required to complete the detection of vulnerabilities in both DVWA and WebGoat. The processing time for each scanner was calculated in seconds. We recorded the time for each scanner in both DVWA and WebGoat and present the result in Table 4. It can be seen from Table 4 that the running time for DVWA ranges from 30 to 360 seconds, and WebGoat ranges from 30 to 900 seconds. The performance differences between the scanners could be due to the URL injection points, with fewer injection points requiring less time than more points. Furthermore, variations in individual tool detection speed (*time*) could also be attributed to the internal security components of the applications. For instance, ZAP took 360 seconds in DVWA, but only 60 seconds in WebGoat. The scan profile of the tools for vulnerability detection could impact on the detection time.

**TABLE 4** Observed running time of scanners

| **Scanners** | **Scan duration** | |
|---|---|---|
| | **DVWA** | **WebGoat** |
| **ZAP** | 360sec | 60sec |
| **Skipfish** | 120sec | 120sec |
| **Acunetix** | 122sec | 120sec |
| **Arachni** | 60sec | 900sec |
| **WebInspect** | 180sec | 181sec |
| **Vega** | 60sec | 60sec |
| **AppScan** | 62sec | 70sec |
| **Iron WASP** | 60sec | 30 sec |

### 4.3 | Vulnerability severity

The vulnerabilities detected in DVWA and WebGoat were ranked according to their severity levels [82], with *high severity* meaning the impact of the vulnerability is devastating; *medium* meaning that the impact is dangerous; *low* meaning that the impact is minor; and *informational severity* having a negligible impact. 146 vulnerabilities were found in DVWA, of which 28 were of high severity; 29 medium; 50 low; and 39 informational. Acunetix and AppScan found the highest number of high-severity vulnerabilities in DVWA (10), followed by WebInspect (8). Not all open-source scanners found high-severity vulnerabilities, but this could be attributed to the licensing and profile settings of the tools. OWASP ZAP, for example, detected 30 vulnerabilities in DVWA (five medium, 20 low, and five informational).

IronWasp, an open-source web application security tool, detected the least number of vulnerabilities. 109 vulnerabilities were found in WebGoat, of which 23 were of high severity; 26 medium; 23 low; and 37 informational. Acunetix, WebInspect, Vega, and AppScan found ten, seven, one, and five high-severity vulnerabilities, respectively. The different severity ratings of vulnerabilities detected by the scanners in DVWA and WebGoat could be ascribed to the internal security architecture of the two web applications. The results also indicate that open-source scanners could not detect high-severity web vulnerabilities.

## 5 | DISCUSSION

This section presents a detailed analysis and evaluation of the tools.

### 5.1 | Precision and recall analysis of scanners

In this study, both precision and recall were measured in the range of 0-100%: An effective tool, with no false negatives or false positives, would have a value of 100% for both precision and recall. Figures 6 and 7 show the SVS precision and recall values for DVWA and WebGoat, respectively.

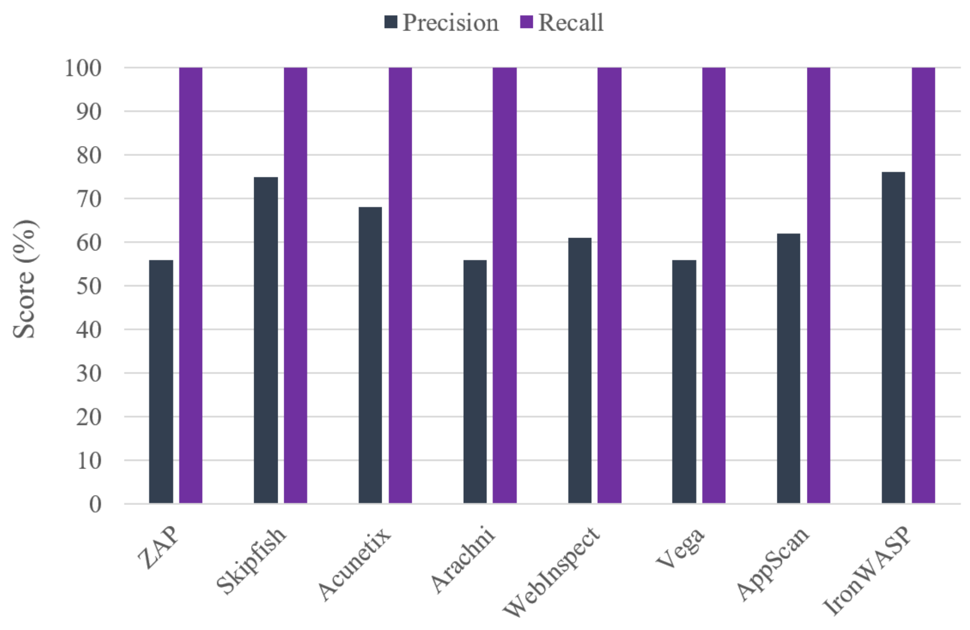

**FIGURE 6** Precision and recall for DVWA

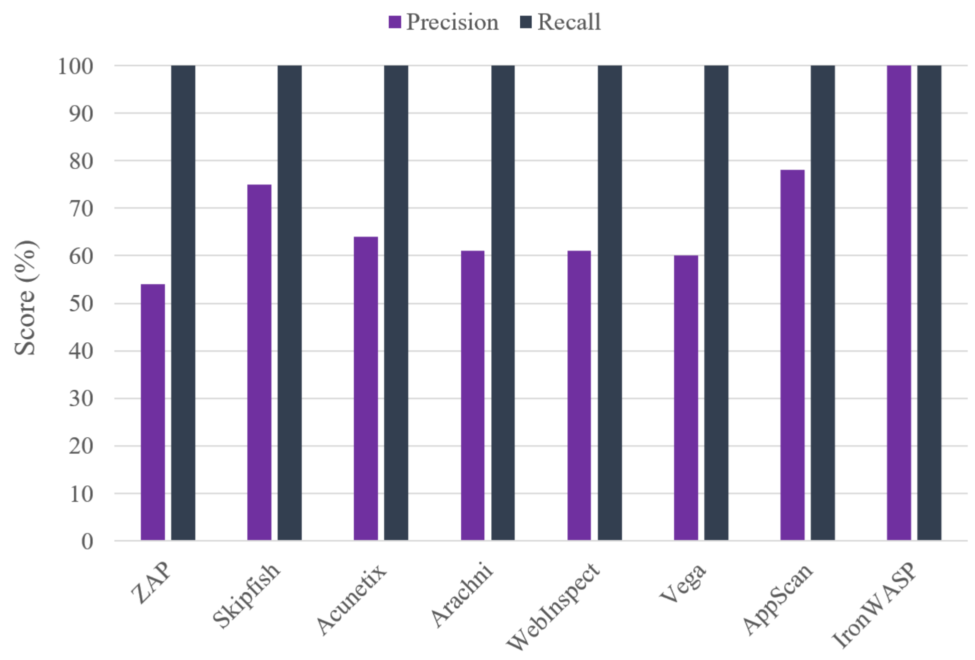

**FIGURE 7** Precision and recall for WebGoat

While the figures show that all scanners achieved a 100% recall score, indicating their ability to detect real vulnerabilities, there is considerable variation in their precision scores. This variation in precision could be attributed to each tool's uniqueness in vulnerability detection. Skipfish, for example, had a precision score of 75% for both DVWA and WebGoat, but Acunetix scored 68% for DVWA and 64% for WebGoat. ZAP, Arachni, and Vega all had precision scores of 56% with DVWA. These scores of less than 100% reflect the scanners flagging as vulnerabilities some issues that were not actual vulnerabilities (false positives).

**Answer to RQ1: Detection capability of scanners**

Both the open-source and commercial scanners were effective at detecting vulnerabilities in web applications, with the main differences between the two groups being in the different levels of precision (false positives). This finding suggests that stakeholders should consider assessing the tools based on the lowest numbers of false positives. Some tools were very effective for a specific type of vulnerability: While Acunetix, for example, was very effective at detecting reflected cross-site scripting (RXXS) vulnerabilities, OWASP ZAP was good at detecting command execution (CMDExec) vulnerabilities.

## 5.2 | OWASP WBE

The OWASP WBE results interpretation guide (Section 3.5.3) provides a graphical representation of a tool's effectiveness, mapping its true positive against its false-positive rates, as shown in Figure 3. In our experiments, as shown in Eqs. 4 and 5, we defined the total true-positive and total false-positive rates as the total number across both DVWA and WebGoat ($TP_t$ and $FP_t$ are the total true and false-positive rates, respectively; $TP_d$ and $FP_d$ are the true-positive and false-positive rates, respectively, for DVWA; and $TP_w$ and $FP_w$ are the true-positive and false-positive rates, respectively, for WebGoat).

$$TP_t = TP_d + TP_w \tag{4}$$

$$FP_t = FP_d + FP_w \tag{5}$$

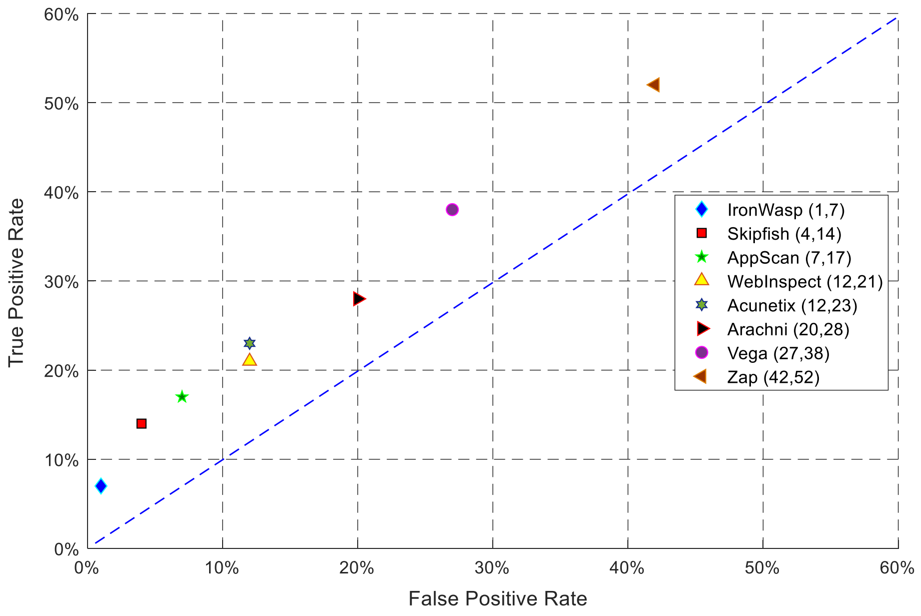

**FIGURE 8** OWASP WBE interpretation guide

Figure 8 presents the WBE results for the WVSs under study. As explained in Section 3.5.3, the scanner's effectiveness is represented by its position. According to ZAP's position at the top right corner, the tool detects and reports that "everything is vulnerable" — both true and false positive rates are high. IronWasp's position corresponds to the "nothing is vulnerable" category — both true and false positive rates are low. The performance of IronWasp could be attributed to it having been designed for a specific type of vulnerability detection. The remaining scanners fell into the "tool reports nothing is vulnerable" category, except Arachni, which was close to the "tool reports vulnerability randomly" category.

**Answer to RQ2: False-positive analysis of scanners**

According to the experimental data, there was no single scanner that offered ideal detection for all vulnerabilities. There were differences in the false-positive rates of vulnerabilities reported by both open-source and commercial scanners, with the rates being relatively higher for open-source tools. This performance difference could be attributed to most commercial scanners having automated scanners and crawlers, which could be more efficient and effective than the manual configuration and intervention necessary for open-source scanners. The generally high false-positive rates reflected an almost random vulnerability detection.

## 5.3 | Youden index

Figure 9 presents the Youden index (Section 3.5.4) of the scanners under study. IronWASP has the highest Youden index (0.83), which indicates its effectiveness detecting known vulnerabilities, with little or no false positives. The next highest scoring scanners were Skipfish, Appscan, Webinspect, and Acunetix, with 0.45, 0.31, 0.23, and 0.21, respectively. The results also indicate that several open-source scanners can function as effectively as some commercial web scanners. Thus, licensing alone should not be used as a standard metric for estimating the effectiveness of a tool.

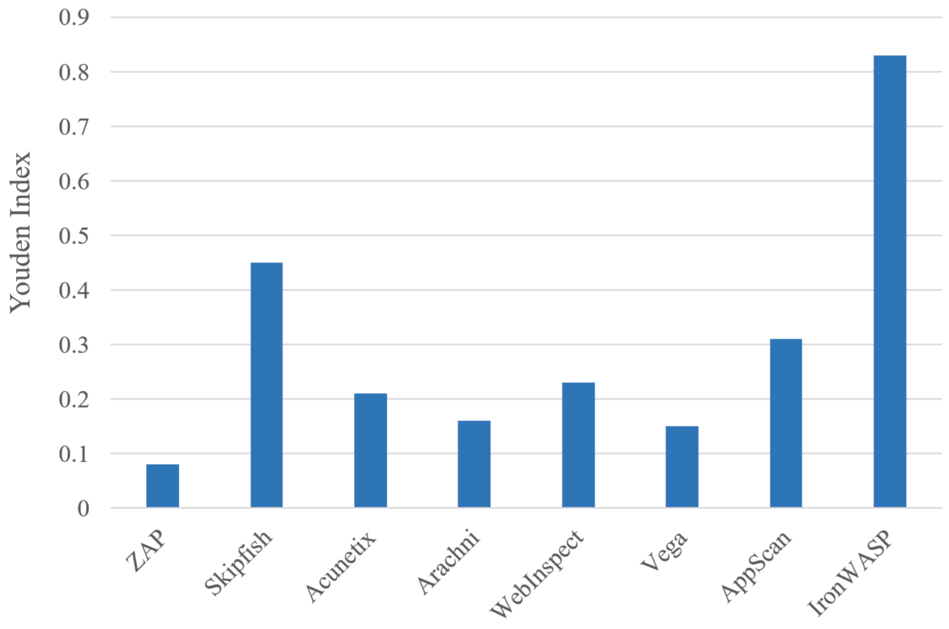

**FIGURE 9** Youden index results

## 5.4 | Web application security scanner evaluation criteria (WASSEC)

Table 5 shows the WASSEC (Section 3.5.5) results for the scanners under test. The results in Table 5 indicate that Acunetix has the best protocol support, followed by Appscan and Skipfish. The differences for session management, however, were much more marginal. Although there are differences in the performance of the scanners, there are similarities in the area of crawling, authentication and testing. Figure 10 shows the average WASSEC results, according to which Acunetix has the best performance, followed by Appscan, with scores of 0.81 and 0.65, respectively. However, the third and fourth-best performers, open-source scanners Skipfish and ZAP — with scores of 0.43 and 0.40, respectively — are also good performers.

TABLE 5 WASSEC results

| Tools | Metrics Protocol support | Session management | Testing | Parsing | Authentication | Crawling |
|---|---|---|---|---|---|---|
| **ZAP** | 7 | 5 | 12 | 2 | 3 | 4 |
| **Skipfish** | 8 | 5 | 13 | 3 | 3 | 4 |
| **Acunetix** | 10 | 6 | 28 | 7 | 7 | 9 |
| **Arachni** | 6 | 5 | 12 | 2 | 3 | 3 |
| **WebInspect** | 7 | 6 | 4 | 0 | 7 | 1 |
| **Vega** | 6 | 5 | 10 | 2 | 3 | 3 |
| **AppScan** | 8 | 6 | 22 | 4 | 6 | 8 |
| **IronWASP** | 6 | 5 | 9 | 3 | 3 | 2 |

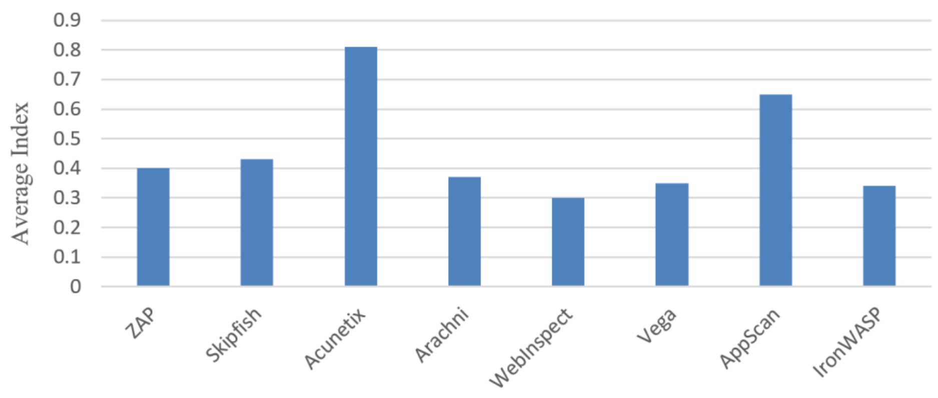

FIGURE 10 Average WASSEC results

**Answer to RQ3: Effectiveness of the scanners for vulnerability detection**

The experimental results show that there is no single WVS that can effectively detect all vulnerability classes. Although the results indicate that the commercial scanners Acunetix and Appscan may be the most effective, the open-source scanners Skipfish and ZAP also performed well, outperforming other commercial WVSs.

## 6 | THREATS TO VALIDITY

A threat to internal validity relates to the number of vulnerabilities used in the experimental analysis, i.e., the total vulnerabilities in DVWA and WebGoat. To mitigate this threat, we estimated the total number of vulnerabilities by the aggregation of each scanner's true-positive to form a true representation for our experiment. There were challenges configuring the tools due to their functionalities not being compatible with the Java platform (new version) employed in this study. We used several versions with limited functionality to validate the effectiveness of the tools: This can affect the vulnerability detection rate compared to the tools with the full versions. A threat to external validity relates to the generalizability of our results because we used vulnerability data from only two vulnerable web applications to verify the efficiency of the eight WVSs studied. Our future work will address this threat by examining other vulnerabilities and implementation tools.

## 7 | CONCLUSION AND FUTURE DIRECTIONS

This paper has reported on a comparative study of the vulnerability detection capabilities of eight web vulnerability scanners (WVSs) using two vulnerable web applications (Damn vulnerable web application (DVWA) and WebGoat). Of the eight WVSs studied, three were commercial scanners

(Acunetix, HP Webinspect, and IBM Appscan), and five were open-source scanners (OWASP ZAP, Skipfish, Arachni, Vega, and IronWASP). Their performance was examined using five metrics: precision; recall; Youden index; OWASP web benchmark evaluation (WBE); and the web application security scanner evaluation criteria (WASSEC). The experimental results show that the commercial scanners were effective at detecting security vulnerabilities, but that there were also open-source scanners (ZAP and Skipfish) that were equally efficient at detecting some vulnerabilities (including command execution, cross-site scripting, and SQL injection). Based on the experimental analysis, we recommend improving the vulnerability detection capabilities of the commercial and open-source scanners, to enhance code coverage and detection rates, and to reduce false positives. The development of WVSs should be standardized, to improve the systems, and promote the production of high-quality tools. Reports generated by scanners should not be difficult for users to interpret and understand (such as the HTML and XML reports provided by ZAP). In our future work, we will extend this study to include more state-of-the-art tools, and to examine performance with different vulnerable web applications.

## ACKNOWLEDGEMENT

The authors would like to thank anonymous reviewers for their comments on earlier versions of this paper.

## REFERENCES

[1] C. Möckel and A. E. Abdallah, "Threat modeling approaches and tools for securing architectural designs of an e-banking application," in *Proceeding of the Sixth International Conference on Information Assurance and Security*, 2010, pp. 149-154.

[2] L. F. Herrera-Quintero, J. C. Vega-Alfonso, K. B. A. Banse, and E. C. Zambrano, "Smart ITS sensor for the transportation planning based on IoT approaches using serverless and microservices architecture," *IEEE Intelligent Transportation Systems Magazine,* vol. 10, pp. 17-27, 2018.

[3] H. Li, W. Wei, and R. Fan, "Deep learning-based QoS prediction for manufacturing cloud service," in *Proceeding of the Chinese Control Conference (CCC)*, 2019, pp. 2719-2724.

[4] A. Wibowo, G. Aryotejo, and M. Mufadhol, "Accelerated mobile pages from Javascript as accelerator tool for web service on E-commerce in the E-business," *International Journal of Electrical & Computer Engineering* vol. 8, pp. 2088-8708, 2018.

[5] K. Kasemsap, "Exploring the role of web-based learning in global education," in *Revolutionizing Education through Web-Based Instruction*, ed: IGI Global, 2016, pp. 202-224.

[6] M. Awad, M. Ali, M. Takruri, and S. Ismail, "Security vulnerabilities related to web-based data," *Telkomnika,* vol. 17, pp. 852-856, 2019.

[7] D. Sagar, S. Kukreja, J. Brahma, S. Tyagi, and P. Jain, "Studying open source vulnerability scanners for vulnerabilities in web applications," *Institute of Integrative Omics and Applied Biotechnology Journal,* vol. 9, pp. 43-49, 2018.

[8] Z. Su and G. Wassermann, "The essence of command injection attacks in web applications," *ACM Sigplan Notices,* vol. 41, pp. 372-382, 2006.

[9] P. Luo, D. Zou, Y. Du, H. Jin, C. Liu, and J. Shen, "Static detection of real-world buffer overflow induced by loop," *Computers & Security,* vol. 89, pp.1-12, 2019.

[10] Z. Jin, Y. Chen, T. Liu, K. Li, Z. Wang, and J. Zheng, "A novel and fine-grained heap randomization allocation strategy for effectively alleviating heap buffer overflow Vulnerabilities," in *Proceedings of the 4th International Conference on Mathematics and Artificial Intelligence*, 2019, pp. 115-122.

[11] L. Miller and C. Pelsser, "A taxonomy of attacks using BGP blackholing," in *European Symposium on Research in Computer Security*, 2019, pp. 107-127.

[12] S. S. Alqahtani, "Automated extraction of security concerns from bug reports," in Proceedings of the *17th International Conference on Privacy, Security and Trust (PST)*, 2019, pp. 1-3.

[13] Q. Hu, B. Du, K. Markantonakis, and G. P. Hancke, "A session hijacking attack against a device-assisted physical-layer key agreement," *IEEE Transactions on Industrial Informatics,* vol. 16, pp. 691-702, 2019.

[14] P. Baral, "Web application scanners: a review of related articles [Essay]," *IEEE Potentials,* vol. 30, pp. 10-14, 2011.

[15] N. Antunes and M. Vieira, "Benchmarking vulnerability detection tools for web services," in *IEEE International Conference on Web Services*, 2010, pp. 203-210.

[16] O. R. Laponina and S. A. Malakhovsky, "Using the ZAP vulnerability scanner to test web applications," *International Journal of Open Information Technologies,* vol. 5, pp. 18-26, 2017.

[17] I. Mantra, M. S. Hartawan, H. Saragih, and A. A. Rahman, "Web vulnerability assessment and maturity model analysis on indonesia higher education," *Procedia Computer Science,* vol. 161, pp. 1165-1172, 2019.

[18] S. Idrissi, N. Berbiche, F. Guerouate, and M. Shibi, "Performance evaluation of web application security scanners for prevention and protection against vulnerabilities," *International Journal of Applied Engineering Research,* vol. 12, pp. 11068-11076, 2017.

[19] F. R. Muñoz, E. A. A. Vega, and L. J. G. Villalba, "Analyzing the traffic of penetration testing tools with an IDS," *The Journal of Supercomputing,* vol. 74, pp. 6454-6469, 2018.

[20] J. Fonseca, M. Vieira, and H. Madeira, "Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks," in *Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007)*, 2007, pp. 365-372.

[21] K. McQuade, "Open source web vulnerability scanners: the cost effective choice," in *Proceedings of the Conference for Information Systems Applied Research ISSN*, 2014, pp. 1508-1516.

[22] L. K. Seng, N. Ithnin, and S. Z. M. Shaid, "Automating penetration testing within an ambiguous testing environment," *International Journal of Innovative Computing,* vol. 8, pp.180-191, 2018.

[23] P. E. Black and E. Fong, "Proceedings of defining the state of the art in software security tools workshop," *NIST Special Publication,* vol. 500, pp. 264-273, 2005.

[24] N. Antunes and M. Vieira, "Detecting SQL injection vulnerabilities in web services," in *Proceedings of the Fourth Symposium on Dependable Computing* 2009, pp. 17-24.

[25] Y. Makino and V. Klyuev, "Evaluation of web vulnerability scanners," in *Proceedings of the IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, 2015, pp. 399-402.

[26] M. Parvez, P. Zavarsky, and N. Khoury, "Analysis of effectiveness of black-box web application scanners in detection of stored SQL injection and stored XSS vulnerabilities," in *Proceedings of the 10th International Conference on Internet Technology and Secured Transactions (ICITST)*, 2015, pp. 186-191.

[27] L. Suto, "Analyzing the accuracy and time costs of web application security scanners," *Application Security Consultant, February* 2010.

[28] N. Antunes and M. Vieira, "Designing vulnerability testing tools for web services: approach, components, and tools," *International Journal of Information Security,* vol. 16, pp. 435-457, 2017.

[29] S. Tyagi and K. Kumar, "Evaluation of static web vulnerability analysis tools," in Proceedings of the *Fifth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, 2018, pp. 1-6.

[30] Y. Liu, Z. Wang, and S. Tian, "Security against network attacks on web application system," in Proceedings of the *China Cyber Security Annual Conference*, 2018, pp. 145-152.

[31] F. A. Saeed, "Using wassec to evaluate commercial web application security scanners," *International Journal of Soft Computing and Engineering (IJSCE),* vol. 4, pp. 177-181, 2014.

[32] R. Mohammed, "Assessment of web scanner tools," *International Journal of Computer Applications,* vol. 5, pp.1-4, 2016.

[33] I. M. Babincev and D. V. Vuletić, "Web application security analysis using the kali Linux operating system," *Vojnotehnički Glasnik,* vol. 64, pp. 513-531, 2016.

[34] N. Suteva, D. Zlatkovski, and A. Mileva, "Evaluation and testing of several free/open source web vulnerability scanners," *Proceedings of the 10th Conference for Informatics and Information Technology (CIIT 2013),* 2013, pp. 221-224.

[35] B. Mburano and W. Si, "Evaluation of web vulnerability scanners based on OWASP benchmark," in Proceedings of the *26th International Conference on Systems Engineering (ICSEng)*, 2018, pp. 1-6.

[36] M. Rennhard, D. Esposito, L. Ruf, and A. Wagner, "Improving the effectiveness of web application vulnerability scanning," *International Journal on Advances in Internet Technology,* vol. 12, pp. 12-27, 2019.

[37] H. S. Abdullah, "Evaluation of open source web application vulnerability scanners," *Academic Journal of Nawroz University,* vol. 9, pp. 47-52, 2020.

[38] "The Open Web Application Security Project (OWASP)", https://owasp.org/, Accessed on March 15, 2017

[39] M. Sridevi and K. Sunitha, "A hybrid framework for secure web applications," in *International Conference on Intelligent Computing and Communication Technologies*, 2019, pp. 140-151.

[40] C. J. Van Rijsbergen, "A non-classical logic for information retrieval," *The Computer Journal,* vol. 29, pp. 481-485, 1986.

[41] E. Fong and V. Okun, "Web application scanners: definitions and functions," in *Proceeding of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, 2007, pp. 280b-280b.

[42] L. D. Paulson, "Building rich web applications with Ajax," *Computer,* vol. 38, pp. 14-17, 2005.

[43] S. Patil, N. Marathe, and P. Padiya, "Design of efficient web vulnerability scanner," in *International Conference on Inventive Computation Technologies (ICICT)*, 2016, pp. 1-6.

[44] S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic, "Secubat: a web vulnerability scanner," in *Proceedings of the 15th International Conference on World Wide Web*, 2006, pp. 247-256.

[45] S. M. Srinivasan and R. S. Sangwan, "Web app security: A comparison and categorization of testing frameworks," *IEEE Software,* vol. 34, pp. 99-102, 2017.

[46] R. Zabicki and S. R. Ellis, "Penetration testing," in *Computer and Information Security Handbook*, ed: Elsevier, 2017, pp. 1031-1038.

[47] E. Fong, R. Gaucher, V. Okun, P. E. Black, and E. Dalci, "Building a test suite for web application scanners," in *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*, 2008, pp. 478-478.

[48] O. Hamed and N. Kafri, "Performance prediction of web based application architectures case study: .NET vs. Java EE," *International Journal of Web Applications,* vol. 1, pp. 146-156, 2009.

[49] M. Salas and E. Martins, "Security testing methodology for vulnerabilities detection of xss in web services and ws-security," *Electronic Notes in Theoretical Computer Science,* vol. 302, pp. 133-154, 2014.

[50] H. Le and P. Loh, "Unified approach to vulnerability analysis of web applications," in *Proceedings of AIP Conference*, 2008, pp. 155-159.

[51] H. Booth, D. Rike, and G. Witte, "The national vulnerability database (NVD): Overview," National Institute of Standards and Technology, pp. 25-35, 2013.

[52] P. K. Kudjo, J. Chen, S. Mensah, and R. Amankwah, "Predicting vulnerable software components via bellwethers," in *Communications in Computer and Information Science*, 2018, pp. 389-407.

[53] A. Tripathi and U. K. Singh, "On prioritization of vulnerability categories based on CVSS scores," in *Proceedings of the 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*, 2011, pp. 692-697.

[54] J. A. Wang and M. Guo, "Vulnerability categorization using Bayesian networks," in *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, 2010, pp. 1-4.

[55] P. K. Kudjo, J. Chen, M. Zhou, S. Mensah, and R. Huang, "Improving the accuracy of vulnerability report classification using term frequency-inverse gravity moment," in *Proceedings of the 19th IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2019, pp. 248-259.

[56] M. Vieira, N. Antunes, and H. Madeira, "Using web security scanners to detect vulnerabilities in web services," in *Proceedings of IEEE/IFIP International Conference on Dependable Systems & Networks, 2009. DSN'09*, 2009, pp. 566-571.

[57] M. Alsaleh, N. Alomar, M. Alshreef, A. Alarifi, and A. Al-Salman, "Performance-based comparative assessment of open source web vulnerability scanners," *Security and Communication Networks,* vol. 2017, pp. 1-14, 2017.

[58] D. H. Woo and H. Lee, "Analyzing performance vulnerability due to resource denial of service attack on chip multiprocessors," in *Workshop on Chip Multiprocessor Memory Systems and Interconnects*, 2007, pp. 1-8.

[59] W. Du and A. P. Mathur, "Vulnerability testing of software system using fault injection," *Purdue University, West Lafayette, Indiana, Technique Report.* pp. 98-02, 1998.

[60] E. Haugh and M. Bishop, "Testing C programs for buffer overflow vulnerabilities," in *NDSS*, 2003.

[61] S. Whalen, M. Bishop, and S. Engle, "Protocol vulnerability analysis," *Department of Computer Science, University of California, Davis, USA, Technical Report CSE-2005-04,* 2005.

[62] S. Calzavara, M. Conti, R. Focardi, A. Rabitti, and G. Tolomei, "Mitch: a machine learning approach to the black-box detection of CSRF vulnerabilities," in *Proceedings of IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019, pp. 528-543.

[63] A. B. Ibrahim and S. Kant, "Penetration testing using SQL injection to recognize the vulnerable point on web pages," *International Journal of Applied Engineering Research,* vol. 13, pp. 5935-5942, 2018.

[64] M. M. Hassan, T. Bhuyian, M. K. Sohel, M. H. Sharif, and S. Biswas, "SAISAN: An automated local file inclusion vulnerability detection model," *International Journal of Engineering & Technology,* vol. 7, pp.4-8, 2018.

[65] P. Sharma, R. Johari, and S. Sarma, "Integrated approach to prevent SQL injection attack and reflected cross site scripting attack," *International Journal of System Assurance Engineering and Management,* vol. 3, pp. 343-351, 2012.

[66] G. Deepa and P. S. Thilagam, "Securing web applications from injection and logic vulnerabilities: Approaches and challenges," *Information and Software Technology,* vol. 74, pp. 160-180, 2016.

[67] G. Deepa, P. S. Thilagam, A. Praseed, and A. R. Pais, "DetLogic: A black-box approach for detecting logic vulnerabilities in web applications," *Journal of Network and Computer Applications,* vol. 109, pp. 89-109, 2018.

[68] W. A. S. Consortium, "WASC threat classification," *Release, Web Application Security Consortium,* 2010, pp. 1-8.

[69] N. A. Aziz, S. N. Z. Shamsuddin, and N. A. Hassan, "Inculcating Secure Coding for beginners," in *Proceedings of the International Conference on Informatics and Computing (ICIC),* , 2016, pp. 164-168.

[70] C. Joshi and U. K. Singh, "Security testing and assessment of vulnerability scanners in quest of current information security landscape," *International Journal of Computer Applications,* vol. 145, pp. 1-7, 2016.

[71] S. Gupta and L. Sharma, "Analysis and assessment of web application security testing tools," in *Proceedings of the 5th National Conference*, 2011, pp. 1-2.

[72] A. Dessiatnikoff, R. Akrout, E. Alata, M. Kaâniche, and V. Nicomette, "A clustering approach for web vulnerabilities detection," in *Proceedings of the 17th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2011),* 2011, pp. 194-203.

[73] S. Chen, "The web application vulnerability scanners benchmark," *Denim Group,* 2014.

[74] F. A. Saeed and E. A. Elgabar, "Assessment of open source web application security scanners," *Journal of Theoretical and Applied Information Technology,* vol. 61, pp. 281-287, 2014.

[75] M. U. J. DAR, J. L. Shah, and G. I. A. Khanday, "Web abuse using cross site scripting (XSS) Attacks," *Journal of Artificial Intelligence Research & Advances,* vol. 6, pp. 69-75, 2019.

[76] Y.-H. Tung, S.-S. Tseng, J.-F. Shih, and H.-L. Shan, "A cost-effective approach to evaluating security vulnerability scanner," in *Proceedings of the 15th Asia-Pacific Symposium on Network Operations and Management (APNOMS 2013),* 2013, pp. 1-3.

[77] Y. Fang, Y. Li, L. Liu, and C. Huang, "DeepXSS: Cross site scripting detection based on deep learning," in *International Conference on Computing and Artificial Intelligence*, 2018, pp. 47-51.

[78] "OWASP Benchmark Project" https://www.owasp.org/index.php/Benchmark, Accessed June 2, 2016.

[79] Y.-H. Tung, S.-S. Tseng, J.-F. Shih, and H.-L. Shan, "W-VST: A Testbed for Evaluating Web Vulnerability Scanner," in *Proceeding of the 14th International Conference on Quality Software*, 2014, pp. 228-233.

[80] W. J. Youden, "Index for rating diagnostic tests," *Cancer,* vol. 3, pp. 32-35, 1950.

[81] W. A. S. Consortium, "Web application security scanner evaluation criteria," *Version,* vol. 1, pp. 1-26, 2009.

[82] P. Saripalli and B. Walters, "Quirc: A quantitative impact and risk assessment framework for cloud security," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing (CLOUD), 2010* on, 2010, pp. 280-288.