
Article:
Althunayyan, M., Saxena, N., Li, S. et al. (2022) Evaluation of black-box web application security scanners in detecting injection vulnerabilities. Electronics, 11 (13). 2049.

https://doi.org/10.3390/electronics11132049

---

electronics

Article

Evaluation of Black-Box Web Application Security Scanners in Detecting Injection Vulnerabilities

Muzun Althunayyan [1,2,*], Neetesh Saxena [1,*], Shancang Li [1] and Prosanta Gope [3]

[1] School of Computer Science and Informatics, Cardiff University, Cardiff CF24 4AX, UK; lis117@cardiff.ac.uk
[2] School of Computer Science and Informatics, Majmaah University, Al Majma'ah 15362, Saudi Arabia
[3] Department of Computer Science, University of Sheffield, Sheffield S1 4DP, UK; p.gope@sheffield.ac.uk
* Correspondence: mzon_th@hotmail.com (M.A.); nsaxena@ieee.org (N.S.)

Abstract: With the Internet's meteoric rise in popularity and usage over the years, there has been a significant increase in the number of web applications. Nearly all organisations use them for various purposes, such as e-commerce, e-banking, e-learning, and social networking. More importantly, web applications have become increasingly vulnerable to malicious attack. To find web vulnerabilities before an attacker, security experts use black-box web application vulnerability scanners to check for security vulnerabilities in web applications. Most studies have evaluated these black-box scanners against various vulnerable web applications. However, most tested applications are traditional (non-dynamic) and do not reflect current web. This study evaluates the detection accuracy of five black-box web application vulnerability scanners against one of the most modern and sophisticated insecure web applications, representing a real-life e-commerce. The tested vulnerabilities are injection vulnerabilities, in particular, structured query language (SQLi) injection, not only SQL (NoSQL), and server-side template injection (SSTI). We also tested the black-box scanners in four modes to identify their limitations. The findings show that the black-box scanners overlook most vulnerabilities in almost all modes and some scanners missed all the vulnerabilities.

Keywords: injection vulnerability; web application; cyber security

Citation: Althunayyan, M.; Saxena, N.; Li, S.; Gope, P. Evaluation of Black-Box Web Application Security Scanners in Detecting Injection Vulnerabilities. *Electronics* **2022**, *11*, 2049. https://doi.org/10.3390/electronics11132049

Academic Editors: Constantinos Kolias, Georgios Kambourakis and Weizhi Meng

Received: 24 May 2022
Accepted: 26 June 2022
Published: 29 June 2022

1. Introduction

According to Internet Live Stats [1], there has been a significant increase in Internet users over the past decade, with approximately 4.7 billion users worldwide. Therefore, it is no surprise that there has also been a significant increase in the number of web applications, with an approximate total over 1.8 billion [2]. Web applications are used by almost all organisations in all sectors for numerous purposes, including e-commerce, e-banking, e-learning, and social networking. Organisations that fail to protect their web applications are at risk of being targeted by attackers. This can result in information disclosure, revenue loss, damaged client relationships and more. According to the latest report by Verizon [3], web applications are a popular target for data breaches.In some organisations, up to 43% of data breaches are related to web applications, more than double the results of the previous year. An insecure web application not only threatens the organisation but it also affects its users. For example, more than 80% of the reported data breaches have resulted in the theft of user credentials [3].

The Open Web Application Security Project (OWASP) [4] report listed the top 10 most common web application vulnerabilities and the injection type is currently ranked first. Moreover, according to a State of the Internet report [5], injection attacks are the top threat, accounting for nearly two-thirds of all attacks in 2019. A famous example is the Equifax breach, in which a vulnerability was exploited, resulting in the personal information of an estimated 143 million American users and approximately 100,000 Canadian users [6].

A variety of techniques can be used to secure web applications. These include firewalls, secure coding practices, and black-box web application vulnerability scanners [7]. In order to automate web application security in large and complex organisations, black-box scanners are ideal. Moreover, Makino and Klyuev [8] argued that testing web application vulnerabilities manually is challenging, time-consuming, error-prone, and expensive. Although many studies have evaluated black-box scanners, more research is needed with modern web applications with the latest technologies to improve the vulnerability detection capabilities of these automated scanners. Identifying the problem is often half the solution. Many studies have sought to identify the limitations of black-box scanners to improve the efficacy and timeliness of detection. Although various studies have evaluated black-box scanners' ability to detect vulnerabilities using different vulnerable web applications, there is still a need to test black-box scanners against modern web applications as technologies advance.

2. State-of-the-Art Review

Many academic and private-sector studies on web application security have been conducted. In 2010, Suto [9] tested the effectiveness of seven black-box scanners by performing point-and-shoot (PaS) and trained scans.Suto observed that the Cenzic Hailstorm scanner demonstrated very high detection accuracy which significantly improved after training. Other scanners only showed moderate improvements. In the same year, Bau et al. [10] evaluated eight commercial web application scanners. The scanners were first tested against well-known vulnerable applications (i.e., Drupal, phpBB, and WordPress), released in 2006. The study showed that the scanners did exceptionally well in detecting information disclosures and session management vulnerabilities. In addition, the scanner's detection rate was about 50% successful in detecting cross site scripting (XSS) and structured query language injection (SQLi) vulnerabilities and very low for Cross-Site Request Forgery (CSRF) and cross channel scripting (XCS) vulnerabilities. Second, the selected scanners' vulnerability accuracy and crawling ability were tested against a custom test-bed application. The authors concluded that crawling web technologies, such as Java applets, SilverLight, and Flash, were challenging, and most of the scanners had poor crawling capabilities. Moreover, 11 black-box scanners were evaluated by Doupé et al. [11] on their ability to discover associated vulnerabilities and crawl complex web pages. The authors developed a practical web application called WackoPicko with many contemporary features. They found that these scanners missed many types of vulnerabilities. Therefore, additional research is required to improve the automated vulnerability detection.

In 2011, Khoury et al. [12] tested and assessed three web scanners in their ability to detect persistent SQLi injection vulnerability. The study showed that the three black-box scanners were poor at detecting persistent SQLi injections even when they were explicitly taught to execute the attack code. Khoury et al. [13] also showed that the scanners were poor at detecting stored SQLi vulnerabilities. They concluded that the significant challenges for scanners in detecting stored SQLi injection was selecting proper input values and not using proper attack codes to exploit these vulnerabilities.

In 2015, Parvez et al. [14] analysed the performance of three black-box web scanners in their ability to detect stored XSS and stored SQLi vulnerabilities using a custom web application and WackoPicko, which was used in most previous studies [11–13]. The research showed that the black-box scanners' XSS detection had improved. The authors confirmed that it was a significant challenge for automated scanners to select suitable attack vectors for both stored XSS and stored SQLi.

Makino and Klyuev [8] evaluated two open-source vulnerability scanners, OWASP Zed Attack Proxy (ZAP) and Skipfish, for the detection of common vulnerabilities. After comparing the results, OWASP ZAP was found to be superior to Skipfish. However, both scanners had limitations, especially with detecting the remote file inclusion (RFI) vulnerability. In 2017, Berbiche et al. [15] assessed the effectiveness of 11 commercial and free web application security scanners against Web Application Vulnerability Scanner

Evaluation Project (WAVSEP) assessment application. Precision, recall and F-measure metrics were applied to evaluate performance. Although the study showed different results for each scanner, all performed better on SQLi and XSS than on local and remote file inclusion.

The aim of this study is to evaluate the detection capability of black-box scanners against injection vulnerabilities, particularly SQLi injection, not only SQL(NoSQL), and server-side template injection (SSTI). Previous studies were limited to testing black-box scanners against traditional (non-dynamic) applications that do not reflect current web architectures. Therefore, this study evaluates them against one of the most modern and sophisticated insecure web applications that use the latest web technology. Moreover, to the best of our knowledge, no study has evaluated the black-box scanner's ability to detect NoSQL and SSTI vulnerabilities.

We summarize our three-fold contribution as follows.

1. We analysed and evaluated the detection accuracy of black-box scanners against SQLi, NoSQL, and SSTI injection vulnerabilities using one of the most modern and sophisticated insecure web applications, representing a real-life e-commerce web application.
2. We tested black-box scanners in four different modes to identify their limitations and gaps in practice.
3. We identified the limitations of the evaluated black-box scanners' ability to detect SQLi, NoSQL, and SSTI vulnerabilities.

3. Web Application Security Testing and Vulnerability

The security of web applications can be tested in two ways: white-box testing [8] and black-box testing [16,17]. Several useful tools are used for both [18]. Several vulnerabilities threaten web applications. OWASP [19] has outlined security concerns for web applications and provides regular reports on the top 10 critical vulnerabilities.

In this work, we are mainly interested in injection attacks. In the latest report by OWASP [4], the injection attack was first on the list. These vulnerabilities arise when an attacker sends hostile data to an interpreter as part of a command or query. Successful injection can lead to data loss, corruption, information disclosure, loss of accountability, denial of access and loss of control of the system [4]. The business impact of injection attacks depends on the application and data. Other common injections include SQLi, NoSQL, operating system command, lightweight directory access protocol injection, expression language, SSTi and object graph navigation library injection. In greater detail, the following section describes three injection vulnerabilities, namely SQLi, NoSQL, and SSTI.

3.1. Structured Query Language (SQLi) Injection

According to OWASP [20], the SQLi web security vulnerability involves injecting an SQL query into an application through a client's input data. A successful SQLi can exploit confidential information, such as passwords, credit card information and personal data. It also enables attackers to modify database data (e.g., insert, update or delete), execute administration operations in the database (e.g., shutting down the database management system and issuing commands to the operating system) [20]. Various SQL injection vulnerabilities, attacks, and techniques occur in different situations. Examples of SQLi include union-based and blind SQLi injections.

3.2. Union-Based SQLi

According to PortSwigger [21], a union-based SQLi attack occurs when a web application is vulnerable to SQLi injection, and the query outputs are returned to the user within the responses of the application. Therefore, the attackers can use the UNION keyword to retrieve data that they are not permitted to access from other tables in the database. The UNION keyword enables the attacker to execute one or more SELECT queries and append the results to the original query. One example, as presented by Al-Khurafi and Al-Ahmad [22], is an online shop web application connected to a database server that

contains an Accounts table to authenticate users and a Customers table with records of all customer information, including names, phone numbers, addresses, orders, and payment information. If the username field parameter is vulnerable to SQLi injection, the attacker can inject the following malicious command in the username field: `", UNION SELECT * FROM Customers - - "` and anything for the password, which will result in the following query [21]:

```
SELECT * FROM Accounts WHERE USERNAME= ' ' UNION SELECT * FROM Customers - - '
AND password = ' anything ' ;
```

The first query will return null because the Accounts table does not contain matching records with an empty username. However, the second query will return all customer records from the Customers table. Because the UNION operator returns the output of both queries, this will allow an attacker to access all data in the Customers table.

3.3. Blind SQLi

According to Banach [23], a blind SQLi indirectly discovers information by analysing server reactions to various injected statements. Blind SQL injection occurs when the database server is setup to display SQL errors. As a result, the web server will show the error in the web application. Then, the attacker will know that there is an SQLi vulnerability. The attack is blind because the results are not directly visible and rely on the analysis of server responses to the injected SQL queries [23]. According to Acunetix [24], when attackers discover an SQLi vulnerability, they may try different requests to extract information about the database in the error responses. Blind SQLi attacks can also be used to build database schema, retrieve data from any table and escalate the attack. Because of this trend, web server administrators tend to remove detailed error messages. However, doing so does not solve the main problem, as the SQLi interpreter can still read users' inputs as part of an SQLi statement. Attackers overcame the lack of error messages by developing new methods of determining whether transactions are interpreted as SQL queries [24]. The two types of blind SQLi techniques are content and time-based [24].

3.4. Content-Based Blind SQLi

According to Acunetix [24], in a content-based blind SQLi, attackers send SQL queries that ask the database TRUE or FALSE questions. They then analyse the responses. Consider the following scenario (Acunetix [24]): an online shop's web page shows items for sale. Their link provides a description of item 26, retrieved from a database. The attacker can manipulate the request as follows:

`http://www.shop.com/item.php?id=26and1=2 (accessed on 8 May 2021)`

The SQL query changes as follows:

```
SELECT column_name_2 FROM table_name WHERE ID = 26 and 1=2 SELECT name,
description, price FROM Store_table WHERE ID = 26 and 1=2
```

This causes the query to return FALSE, and no items are displayed in the list. Then, the attacker proceeds to modify the request to:

`http://www.shop.com/item.php?id=26and1=1 (accessed on 8 May 2021)`

The SQL query changes as follows:

```
SELECT column_name, column_name_2 FROM table_name WHERE
ID = 26 and 1=1 SELECT name, description, price FROM Store_table
WHERE ID=26and 1=1
```

This query returns TRUE and display details about item 26. This is a clear indication that the page is vulnerable.

3.5. Time-Based Blind SQLi

According to Acunetix [24], in time-based blind SQLi attacks, attackers cause the database to perform a time-intensive operation. If the web application does not immediately return a response, then it is deemed vulnerable to blind SQLi. 'sleep' command is a typical time-intensive operation. As shown in the previous example, an attacker will first

test the response time of the web server for a standard query. The following command can then be injected [24]:

`http://www.shop.com/item.php?id=26 (accessed on 8 May 2021) and if(1=1, sleep(10), false)`

If the returned response is delayed by 10s, the web application is deemed vulnerable.

3.6. NoSQLi Injection

NoSQL (not only SQL) refers to non-relational databases that are growing in popularity as back-ends for distributed cloud platforms and web applications [25]. Unlike relational databases, NoSQL does not store data in tables. Instead, it uses other data models, such as graphs, documents and objects better suited for their particular purposes [25]. According to Sachdeva and Gupta [26], many companies have migrated to NoSQL because NoSQL databases provide looser consistency restrictions than traditional SQL databases [27]. Indeed, over the past few years, the popularity of NoSQL databases has grown consistently. For example, MongoDB database was ranked fifth among the 10 most popular databases in September 2020 (see Table 1) according to DB-Engine 2020 [28].

Table 1. Most popular databases.

Sep 2021	Database Management System
1	Oracle
2	MySQL
3	Microsoft SQLServer
4	PostgreSQL
5	MongoDB
6	IBM DB2
7	Redis
8	Elasticsearch
9	SQLite
10	Cassandra

Although NoSQL uses JavaScript Object Notation (JSON) query instead of SQL, that does not mean it is resistant to the threat of injection attacks [27]. Instead of using a standard query language, as with relational databases, NoSQL query syntax is product-specific, and commands are written in the application's programming language (e.g., Python, PHP, JavaScript, or Java). Consequently, a successful NoSQL injection attack will enable the attacker to execute a malicious command in the database and the application, which escalates the danger [25]. Hou et al. [27] reported that the NoSQL database system allows users to change data attributes at any time, and data can be added anywhere. In general, NoSQL attacks are like those of SQLi; only the grammar form changes. Because the attacker's command is inserted and executed on the server side and in the language of the web application, the impact of a successful NoSQL injection attack can be hazardous and allows for arbitrary code execution [25]. SQL and NoSQL query statements are shown below. We query the customer number as an example.

SQL Query:

```
"SELECT * FROM Customers WHERE (CustomerNo = ' " + Customer_Number + " '); ' "
```

NoSQL Query:

```
db.collection.find (CustomerNo:  Customer_Number)
```

As shown above, an attacker may input malicious codes into the input boxes in a web application, which can cause an injection attack.

3.7. Server-Side Template Injection

Template systems, such as Twig and FreeMarker, are widely used to embed dynamic content in web pages and emails [29]. Unsafely embedding user input in templates instead of passing it in as data can cause SSTIs [30]. PortSwigger [29] stated that the impact of SSTIs are more dangerous than a typical client-side template injection because the attacker payload is executed on the server side and explicitly targets the web servers inner processes. For example, an attacker may access and read sensitive data and arbitrary files on the server. According to PortSwigger [29], the template injection vulnerability can be caused by developer error or exposure to intentional templates designed to deliver rich functionality. To illustrate this risk, consider a marketing application that sends email messages to a group of subscribers to welcome them by name using a Twig template. As seen in the illustration below [29], when the name is passed directly to the template, everything works well:

```
    $output = $twig->render("Dear first_name,",
array("first_name" =>$user.first_name) );
```

However, problems occur when users are allowed to customize emails like this:

```
$output=$twig->render($_GET['custom_email'],
array("first_name" =>$user.first_name) );
```

In the above example, the user can control the content of the template via the custom_email GET parameter, rather than a value passed into it.

4. Our Approach towards Vulnerability Scanners

This section presents our approach in terms of selecting the black-box scanners.

4.1. Application Vulnerability Scanners

Doupé et al. [11] defined web application vulnerability scanners (WAVS) as "automated tools that are used to scan web applications and detect web vulnerabilities, also known as black-box vulnerability scanners". In addition, they are often known as point-and-shoot (PaS) penetration testing tools that test web applications automatically. Black et al. [31] provided a list of functional requirements that all web vulnerability scanners should meet:

- Can identify a specific set of security vulnerabilities in a web application;
- Can generate a text report describing the attack for each vulnerability identified;
- Has a low rate of false-positive results.

4.2. Architecture of Web Application Vulnerabilities Scanners

At a high-level, a WAVS consists of three modules: crawling, attacking and analysis. Below is a brief explanation of each.

- The crawling module uses a crawler to navigate a web application to identify and recover web pages, input vectors (e.g., input fields of hypertext markup language forms), GET/POST request parameters and cookies. Next, the crawler generates an indexed list of all accessed uniform resource locators (URLs). The detection of a web vulnerability ultimately depends on the quality of the crawler. If it is ineffective, the scanners will not be able to detect the vulnerability [11,32].
- The attacking (fuzzing) module uses a fuzzer to analyse the URLs and input vectors identified by the crawler then sends potential attack patterns to the entry points. The fuzzer creates a list of potentially vulnerable values to trigger a vulnerability for each entry and type. For example, the fuzzer component tries to inject JavaScript malicious code to test the presence of an XSS vulnerability [11,32].
- The analysis module analyses the results obtained in the previous step to detect existing vulnerabilities and provide other modules with comments. For example, if the returned page contains a database error message in response to the input tests for an SQLi injection. In that case, the analysis module will predict a potential SQLi vulnerability in that page [11,32].

4.3. Web Application Security Scanner Evaluation Criteria (WASSEC)

The Web Application Security Consortium developed WASSEC, a vendor-neutral document to help security professionals evaluate web application scanners and choose the most suitable tool [33]. The following list describes the features that should be considered when evaluating web application security scanners:

- Protocol Support: The scanner must support all communication protocols that are frequently used by web applications. Moreover, proxy capabilities, such as the hypertext transfer protocol(HTTP) and Socks proxies, should be supported.
- Authentication: The scanner should be able to maintain all authentication methods commonly used in a web application.
- Session Management: During a security scan, a scanner should maintain a valid session with the application.
- Crawling: The scanner should have a feature that can crawl a web application thoroughly based on the user-defined configuration.
- Parsing: To obtain information about the functionality and layout of the scanned web application, the scanner should be able to parse the most widely used web technologies.
- Testing: The scanner should be able to detect the security vulnerabilities and architectural flaws in a web application. It should also provide the user with configuration options to customize a scan.
- Command and control: The scanner should have command and control functions that enhance the user experience. For example, it schedules scans, pause and restart them, and schedule several scans simultaneously.
- Reporting: After each scan, a scanner should be able to produce a custom report.

4.4. Evaluation Metrics

Several evaluation metrics are used to measure the detection accuracy of black-box scanners.

- True positives (TPs) are the vulnerabilities detected by a scanner that truly exist in the code [34].
- False positives (FPs) are vulnerabilities detected by a scanner that do not exist [34]. FPs pose a significant problem to users. If the FPs high, the user inspects each reported vulnerability manually to assess its validity [17].
- False negatives (FNs) are the vulnerabilities that actually exist in the code but are not detected by the scanner [34].
- Precision is the ratio of correctly detected vulnerabilities to the total number of detected vulnerabilities, which is represented as follows [34,35]:

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

- Recall is the ratio of correctly detected vulnerabilities to the number of total existing vulnerabilities, which is represented as follows [34,35]:

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

- F-measure is the harmonic mean of precision and recall [36], which is represented as follows:

$$\text{F-Measure} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

5. Research Design

As depicted in Figure 1, our methodology consists of five steps. First, we review the available vulnerable applications and select the web application that meets our requirements. Second, we choose the black-box scanners to be evaluated. Third, we choose the

metrics used to measure the selected scanners' accuracy in detecting injection vulnerabilities. Fourth, we setup the environment to run the experiment. Fifth, we analyse the final results. Table 2 lists the general characteristics of the tested scanners.

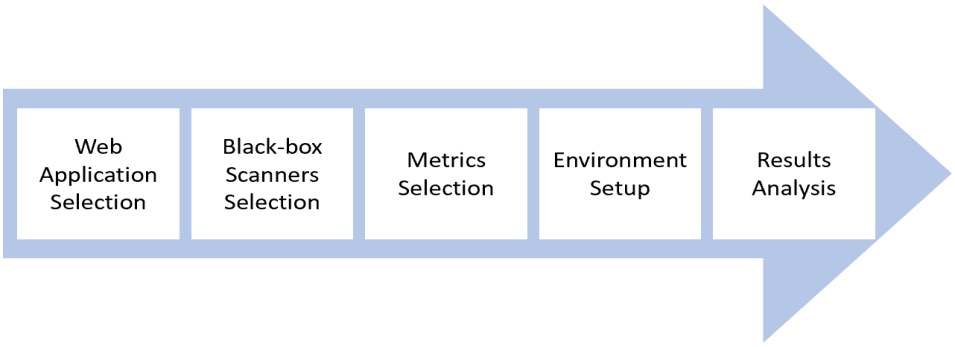

Figure 1. Our five-step approach.

Table 2. General characteristics of the tested scanners.

Scanner	Vendor	Version	License
Burp Suite Pro	PortSwiger	2020.7	Commercial
ZAP	OWASP	2.9.0	Apache v 2.0
Vega	Subgraph	1.0	Open-source
Skipfish	Google	2.10b	Free
Wapiti	Informática Gesfor	3.0.3	Open-source

5.1. Web Application Selection

The first step of our study chooses a web application with a list of known vulnerabilities to test the scanners. Two requirements were used to choose the vulnerable web application: (1) clearly defined vulnerabilities and (2) representative of modern web application technologies. After viewing the OWASP Vulnerable Web Applications Directory (VWAD) project, which maintains a list of all existing insecure web applications available [37], we found that the OWASP Juice Shop application [38] meets our requirements.

5.2. Black-Box Scanners Selection

There is a wide range of commercial and open-source scanners available each with its strengths and limitations. To select the scanners to be tested, we reviewed the currently available WAVS on the market. In this study, we aimed to evaluate both commercial and open-source scanners. We selected five black-box scanners, one commercial and four open-source: Burp Suite Professional, OWASP ZAP, Vega, Skipfish, and Wapiti. The scanners were selected based on their availability and ability to detect injection vulnerabilities. The following list provides a more detailed description of the selected scanners.

- ZAP is a free and open-source penetration testing tool for detecting vulnerabilities in web applications. It has a proxy feature for intercepting and inspecting messages sent between the client and the web application [39].
- Burp Suite Professional is a commercial web security tool that can be used to test all OWASP Top 10 vulnerabilities. It is capable of both passive and active analysis. In addition, its powerful proxy/history enables penetration testers to modify all secure HTTPS communications passing using the browser [40].
- Vega is a free and open-source tool for testing the security of web applications and detecting vulnerabilities. Moreover, it provides an automated scan for quick tests and has an intercepting proxy component [41].

- Skipfish is a free and open-source vulnerability scanner that prepares an interactive sitemap for the scanned web application. It runs repetitive crawls and dictionary-based scans. The obtained map is annotated with the output from several active scans [42].
- Wapiti is a free, open-source, and command-line application that scans the security of web applications. It conducts black-box scans by crawling the web pages of the target web application and looks for forms into which it may inject data. When the list of URLs, forms and their inputs has been collected, Wapiti acts like a fuzzer and injects payloads to see whether there is a potential vulnerability [43].

5.3. Metrics Selection

To determine the vulnerability detection accuracy of the evaluated scanners, we calculated the number of TPs, FPs and FNs produced by each scanner. Furthermore, three metrics were calculated: precision, recall, and F-measure. These metrics were selected based on Antunes and Vieira's [44] recommendation to use recall to assess which tool detects the highest number of vulnerabilities or, to put it another way, which leaves the feweest vulnerabilities undetected. Precision is the recommended tiebreaker. Antunes and Vieira [44] suggested using the F-measure to select a tool that detects a high number of vulnerabilities while reporting a low number of FPs; recall is the recommended tiebreaker.

6. Experimental Setup and Execution

This section presents key details about the experiments and their execution.

6.1. Experimental Design

The following briefly describes the experimental workflow, which is illustrated in Figure 2.

1. We started every scan by choosing from the four scanning modes.
2. We ran the target web application (OWASP Juice Shop).
3. We connected the target web application to the evaluated scanner.
4. The crawler component of the scanner then explored the web application pages and identified forms and entry points.
5. The scanner sent potential payload attacks to the entry points.
6. The analysis module of the scanner generated analytical reports.
7. We analysed these reports manually.

Because we had four modes, these steps were repeated four times for each scanner if the scanner had a proxy component. Otherwise, the scanner ran on two modes only.

6.2. Owasp Juice Shop Application

OWASP Juice Shop is an open-source project created by Björn Kimminich and hosted by the OWASP. It was written in Node.js, Angular and Express [45] and was the first application written entirely in JavaScript in the OWASP VWAD. Although Juice Shop is an intentionally vulnerable web application that was created for awareness and training, it gives the impression of a functionally complete e-commerce web application that could exist in the real world. Juice Shop emulates a small online shop that offers fruit and vegetable juice and related products. Users can create an account, log in, place an order, write and read product reviews, track the order and more. Figure 3 presents the architecture. In addition to the heavy use of JavaScript, which distinguishes Juice Shop from other vulnerable applications, it also uses the latest web technology. For example, the common Angular framework is used in the front end to create a single-page application. In terms of authentication, Juice Shop uses OAuth 2.0, which enables users to sign in with their Google accounts. Moreover, when users successfully logs in using their credentials, a JSON web token (JWT) is returned and is included in subsequent requests, allowing the user to access services that are only authorised with that token. In terms of data storage, a file-based SQLite engine is used as the primary database. MarsDB, a JavaScript derivative of the

widely used MongoDB NoSQL database, is also used for additional data storage. JavaScript is the primary programming language in the back end. Additionally, the necessary back-end functionality of the application is delivered to the client via a RESTful API.

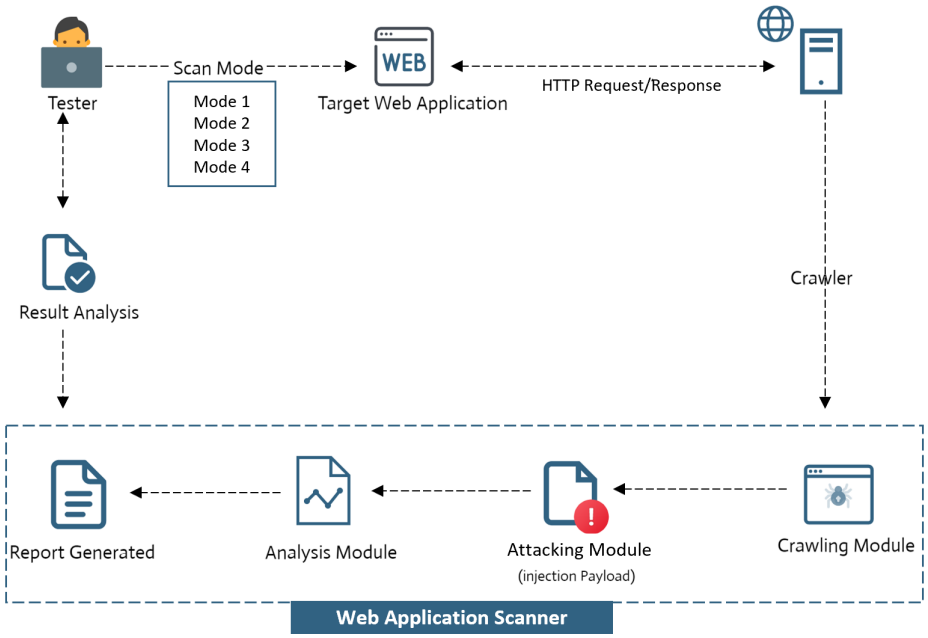

Figure 2. Experimental workflow.

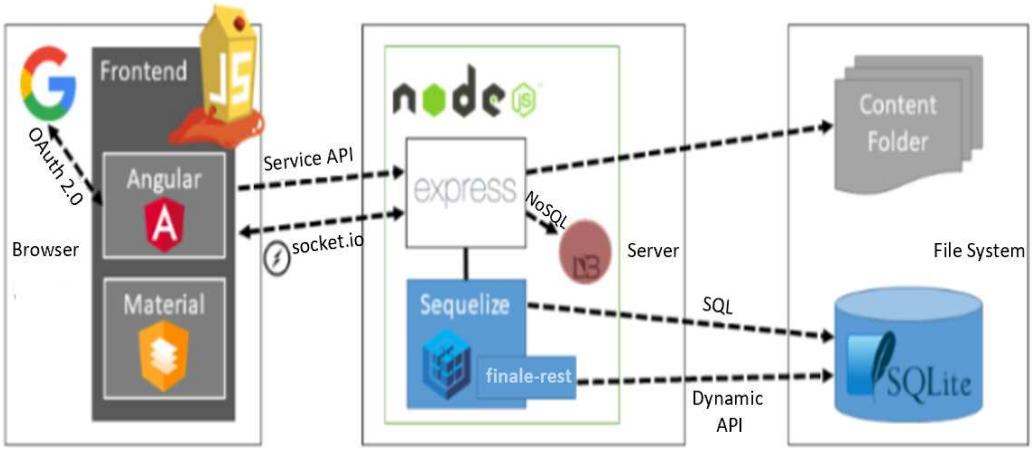

Figure 3. OWASP Juice Shop application architecture [45].

6.3. Owasp Juice Shop Tested Vulnerabilities

Most vulnerabilities in the OWASP Juice Shop application were derived from well-known lists or documents, including the OWASP Top 10, OWASP API Security Top 10, and MITRE's Common Weakness Enumeration [45].

The OWASP Juice Shop application contains a variety of vulnerabilities that are categorised into 14 types (see Figure 4) [45]. However, in this study, we only tested the injection vulnerability because it was ranked first by the OWASP (2017) [4] report for the top 10 web application vulnerabilities. Furthermore, injection attacks can severely impact the data of the hosted web application, including data loss or corruption, financial loss and even the complete loss of control of the system [22]. The OWASP Juice Shop application contains seven injection vulnerabilities, particularly SQLi, NoSQL and SSTI. Table 3 provides details on the tested vulnerabilities.

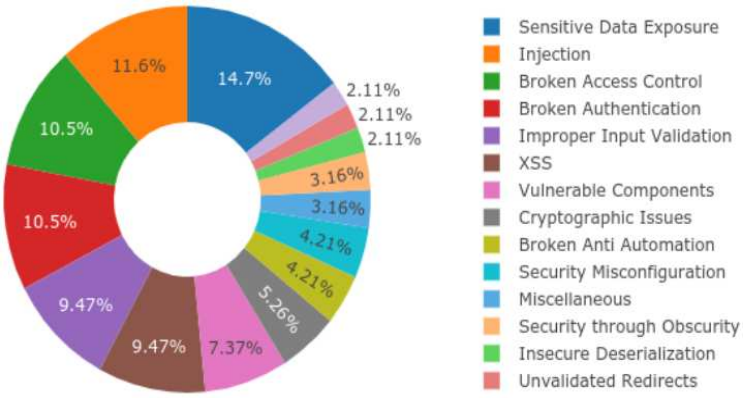

Figure 4. OWASP Juice Shop vulnerability categories [45].

Table 3. Injection vulnerabilities in OWASP Juice Shop.

Vulnerability	Number	Login Required
Blind SQLi	1	0
Union-based SQLi Injection	1	0
SQLi Injection	1	0
NoSQL Injection	3	2
SSTi	1	1
Total Number of Vulnerabilities	7	

6.4. Web Application Vulnerability Scanner Modes

Each scanner ran in four modes to ensure that the entire web application was mapped included pages requiring authentication as some vulnerabilities could only be accessed by an authenticated user and to determine whether a scanner failed to detect a vulnerability because it could not access the vulnerable page due to its poor crawling ability or because it was unable to detect this type of vulnerability.

In In mode 1, we only gave the scanner the OWASP Juice Shop URL. However in mode 2, we gave the scanner the URL and valid login credentials. In mode 3, we provided the scanner with the URL of the application. Moreover, if the scanner had a proxy component, it was used in the third mode to manually view all the web application pages that do not require authentication. In mode 4, we provided the scanner with the application's URL and valid login credentials. In this mode, if the scanner had an intercepting proxy, it was used to access all pages manually, including those requiring authentication. More details about the applied algorithms in the four modes are listed below (Algorithms 1–4).

Algorithm 1 Mode 1: Point-and-shoot (PaS)

Description: The scanner is only provided with the OWASP Juice Shop application's entry URL for scanning.
Input
OWASP Juice Shop URL: http://localhost:3000 (accessed on 8 May 2021)
Output
Scan report
Start
Set the scanner to work with a browser
Launch the browser
Open the OWASP Juice Shop application
Add the application URL to the target scope
Run the crawler
Run the active scan
End

Algorithm 2 Mode 2: Unauthenticated scan with proxy

Description: As in mode 1, the scanner is provided with the application's entry URL, with the exception of using a proxy component (if available) to visit all the application pages that do not need authentication manually and to fill in any forms.
Input
OWASP Juice Shop URL: http://localhost:3000 (accessed on 8 May 2021)
Output
Scan report
Start
Set the scanner to work with a browser
Launch the browser
Open the OWASP Juice Shop application
Add the application URL to the target scope
Run the crawler
Use proxy to visit each unauthenticated page
Fill in any forms
Run the active scan
End

Algorithm 3 Mode 3: Point-and-shoot with user credentials

Description: The scanner is given the OWASP Juice Shop application entry URL and provided with valid login credentials (such as username and password or authentication token) to help the scanner access the authenticated pages.
Input
OWASP Juice Shop URL: http://localhost:3000 (accessed on 8 May 2021)
User credentials
Output
Scan report
Start
Set the scanner to work with a browser Launch the browser
Open the OWASP Juice Shop application
Add the application URL to the target scope
Log in the application with legitimate credentials
Configure the scanner using user credentials in scanning
Run the crawler
Run the active scan
End

Algorithm 4 Mode 4: Authenticated scan with proxy

Description: As in mode 2, the scanner is provided with the application's URL and valid login credentials to help the scanner access the authenticated pages. Moreover, if the scanner has a proxy component, it is used to visit every page manually, including pages that need authentication, and to fill in any forms.
Input
OWASP Juice Shop URL: http://localhost:3000 (accessed on 8 May 2021)
User credentials
Output
Scan report
Start
Set the scanner to work with a browser
Launch the browser
Open the OWASP Juice Shop application
Add the application URL to the target scope
Log in the application with legitimate credentials
Configure the scanner using user credentials
Run the crawler
Use a proxy to visit every page in the application
Fill in any forms Run the active scan
End

7. Results Analysis

This section presents the results obtained and analysed from experimental testing.

7.1. Experimental Results

This section presents the results of running the five scanners in four different modes. We analysed the alerts and reports generated by each scanner and converted them manually into FN, TP and FP values. Thus, any vulnerability missed by a scanner was considered an FN. Each vulnerability that was indeed detected and reported by a scanner was considered a TP. If the scanner detected an injection vulnerability that did not exist in the application, it was considered an FP. Next, we present the FN, TP and FP values for all scanners in each mode.

7.2. Mode 1 Results

The results of running the scanners in this mode are listed in Table 4.

- FNs: It is clear from Table 4 that the number of undetected vulnerabilities (FN) was significantly higher than the detected (TP) and wrongly detected (FP) values in this mode.
- TPs: As we can see from Table 4, most scanners failed to detect any known vulnerability, apart from the ZAP scanner, which found one SQLi injection vulnerability in the home page.
- FPs: Because we only scanned injection vulnerabilities, no scanner detected any injection vulnerability that did not really exist.

Table 4. FN, TP, and FP results in mode 1.

Scanner	FN	TP	FP
ZAP	6	1	0
Burp Suite	7	0	0
Vega	7	0	0
Skipfish	7	0	0
Wapiti	7	0	0

7.3. Mode 2 Results

The results of running the scanners in this mode are shown in Table 5.

- FNs: The number of undetected vulnerabilities (FNs) in this mode was the same as in Mode 1, with the exception of the Burp Suite Professional scanner, which detected one vulnerability.Thus, the number of FNs decreased by one.
- TPs: In Mode 2, both the ZAP and Burp Suite scanners detected only one vulnerability. The one detected by ZAP was already found during Mode 1 scanning. The Burp Suite was likely able to detect the vulnerability because we filled in the login form manually to provide the scanner with privileged credentials; then, the scanner was able to access the vulnerable login page.
- FPs: None of the scanners detected any injection vulnerabilities that did not really exist in the application.

Table 5. FN, TP, and FP results in mode 2.

Scanner	FN	TP	FP
ZAP	6	1	0
Burp Suite	6	1	0
Vega	7	0	0
Skipfish	7	0	0
Wapiti	7	0	0

7.4. Mode 3 Results

The results of running the scanners in this mode are shown in Table 6

- FNs: ZAP scanner achieved the fewest number of FNs (five).
- TPs: Table 6 shows that using the scanner in a proxy mode to access the unauthenticated pages in the application manually increased the number of detected vulnerabilities by one for the ZAP scanner only. However, other scanners that had a proxy (i.e.Burp Suite Professional and Vega) did not detect any, even when we manually accessed the vulnerable pages.
- FPs: None of the scanners detected any injection vulnerabilities that did not really exist in the application.

Table 6. FN, TP, and FP results in mode 3.

Scanner	FN	TP	FP
ZAP	5	2	0
Burp Suite	7	0	0
Vega	7	0	0
Skipfish	-	-	-
Wapiti	-	-	-

7.5. Mode 4 Results

The results of running the scanners in this mode are displayed in Table 7.

- FNs: Vega had the highest number of undetected vulnerabilities. In contrast, ZAP and Burp Suite had the fewest FN values (five each).
- TPs: Both ZAP and Burp Suite detected two injection vulnerabilities.
- FPs: None of the scanners detected any injection vulnerabilities that did not really exist in the application.

Table 7. FN, TP, and FP results in mode 4.

Scanner	FN	TP	FP
ZAP	5	2	0
Burp Suite	5	2	0
Vega	7	0	0
Skipfish	-	-	-
Wapiti	-	-	-

7.6. Results Summary

In Mode 1, where only the application URL was given to the scanner, ZAP scanner was the only scanner that detected a vulnerability (i.e., SQLi) which was located on the application home page. In contrast, all other scanners could not detect any vulnerabilities.

In Mode 2, ZAP reported the same vulnerability that was detected in Mode 1, even when we provided the scanner with valid user session tokens. This indicates that the scanner could not use the session tokens properly for authentication. Moreover, the Burp Suite detected the SQLi on the login page. In Modes 1 and 2, Vega, Skipfish, and Wapiti scanners could not detect any vulnerabilities, including the one on the home page.

In Mode 3, when we accessed the web pages manually through a proxy, ZAP was able to find two injection vulnerabilities on the login page. Although Burp suite and Vega had a proxy, they could not detect any vulnerabilities.

In Mode 4, the Burp Suite found SQLi vulnerability in the email field, which was already discovered in Mode 2. Additionally, the Burp Suite detected a NoSQL vulnerability in the http://localhost:3000/#/track-order (accessed on 8 May 2021). Page that was defined as a server-side JavaScript code injection. Like ZAP, the Burp Suite could not use the login credentials to access the authenticated pages. Even when a proxy accessed all application pages, other vulnerabilities were missed by ZAP and Burp Suite Professional. ZAP detected the same vulnerabilities that were already found in previous modes. In contrast, Vega overlooked all injection vulnerabilities. Skipfish and Wapiti hsd no proxy component to test. Table 8 lists the types of vulnerabilities detected and not detected by the various scanners.

Table 8. Types of detected and undetected vulnerabilities.

	Burp Suite	ZAP	Vega	Skipfish	Wapiti
Blind SQLi	❍	●	❍	❍	❍
Union-based SQLi	❍	❍	❍	❍	❍
SQLi	●	●	❍	❍	❍
NoSQL	◗	❍	❍	❍	❍
SSTi	❍	❍	❍	❍	❍

●: All vulnerabilities are detected, ❍: none of the vulnerabilities are detected, ◗: some vulnerabilities are detected and some are not.

8. Evaluation

This section assesses the vulnerability detection accuracy of the tested scanners by calculating the precision, recall, and F-measure metrics.Moreover, this section discusses the limitations and the key gaps of the scanners considered in this work.

8.1. Precision, Recall, and F-Measure

Based on the obtained FN, TP and FP results, precision, recall and F-measures of each scanner were calculated (see Table 9). The best result obtained from each scanner is highlighted.

According to Antunes and Vieira [34], the lower the number of FPs, the higher the precision. Consequently, the scanner can detect vulnerabilities more accurately. In contrast, the higher the recall, the lower the number of FNs. Consequently, the scanner can detect vulnerabilities more accurately [34]. Precision refers to the percentage of correct relevant vulnerabilities compared to the total number of detected vulnerabilities [15]. As can be seen in Table 9, the Burp Suite Professional and ZAP scanners had the highest precision (100%), whereas the others had the lowest precision (0%). The recall is the percentage of the correctly identified vulnerabilities compared to the total number of actual vulnerabilities [15]. From Table 9, we can see that the recall of the Burp Suite Professional in Mode 4 and ZAP in Modes 3 and 4 occupies a higher percentage (29%) than other scanners (0%) in all modes. The last column of Table 9 shows the F-measure of each scanner in detecting injection vulnerabilities. According to Idrissi et al. [15], the F-measure metric indicates each scanner's effectiveness as it incorporates precision and recall into a single measure. As can be seen in Table 9, the Burp Suite Professional in Mode 3 and ZAP in Modes 3 and 4 had the same F-measure and the highest efficiency (44%) compared with the others. Vega, Skipfish and Wapiti had F-measures of 0% in all modes. Figure 5 demonstrates the different modes' precision, recall and F-measure results.

Table 9. Precision, recall and F-measure values for all scanners.

Mode	Scanner	Precision%	Recall%	F-Measure%
Mode 1	ZAP	100%	14%	25%
	Burp Suite Pro	0%	0%	0%
	Vega	0%	0%	0%
	Skipfish	0%	0%	0%
	Wapiti	0%	0%	0%
Mode 2	ZAP	100%	14%	25%
	Burp Suite Pro	100%	14%	25%
	Vega	0%	0%	0%
	Skipfish	0%	0%	0%
	Wapiti	0%	0%	0%
Mode 3	ZAP	100%	29%	44%
	Burp Suite Pro	0%	0%	0%
	Vega	-	-	-
	Skipfish	-	-	-
	Wapiti	-	-	-
Mode 4	ZAP	100%	29%	44%
	Burp Suite Pro	100%	29%	44%
	Vega	-	-	-
	Skipfish	-	-	-
	Wapiti	-	-	-

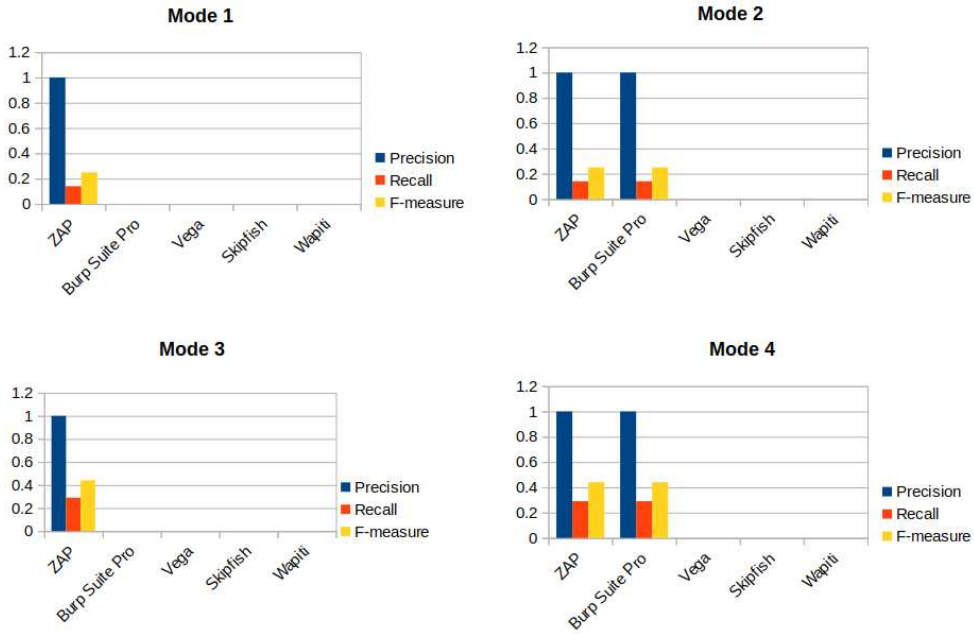

Figure 5. Obtained results of precision, recall, and F-measure in different modes.

8.2. Discussion: Finding Key Gaps

This section discusses the potential reasons for the scanners' limitations. The first thing worth noting is that black-box scanners aim to solve a challenging problem within a challenging task [7]. Thus, they are likely to miss many vulnerabilities. Our results demonstrated that two scanners, namely ZAP and Burp Suite Professional, detected no more than two injection vulnerabilities out of seven known ones. Approximately 72% of the existing vulnerabilities went undetected. In contrast, Vega, Skipfish and Wapiti scanners overlooked all vulnerabilities.

No scanner detected the vulnerabilities, even when the vulnerable pages were manually visited through a proxy. As stated by Kimminich [45], OWASP Juice Shop uses a MongoDB derivate as its NoSQL database, which is vulnerable to injection attacks. The OWASP Juice Shop has three NoSQL vulnerabilities, two of which were not detected by any scanner. The identification of these vulnerabilities by an attacker can lead to manipulations, the updating of multiple product reviews at the same time, and denial-of-service attacks via the injection of malicious commands into the URL. Moreover, the SSTI vulnerability was missed by all scanners.

From these results, it is clear that the scanners failed to detect most vulnerabilities. This appears to be mainly caused by two reasons. First, the scanners had a limited ability to crawl the application because OWASP Juice Shop, like many modern web applications, was created dynamically using JavaScript, which is a big challenge for crawlers. Doupé et al. [11] also stated that modern web applications present crawling challenges to black-box scanners. As a result, many vulnerable forms are overlooked during poor crawling. The evaluated scanners had varying crawling capabilities. For instance, ZAP and Burp Suite Professional had the best ability to crawl dynamic JavaScript. Consequently, they had higher accuracies of vulnerability detection. Thus, providing scanners with a proxy component could help them detect more vulnerabilities. There is no doubt that the effectiveness of black-box scanners in crawling modern web applications is improving; however, progress is slow [7]. Second, another reason for missing vulnerabilities was that although the black-box scanners supported the detection of injection vulnerabilities, they could not detect all injection issues. For instance, the ZAP scanner did not support SSTI detection, and Burp Suite Professional did not support NoSQL.

Although we did not evaluate the authentication feature of the scanners, we found that all failed to maintain an authenticated state. When we conducted authenticated scans in Modes 2 and 4, nearly all scanners could not use the given privileged credentials properly to indicate the user's identity. As a result, none could access the authenticated pages. The ability of scanners to maintain JWT authentication is interesting and should be evaluated thoroughly in future studies. Although our web vulnerability scanner evaluation included five scanners, our results may not be generalisable to all other black-box scanners.

9. Comparison with Previous Work

Several researchers have tested black-box scanners against vulnerable web applications.

Doupé et al. [11] evaluated 11 black-box scanners against the WackoPicko application. The authors ran the scanners in three modes: initial, configured, and manual. In the initial mode, the scanners were operated in a PaS mode and were given valid login credentials. In the last mode, the scanners were set to a proxy mode, whereas the application pages were browsed manually. Doupé et al. [11] reported that it is a significant challenge for scanners to crawl modern web applications. Various vulnerabilities were detected only in the manual mode. Doupé et al. [11] also reported that the low crawling coverage was likely due to the web technologies used in the application. Although we tested different scanners and web applications, our findings are similar to those of Doupé et al. [11], where the scanners overlooked at least 50% of the vulnerabilities. However, in our study, the percentage of undetected vulnerabilities was considerably higher.

Idrissi et al. [15] assessed 11 black-box scanners using the same evaluation metrics applied in our study. However, they used a different application, WAVSEP, to measure the scanners' ability to detect SQLi, reflected XSS, remote file inclusion and path traversal/local file inclusion vulnerabilities. Therefore, we only compared our results to their SQLi results. In Idrissi et al.'s study [15], the F-measure for all scanners was between 70 and 100%; in ours, it was between 0 and 44%, which is considerably lower. Concerning recall, in Idrissi et al.'s study [15], it was between 60 and 100%; in ours, it was between 0 and 29%. The significant difference between our findings and those of Idrissi et al. [15] is not surprising, as OWASP Juice Shop is a much more challenging application for scanners than WAVSEP. Furthermore,

we only considered Idrissi et al.'s [15] SQLi results against all injection vulnerabilities in our study.

Makino and Klyuev [8] evaluated two open-source vulnerability scanners: OWASP ZAP and Skipfish using the Damn Vulnerable Web Application (DVWA) and WAVSEP as test applications. The authors tested the scanners against a list of vulnerabilities, including SQLi, blind SQLi, reflected XSS, persistent XSS, local file injection, remote file injection, command execution and cross-site request forgery. Our results are consistent with those of Makino and Klyuev [8] in some aspects. First, OWASP ZAP was found to be superior to Skipfish in detecting vulnerabilities. Second, although the scanners detected some injection vulnerabilities, such as SQLi, they missed others.

Khoury et al. [12] reached a similar conclusion after evaluating three black-box scanners that supported persistent SQLi vulnerability detection. For this purpose, the authors built a custom application, called MatchIt, finding that the scanners could not identify existing vulnerabilities. Khoury et al. [12] emphasized that configuring a scanner with a username and password could enhance the overall results because pages that require authentication are more likely to be accessed. In comparison, our study tested black-box scanners against one of the most modern and sophisticated insecure web applications, representing a real-life e-commerce web application. Furthermore, we explored four modes to identify the reasons for the scanner limitations. We hope that this study will provide valuable insights into how black-box scanners' ability to detect injection vulnerabilities might be improved.

10. Conclusions and Future Work

This study evaluated the detection accuracy of black-box scanners against SQLi, NoSQL and SSTI injection vulnerabilities. To achieve this, we evaluated the detection capability of five black-box web scanners against one of the most modern and vulnerable web applications. We measured the vulnerability detection accuracy feature of each scanner using three evaluation metrics: precision, recall and F-measure. We found that ZAP and Burp Suite Professional were superior. In addition, the evaluated black-box scanners overlooked most existing vulnerabilities in most modes, and some scanners could not detect any. This appeared to be caused by two reasons. First, scanners have a limited ability to crawl dynamic modern web applications. Thus, providing scanners with a proxy component helps them detect more existing vulnerabilities. Second, although the black-box scanners support detecting injection vulnerabilities, they cannot detect all types of injection flaws. In future work, improvements should be made from the following perspectives:

1. Evaluate more black-box scanners. In this paper, we evaluated only five. However, there are a growing number on the market. Therefore, future studies can scan the same application, OWASP Juice Shop, with a different selection of the most widely used scanners.
2. Test other vulnerabilities. The OWASP Juice Shop application contains 14 categories of vulnerabilities and includes some that no research has ever tested. In this paper, we focused only on injection vulnerabilities. However, future research should include other types.
3. Analyse the identified vulnerability detection limitations. Because our work evaluated open-source scanners, other researchers now have the opportunity to access and review the source code. Therefore, future studies should examine the design issues and limitations in greater detail.

Author Contributions: Conceptualization, M.A. and N.S.; methodology, M.A., N.S. and S.L.; validation, M.A., N.S. and P.G.; writing—original draft preparation, M.A. and N.S.; writing—review and editing, M.A., N.S., S.L. and P.G.; supervision, N.S.; All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Internet Users. Available online: https://www.internetlivestats.com/watch/internet-users (accessed on 8 May 2021).
2. Total Number of Websites. Available online: https://www.internetlivestats.com/watch/websites (accessed on 8 May 2021).
3. Verizon. 2020 Data Breach Investigations Report. Technical Report, Verizon, 2020. Available online: https://www.cisecurity.org/wp-content/uploads/2020/07/The-2020-Verizon-Data-Breach-Investigations-Report-DBIR.pdf (accessed on 8 May 2021).
4. OWASP. *OWASP Top 10 -2017 the Ten Most Critical Web Application Security Risks*; Technical Report; OWASP: Wakefield, MA, USA, 2017.
5. State of the Internet Security. Available online: https://www.akamai.com/uk/en/multimedia/documents/state-of-the-internet/soti-security-a-year-in-review-report-2019.pdf (accessed on 8 May 2021).
6. Braga, M. What We Know about the Equifax Breach—And What We Don't. 2017. Available online: https://www.cbc.ca/news/science/equifax-canada-breach-sin-cybersecurity-what-we-know-1.4297532 (accessed on 8 May 2021).
7. Khalil, R.F. Why Johnny Still Can't Pentest: A Comparative Analysis of Open-Source Black-box Web Vulnerability Scanners. Ph.D. Thesis, Université d'Ottawa/University of Ottawa, Ottawa, ON, Canada, 2018.
8. Makino, Y.; Klyuev, V. Evaluation of web vulnerability scanners. In Proceedings of the 2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Warsaw, Poland, 24–26 September 2015; Volume 1, pp. 399–402.
9. Suto, L. Analyzing the Accuracy and Time Costs of Web Application Security Scanners. 2010. Available online: http://www.think-secure.nl/uk/Accuracy_and_Time_Costs_of_Web_App_Scanners.pdf (accessed on 8 May 2021).
10. Bau, J.; Bursztein, E.; Gupta, D.; Mitchell, J. State of the art: Automated black-box web application vulnerability testing. In Proceedings of the 2010 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 16–19 May 2010; pp. 332–345.
11. Doupé, A.; Cova, M.; Vigna, G. Why Johnny can't pentest: An analysis of black-box web vulnerability scanners. In Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Cagliari, Italy, 14–16 July 2021; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6201, pp. 111–131.
12. Khoury, N.; Zavarsky, P.; Lindskog, D.; Ruhl, R. Testing and Assessing Web Vulnerability Scanners for Persistent SQL Injection Attacks. In Proceedings of the First International Workshop on Security and Privacy Preserving in E-Societies, New York, NY, USA, 9 June 2011; Association for Computing Machinery: New York, NY, USA, 2011; pp. 12–18. [CrossRef]
13. Khoury, N.; Zavarsky, P.; Lindskog, D.; Ruhl, R. An analysis of black-box web application security scanners against stored SQLi injection. In Proceedings of the 2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing, Boston, MA, USA, 9–11 October 2011; pp. 1095–1101.
14. Parvez, M.; Zavarsky, P.; Khoury, N. Analysis of effectiveness of black-box web application scanners in detection of stored SQLi injection and stored XSS vulnerabilities. In Proceedings of the 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST), London, UK, 14–16 December 2015; pp. 186–191.
15. Idrissi, S.; Berbiche, N.; Guerouate, F.; Shibi, M. Performance evaluation of web application security scanners for prevention and protection against vulnerabilities. *Int. J. Appl. Eng. Res.* **2017**, *12*, 11068–11076.
16. Sutton, M.; Greene, A.; Amini, P. *Fuzzing: Brute Force Vulnerability Discovery*, 3rd ed.; Pearson Education: London, UK, 2007.
17. Doupé, A.; Cavedon, L.; Kruegel, C.; Vigna, G. Enemy of the state: A state-aware black-box web vulnerability scanner. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12), Bellevue, WA, USA, 11–13 August 2021*; USENIX Association: Bellevue,WA, USA, 2012; pp. 523–538.
18. Muñoz, F.R.; Cortes, I.I.S.; Villalba, L.J.G. Enlargement of vulnerable web applications for testing. *J. Supercomput.* **2017**, *74*, 6598–6617. [CrossRef]
19. OWASP Foundation. Available online: https://owasp.org (accessed on 8 May 2021).
20. SQL Injection. Available online: https://owasp.org/www-community/attacks/SQL_Injection (accessed on 8 May 2021).
21. SQL Injection UNION Attacks. Available online: https://portswigger.net/web-security/sql-injection/union-attacks (accessed on 8 May 2021).
22. Al-Khurafi, O.B.; Al-Ahmad, M.A. Survey of web application vulnerability attacks. In Proceedings of the 2015 4th International Conference on Advanced Computer Science Applications and Technologies (ACSAT), Kuala Lumpur, Malaysia, 8–10 December 2015; pp. 154–158.
23. Banach, Z. How Blind SQL Injection Works. 2020. Available online: https://www.invicti.com/blog/web-security/how-blind-sql-injection-works/ (accessed on 8 May 2021).
24. Acunetix. What Are Blind SQL Injections. Available online: https://www.acunetix.com/websitesecurity/blind-sql-injection/ (accessed on 8 May 2021).
25. Banach, Z. What Is NoSQL Injection and How Can You Prevent It? 2020. Available online: https://www.invicti.com/blog/web-security/what-is-nosql-injection/ (accessed on 8 May 2021).

26. Sachdeva, V.; Gupta, S. Basic NOSQL Injection Analysis And Detection On MongoDB. In Proceedings of the 2018 International Conference on Advanced Computation and Telecommunication (ICACAT), Bhopal, India, 28–29 December 2018; pp. 1–5.
27. Hou, B.; Qian, K.; Li, L.; Shi, Y.; Tao, L.; Liu, J. MongoDB NoSQL injection analysis and detection. In Proceedings of the 2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud), Beijing, China, 25–27 June 2016; pp. 75–78.
28. Knowledge Base of Relational and NoSQL Database Management Systems. Available online: https://db-engines.com/en (accessed on 8 May 2021).
29. Server-Side Template Injection. Available online: https://portswigger.net/web-security/server-side-template-injection (accessed on 8 May 2021).
30. Kettle, J. Server-Side Template Injection. 2015. Available online: https://portswigger.net/research/server-side-template-injection (accessed on 8 May 2021).
31. Black, P.E.; Fong, E.; Okun, V.; Gaucher, R. Software assurance tools: Web application security scanner functional specification version 1.0. *Nist Spec. Publ.* **2008**, *2008*, 269–500.
32. Kagorora, F.; Li, J.; Hanyurwimfura, D.; Camara, L. Effectiveness of web application security scanners at detecting vulnerabilities behind ajax/json. *Int. J. Innov. Res. Sci. Eng. Technol.* **2015**, *4*, 4179–4188.
33. Web ApplicationSecurity Consortium and others; Web application security scanner evaluation criteria. *Web Appl. Secur. Consort.* **2009**, *1*, 1–26.
34. Antunes, N.; Vieira, M. Benchmarking vulnerability detection tools for web services. In Proceedings of the 2010 IEEE International Conference on Web Services, Miami, FL, USA, 5–10 July 2010; pp. 203–210.
35. Antunes, N.; Vieira, M. Assessing and comparing vulnerability detection tools for web services: Benchmarking approach and examples. *IEEE Trans. Serv. Comput.* **2014**, *8*, 269–283. [CrossRef]
36. Powers, D.M. Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. *arXiv* **2011**, arXiv:2010.16061v1.
37. Siles, R.; Bennetts, S. OWASP Vulnerable Web Application Directory Project. 2019. Available online: https://owasp.org/www-project-vulnerable-web-applications-directory/ (accessed on 8 May 2021).
38. Kimminich, B. *OWASP Juice Shop Project*; Technical report; OWASP: Bel Air, MD, USA, 2020.
39. OWASP Zed Attack Proxy (ZAP). Available online: https://www.zaproxy.org (accessed on 8 May 2021).
40. Burp Suite Scanner. Available online: https://portswigger.net/burp/vulnerability-scanner (accessed on 8 May 2021).
41. Vega Vulnerability Scanner. Available online: https://subgraph.com/vega/ (accessed on 8 May 2021).
42. skipfish(1)—Linux Man Page. Available online: ps://linux.die.net/man/1/skipfish (accessed on 8 May 2021).
43. The Web-Application Vulnerability Scanner. Available online: https://wapiti.sourceforge.io (accessed on 8 May 2021).
44. Antunes, N.; Vieira, M. On the metrics for benchmarking vulnerability detection tools. In Proceedings of the 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Rio de Janeiro, Brazil, 22–25 June 2015; pp. 505–516.
45. Kimminich, B. Pwning OWASP Juice Shop. 2017. Available online: https://pwning.owasp-juice.shop/ (accessed on 8 May 2021).