Enhancing web application security through automated penetration testing with multiple vulnerability scanners.

Abdulghaffar, Khaled ; Elmrabit, Nebrase; Yousefi, Mehdi

Published in:
Computers

DOI:
10.3390/computers12110235

Publication date:
2023

Document Version
Publisher's PDF, also known as Version of record

computers

MDPI

Article

Enhancing Web Application Security through Automated Penetration Testing with Multiple Vulnerability Scanners

Khaled Abdulghaffar [1], Nebrase Elmrabit [1,*] and Mehdi Yousefi [2]

[1] Department of Cyber Security and Networks, Glasgow Caledonian University, Glasgow G4 0BA, UK; kabdul203@caledonian.ac.uk
[2] School of Computing and Digital Technology, Birmingham City University, Birmingham B4 7XG, UK; mehdi.yousefi@bcu.ac.uk
* Correspondence: nebrase.elmrabit@gcu.ac.uk; Tel.: +44-141-273-1940

Abstract: Penetration testers have increasingly adopted multiple penetration testing scanners to ensure the robustness of web applications. However, a notable limitation of many scanning techniques is their susceptibility to producing false positives. This paper presents a novel framework designed to automate the operation of multiple Web Application Vulnerability Scanners (WAVS) within a single platform. The framework generates a combined vulnerabilities report using two algorithms: an automation algorithm and a novel combination algorithm that produces comprehensive lists of detected vulnerabilities. The framework leverages the capabilities of two web vulnerability scanners, Arachni and OWASP ZAP. The study begins with an extensive review of the existing scientific literature, focusing on open-source WAVS and exploring the OWASP 2021 guidelines. Following this, the framework development phase addresses the challenge of varying results obtained from different WAVS. This framework's core objective is to combine the results of multiple WAVS into a consolidated vulnerability report, ultimately improving detection rates and overall security. The study demonstrates that the combined outcomes produced by the proposed framework exhibit greater accuracy compared to individual scanning results obtained from Arachni and OWASP ZAP. In summary, the study reveals that the Union List outperforms individual scanners, particularly regarding recall and F-measure. Consequently, adopting multiple vulnerability scanners is recommended as an effective strategy to bolster vulnerability detection in web applications.

Keywords: web applications; web application cyber security; vulnerability scanners; automate penetration testing

Citation: Abdulghaffar, K.; Elmrabit, N.; Yousefi, M. Enhancing Web Application Security through Automated Penetration Testing with Multiple Vulnerability Scanners. *Computers* **2023**, *12*, 235. https://doi.org/10.3390/computers12110235

Academic Editors: Ömer Aslan and Refik Samet

Received: 9 October 2023
Revised: 7 November 2023
Accepted: 10 November 2023
Published: 15 November 2023

1. Introduction

Discovering Web Application Vulnerabilities (WAV) before cyber attacks occur is a continuous challenge to organisations [1]. Furthermore, developing a secure web application without vulnerabilities is a challenge for security analysts. This makes web applications an attractive target for cyber attacks, with an average of 10 million web attacks per day [2], a figure that will only increase the number of users of web applications increases [3].

A WAV is defined as "a flaw in the application that stems from coding defects and causes severe damage to the application upon exploitation" [4]. To identify and mitigate vulnerabilities that may be exploited by attackers, a penetration testing method or ethical hacking is used [5]. The Open Web Application Security Project (OWASP) provides the standard for such penetration testing methodology to test web applications and could be used to evaluate the effectiveness of web vulnerability scanners [6,7].

Web Application Vulnerability Scanners (WAVS) are tools used by penetration testers. It is used to conduct web application evaluations with the primary goal of identifying and mitigating potential vulnerabilities to prevent security breaches. Security measures should be incorporated throughout the development life cycle rather than being added and tested only at the final stages of the development life cycle [8].

The challenge here is that when using different WAVS, they will not offer the same vulnerability results for the same target [1,8]. These differences arise from the different levels of precision, speed, and coverage in terms of finding various vulnerabilities, for example, with the respect to XSS and SQLi attacks [8,9]. These differing results mean that penetration testers need to utilise multiple WAVS and thus, the accuracy and detection coverage of the penetration testing report depend on the testers' knowledge and experience of the most effective WAVS in particular situations. This manual approach to Web Application Penetration Testing (WAPT) can be time-consuming, costly, and subject to human error [10].

Penetration testing, a cornerstone of cybersecurity practices, involves the authorised evaluation of a system's security attributes. It is characterised by the ethical authorisation to execute simulated attacks for the purpose of identifying vulnerabilities. This well-established technique is facilitated by a variety of tools that help automate repetitive tasks. The process is manual, commencing with a comprehensive examination of the target system. Penetration testers utilise ethical hacking techniques to exploit any vulnerabilities they uncover. These simulated attacks mirror real-world cyber threats within a controlled environment, with the goal of breaching the system's defences and demonstrating proof-of-concept attacks that compromise network resources. The industry is witnessing a shift toward the automation of penetration testing, known as automated penetration testing [11].

With the increasing availability of state-of-the-art hacking tools, threat actors have the motivation, opportunity, and capability to launch automated attacks [12,13]. Similarly, an automated approach has been the recent trend for WAPT because it eliminates human error and provides a cost and time-effective solution to creating detailed vulnerability reports [14].

This leads to our research question: How can vulnerability detection rates be improved through the development of an automated framework that combines the results of multiple WAVS into a single vulnerability report, compared to using a single scanner? Therefore, this paper aims to develop an automated framework that enhances vulnerability detection by aggregating results from multiple WAVS into a consolidated vulnerability report.

To achieve this paper's aims, a list of objectives has been presented. A review of the existing scientific literature on open-source WAVS and OWASP 2021. Develop an automated framework for enhancing vulnerability detection in web applications. Address the challenges of varying results obtained from different WAVS. Finally, Combine the results of multiple WAVS into a consolidated vulnerability report to improve detection rates.

A new framework is proposed that could be used to automate multiple WAVS and to produce a combined vulnerability report that compares and combines the results of the vulnerabilities discovered by multiple WAVS. The main contributions of this paper are summarised as follows:

- Conducting an extensive and rigorous review of the existing scientific literature on open-source WAVS while also discussing the current works that apply to WAVS and their weaknesses.
- Introducing a novel framework with the objective of augmenting precision, recall, and the F-measure within vulnerability detection compared to individual scanners. This is accomplished by combining outcomes from multiple WAVS.

The rest of this paper is structured as follows. In Section 2, we review the relevant work in the area of WAVS, web application security standards, automated penetration testing techniques, and benchmark targets. In Section 3, we describe the methodology used in our work, functional and non-functional requirements, algorithms, and the detailed experimental design. In Section 4, we present the experimental results for benchmark targets and discuss the findings. Finally, Section 5 concludes this paper.

2. Background and Related Work

Although significant work has taken place in recent years to mitigate cyber attacks and enhance web application security, threat actors are still able to deploy sophisticated approaches to bypass the current defensive mechanism [15]. In this section, we will review

the current web application architecture and technologies, and investigate the associated security standards. We then discuss the use of benchmark targets, investigate the widely used WAVS and automated penetration testing techniques, and finally present the current evaluation metrics.

2.1. Web Application Architecture and Technologies

A web application is a computer program that is stored on a server. Web applications consist of three main components: Front-end, back-end, and database [16]. The front-end of a web application is what the users of the web application can see when they visit a website. Thus, the front-end is considered to be part of the client-side code [17]. Frontend technologies, such as React.js and AngularJS, are built with advanced mechanisms to prevent commonly known vulnerabilities. Node.js and React.js are the most commonly used technologies for developing web applications [18].

Both the back-end and database components are considered to be part of the server-side code [17]. The back-end of a web application is responsible for processing and supplying the data that is shown on the front-end. Commonly used back-end languages are JavaScript, Java, PHP, Python, and Ruby. The database of the web application is where data is stored and retained. There are two types of databases: structured databases, such as MySQL and PostgresSQL, and non-structured databases like MongoDB.

When developing a web application, either single or multiple programming languages can be used, but, based on Stack Overflow 2022 survey responses, it was revealed that JavaScript is the most used programming language in modern web applications [18]. Accordingly, the project framework proposed in this paper is developed using the JavaScript language. It is also easier to develop the front-end and back-end using a single programming language.

2.2. Web Application Security Standards

Web applications that have been developed without following the latest security guidelines and best practices might be vulnerable to various cyber-attacks, and indeed, in general, the understanding of web application architectures and technologies among developers is insufficient to shield web applications from cyber threats. Various security guides and communities, such as OWASP, Common Weakness Enumeration (CWE™), and Web Security Consortium application (WASC), attempt to address these deficiencies by providing an updated list of web vulnerabilities.

Among these, CWE encompasses software weaknesses as well as web application weaknesses, while WASC describes all possible attacks without focusing on which attacks are the most common ones. OWASP, on the other hand, focuses on the top ten most critical vulnerabilities. For this reason, and since other studies [6,8,16,19–22] have used OWASP for web application security guidelines, we also adopted OWASP as our preferred security guideline for use in this project.

A list of Web application security vulnerabilities for OWASP Top 10 2021 [23] are: Broken Access Control; Cryptographic Failures; Injection; Insecure Design; Security Misconfiguration; Vulnerable Components; Identification and Authentication Failures; Software and Data Integrity Failures; Security Logging and Monitoring Failures; Server-Side Request Forgery.

This study focuses on the top five vulnerabilities listed in the OWASP Top Ten for 2021. Research conducted by [24,25] demonstrates that the vulnerabilities prioritised by OWASP 2021 play a significant role in addressing a substantial percentage of WAV attacks. Broken Access Control includes issues such as 19% Insufficient Authorisation and 15% Predictable Resource Location. Injection-related vulnerabilities are represented by 18% SQL Injection and 12% Cross-Site Request Forgery (CSRF). The category of Insecure Design includes 30% Insufficient Authentication. Security Misconfiguration. Vulnerable Components encompass 65% Cross-Site Scripting (XSS) and 46% Information Leakage.

- Broken Access Control is a vulnerability in web applications where inadequate access controls, such as user permissions, lead to unauthorised access to resources or actions. This results in users bypassing restrictions and gaining entry to sensitive data or actions they should not have access to.
- Cryptographic Failures are a weakness in encryption and decryption processes. These weaknesses include the use of weak encryption algorithms, improper key management, or incorrect application of cryptographic methods, allowing attackers to decipher encrypted information.
- Injection vulnerabilities involve the insertion of untrusted data into a program's code or commands, unintentionally triggering malicious actions. Common examples include SQL Injection, where attackers manipulate database queries, and Cross-Site Scripting (XSS), where they inject harmful scripts into web applications, undermining system functionality and data security.
- Security Misconfiguration is a security flaw that occurs when an application, system, or network is not properly configured to protect against known vulnerabilities. It results from the failure to apply secure settings, patches, and access controls.
- Vulnerable Components are software or hardware elements within an application or system with known security issues. Attackers target these components due to their recognised vulnerabilities. Examples include outdated software libraries, frameworks, or third-party components with flaws, which, if exploited, can lead to system compromise.

2.3. Benchmarks Targets

Benchmark targets are used to evaluate WAVS, for example, NodeGoat, and Damn Vulnerable Node.js Application (DVNA) [26]. But it is important that these benchmarks should be up to date, and that they consider the latest OWASP vulnerabilities [6]. Currently, the most up-to-date test targets are Juice Shop [27] and NodeGoat [28] . Juice Shop is written entirely using JavaScript and is powered by Node.js, Express, and Angular. Node.js is also entirely written using JavaScript.

Based on [27], a total of 17 vulnerabilities were identified within NodeGoat, covering only OWASP 2021 vulnerability categories A1, A2, A3, A5, and A6. The vulnerabilities were distributed as follows. Broken Access Control had 5 vulnerabilities, Cryptographic Failures had none, Injection had 3, Security Misconfiguration had 6, and Vulnerable and Outdated Components had 3 vulnerabilities. Moreover, a total of 39 vulnerabilities were identified within Juice Shop, again scoped only to OWASP 2021 vulnerability categories A1, A2, A3, A5, and A6. These were distributed as follows: Broken Access Control had 10 vulnerabilities, Cryptographic Failures had 5, Injection had 11, Security Misconfiguration had 4, and Vulnerable and Outdated Components had 9. Table 1 summarises the NodeGoat and OWASP Juice Shop vulnerabilities.

The main reasons for using NodeGoat and Juice Shop are that they have been updated recently, they contain vulnerabilities from among the OWASP 2021 vulnerabilities, and they have been designed with modern web applications technologies such as Node.js, AngularJS, and React.js. It is worth noting that these are the same technologies that have been mentioned in Section 2.1. This is significant because if the framework architecture and targets are built using the same architecture it is possible to use the framework itself as a target. That, in turn, makes it possible to utilise the multiple WAVS framework as a benchmark target to validate itself, in addition to using NodeGoat and Juice Shop, which adds an extra level of efficiency to the project.

Table 1. NodeGoat & OWASP Juice Shop Vulnerabilities.

OWASP 2021 Vulnerability	Vulnerability Category	NodeGoat		OWASP Juice Shop Vulnerabilities	
		Challenge/ Vulnerability Name	Total Vul	Challenge/ Vulnerability Name	Total Vul
A01:2021	Broken Access Control	Application Error Disclosure Server Leaks Information Session Management Password Guessing Attacks	5	Admin Section CSRF Easter Egg Five-Star Feedback Forged Feedback Forged Review Manipulate Basket Product Tampering SSRF View Basket	10
A02:2021	Cryptographic	N/A	N/A	Forged Coupon Imaginary Challenge Nested Easter Egg Premium Paywall Weird Crypto	5
A03:2021	Injection	Cross-Site Scripting (DOM Based) Cross-Site Scripting (Reflected) SQL Injection	3	Christmas Special Database Schema, Ephemeral Accountant Login Admin Login Bender Login Jim NoSQL DoS NoSQL Exfiltration NoSQL Manipulation SSTi User Credentials	11
A05:2021	Security Misconfiguration	Backup File Disclosure CSP: Wildcard Directive Content Security Policy (CSP) Header Not Set Permissions Policy Header Not Set X-Content-Type-Options Header Missing Missing 'X-Frame-Options' header	6	Cross-Site Imaging Deprecated Interface Error Handling Login Support Team	4
A06:2021	Vulnerable Components	Vulnerable JS Library Cross-Domain JavaScript Source File Inclusion Insecure version of the Marked library	3	Arbitrary File Write Forged Signed JWT Frontend Typosquatting Kill Chatbot Legacy Typosquatting Local File Read Supply Chain Attack Unsigned JWT Vulnerable Library	9

2.4. Investigate Widely Used WAVS

Web application vulnerability scanning can be classified into three main types. Black-box scanning performs a scan on a given target without accessing the internal source code of the targeted web application. WAVS is an example of this type. White-box scanning, meanwhile, gains complete access to the source code of the web application before it performs the scan. Finally, grey-box scanning offers the penetration tester partial knowledge of the application source code. Information such as hidden path and software version can be supplied [6,29]. WAVS are available on an open-source and commercial basis [8]. Open-

source scanners are free to use and allow users to access (and thus evaluate) their source code. Commonly used open-source WAVS include OWASP ZAP, Wapiti, Vega, W3AF, and Arachni [30,31]. The main commercial WAVS, meanwhile, are BurpSuite, Acunetix and Netsparker [32].

Based on the research findings by Albalawi et al. [33] and the work of Shahid [19], the widely used WAVS that are up to date and recognised for their proficiency in detecting OWASP 2021 vulnerabilities are as follows:

- Acunetix is one of the most widely recognised commercial WAVS. It is adept at uncovering an array of vulnerabilities, including but not limited to SQL injections, Cross-Site Scripting (XSS), Host Header Injection, and an extensive list comprising over 3000 web-related vulnerabilities.
- BurpSuite is available in both free and commercial licenses. BurpSuite takes on the role of a Man In The Middle (MITM), HTTP requests. This process allows it to capture and analyse requests originating from the target web application server.
- OWASP ZAP, The OWASP Zed Attack Proxy, is one of the most widely recognized open-source WAVS. It was created by the OWASP team.
- Arachni is a free and open-source WAVS. It can navigate complex pathways dictated by a web application's complexity. Arachni excels at identifying vulnerabilities from the OWASP 2021 list, making it a valuable asset in enhancing web application security.
- Nikto is an open-source WAVS. It scans server configuration files. One limitation of Nikto is it primarily focuses on server configuration and may not cover all aspects of web application security, leaving potential vulnerabilities unaddressed.

Alazmi et al. [32] used Arachni and OWASP ZAP to address the effectiveness of WAVS. Given that WAVs used for this project needed to be automated from a framework, an important selection criterion was the ability to run from the command-line interface (CLI). Since both Arachni and OWASP ZAP can start from the CLI, and also since they have both recently been updated, these two WAVs were selected for use within the proposed framework. Based on the sources [34,35].

Hance et al. [12] suggested a novel attack framework utilising a distributed attack platform that incorporated a control scheme for automating vulnerability detection. Qiu et al. [14] developed an automated penetration testing algorithm that exploited vulnerabilities based on a scanning report. Zhou et al. [36] proposed the Network Information Gain Based Automated Attack Planning (NIG-AP) algorithm to automate penetration testing phases that use the reward system. Minh et al. [37], meanwhile, automated vulnerability assessments at the commit level, triggering them with each new commit made to the codebase.

2.5. Performance Metrics

Evaluation metrics provide the ability to evaluate the effect of the proposed framework properly. The evaluation analysis undertaken in this paper will be based on the results produced by the framework. To ensure appropriate evaluation of the multiple WAVS, four criteria were used in selecting the evaluation metrics. Consistency: in other words, metric parameters should remain constant throughout the entire testing period. Repeatable and reproducible: anyone should be able to replicate the experiment and obtain the same results. Comparable: metrics should give the ability to compare values against different results over a period of time. Meaningful: they should provide data that is meaningful and relevant to the context of the test.

Based on existing studies [8,31,38,39], metrics such as precision, recall, and F-measure were benchmarked and used to evaluate the multiple WAVS. Precision is the ratio of true positives (TP) divided by the sum of true positives and false positives (FP). The precision metrics are limited to the TP and FP, and thus do not take account of the true negative (TN) values. For this reason, recall is also used as an evaluation metric, defined as the ratio of true positives divided by the sum of true positives and false negatives. The last metric is the F-measure, which is used to measure the performance of the framework, and is defined

as the harmonic mean of precision and recall. A higher F-measure means a better balance between precision and recall. These are defined as follows:

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

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

$$\text{F-Measure} = 2 \times \left(\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \right)$$

To evaluate the multiple WAVS values such as true positives, false negatives, and false positives need to be calculated manually. True positives are the vulnerabilities correctly detected by the WAVS. False negatives are vulnerabilities that exist but have not been detected by WAVS. False positives are vulnerabilities that do not exist but have been mistakenly detected by multiple WAVS. True negatives are the non-vulnerable threats that the multiple WAVS correctly ignored. True negative values cannot be calculated, however, because the benchmark targets were not designed to provide cases for components that have no vulnerabilities or negative cases [26]. This means that the full accuracy of the framework cannot be calculated. Overall, however, precision, recall, and F-measure are considered good metric choices to evaluate the proposed framework.

3. Experiments

This section presents the experimental design in detail, including the environment used to run the presented methods and the general structure of the experimental design. Finally, we introduce the two algorithms used in the WAVS framework.

3.1. Methodology

Choosing an appropriate software development cycle is crucial to the project's success. As mentioned in [40,41], there are various software development approaches such as the prototype-based model, V model, classical waterfall, and iterative waterfall model. The prototype-based model is best when there is no goal or outcome defined, but since the goal and the scope of this project are clear the prototype-based model is not appropriate in this case. The iterative waterfall model allows for multiple iterations, unlike the classical waterfall model. The iterative waterfall methodology also allows tasks to be broken down into phases, where each new phase starts after the previous one is finished [42,43]. Since this project follows a structured approach, and the steps are clear and pre-defined, the iterative waterfall model is the most appropriate choice.

The experiment was carried out in the following manner:

- Launch the Juice Shop target and initiate an OWASP ZAP scan on the target. Afterwards, collect the scan results, as well as the Precision, Recall, and F-measure metrics.
- Launch the Juice Shop target once more, this time running Arachni for the scan. Collect the results of the scan, along with the Precision, Recall, and F-measure metrics.
- Commence the scan process on the Juice Shop target by configuring the WAVS framework to target it. This created the Union List and Intersection List. Following this, gather the results, including the Precision, Recall, and F-measure metrics.
- Repeat the entire process using the NodeGoat target and the WAVS framework.
- Compare the results of the Union list, Intersection list, Arachni list, and OWASP ZAP list and draw conclusions.

The target selection criteria for benchmark targets are as follows:

- Relevant and up to date: we chose the benchmark targets based on their relevance. We ensured that they aligned with the latest OWASP vulnerabilities. This entails focusing on benchmark targets updated recently with vulnerabilities from OWASP 2021 vulnerability categories.

- Build using Modern Web Technologies: The selected benchmark targets, NodeGoat and Juice Shop, are characterized by their utilisation of modern web application technologies. This approach, as detailed in the provided references [6,26,27], contributes to the robustness and comprehensiveness of the WAVS evaluation process.

The selection of Arachni and OWASP ZAP, being widely used within the cybersecurity domain, carry credibility as mentioned in the comprehensive studies conducted by Alazmi et al. [32] and Zapotek et al. [34]. The open-source nature of Arachni and OWASP ZAP enhances the transparency and collaborative essence of these tools, which integrates with the project's emphasis on open, community-driven security solutions.

3.2. Requirements

Functional Requirements

- The Combination Algorithm class should create two lists: Union and Intersection.
- The progress of the scan should be communicated to the user.

Non-functional Requirements

- The framework should operate without the need for Personally Identifiable Information (PII)
- The framework Combination Algorithm should produce the same results when run against the same target.
- The Vulnerability Scanner Abstract Class should have the ability to be extended by the addition of further WAVS.

3.3. Experimental Design

At a high-level architecture overview, the framework operates through the interaction between its three core components as shown in Figure 1. Users (Actors) communicate with the front-end web application, which allows the users to interact with the back-end APIs using the front-end interface. The user (Actor) does not need to understand how to start Arachni or OWASP ZAP because all of this logic is managed by the framework. The user (Actor) has to fill in the input field for the target and click the start button.

The back-end initiates the WAVS to perform the necessary scanning tasks after receiving HTTP requests from the front-end. The back-end then starts the WAVS to scan the specified target and generate the results. Then, the back-end returns this information to the front-end, allowing users to view and analyse the results. This communication between the front-end, back-end, and WAVS ensures a user-friendly experience.

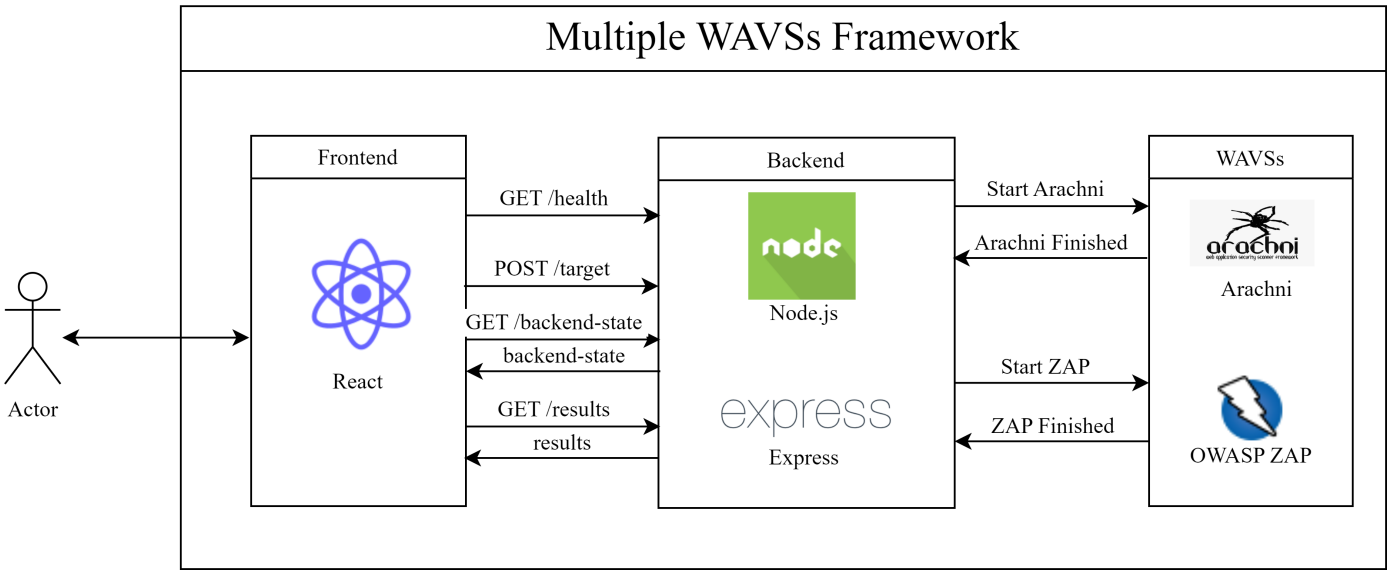

Figure 1. Diagram of the Multiple WAVS Framework.

In more detail, the multiple WAVS framework sequence is as follows:

The process begins with the WAVS framework front-end, built using React. This serves as the user interface. The back-end, developed using Node.js with Express, handles the communication between the front-end, the vulnerability scanners (WAVS), and the overall process flow. The WAVS, represented by Arachni and OWASP ZAP, are initiated to perform vulnerability scans on the specified target.

Start Arachni: the Arachni scanner begins its scan on the provided target. Start ZAP: the OWASP ZAP scanner commences its vulnerability assessment on the same target. Start Arachni: upon completing the scan, Arachni signals that it has finished scanning. Start ZAP: similarly, OWASP ZAP indicates that it has completed the scanning process. GET /backend-state: the front-end queries the back-end for the current state of the scanning process. GET /health: a health check is conducted to ensure that the back-end is operational. POST /target: the front-end sends a request to the back-end with the target for scanning.

GET /results: the back-end retrieves the results of the scans for the specified target. Back-end state results: the back-end combines the state of the process with the scan results. Actor: the user, or an automated process, interacts with the front-end to initiate scans and monitor progress. Arachni Finished: after Arachni completes its scan, its status is updated. Actor Arachni Finished: the user or automated process is notified of Arachni's completion. Arachni and ZAP Results: the back-end fetches the scan results from both Arachni and OWASP ZAP. Combined Results: the back-end combines the results from both scanners. Results: the consolidated results are prepared for further analysis or presentation.

3.4. Algorithms

3.4.1. Combination Algorithms

Algorithm 1 is a novel algorithm proposed for this paper. This algorithm compares the results generated from the different WAVSs and then it compares them against each other to produce a final combined report that contains all of the results from different WAVSs. The concept of the algorithm is to loop over the generated vulnerability results and store them in four different lists. Union list, Intersection list, Arachni list, OWASP ZAP list.

1. The Union list represents the combined set of vulnerabilities identified by the WAVSs, in this case, Arachni and OWASP ZAP. This is the important list that is used to validate the hypothesis by combining all vulnerabilities identified by either scanner. By considering the union list, researchers can assess the effectiveness of using multiple scanners together in enhancing vulnerability detection coverage and accuracy.
2. Intersection list: this list contains the vulnerabilities detected by both scanners, representing a higher confidence set of vulnerabilities.
3. Arachni list: the unique list for Arachni contains vulnerabilities detected by the Arachni scanner. This list is used to evaluate Arachni against the union list.
4. OWASP ZAP list: similar to Arachni's list, the OWASP ZAP list contains vulnerabilities detected by the OWASP ZAP scanner.

3.4.2. Automation Algorithms

Algorithm 2 presented in this paper outlines a systematic process for analysing the results of various Web Application Vulnerability Scanners (WAVSs). The algorithm iterates through each WAVS in a loop, starting its execution and evaluating its success. When a WAVS succeeds, it executes and records its findings, including the target value and results. If result availability is confirmed, the algorithm aggregates all WAVS results and checks their retrieval status. Once all results are retrieved, the algorithm employs a process of combining and filtering the results, producing a refined report of identified vulnerabilities.

Algorithm 1 Combination Algorithm

```
Input: WAVSs Results.
Output: The vulnerabilities results.
Arguments:
results1: the result list generated by WAVS 1.
results2: the result list generated by WAVS 2.
function CombinationAlgorithm(results1, results2)
    intersectionList ← Create Intersection List(results1 ∩ results2)
    unionList ← Create Unique List(results1 ∪ results2)
    resultList1 ← Create Single List(results1)
    resultList2 ← Create Single List(results2)
    while ValidResults do
        if results1 and results2 then
            groupResults1ByCweId ← GROUP BY CWEID(results1)
            groupResults2ByCweId ← GROUP BY CWEID(results2)
            for all cweid in groupResults1ByCweId do
                if cweid ∈ groupResults2ByCweId then
                    intersectionList.add(cweid)
                else
                    uniqueList.add(cweid)
                end if
                resultList1.add(cweid)
            end for
            for all cweid in groupResults2ByCweId do
                if cweid ∉ groupResults1ByCweId then
                    unionList.add(cweid)
                end if
                resultList1.add(cweid)
            end for
        else
            if results1 then
                unionList ← Add to Union List(results1)
                resultList1 ← Add to Single List(results1)
            end if
            if results2 then
                unionList ← Add to Union List(results2)
                resultList2 ← Add to Single List(results2)
            end if
        end if
        Perform Additional Processing()
    end while
    finalCombinedResults ← Combine and Filter(resultList1, resultList2)
    if finalCombinedResults then
        storeResults(finalCombinedResults)
        generateDetailedReport(finalCombinedResults)
    else
        generateFailureReport()
    end if
end function
```

Algorithm 2 Automation Algorithm

```
Input: Void.
Output: The vulnerabilities results.
function AUTOMATIONALGORITHM
    for i = 0 to the total count of WAVS do
        startCurrentWAV([i])
        if Success then
            Execute WAVS with the current target value
            Retrieve the current target value
            Store the Results of the current WAVS
        else
            Execute Combination Algorithm
            End with success message update
        end if
    end for
    while ¬allWAVSResults do
        if resultAvailable then
            allWAVSResults ← fetchAllWAVSResults()
        else
            Wait for Results allWAVSResults
        end if
    end while
    if allWAVSResults then
        results1 ← Scanner 1 results extracted from allWAVSResults(allWAVSResults)
        results2 ← Scanner 2 results extracted from allWAVSResults(allWAVSResults)
        combinedResults ← CombinationAlgorithm(results1, results2)(allWAVSResults)
▷ Invoke Algorithm 1 Combination Algorithm
        if combinedResults then
            filteredResults ← Filter Results(combinedResults)
            storeResults(filteredResults)
            generateSummaryReport(filteredResults)
        end if
    end if
end function
```

4. Experiment Results

The experimental results are presented and discussed in this section, which includes the Union List, Intersection List, Arachni's List, and OWASP ZAP's List.

4.1. Target NodeGoat

A total of 17 vulnerabilities were identified within NodeGoat. These are listed in Figure 2. These 17 vulnerabilities are the total amount of vulnerabilities.

The Union List has the highest F-measure at 73%, as shown in Table 2 and Figure 2. This means that it has a good balance between precision and recall. With a precision score of 63%, it reported 37% false positives. Its recall of 88% means that it missed 12% of the true positives. The Intersection List's precision score was 100%, meaning it did not report any false positives. its recall was only 24%, however, indicating that it missed 76% of true positives. This led to a very low F-measure of 38%. Arachni's precision was 100%, similar to the Intersection List, but its recall was just 18%, resulting in a lower F-measure of 30%. Despite its high-precision score, therefore, Arachni cannot detect all true positives. OWASP ZAP reported a high F-measure of 70%, lower than the Union List. Its precision of 61% means it reported 39% false positives. With the second-highest recall after the Union List at 82%, OWASP ZAP demonstrates an impressive performance in detecting true positives.

Table 2. Table Summary of the evaluation metrics.

List	Precision			Recall			F-Measure		
	Node-Goat	Juice Shop	WAVS	Node-Goat	Juice Shop	WAVS	Node-Goat	Juice Shop	WAVS
Union	63%	78%	86%	88%	18%	100%	73%	30%	92%
Intersection	100%	100%	100%	24%	5%	83%	38%	10%	91%
Arachni	100%	100%	100%	18%	5%	33%	30%	10%	50%
OWASP ZAP	61%	57%	80%	82%	10%	67%	70%	17%	73%

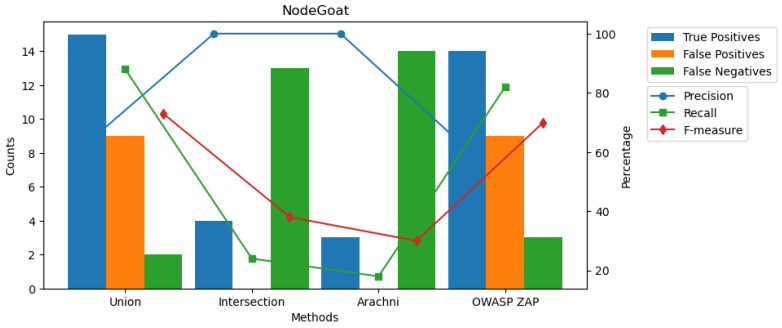

Figure 2. NodeGoat performance metrics with TP, FP and FN comparison.

4.2. Target Juice Shop

A total of 39 vulnerabilities were identified within Juice Shop provided in Figure 3. These 39 vulnerabilities are the total amount of vulnerabilities.

The Union List has the highest F-measure at 30% as shown in Table 2 and Figure 3. This means a low balance between precision and recall. With a precision score of 78%, it reported 22% false positives. Its recall of 18% means that it missed 82% of true positives. The Intersection List's precision score was 100%, meaning it did not report any false positives. Its recall was only 5%, however, which means that it missed 95% of true positives. This led to a very low F-measure of 10%. Arachni's precision was 100%, similar to the Intersection List. Again, however, its recall was just 5%, resulting in a low F-measure of 10%. This means that, despite its high-precision score, Arachni cannot detect all true positives. OWASP ZAP reported a relatively high F-measure of 17%, lower than the Union List. Its precision of 57% means it reported 43% false positives. With the second-highest recall after the Union List at 10%, OWASP ZAP demonstrates a medium performance in detecting true positives.

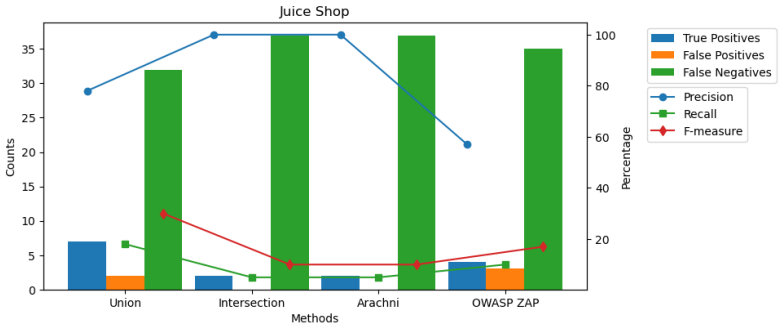

Figure 3. Juice Shop performance metrics with TP, FP, and FN comparison.

4.3. Target WAVS Framework

A total of six vulnerabilities were identified within the WAVS framework provided in Figure 4. These six vulnerabilities are the total amount of vulnerabilities.

The Union List has the highest F-measure at 92%, as shown in Table 2 and Figure 4, indicating a high balance between precision and recall. With a precision score of 86%, it reported 14% false positives. Its recall of 100% means that it successfully detected all true

positives. The Intersection List's precision score was 100%, meaning it did not report any false positives. its recall of 83%, however, indicated that it missed 17% of true positives. This led to a high F-measure of 91%, similar to the Union List. Arachni's precision was 100%, similar to the Intersection List. Its recall was only 33%, however, resulting in an F-measure of 50%. Again, despite its high-precision score, Arachni cannot detect all true positives. OWASP ZAP reported an F-measure of 73%, lower than the Union List. Its precision of 80% means it reported 20% false positives. With a recall of 67%, OWASP ZAP demonstrates a good performance in detecting true positives.

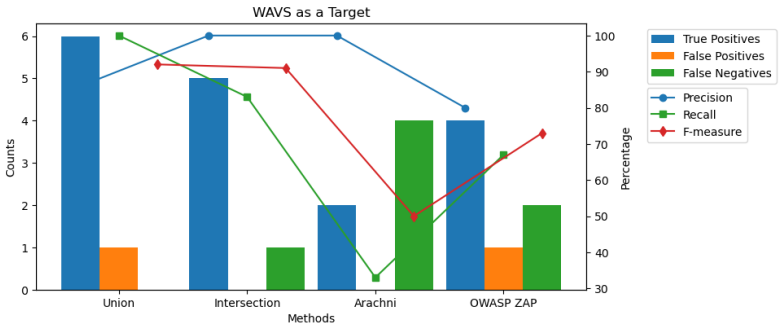

Figure 4. WAVS framework performance metrics with TP, FP, and FN comparison.

4.4. Discussion

The Union List achieved the highest F-measure across all targets. This means that it offers the best balance between precision and recall. The Intersection List had the best precision score, but its recall score varied from target to target, which led to different F-measures. Arachni had a perfect score of 100% precision for all three targets, but it recorded the lowest recall. OWASP ZAP had a better recall than both the Intersection List and Arachni, but still did not surpass the Union List's performance. This means that the Union List performed better than either Arachni and OWASP ZAP individually.

The practical significance of scanner performance and the use of the Union List in the context of web application security is as follows:

- Union List's High F-Measure: The Union list demonstrated the highest F-measure across all targets, indicating a good balance between precision and recall. This is important because it shows that the Union List is effective in identifying vulnerabilities without high ratios of false positives or false negatives.
- Union List vs. Intersection List: The Union performed better in terms of recall and F-measure compared to the Intersection List, even though the latter had better precision. This demonstrates the Union List's ability to offer a better balance between precision and recall.
- Importance of Combining Scanners: The Union list was generated by OWASP ZAP, while Arachni failed to detect the most true positives. This emphasises the value of combining multiple scanners to improve overall detection rates. In conclusion, the research findings suggest that the Union List, generated by the WAVS framework, is a practical choice for web application security assessments due to its balanced performance. It is able to outperform individual scanners. The use of multiple scanners in combination is also recommended for comprehensive vulnerability detection.

When compared to the state-of-the-art studies, exemplified by Anhar et al. [26], the comprehensive Union List WAVS framework surpassed all findings presented in their research.

In the study conducted by Mburano et al. [6] concluded that ZAP outperformed Arachni in categories such as SQL Injection (SQLI) and Cross-Site Scripting (XSS). In contrast, Arachni exhibited superior performance in the Lightweight Directory Access Protocol (LDAP) category. These findings substantiate the results obtained from the Union List, affirming the framework's efficacy in identifying a broader spectrum of vulnerabilities and achieving a higher F-measure. The framework's efficacy can be attributed to several key

factors. This collective knowledge base uses the framework to identify a more extensive spectrum of vulnerabilities, as it benefits from the diversity of the WAVS.

When comparing the Union List to the Intersection List, the latter performed better against all the targets in terms of precision, while the former performed better in terms of recall and F-measure. This indicates that the Union List offers better balance and detection. It is important to note that the Intersection List reported quite similar findings to Arachni, probably because most of the vulnerabilities found by Arachni were also found by OWASP ZAP. It is also important to note that the Union List result was mainly generated by OWASP ZAP, while Arachni failed to detect most of the true positives. Indeed, this is why OWASP ZAP outperformed Arachni in terms of recall and F-measure. Overall, therefore, our results highlight the benefits of utilising multiple vulnerability scanners in conjunction with one another.

The Union List leverages this diversity to ensure a well-rounded vulnerability assessment. The higher F-measure achieved by the framework is a testament to its ability to balance precision and recall effectively. By combining the results of multiple WAVS, the framework minimises false negatives while maintaining a high level of precision. This means that it not only detects a broader range of vulnerabilities but also does so with a lower rate of false positives, enhancing the accuracy of vulnerability identification.

After testing and generating all the results, it was proven that the Union List generated by the WAVS framework achieved the best performance evaluation in terms of recall and F-measure.

5. Conclusions

In this paper, we have comprehensively evaluated the performance of different vulnerability scanners. Our investigation involved comparing four lists—the Union list, the Intersection list, the Arachni list, and the OWASP ZAP list—considering their individual contributions to vulnerability detection across multiple targets.

The Union List consistently achieved the highest F-measure performance across all targets. This underscores the significance of combining multiple scanners for an enhanced security assessment. The study supports the Union List, created by the WAVS framework, as a practical choice for web application security. It excels in maintaining precision-recall balance and outperforming individual scanners. Using multiple scanners in combination is strongly recommended for comprehensive vulnerability detection in web applications.

The Union List emerged as the best performer, exemplifying an exceptional balance between precision and recall, which shows the significance of the integration of multiple scanners. Through these comprehensive evaluations, our study demonstrates the potential of integrating multiple scanners and offers valuable insights into vulnerability scanning methodologies that will help to enhance vulnerability assessment practices.

The main limitation in our study pertains to the constrained availability of up-to-date, free, and open-source WAVS suitable for integration into our framework. Furthermore, our study faces the challenge of having a limited selection of benchmark targets that encompass the latest OWASP 2021 vulnerabilities, which are essential for evaluating the framework. This constraint reflects the dynamic nature of the cybersecurity domain, where vulnerabilities evolve continuously. In accordance with [32]'s findings in their 2022 publication, six studies have undertaken assessments of OWASP ZAP, while only two have evaluated Arachni. Upon reviewing this body of research, one study has explored OWASP 2021 vulnerabilities. In consideration of these limitations, we urge future research to investigate the integration of commercial WAVS into this framework and benchmark against the most up-to-date OWASP 2021 vulnerabilities.

This study offers valuable insights into vulnerability scanning methodologies, shedding light on their practical applications and how they can significantly enhance vulnerability assessment practices in real-world scenarios in the web application cybersecurity domain. This research underscores the tangible benefits of using these methodologies to

improve the security and reliability of web applications, making them more robust against potential threats and vulnerabilities.

Learning how to integrate multiple WAVS, rather than relying solely on single open-source scanners, will be a key area for future work. Additionally, we are now seeking to incorporate automated exploits to validate identified vulnerabilities to generate comprehensive reports based on the success of those exploit attempts.

Author Contributions: Conceptualization, K.A. and N.E.; methodology, K.A. and N.E.; software, K.A.; validation, K.A., N.E. and M.Y.; formal analysis, K.A. and N.E.; investigation, K.A., N.E and M.Y.; resources, K.A. and N.E.; data curation, K.A.; writing—original draft preparation, K.A.; writing—review and editing, K.A., N.E. and M.Y.; visualization, K.A., N.E. and M.Y.; supervision, N.E.; project administration, N.E. and M.Y.; funding acquisition, N.E. All authors have read and agreed to the published version of the manuscript.

Funding: Glasgow Caledonian University provided the funding for open access for this research.

Data Availability Statement: Data available on request.

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

References

1. Altulaihan, E.A.; Alismail, A.; Frikha, M. A Survey on Web Application Penetration Testing. *Electronics* **2023**, *12*, 1229. [CrossRef]
2. Sadqi, Y.; Maleh, Y. A systematic review and taxonomy of web applications threats. *Inf. Secur. J. Glob. Perspect.* **2022**, *31*, 1–27. [CrossRef]
3. Trickel, E.; Pagani, F.; Zhu, C.; Dresel, L.; Vigna, G.; Kruegel, C.; Wang, R.; Bao, T.; Shoshitaishvili, Y.; Doupé, A. Toss a Fault to Your Witcher: Applying Grey-box Coverage-Guided Mutational Fuzzing to Detect SQL and Command Injection Vulnerabilities. In Proceedings of the 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 21–25 May 2023; pp. 2658–2675. [CrossRef]
4. Deepa, G.; Thilagam, P.S. Securing web applications from injection and logic vulnerabilities: Approaches and challenges. *Inf. Softw. Technol.* **2016**, *74*, 160–180. [CrossRef]
5. Alhamed, M.; Rahman, M.M.H. A Systematic Literature Review on Penetration Testing in Networks: Future Research Directions. *Appl. Sci.* **2023**, *13*, 6986. [CrossRef]
6. Mburano, B.; Si, W. Evaluation of Web Vulnerability Scanners Based on OWASP Benchmark. In Proceedings of the 2018 26th International Conference on Systems Engineering (ICSEng), Sydney, Australia, 18–20 December 2018; pp. 1–6.
7. 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.
8. Idrissi, S.E.; 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.
9. 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.
10. Singh, N.; Meherhomji, V.; Chandavarkar, B.R. Automated versus Manual Approach of Web Application Penetration Testing. In Proceedings of the 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kharagpur, India, 1–3 July 2020; pp. 1–6. [CrossRef]
11. Hu, Z.; Beuran, R.; Tan, Y. Automated Penetration Testing Using Deep Reinforcement Learning. In Proceedings of the 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), Genoa, Italy, 7–11 September 2020; pp. 2–10. [CrossRef]
12. Hance, J.; Milbrath, J.; Ross, N.; Straub, J. Distributed Attack Deployment Capability for Modern Automated Penetration Testing. *Computers* **2022**, *11*, 33. [CrossRef]
13. Elmrabit, N.; Zhou, F.; Li, F.; Zhou, H. Evaluation of Machine Learning Algorithms for Anomaly Detection. In Proceedings of the 2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), Dublin, Ireland, 15–19 June 2020; pp. 1–8. [CrossRef]
14. Qiu, X.; Wang, S.; Jia, Q.; Xia, C.; Xia, Q. An automated method of penetration testing. In Proceedings of the 2014 IEEE Computers, Communications and IT Applications Conference, Beijing, China, 20–22 October 2014; pp. 211–216.
15. Shahid, W.B.; Aslam, B.; Abbas, H.; Khalid, S.B.; Afzal, H. An enhanced deep learning based framework for web attacks detection, mitigation and attacker profiling. *J. Netw. Comput. Appl.* **2022**, *198*, 103270. [CrossRef]
16. Lala, S.K.; Kumar, A.; Subbulakshmi, T. Secure Web development using OWASP Guidelines. In Proceedings of the 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 6–8 May 2021; pp. 323–332. [CrossRef]

17. Schutt, K.; Balci, O. Cloud software development platforms: A comparative overview. In Proceedings of the 2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA), Towson, MD, USA, 8–10 June 2016; pp. 3–13. [CrossRef]
18. Stack Overflow Developer Survey. 2021. Available online: https://insights.stackoverflow.com/survey/2021#most-popular-technologies-language (accessed on 31 August 2023).
19. Shahid, J.; Hameed, M.K.; Javed, I.T.; Qureshi, K.N.; Ali, M.; Crespi, N. A Comparative Study of Web Application Security Parameters: Current Trends and Future Directions. *Appl. Sci.* **2022**, *12*, 4077. [CrossRef]
20. Mateo Tudela, F.; Bermejo Higuera, J.R.; Bermejo Higuera, J.; Sicilia Montalvo, J.A.; Argyros, M.I. On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications. *Appl. Sci.* **2020**, *10*, 9119. [CrossRef]
21. Antonelli, D.; Cascella, R.; Perrone, G.; Romano, S.P.; Schiano, A. Leveraging AI to optimize website structure discovery during Penetration Testing. *arXiv* **2021**, arXiv:2101.07223.
22. Kiruba, B.; Saravanan, V.; Vasanth, T.; Yogeshwar, B.K. OWASP Attack Prevention. In Proceedings of the 2022 3rd International Conference on Electronics and Sustainable Communication Systems (ICESC), Coimbatore, India, 17–19 August 2022; pp. 1671–1675. [CrossRef]
23. OWASP Foundation Top Ten Project. 2021. Available online: https://owasp.org/www-project-top-ten/ (accessed on 31 August 2023).
24. Zaitseva, E.; Hovorushchenko, T.; Pavlova, O.; Voichur, Y. Identifying the Mutual Correlations and Evaluating the Weights of Factors and Consequences of Mobile Application Insecurity. *Systems* **2023**, *11*, 242. [CrossRef]
25. Alsaffar, M.; Aljaloud, S.; Mohammed, B.A.; Al-Mekhlafi, Z.G.; Almurayziq, T.S.; Alshammari, G.; Alshammari, A. Detection of Web Cross-Site Scripting (XSS) Attacks. *Electronics* **2022**, *11*, 2212. [CrossRef]
26. Al Anhar, A.; Suryanto, Y. Evaluation of Web Application Vulnerability Scanner for Modern Web Application. In Proceedings of the 2021 International Conference on Artificial Intelligence and Computer Science Technology (ICAICST), Yogyakarta, Indonesia, 29–30 June 2021; pp. 200–204. [CrossRef]
27. Kimminich, B. The OWASP Juice Shop Project. 2023. Available online: https://owasp.org/www-project-juice-shop/ (accessed on 31 August 2023).
28. Karande, C. OWASP NodeGoat project. Available online: https://github.com/OWASP/NodeGoat (accessed on 31 August 2023).
29. Antunes, N.; Vieira, M. Penetration testing for web services. *Computer* **2013**, *47*, 30–36. [CrossRef]
30. Albahar, M.; Alansari, D.; Jurcut, A. An Empirical Comparison of Pen-Testing Tools for Detecting Web App Vulnerabilities. *Electronics* **2022**, *11*, 2991. [CrossRef]
31. Alsaleh, M.; Alomar, N.; Alshreef, M.; Alarifi, A.; Al-Salman, A. Performance-Based Comparative Assessment of Open Source Web Vulnerability Scanners. *Secur. Commun. Netw.* **2017**, *2017*, 6158107. [CrossRef]
32. Alazmi, S.; De Leon, D.C. A Systematic Literature Review on the Characteristics and Effectiveness of Web Application Vulnerability Scanners. *IEEE Access* **2022**, *10*, 33200–33219. [CrossRef]
33. Albalawi, N.; Alamrani, N.; Aloufi, R.; Albalawi, M.; Aljaedi, A.; Alharbi, A.R. The Reality of Internet Infrastructure and Services Defacement: A Second Look at Characterizing Web-Based Vulnerabilities. *Electronics* **2023**, *12*, 2664. [CrossRef]
34. Laskos, T. Arachni—Web Application Security Scanner Framework. Available online: https://github.com/Arachni (accessed on 31 August 2023).
35. ZAPping the OWASP Top 10. 2021. Available online: https://www.zaproxy.org/docs/guides/zapping-the-top-10-2021/ (accessed on 31 August 2023).
36. Zhou, T.-Y.; Zang, Y.-C.; Zhu, J.-H.; Wang, Q-X. NIG-AP: A new method for automated penetration testing. *Front. Inf. Technol. Electron. Eng.* **2019**, *20*, 1277–1288. [CrossRef]
37. Minh Le, T.H.; Hin, D.; Croft, R.; Ali Babar, M. DeepCVA: Automated Commit-level Vulnerability Assessment with Deep Multi-task Learning. In Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 15–19 November 2021; pp. 717–729. [CrossRef]
38. Urbano, L.; Perrone, G.; Romano, S.P. Reinforced WAVSEP: A Benchmarking Platform for Web Application Vulnerability Scanners. In Proceedings of the 2022 International Conference on Electrical, Computer and Energy Technologies (ICECET), Prague, Czech Republic, 20–22 July 2022; pp. 1–6. [CrossRef]
39. 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. [CrossRef]
40. Huo, M.; Verner, J.; Zhu, L.; Babar, M.A. Software quality and agile methods. In Proceedings of the 28th Annual International Computer Software and Applications Conference, Hong Kong, China, 28–30 September 2004; pp. 520–525.
41. Mitchell, S.M.; Seaman, C.B. A comparison of software cost, duration, and quality for waterfall vs. iterative and incremental development: A systematic review. In Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, Lake Buena Vista, FL, USA, 15–16 October 2009; pp. 511–515.

42. Trivedi, P.; Sharma, A. A comparative study between iterative waterfall and incremental software development life cycle model for optimizing the resources using computer simulation. In Proceedings of the 2013 2nd International Conference on Information Management in the Knowledge Economy, Chandigarh, India, 19–20 December 2013; pp. 188–194.
43. Chandra, V. Comparison between various software development methodologies. *Int. J. Comput. Appl.* **2015**, *131*, 7–10. [CrossRef]