electronics

MDPI

Article

Intelligent Platform for Automating Vulnerability Detection in Web Applications

Diogo Moreira *, João Pedro Seara, João Pedro Pavia and Carlos Serrão

Information Sciences, Technologies and Architecture Research Center (ISTAR), ISCTE-Lisbon University Institute, 1649-026 Lisbon, Portugal; joao_pedro_seara@iscte-iul.pt (J.P.S.); joao.pedro.pavia@iscte-iul.pt (J.P.P.); carlos.serrao@iscte-iul.pt (C.S.)

* Correspondence: dsmaa1@iscte-iul.pt

Abstract: In a world increasingly dependent on technology and in an era where connectivity is omnipresent, Web applications have become an essential part of our everyday life. The evolution of these applications, combined with the exponential increase in the number of users, has brought with it not only convenience but also significant challenges in terms of security. Ensuring the security of Web applications and their data is increasingly a priority for companies, although many companies lack the know-how, time, and money to do so. This research project studied and developed a system with the aim of automating the process of detecting vulnerabilities in Web applications by exploiting the benefits of the interoperability of the two forms of automation of the tool selected to carry out this analysis. The developed solution is low-cost and requires very little user intervention. In order to validate and evaluate the developed platform, experiments were carried out on applications with different types of vulnerabilities known in advance and on real applications. It is essential to guarantee the security of Web applications, and the developed system proved capable of automating the detection of vulnerability risks and returning the results in a relatively simple way for the user.

Keywords: Web application; vulnerability; security; scanner; automation; detection

Academic Editor: George Angelos Papadopoulos

Received: 20 November 2024
Revised: 24 December 2024
Accepted: 25 December 2024
Published: 27 December 2024

Citation: Moreira, D.; Seara, J.P.; Pavia, J.P.; Serrão, C. Intelligent Platform for Automating Vulnerability Detection in Web Applications. *Electronics* **2025**, *14*, 79. https://doi.org/10.3390/electronics14010079

1. Introduction

The dependence on technology in an increasingly interconnected global landscape has fundamentally transformed daily life, particularly with the exponential rise in Internet use, now encompassing around 4.7 billion users worldwide [1]. Concurrently, Web applications have proliferated, reaching approximately 1.8 billion globally [1]. This rapid growth has been accompanied by a transformation of Web applications [2], evolving from simple static pages into sophisticated, dynamic information systems. Initially, such applications were built using elementary models and languages that posed relatively low security risks, simplifying protection and analysis efforts [3]. However, contemporary applications have become highly interactive and user-centric, accessible across multiple devices and browsers, thereby allowing numerous domains of applications, including banking, healthcare, commerce, and education [1].

This diversity, coupled with our dependence on these systems, has introduced significant security concerns, complicating update management and defense against newly emerging threats [3]. According to a study by Statista [4], 30% of cybersecurity incidents reported by organizations occurred due to hacking, 16% due to misuse, 13% due to malware, and the remainder divided between other actions (social and errors).

The expanded attack surface has fostered a proliferation of vulnerabilities, with studies indicating that the average Web application contains 33 vulnerabilities, 19% of which may permit attackers to take control of the application or underlying operating system [5]. Furthermore, the incidence of Web application attacks has surged by approximately 88%, surpassing projected annual growth rates [3]. Given the accessibility of Web applications through networks, they are susceptible to a multitude of attacks; any vulnerability, once detected, can result in severe financial, reputational, or even physical consequences. In the event of a data breach, an attacker could compromise an application's database, facilitating data sharing or creating additional entry points for future attacks [6].

To systematically address and mitigate these vulnerabilities, many security practitioners refer to the Open Worldwide Application Security Project (OWASP), which provides a structured "OWASP Top 10" list that highlights the most critical risks for Web applications [7]. Additionally, many organizations deploy vulnerability scanners to aid in software security, enabling continuous code monitoring throughout development and deployment. These tools are indispensable for secure software development, as they automate vulnerability detection processes [6]. Security analysis methods have similarly advanced, integrating more refined protection and detection techniques, as traditional manual testing approaches are neither scalable nor adequate [8].

This study is motivated by the need for the automation of Web vulnerability detection, thereby reducing the necessity for extensive user intervention. The main goal is to design an efficient, scalable, and free solution that facilitates the detection and reporting of security flaws in a structured, user-friendly format. Existing vulnerability scanners typically support various types of analyses, such as crawling, active scanning, and authentication testing. However, these tools often require additional configurations or manual adjustments from users to tailor the scans to specific needs. In contrast, the proposed platform provides a comprehensive analysis of Web applications without requiring any additional configurations from the user. Users only need to complete two straightforward steps: populate a target file with basic application details and execute a single command to initiate the process. This streamlined approach ensures that even complex analyses, encompassing multiple scanning techniques, are conducted automatically. Additionally, the platform delivers results in a clear and organized format, making them accessible to users with limited technical expertise, thereby addressing the common barriers associated with existing tools. By incorporating two distinct automation strategies from an existing vulnerability analysis tool, this platform optimizes vulnerability identification and reporting, addressing critical challenges in Web application security. This approach ensures a more automated, cost-effective, and accessible method for security analysis. In doing so, this research contributes to the advancement of application security by simplifying and enhancing the automation of vulnerability detection in Web applications.

This work focuses specifically on the security of Web applications based on the methodology developed Seara and Serrao [9,10], who designed a distributed system for detecting vulnerabilities in general systems. The Intelligent System for Automation of Security Audits (SIAAS) tool was designed as an accessible vulnerability scanning system that does not require specialized knowledge to operate, employing a server–agent architecture to enhance system scalability. This architecture comprises multiple agents connected to a central server through secure HTTPS communication. The agents analyze the local network, identify hosts, and conduct vulnerability scans using the "Nmap" tool. The collected data are subsequently transmitted to the central server, allowing for centralized management of the results. Additionally, a command-line interface (CLI) was developed to facilitate human interaction with the server API, enabling configuration management and data visualization. The solution in this study further evolves the concept by specifically targeting Web

application security while preserving the low resource usage and zero-cost accessibility that characterize the original framework. This advancement ensures the solution's scalability and practicality, particularly for small organizations or individuals seeking robust security measures without incurring additional costs.

The article is organized as follows: An introduction to the topic, outlining the motivation and objectives of the research is provided in the next section. Section 2 reviews related work, examining methods and findings from previous studies to underscore the relevance of the current approach. Following this, Section 3 provides a detailed account of the system architecture and implementation process. Section 4 then presents the findings from validation tests conducted solely within internal environments. Finally, Section 5 summarizes the achievements of this work and suggests directions for future research.

2. Related Work

Over recent years, the frequency of attacks targeting Web applications has escalated significantly, making these applications prime targets for cyber attacks. For instance, in 2017, a study reported an average of 44 daily attacks on a small business website, with projections indicating an increase of over 50% by 2019 [11]. A study by Statista shows that the sectors most attacked between 2022 and 2023 were the finance and information sectors, followed by the professional sector and public administration [12]. Ensuring Web application security is critical for safeguarding the sensitive data such apps contain, yet the rapid growth in Web applications limits the capacity of security professionals to manually inspect each one. This reality highlights the need for automated tools to detect and remediate vulnerabilities effectively [13]. Additionally, the lack of security emphasis during the development phase further contributes to vulnerabilities, as current platforms for Web development and analysis offer limited support for security, often leading to errors. This lack of support places additional demands on developers already constrained by market pressures, time limitations, and insufficient security knowledge [14]. According to [15], Web applications are often developed without adequate attention to secure coding practices, and less than 50% are tested for security before deployment.

To understand Web application security, it is essential to first define a Web application, including its architecture and operational framework. A Web application is essentially software accessed through a Web browser [5], relying on various components and technologies that constitute the Web platform. This ecosystem includes HTTP protocols, Web servers, server-side technologies (e.g., CGI, PHP, and ASP), browsers, and client-side technologies like JavaScript and Flash [14]. The architecture of Web applications typically includes both server-side and client-side code, enabling dynamic information delivery and interaction. Server-side code generates HTML pages, often interacting with databases to store and retrieve data, while client-side code embedded in HTML pages (such as AJAX) allows users to engage dynamically with the content. Different applications serve diverse business objectives, and thus, architectural choices must align with the specific business functions they support. For example, recent trends indicate a shift towards Web APIs, which enable local applications to interact with the server, enhancing efficiency [6].

Within this complex architecture, Web application vulnerabilities arise from various sources. A vulnerability represents a flaw or weakness that an attacker can exploit to compromise the integrity of a system [16]. Web application vulnerabilities often stem from not only bugs within the application itself but also from malware present in the user's system, potentially leading to data theft or unauthorized access. Weak security practices, such as using simple passwords, also contribute to vulnerabilities by creating exploitable entry points [16]. As vulnerabilities have grown, standardized categorization methods have emerged to help security teams prioritize risks. Key among these is the OWASP Top

10, a widely recognized consensus on critical security risks for Web applications [7]. This categorization, maintained by the Open Worldwide Application Security Project (OWASP), serves as a reference for identifying and mitigating the most prevalent vulnerabilities in Web applications. The OWASP Top 10 highlights the most critical Web application security risks, categorized as follows: broken access control, where flawed permissions allow unauthorized actions or privilege escalation; cryptographic failures, involving inadequate encryption of sensitive data, risking confidentiality and integrity; injection, such as SQL injection, exploiting unvalidated inputs to execute malicious commands; insecure design, arising from applications built without robust security considerations; security misconfiguration, where incomplete or improper system setups create vulnerabilities; vulnerable and outdated components, i.e., using unsupported or unpatched software, which increases the attack surface; identification and authentication failures, exposing flaws in session management or authentication processes; software and data integrity failures, where a lack of integrity checks can lead to code tampering or data corruption; securing, logging, and monitoring failures, with insufficient logging making attacks harder to detect and respond to; and Server-Side Request Forgery (SSRF), enabling attackers to manipulate server requests to access unauthorized resources. Each category represents a specific set of weaknesses that, if exploited, can compromise the security of applications and sensitive data. Additionally, the Common Weakness Enumeration (CWE) complements OWASP by listing the top 25 software risks, focusing on design elements that may lead to vulnerabilities [17].

For effective vulnerability analysis, two primary approaches are commonly used: black-box and white-box testing. In black-box testing, which does not require knowledge of the internal code, testers analyze application functionality based on inputs and outputs. White-box testing, on the other hand, requires full access to the application's code, enabling a more comprehensive analysis of the code structure, cycles, and conditions [18]. Another critical aspect of vulnerability analysis is penetration testing, a method to identify security weaknesses [19]. Penetration tests can be manual, involving skilled individuals who use specific tools and experience to identify vulnerabilities, or automated, using scripts and vulnerability scanners that facilitate rapid assessment. While automated testing offers speed and consistency, it is prone to false positives and may miss complex vulnerabilities detectable only through manual testing [20].

Another emerging solution is the concept of cyber ranges, which offer virtualized environments for testing and training in cybersecurity scenarios. Cyber ranges provide a practical, hands-on approach to training security professionals by simulating real-world attack and defense scenarios [21]. However, traditional cyber ranges often suffer from scalability and flexibility limitations. Many rely on manual setups or preconfigured "in-a-box" solutions that are expensive and lack agility. To address these limitations, frameworks integrating Infrastructure as Code (IaC) practices, such as Ansible, are gaining traction. For example, Ansible has been used to automate cyber-range workflows, enabling the rapid development and deployment of both network- and application-specific attack scenarios in cloud environments. This approach ensures scalability, efficiency, and reduced error rates while lowering costs [21].

Vulnerability scanners are essential tools for automated Web application security testing. These tools fall into two categories: Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST). SAST tools follow a white-box approach, examining the application's source code for vulnerabilities through data flow analysis [22,23]. However, SAST requires compatibility with the application's language and framework, limiting its scope [20]. DAST tools, which use a black-box approach, simulate user interactions with the application via the Web interface, typically progressing through several phases: configuration, crawling (mapping the application structure), attack (testing for vulnera-

bilities), and analysis (evaluating responses for security risks) [24–26]. Studies assessing scanners' performance indicate that static scanners generally outperform dynamic ones in identifying vulnerabilities and locating their exact positions within the code, though dynamic scanners excel in detecting injection vulnerabilities [6]. However, as Web security threats evolve, such as the rise of broken access control risks, dynamic scanners are less effective in detecting these critical issues compared to static tools. For the purposes of this research, we focus exclusively on non-commercial dynamic scanners, considering the following: Arachni, Dirb, Nikto, Nmap, ZAP, Skipfish, Vega, W3af, Wapiti, and Whatweb.

The rapid advancement of Web application security tools has led to the emergence of various penetration testing and vulnerability assessment solutions. However, studies consistently demonstrate that no single tool can comprehensively detect all potential vulnerabilities, emphasizing the importance of utilizing multiple scanners in combination. To address this limitation, platforms integrating multiple tools have been developed to capitalize on the unique strengths of each scanner during different stages of vulnerability detection. For example, a Python-based vulnerability detection platform integrates tools like Nmap, Nikto, Whatweb, and Dirb, with each focusing on specific aspects of the scanning process. This multi-tool approach significantly enhances both the efficiency and thoroughness of vulnerability detection [27]. Similarly, ref. [28] proposed a framework combining OWASP ZAP and Arachni, achieving superior accuracy in identifying vulnerabilities compared to using standalone tools. These integrations highlight the potential of interoperability in creating comprehensive and effective security assessment frameworks.

Despite the advancements in vulnerability scanners, several gaps remain. Many existing tools require significant expertise to operate, limiting their accessibility to non-experts. Furthermore, the results provided by these tools are often complex and difficult to interpret for users with limited technical knowledge. Additionally, most tools focus on a narrow set of vulnerabilities, leaving critical gaps in the detection of more complex or emerging threats. This literature review identified the need for a more automated and user-friendly solution that can detect a broader range of vulnerabilities with minimal user intervention. The platform proposed in this article addresses this gap by integrating multiple automation techniques of a scanner to enhance the coverage and usability of vulnerability detection tools.

ZAP was ultimately chosen after being tested alongside these other scanners based on several key factors that set it apart. First, scientific studies and benchmarking articles consistently highlight ZAP's superior performance in detecting vulnerabilities in Web applications compared to other scanners [3,6,29,30]. Additionally, ZAP offers advanced automation solutions. While it features a user-friendly graphical interface, this aspect had no bearing on its selection, as the interface was not intended to be used in this research. Instead, the decision was based on its robust API and automation frameworks, which provide full control over the scanner without requiring interaction through the interface. This flexibility was essential for enabling comprehensive automation and integrating the scanner seamlessly into the proposed platform. Another critical factor was ZAP's active and supportive community, which proved invaluable in addressing challenges, such as defining and implementing a complete and accurate automation plan. This community-driven support facilitated rapid problem solving and knowledge sharing during the development process. Finally, ZAP allows for granular control over scan results, enabling the customization of outputs. This capability was crucial for tailoring the system to provide clear and organized reports that meet the specific needs of the users.

3. Design and Development

This section details the design and development of the proposed system for automating vulnerability detection in Web applications. The system was developed in Python and integrates with the ZAP vulnerability analysis tool via a REST API also implemented in Python to enable automated vulnerability detection across multiple Web applications. The choice of ZAP over other well-known scanners was informed by its strong track record in detecting Web application vulnerabilities, as highlighted in the related work. The primary goal of this research is to create a fully automated system that identifies and reports potential vulnerabilities in Web applications with minimal (ideally zero) operator intervention or specialized cybersecurity knowledge while remaining cost-effective for organizations. This cost-effectiveness is further enhanced by the system's ability to run on resource-constrained hardware, such as a Raspberry Pi or virtual machine. Additional objectives include integrating the new system with previously developed tools, influencing the selection of the technologies and software used. Key characteristics guiding the system's implementation include the use of open-source tools, a plug-and-play characteristic, scalability, and secure communication via HTTPS with authenticated data storage and access. The code of the developed system is hosted on GitHub and can be found at the following URLs:

- https://github.com/DiogoMoreira4/siaas-zap.git (accessed on 22 December 2024).
- https://github.com/DiogoMoreira4/siaas-server.git (accessed on 22 December 2024).
- https://github.com/DiogoMoreira4/siaas-cli.git (accessed on 22 December 2024) .

3.1. Technologies Used

The system leverages a carefully selected stack of open-source technologies, each chosen to meet the objectives of cost-efficiency, flexibility, and performance:

- Python: Selected for its simplicity and vast library support, especially for integrating with ZAP via its API. Python 3.8.10 was used for development.
- ZAP: The core scanning tool used to detect vulnerabilities in Web applications. ZAP 2.15.0 was integrated into the system using its REST API.
- Linux: The system was deployed on Ubuntu 20.04 LTS to ensure stability, security, and cost-effectiveness, as Ubuntu is a free and open-source operating system, unlike Windows, which requires paid licenses. While optimized for Ubuntu, the system can also run on other Debian-based distributions, offering flexibility in deployment.
- MongoDB: A NoSQL database used to store the results of vulnerability scans. MongoDB was chosen for its compatibility with JSON documents, which align well with the data generated by ZAP scans.

These tools collectively create a robust foundation for the automated vulnerability detection system, supporting all core functionality while meeting the objectives of scalability and user accessibility.

3.2. System Architecture

The system architecture for automated vulnerability detection in Web applications comprises three core modules: SIAAS-server, SIAAS-CLI, and siaas-zap. These modules interact seamlessly to enable automated scanning, with a centralized API facilitating communication across the system:

1. **SIAAS-server**: The central server module manages communication between the system components and the user. It is responsible for processing user requests, managing configuration files, and storing the results of vulnerability scans.

2. **SIAAS-CLI**: A command-line interface that allows users to interact with the system without needing to access a graphical user interface. It simplifies the process of setting up scans and retrieving results.
3. **SIAAS-ZAP**: This module handles the integration with ZAP, automating the scanning of Web applications for vulnerabilities. It ensures that multiple targets can be analyzed concurrently by creating separate instances of ZAP for each target.

The interaction between these components is illustrated in the system's architecture diagram (Figure 1). The first step is to configure the *target.ini* file with the correct information about the Web applications to be scanned (1). For applications that require login, the user must provide details such as the application name, URL, username, password, and login URL. If no login is required, the user only needs to specify the application name and URL. Once the configuration file is set, the user initiates the scan (2) by running the following command: `sudo systemctl start zap_manager`.

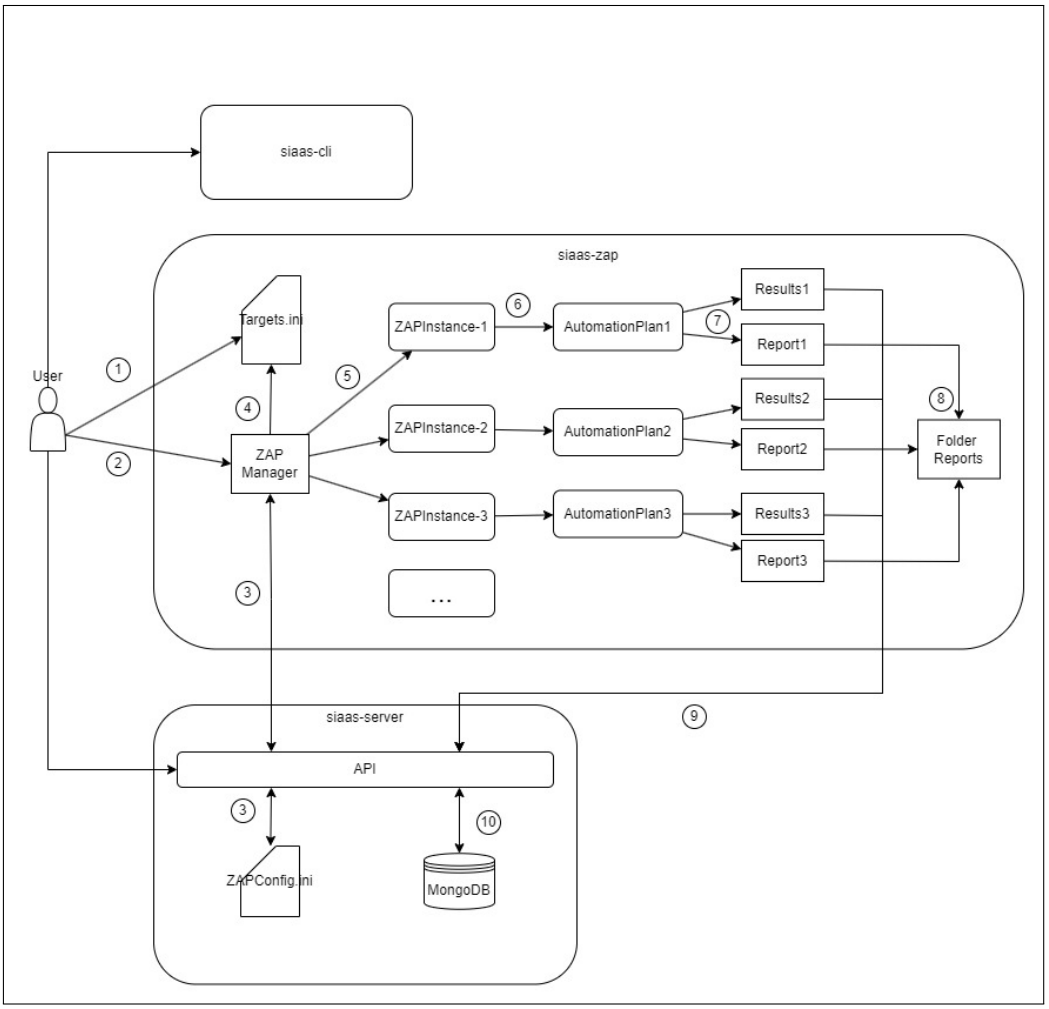

Figure 1. Overall architecture and flow of the developed system.

This command starts the `zap_manager` process, which retrieves automation plan configurations from the SIAAS-server through an API call (3) and reads the `target.ini` file (4). The `zap_manager` creates an instance of ZAP for each target (5). Each ZAP instance runs in a separate directory and port, analyzing targets sequentially to ensure performance and prevent system overload. This ensures the system remains cost-efficient and can run on limited hardware resources. The automation plan is then created for each target, and necessary fields are filled based on the ZAP configuration and target details (6). When a scan is complete, the Python API of ZAP generates a report summarizing vulnerabilities and

outputs a JSON file with the URLs discovered during crawling, vulnerability alerts, and a summary of the automation plan's execution (7). These reports are stored in a reports folder within the SIAAS-ZAP module (8). The JSON object is then sent to the server via an HTTPS request (9), ensuring data security before being stored in the MongoDB database (10).

3.3. Functionality and Automation

SIAAS-ZAP is the module responsible for automating the detection and analysis of vulnerabilities in Web applications. The system was designed to be used by operators without requiring deep expertise. To achieve this, understanding how ZAP works and its available automation options was crucial. The goal was not only to ensure quality results but also to achieve a high degree of automation in the detection process. After evaluating various automation options provided by ZAP, it was determined that the automation framework combined with the ZAP Python API was the best approach. While more complex to implement, this combination offers the greatest flexibility and access to all of ZAP's functionalities [31].

3.3.1. Automation Framework

The automation framework enables control of ZAP using a YAML file, supporting three important domains:

- Environment: Specifies the application on which the jobs (scans) operate;
- Authentication: Crucial for analyzing applications requiring login, allowing for the use of all authentication mechanisms supported by ZAP;
- Jobs: Represent the different ZAP functionalities, such as spider, spiderAjax, and activeScan.

The Framework's flexibility ensures the automation plan is fully adjustable to the user's needs, supporting complex, authentication-based tests that extend ZAP's scanning capabilities across diverse scenarios.

3.3.2. ZAP Python API

The ZAP Python API provides near-complete control over ZAP's scanning functionalities, akin to those available in ZAP's desktop interface. The API is central to the system's automation capabilities, allowing for the implementation of key functionalities, such as initiating scans, monitoring progress, and retrieving results.

The following API endpoints are utilized in the solution:

- Executing the automation plan: "`GET /JSON/automation/action/runPlan/`" is used to trigger the predefined automation plan, enabling the system to start a comprehensive vulnerability analysis automatically.
- Monitoring analysis progress: To track the progress of the ongoing scans, the "`GET /JSON/automation/view/planProgress/`" and "`GET /JSON/ascan/view/scanProgress/`" endpoints are employed. These endpoints provide real-time updates on the execution of the automation plan and the active scanning phase, respectively.
- Building the results: Once the scan is completed, the "`GET /JSON/core/view/urls/`" and "`GET /JSON/core/view alerts/`" endpoints are used to retrieve discovered URLs and detected vulnerabilities. These data form the basis of the comprehensive and user-friendly reports generated by the system.

The use of these endpoints allow the system to achieve a high level of automation, seamlessly integrating ZAP's capabilities into the proposed platform. The API's flexibility ensures that all scanning tasks can be orchestrated programmatically, without requiring manual intervention, aligning with the research goal of creating a fully automated solution for vulnerability detection.

3.3.3. Workflow and Key Functionalities

The following demonstrates some of the main flows and key functionalities of the solution.

- Target File Reading: When the `zap_manager` service is started, it first reads the *targets.ini* file. This file contains the necessary details to scan the Web applications or sites the user wishes to analyze as shown in Figure 2. The user must edit the *targets.ini* file before starting the service. The function creates a list of target objects, each with a boolean attribute `has_auth`. If the user provides credentials for authentication, the `has_auth` value is set to true; otherwise, it remains false.

```
[Security Tweets]
name:securitytweets
url:http://testhtml5.vulnweb.com

[Acuart]
name:acuart
url:http://testphp.vulnweb.com
username:test
password:test
loginpage:http://testphp.vulnweb.com/login.php
```

Figure 2. Example of the file to fill in with target information.

For example, in the targets.ini file, if the "SecurityTweets" target does not require authentication, the `has_auth` attribute is set to false. Conversely, for the "Acuart" target, which includes login credentials, `has_auth` is true, and the corresponding automation plan includes authentication.

- Automation Plan Modification: Once the targets are read, the system modifies the necessary fields in the corresponding automation plan. For targets without authentication (e.g., "SecurityTweets"), the system uses a basic automation plan, while targets with authentication use a plan adapted to the provided login credentials. These plans are YAML files, as illustrated in Figure 3, that control ZAP's behavior, such as the maximum time allowed for each Spider and ActiveScan job. These configurations are stored server-side and can be modified by the user via command-line instructions. These automation plans are divided into two parts: the env part, corresponding to the environment and context in which we want to analyze the application, and the jobs part, which identifies which functionalities and analyses we want to carry out during the execution of the automation plan. In the following, we explain in more detail the three main jobs used, spider, ajax spider, and active scan.
- Authentication: Authentication is a key feature of the system, as it allows for a more thorough and accurate vulnerability analysis. ZAP supports various authentication methods:
  - Manual Authentication: The user logs in manually via a browser while using ZAP as a proxy (not used in this project);
  - Form-based Authentication: Handles login forms where credentials are submitted (username and password);
  - JSON-based Authentication: Similar to form-based but submits credentials as a JSON object;
  - HTTP/NTLM Authentication: Used for environments requiring authentication through HTTP headers (common in corporate settings);

  - Script-based Authentication: For more complex scenarios, allowing users to define custom scripts to manage the login process.

  Additionally, browser-based authentication was implemented through the automation framework. The browser-based authentication in ZAP leverages Selenium, a browser automation tool, to handle login processes that require interaction with a graphical user interface. Through this method, ZAP launches a browser instance (either visible or headless) to simulate user actions such as filling in login credentials, clicking buttons, and interacting with dynamic elements like JavaScript-based forms or CAPTCHA solutions. Once the authentication is successful, ZAP captures session cookies or tokens from the browser, enabling automated authenticated scans of protected resources. This approach is particularly effective for complex login flows, such as Single Sign-On (SSO) or applications with heavy JavaScript frontends, although it has limitations, such as the inability to manage logins triggered exclusively by the "Enter" key or scenarios involving non-graphical APIs. By using this type of authentication, the system handles authentication intelligently, making it possible to automate this process.

- Spider: The Spider process involves discovering URLs by sending HTTP requests through a pool of threads managed by the SpiderController. Responses are analyzed by parsers to extract new URLs and resources, which are added to the thread pool for further crawling. The process continues until all URLs are visited or manually stopped, ensuring thorough coverage of the site.
- Ajax Spider: The Ajax Spider operates similarly to the traditional spider but uses a WebDriver (such as Selenium) to automate a browser, allowing ZAP to interact with dynamic elements and run JavaScript within the target application. This enables ZAP to discover and analyze URLs that require AJAX requests and other dynamic interactions.
- Active Scan: The active scan component performs active attacks on the Web application to detect vulnerabilities. It sends various HTTP/HTTPS requests, including malicious payloads, to test for weaknesses such as SQL injection and Cross-Site Scripting (XSS). The scan policy defines the strength of the attacks and the alert thresholds for vulnerabilities. For example, during an attack on the "JuiceShop" application, the system sends a POST request with an injection payload in the email parameter. If the server responds with an SQL error, ZAP generates a high-confidence alert indicating a successful SQL injection attack.
- Results: Once the analysis is complete, the SIAAS-ZAP module generates two types of reports: a detailed ZAP report and a JSON file with four main components (target, URLs, alerts, and plan). The target component identifies the application being analyzed, while the URL component lists all discovered URLs from the Spider and Ajax Spider processes. The alerts component contains all the vulnerability alerts triggered during both passive and active scans. The plan component summarizes the automation process, showing whether the scan was executed as expected and providing details about the duration and execution of each job. These results are accessible through the API and command-line interface, allowing users to view scan progress and outcomes.

```yaml
---
env:
  contexts:
  - name: context
    urls:
    - http://exemplo
    includePaths:
    - http://exemplo.*
    excludePaths:
    authentication:
      method: "browser"
      parameters:
        loginPageUrl: http://exemplo/login.php
        loginPageWait: 5
        browserId: "firefox-headless"
      verification:
        method: "autodetect"
        loggedInRegex:
        loggedOutRegex:
        pollFrequency:
        pollUnits:
        pollUrl:
        pollPostData: ""
    sessionManagement:
      method: autodetect
      parameters: {}
    technology:
      exclude: []
    users:
    - name: admin
      credentials:
        password: password
        username: user
  parameters:
    failOnError: true
    failOnWarning: false
    progressToStdout: true
  vars: {}
jobs:
- parameters:
    scanOnlyInScope: true
    enableTags: false
    disableAllRules: false
  rules: []
  name: "passiveScan-config"
  type: "passiveScan-config"
```

Figure 3. Example of a part of the automation plan file after it has been filled in with target information.

The results in the JSON file can be consulted in the server's API and in the command line. Figure 4 shows the structure of how the results are made available to the user via the server API.

As previously mentioned, the results are divided into four components: a target component that identifies the name associated with the application to be analyzed; the URL component, which shows the user all the URLs found during the crawling process, i.e., the results of the Spider and Ajax Spider jobs; the alerts component, where the user can find all the alerts discovered during the analysis of the application; and, finally, the plan component, which presents a summary of what happened to the respective target during the execution of the automation plan. As shown in Figure 5, in the plan component of each result from

the analysis of a target, there is information that helps the user understand whether the automation plan went as expected or whether errors or warnings were observed. We also receive information such as the start and end time of the scan, the jobs that were run during that automation plan, and some information about those jobs.

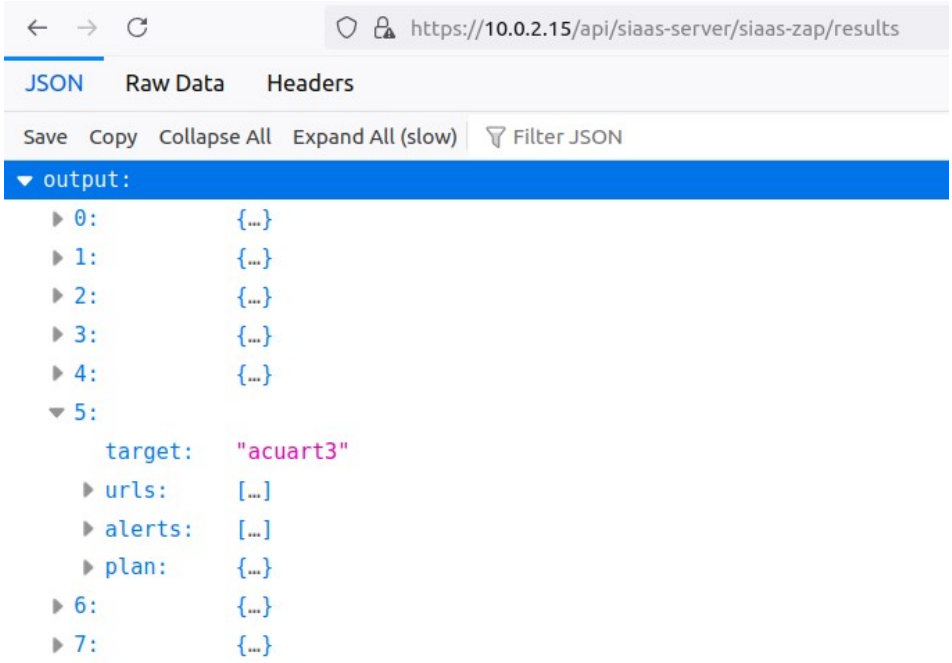

Figure 4. Visualization of results via the server API.

The URL component (Figure 6), shows all the URLs related to the target that were found by ZAP in the crawling process. The number of URLs we see in the plan component in Figure 5 is sometimes higher because it takes into account some URLs that do not belong to the target, i.e., they are found but are not related to the target.

The alerts component shown in Figure 7 is where the alerts that ZAP identified during the entire application analysis process are stored, from passive scans looking for URLs, as well as during the actual attack on the application in the active scanning process. Figure 7 shows the structure of the generated alert.

The developed system identifies potential vulnerabilities in targets and generates alerts, which are presented to the user. These alerts include key attributes such as risk, alert, description, tags, confidence, and solution, as shown in Figure 7. The alert field specifies the category or class of vulnerability associated with the alert. The confidence field indicates the scanner's level of certainty about the triggered alert, with levels categorized as high, medium, and low. The description provides the rationale behind the alert, while risk assigns a severity level to the vulnerability, ranging from high, medium, and low to informational. The solution field offers potential remediation to address the identified vulnerability. Lastly, the tags field maps the detected vulnerability to OWASP and CWE taxonomies, providing links for detailed information. It is essential to clarify that the system does not validate the existence of the vulnerabilities; its primary function is automating their detection. The confidence field plays a crucial role by guiding the user on the likelihood of the identified vulnerabilities. Specifically, a high level suggests that ZAP has clear evidence of a successful attack or a lack of security mechanisms, indicating a strong probability of vulnerability. A medium level indicates the presence of signs of a vulnerability without confirmation of exploitation. Finally, a low level implies suspicious behavior or configuration errors detected by ZAP but with no definitive proof of exploitability.

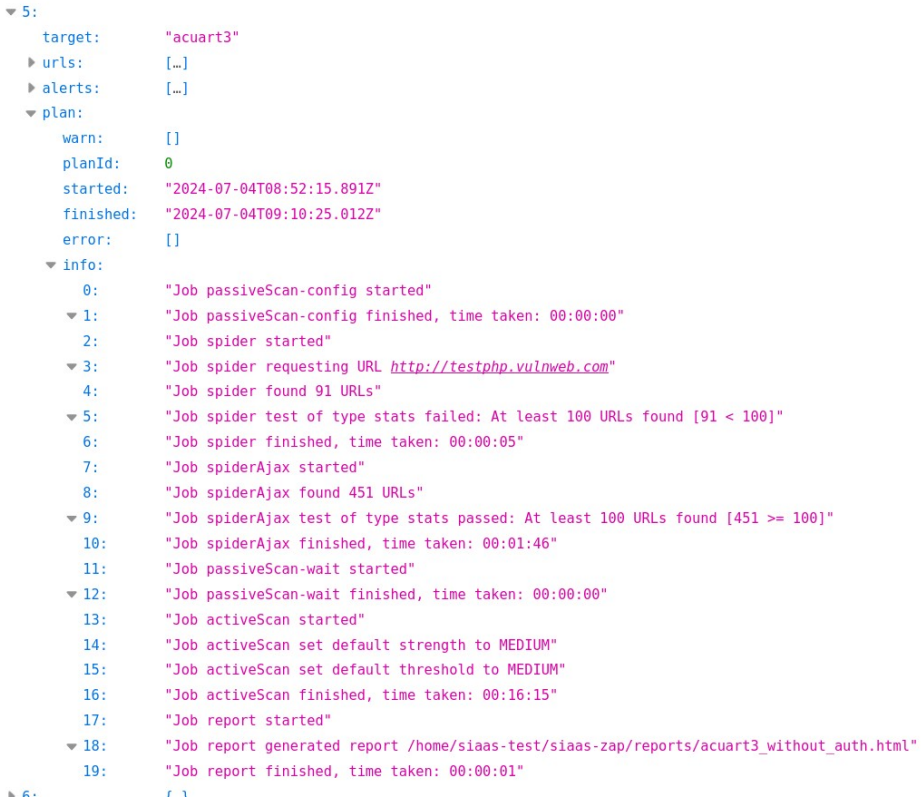

Figure 5. Visualization of the "plan" component via the server API.

Figure 6. Visualization of the "URL" component via the server API.

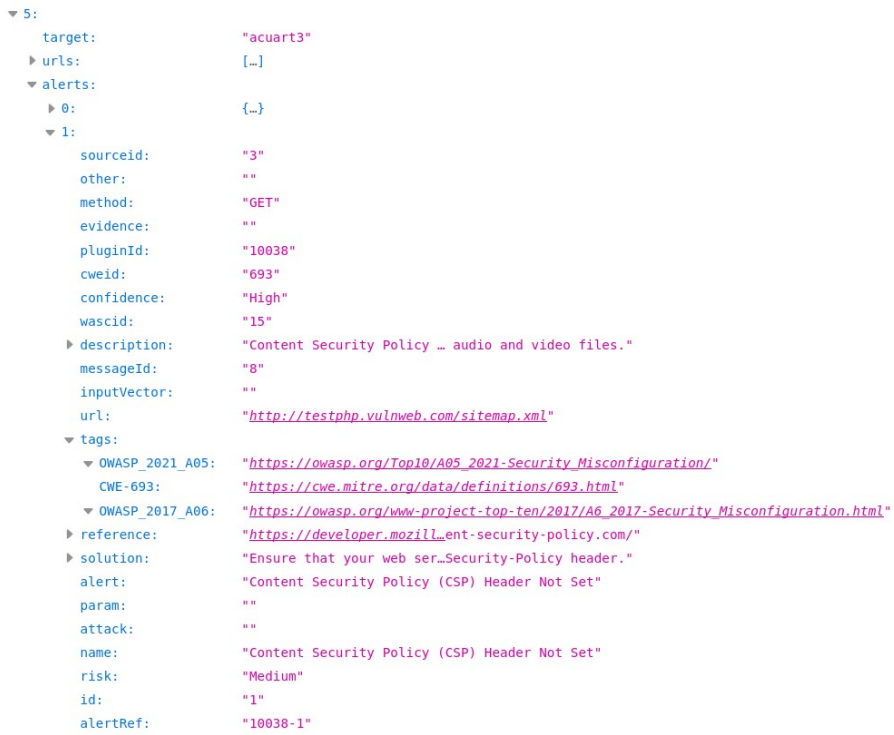

Figure 7. Visualization of the "alerts" component via the server API.

4. Tests and Discussion of Results

This section describes the testing conditions, the analyzed targets, and the outcomes of tests conducted with our developed automated vulnerability detection system. The system, designed to streamline vulnerability identification for websites and Web applications, uses ZAP as the primary tool for conducting analyses and generating reports. The results outline the specific risks for which the tested applications are vulnerable, categorized according to the OWASP Top 10 standard. The tests were executed on a single computer equipped with an AMD Ryzen 7 5800H processor with Radeon Graphics running at 3.20 GHz, with 16 GB of RAM and a 960 GB SSD. This machine hosted a virtual machine (VM) via VirtualBox, allocated with 6 GB of RAM, four processors, and 25 GB of storage, running Ubuntu 20.04 LTS (Focal Fossa).

To assess the system's viability, suitability, and versatility, several tests were performed on various websites and Web applications. In compliance with the Portuguese cybercrime law [32], which prohibits vulnerability scanning without owner consent, testing was restricted to three specific groups of applications:

- Applications with Known Vulnerabilities: These targets were selected to evaluate vulnerability scanning tools, each with prior authorization for testing. Most applications were Web-hosted, with DVWA running locally in a separate VM. The applications tested included:
  - Acuart (http://testphp.vulnweb.com) (accessed on 22 December 2024);
  - Acuforum (http://testasp.vulnweb.com) (accessed on 22 December 2024);
  - Altoro Mutual (http://demo.testfire.net/index.jsp) (accessed on 22 December 2024);
  - bWAPP (https://demo.weblock.ru) (accessed on 22 December 2024);
  - DVWA (http://10.0.2.4/DVWA) (accessed on 22 December 2024);
  - JuiceShop (https://juice-shop.herokuapp.com) (accessed on 22 December 2024);
  - Rest API (http://rest.vulnweb.com) (accessed on 22 December 2024);
  - SecurityTweets (http://testhtml5.vulnweb.com) (accessed on 22 December 2024).

- CMS Applications: The system was further tested on real content management systems running locally in containers, ensuring compliance with ethical standards by avoiding live user interaction. This setup enabled the system to perform vulnerability detection and validation under realistic conditions.
  - WordPress (http://localhost:8070) (accessed on 22 December 2024);
  - Drupal (http://localhost:8071) (accessed on 22 December 2024);
  - Joomla (http://localhost:8072) (accessed on 22 December 2024).
- Real Site: The system was also tested on a real site in a quality environment, which included authentication. The analyzed site was ISCTE's Fenix, at the following URL: https://fenix-qua.iscte-iul.pt) (accessed on 22 December 2024).

These tests aimed to assess the system's ability to detect a wide range of vulnerabilities in both controlled and real-world environments and applications.

4.1. Deliberately Vulnerable Applications

In this subsection, we analyze the alerts generated in tests on purposefully vulnerable applications. The alerts are grouped according to the OWASP Top 10 categorization. These are the alerts that were generated with the high and medium risk-level property.

Table 1 provides a comprehensive overview of high- and medium-risk vulnerabilities identified across various intentionally vulnerable Web applications, analyzed using the OWASP 2021 Top 10. The applications included in this analysis—Acuart, Acuforum, Altoro, bWAPP, DVWA, JuiceShop, RestAPI, and SecurityTweets (ST)—are purposefully designed to contain a range of vulnerabilities, making them ideal for testing the detection capabilities of the developed system. This table highlights the vulnerabilities detected by the scanner, focusing on critical areas within access control, injection vulnerabilities, and security misconfiguration.

The primary classes of detected vulnerabilities fall within `OWASP_2021_A01` (broken access control), `OWASP_2021_A03` (injection), and `OWASP_2021_A05` (security misconfiguration). Under broken access control `(OWASP_2021_A01)`, the analysis revealed several vulnerabilities related to insufficient enforcement of access restrictions, including issues such as CORS misconfiguration, cross-domain misconfigurations, bypassing of HTTP 403 errors, and the absence of anti-CSRF tokens. These weaknesses, present across applications like Acuart, Acuforum, and bWAPP, indicate a consistent pattern of poor access control configurations, which could allow for unauthorized access to sensitive application functions or data.

Injection vulnerabilities `(OWASP_2021_A03)` were also prevalent, particularly across applications such as DVWA, Altoro, and Acuart. The system identified multiple types of injection attacks, including SQL injection, advanced SQL injection, and cross-site scripting (XSS), which expose applications to the risk of unauthorized data access and potential data corruption. Additionally, vulnerabilities such as NoSQL injection and remote file inclusion were noted, reflecting the diverse range of injection risks that can compromise application integrity. These findings underscore the need for rigorous input validation and secure coding practices to prevent attackers from injecting malicious code or commands into vulnerable applications.

Security Misconfiguration `(OWASP_2021_A05)` emerged as another major category of detected vulnerabilities, with recurring issues like missing CSP headers, backup file disclosure, insecure HTTP methods, and the lack of anti-clickjacking headers across several applications, including JuiceShop and bWAPP. Security misconfigurations make applications vulnerable to a variety of attacks, often due to incomplete or improperly implemented security controls. The presence of these vulnerabilities across multiple applications suggests a lack of consistency in configuration practices, exposing systems to risks that could otherwise be mitigated through careful configuration management.

Table 1. Overview of vulnerability alerts generated by the system in tests carried out on intentionally Vulnerable Web Applications.

	Acuart	Acuforum	Altoro	bWAPP	DVWA	JuiceShop	RestAPI	ST
OWASP-2021-A01								
CORS Misconfiguration	X					X		X
Cross-Domain Misconfiguration	X					X		X
Bypassing 403		X						X
Absence of Anti-CSRF Tokens	X	X	X	X	X			X
Path Traversal		X			X			
Directory Browsing					X			
Session ID in URL Rewrite						X		
OWASP-2021-A03								
Cross-Site Scripting	X	X	X		X			
Advanced SQL Injection	X	X				X		X
SQL Injection	X	X			X	X		
External Redirect		X						
Integer Overflow Error		X						
Cookie Slack Detector			X					
NoSQL Injection					X			X
Remote File Inclusion					X			
XSLT Injection					X			
Open Redirect						X		
OWASP-2021-A04								
Exponential Entity Expansion								X
Parameter Tampering					X			
OWASP-2021-A05								
CSP Header Not Set	X	X	X	X	X	X	X	X
Anti-CSRF Token Check	X	X	X		X		X	X
Backup File Disclosure	X							
HTTP Only Site	X	X			X		X	X
Missing Anti-clickjacking Header	X	X	X	X	X	X	X	X
Source Code Disclosure	X	X	X		X			X
Sub-Resource Integrity Missing			X		X		X	X
Insecure HTTP Method			X					
Relative Path Confusion			X		X			
Web Cache Deception			X			X	X	
Proxy Disclosure				X				
Application Error Disclosure					X			
OWASP-2021-A09								
Vulnerable JS Library							X	X
OWASP-2021-A10								
Server-Side Request Forgery					X			

Beyond the primary categories where the system generated the most alerts, it was also able to detect and report vulnerabilities in additional categories. Although these detections were less frequent, it is significant to note that the system demonstrated the capability to identify vulnerabilities across other areas beyond those mentioned previously. This includes alerts within `OWASP_2021_A04` (insecure design), where issues such as parameter tampering and exponential entity expansion were identified. Additionally, some vulnerabilities were detected in `OWASP_2021_A09` (security logging and monitoring failures) and `OWASP_2021_A10` (server-side request forgery). While these alerts were generated in smaller numbers, they emphasize the system's broader detection range, capable of identifying a diverse array of vulnerability types.

However, there were categories in which the system did not generate any alerts, specifically `OWASP_2021_A02` (cryptographic failures), `OWASP_2021_A06` (vulnerable and outdated components), `OWASP_2021_A07` (identification and authentication failures), and `OWASP_2021_A08` (software and data integrity failures). This absence of alerts could imply that the tested applications were adequately protected against these categories of vulnerabilities, or it may indicate that the system lacks the ability to detect issues within these specific classes.

4.2. Content Management Systems (CMSs)

As in the previous section, here, we analyze the alerts generated in the content management system tests. The alerts are grouped according to the OWASP Top 10 categorization, and the alerts to be analyzed were generated with the high and medium risk-level property.

Table 2 summarizes the results of the vulnerability assessments conducted on three content management systems (CMSs): WordPress, Drupal, and Joomla. The detected vulnerabilities are categorized according to the OWASP 2021 Top 10 framework, specifically under categories `OWASP_2021_A01`, `OWASP_2021_A03`, and `OWASP_2021_A05`, each representing a common class of Web security risks.

Table 2. Overview of vulnerability alerts generated by the system in tests carried out on content management systems.

	Wordpress	**Drupal**	**Joomla**
OWASP-2021-A01			
CORS Misconfiguration	X		
Bypassing 403	X		X
OWASP-2021-A03			
SQL Injection	X	X	X
Buffer Overflow	X		
Integer Overflow Error	X		
SQL Injection - SQLite		X	
Server-Side Template Injection			X
Advanced SQL Injection			X
OWASP-2021-A05			
Content Security Policy (CSP) Header Not Set	X	X	X
Source Code Disclosure - SQL	X		
Missing Anti-clickjacking Header	X		
Backup File Disclosure			X

Under `OWASP_2021_A01`, which addresses broken access control, WordPress was found to have vulnerabilities related to CORS misconfiguration and bypassing 403 errors. Additionally, an SQL injection vulnerability was identified, highlighting potential risk points for unauthorized access within the WordPress setup.

In `OWASP_2021_A03`, focused on injection vulnerabilities, a broader range of security issues was detected. WordPress was found to be vulnerable to buffer overflow, SQL injection (specifically SQLLite-based), and integer overflow errors. Moreover, an advanced SQL injection was identified, indicating a severe security risk. For Joomla, a server-side template injection vulnerability was detected, which could allow attackers to execute server-side code within the application.

Finally, `OWASP_2021_A05` pertains to security misconfigurations. For WordPress, several issues were noted, including the absence of a Content Security Policy (CSP) header, source code disclosure, and a missing anti-clickjacking header. Additionally, a backup file disclosure vulnerability was observed, which could expose sensitive configuration files. For Joomla, the system also lacked a CSP header, indicating similar risks in header security configurations.

These results underscore the critical vulnerabilities present in common CMS platforms and highlight the importance of implementing robust security configurations, especially regarding access control and injection prevention measures. The findings suggest that WordPress may require significant improvements in handling CORS configurations, SQL injections, and security header implementations, while Joomla shows specific risks in template injections and security headers. These results emphasize the need for continuous security monitoring and updates within CMS environments to protect against common Web application vulnerabilities.

4.3. Fenix ISCTE Site

During the analysis of the ISCTE website, vulnerabilities were identified across five categories of the OWASP Top 10: broken access control (A01), injection (A03), insecure design (A04), security misconfiguration (A05), and software and data integrity failures

(A08). As we were dealing with a real site, extra validation was carried out to see if the generated alerts really did point to security flaws in the site that could be exploited. High-confidence alerts included issues such as "Content Security Policy (CSP) Header Not Set", "Sub Resource Integrity Attribute Missing", "Server Leaks Version Information via 'Server' HTTP Response Header Field", and "Strict-Transport-Security Header Not Set". While these indicate potential areas for improvement, none was deemed to pose an immediate threat due to the presence of alternative protective mechanisms. A critical risk alert, "Path Traversal", was also triggered but with a low confidence level, meaning the system flagged suspicious behavior but could not gather sufficient evidence to confirm or exploit the vulnerability. This suggests a potential false positive. A deeper investigation confirmed that despite the identified alerts, the application maintains a secure posture. For example, while the "Content Security Policy (CSP) Header Not Set" alert was accurate, the application employed other robust measures to restrict resource loading effectively. Overall, the analysis concluded that the ISCTE website implements adequate security controls, mitigating the highlighted risks.

4.4. Comparison of Analysis Duration

The time taken to run each analysis was recorded and is interesting to observe. The duration of the analyses was only set for demonstration and study purposes, and this variable always depends on the capacity of the hardware on which the SIAAS system runs.

Looking at the graph in Figure 8, we can conclude that the duration of the analyses carried out on the purposely vulnerable applications were all relatively quick (less than an hour), with the "bWAPP" target having the longest analysis of around 58.38 min. The majority of the scans took less than 20 min to complete, with the average duration of scans on purposefully vulnerable applications being approximately 25.7 min.

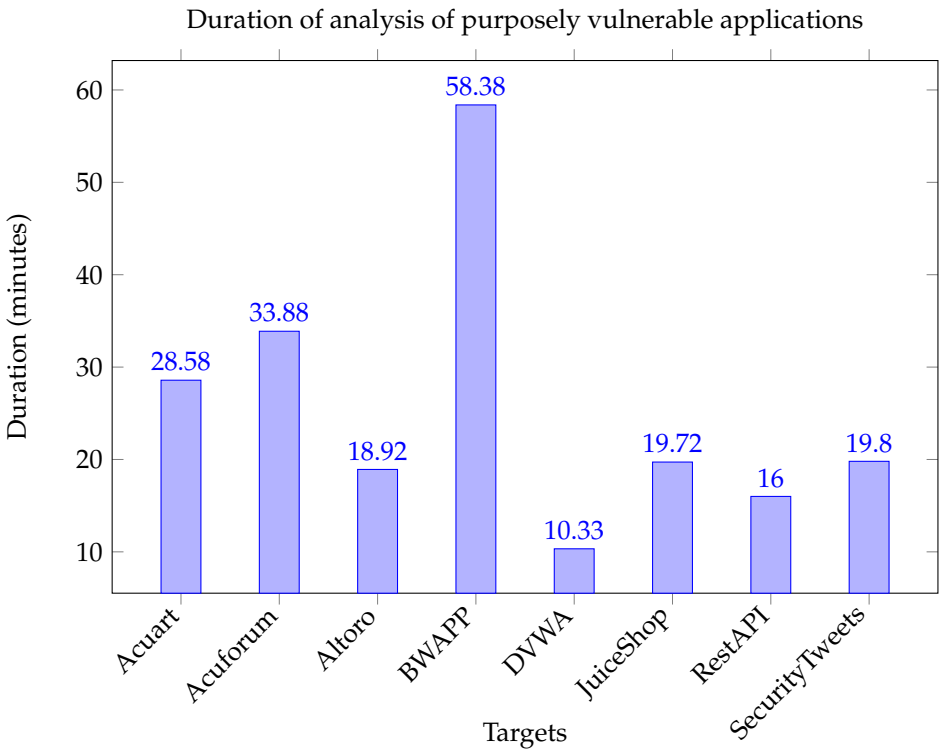

Figure 8. Graph of the duration of analysis of intentionally vulnerable applications by target.

With regard to the duration of the analyses carried out on the Fenix Iscte target and the CMSs, these results were much longer than the times for the first group of applications

tested. As we can see in Figure 9, the longest duration was that of the real site, Fenix Iscte, at 303.1 min, about six times longer than the average of the purposely vulnerable applications. This difference can be explained by the fact that these applications are larger and have complex authentication and permissions logic, which naturally requires more processing time for an exhaustive analysis. Purposely vulnerable applications, on the other hand, are simpler and designed to facilitate fault detection, resulting in shorter analysis times.

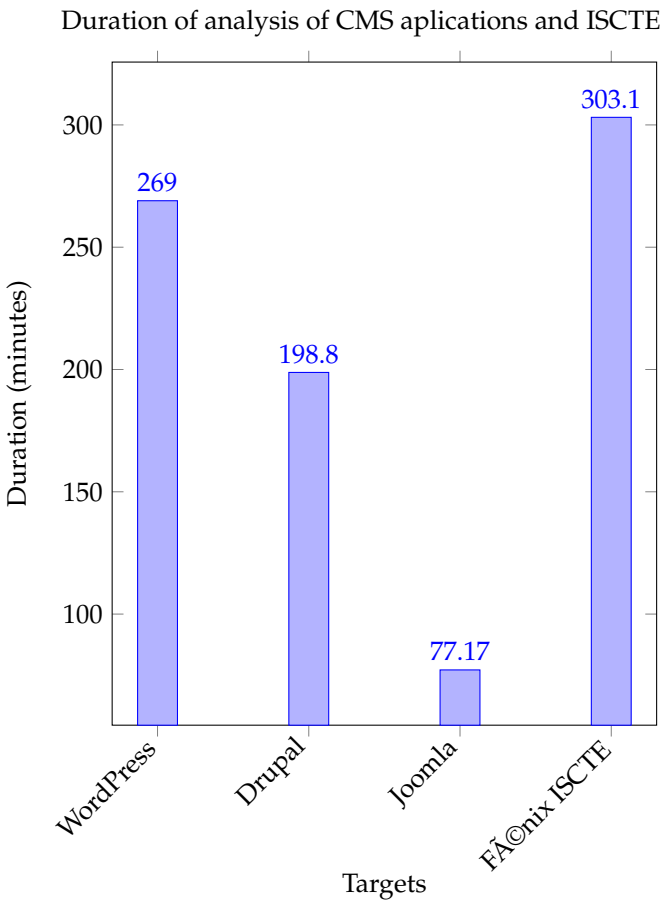

Figure 9. Graph of the duration of analysis of CMS applications and the ISCTE site by target.

5. Conclusions

The complexity of Web application security is underscored by the wide range of vulnerabilities that attackers can exploit. The literature review highlighted the evolution of Web applications, from basic static models to complex, interactive systems. This technological advancement has led to a substantial increase in the number of identified vulnerabilities per application. Through this research, not only were the primary risks and vulnerabilities in Web applications identified, but a system was developed to automate the detection of these vulnerabilities.

Our research findings confirm that developing a platform using a single open-source tool to automate vulnerability detection in Web applications, without requiring high levels of technical expertise, is feasible. The system implements an automated workflow, with sequential steps that allow for comprehensive vulnerability detection and present results to the user in an organized and intuitive manner.

The automated vulnerability detection system demonstrated to be a valuable and effective tool, particularly in meeting its goal of simplifying vulnerability detection for users with limited cybersecurity knowledge. The system successfully identified and categorized

critical vulnerabilities according to the OWASP Top 10 framework, producing results that can be understood by non-specialized users.

Additionally, tests on real-world applications, such as Fenix ISCTE, WordPress, Drupal, and Joomla, running in a controlled environment, validated the system's efficacy in handling authentication processes and detecting security flaws in complex applications. However, limitations were observed in performing authenticated scans on certain types of applications, which represents an area for future improvement.

This research aims to improve application security by providing a straightforward, cost-free solution that informs users—individuals or businesses—about vulnerabilities in their Web applications. By utilizing a single open-source tool, this system enables a more accessible vulnerability detection process, even for users with limited technical experience. This approach allows users to test their applications, identify potential security issues, and take corrective action without relying on complex or costly commercial solutions, thereby promoting safer development and deployment of Web applications. In scenarios where authentication does not work as expected, one potential solution is to test the application in a secure environment with the authentication process temporarily disabled. This approach ensures that the system can perform a thorough analysis of the application's security without being hindered by authentication barriers, providing valuable insights while maintaining the integrity of the testing process.

Considering the limitations identified in this study, future research efforts could aim to advance the system's capabilities in two key areas. First, integrating additional vulnerability analysis tools, such as Nikto or Arachni, would enable a more comprehensive and precise analysis, broadening the range of vulnerabilities the system can detect. Due to the system's design, which prioritizes low computational resource usage, analyses targeting the same application would need to be conducted sequentially. This approach ensures that the system remains lightweight while still allowing the combined strengths of multiple tools to be utilized. Moreover, integrating the results from different tools would require normalization of their outputs to ensure consistency and coherence. This normalization is achievable thanks to the granular control over the analysis process provided by the developed system, which guarantees that results from diverse tools can be aligned and consolidated effectively.

Secondly, developing a management interface to customize results could enhance targeted vulnerability management. This interface would allow users to generate custom reports or structure the output to meet specific needs, further improving usability. Consolidating results from multiple tools within this interface would also facilitate meaningful insights, leveraging the normalized outputs to provide a unified view of detected vulnerabilities.

By addressing these areas, the system could evolve into a more robust and flexible tool for vulnerability detection, capable of accommodating a wider variety of security contexts and user requirements. Such developments would ensure the system's continued relevance and effectiveness, even as security challenges grow increasingly complex.

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

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Acknowledgments: We would like to thank Iscte-Instituto Universitário de Lisboa and ISTAR for providing some resources to perform this research.

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

References

1. Althunayyan, M.; Saxena, N.; Li, S.; Gope, P. Evaluation of Black-Box Web Application Security Scanners in Detecting Injection Vulnerabilities. *Electronics* **2022**, *11*, 2049. [CrossRef]
2. Djeki, E.; Degila, J.; Bondiombouy, C.; Alhassan, M.H. Security Issues in Digital Learning Spaces. In Proceedings of the 2021 IEEE International Conference on Computing, ICOCO 2021, Kuala Lumpur, Malaysia, 17–19 November 2021; pp. 71–77. [CrossRef]
3. 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]
4. Petrosyan, A. Distribution of Cyber Incidents in Organizations Worldwide as of September 2023, by Type. *Statista*. 2024. Available online: https://www.statista.com/statistics/1483769/global-cyber-incidents-by-type/ (accessed on 22 December 2024).
5. Muralidharan, M.; Babu, K.B.; Sujatha, G. W3BnNr: An Automated tool for information gathering, vulnerability scanning, attacking and reporting for injection attacks on web application. In Proceedings of the ACCTHPA 2023—Conference on Advanced Computing and Communication Technologies for High Performance Applications, Ernakulam, India, 20–21 January 2023. [CrossRef]
6. Lavens, E.; Philippaerts, P.; Joosen, W. A Quantitative Assessment of the Detection Performance of Web Vulnerability Scanners. In Proceedings of the ACM International Conference Proceeding Series, Association for Computing Machinery, Vienna, Austria, 23–26 August 2022. [CrossRef]
7. OWASP. OWASP Top 10. OWASP Foundation. 2021. Available online: https://owasp.org/www-project-top-ten/ (accessed on 22 December 2024).
8. Sonmez, F.O.; Kilic, B.G. Holistic Web Application Security Visualization for Multi-Project and Multi-Phase Dynamic Application Security Test Results. *IEEE Access* **2021**, *9*, 25858–25884. [CrossRef]
9. Seara, J.P.; Serrao, C. Automation of System Security Vulnerabilities Detection Using Open-Source Software. *Electronics* **2024**, *13*, 873. [CrossRef]
10. Seara, J.P.; Serrao, C. Intelligent System for Automation of Security Audits (SIAAS). *EAI Endorsed Trans. Scalable Inf. Syst.* **2023**, *11*, 1. [CrossRef]
11. Truong, D.; Tran, D.; Nguyen, L.; Mac, H.; Tran, H.A.; Bui, T. Detecting web attacks using stacked denoising autoencoder and ensemble learning methods. In Proceedings of the ACM International Conference Proceeding Series, Association for Computing Machinery, Hanoi Ha Long Bay, Vietnam, 4–6 December 2019; pp. 267–272. [CrossRef]
12. Petrosyan, A. Global Industry Sectors Most Targeted by Basic Web Application Attacks from November 2022 to October 2023 *Statista* 2024. Available online: https://www.statista.com/statistics/221293/cyber-crime-target-industries/ (accessed on 22 December 2024).
13. Nirmal, K.; Janet, B.; Kumar, R. It's more than stealing cookies—Exploitability of XSS. In Proceedings of the 2018 International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 14–15 June 2018.
14. Li, X.; Xue, Y. A Survey on Web Application Security. *Nashville, TN USA* **2011**, *25*, 1–14. Available online: https://www.isis.vanderbilt.edu/sites/isis.vanderbilt.edu/files/bibcite_files/main_0_0.pdf (accessed on 22 December 2024).
15. Nocera, S.; Romano, S.; Francese, R.; Scanniello, G. Training for Security: Results from Using a Static Analysis Tool in the Development Pipeline of Web Apps. In Proceedings of the International Conference on Software Engineering, Lisbon, Portugal, 14–20 April 2024; pp. 253–263. [CrossRef]
16. Goe, D. Detection of Web Appication Vulnerability Based on RUP Model. In Proceedings of the 2015 National Conference on Recent Advances in Electronics & Computer Engineering (RAECE), Roorkee, India, 13–15 February 2015.
17. MITRE. CWE Top 25 Most Dangerous Software Weaknesses. *Mitre*. 2023. Available online: https://cwe.mitre.org/top25/ (accessed on 22 December 2024).
18. Verma, A.; Khatana, A.; Chaudhary, S. A Comparative Study of Black Box Testing and White Box Testing. *Artic. Int. J. Comput. Sci. Eng.* **2017**, *5*, 301–304. [CrossRef]
19. Hassan, M.M.; Mustain, U.; Khatun, S.; Karim, M.S.A.; Nishat, N.; Rahman, M. Quantitative Assessment of Remote Code Execution Vulnerability in Web Apps. In *Proceedings of the Lecture Notes in Electrical Engineering*; Springer: Berlin/Heidelberg, Germany, 2020; Volume 632, pp. 633–642. [CrossRef]

20. 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.
21. Acheampong, R.; Balan, T.C.; Popovici, D.M.; Rekeraho, A. Security Scenarios Automation and Deployment in Virtual Environment using Ansible. In Proceedings of the 14th International Conference on Communications, COMM 2022, Bucharest, Romania, 16–18 June 2022. [CrossRef]
22. Beba, S.; Karlsen, M.M.; Li, J.; Zhang, B. Critical Understanding of Security Vulnerability Detection Plugin Evaluation Reports. In Proceedings of the Asia-Pacific Software Engineering Conference, APSEC, Taipei, Taiwan, 6–9 December 2021; pp. 275–284. [CrossRef]
23. Al-Kahla, W.; Shatnawi, A.S.; Taqieddin, E. A Taxonomy of Web Security Vulnerabilities. In Proceedings of the 2021 12th International Conference on Information and Communication Systems, ICICS 2021, Valencia, Spain, 24–26 May 2021; pp. 424–429. [CrossRef]
24. Idrissi, S.E.; Berbiche, N.; Guerouate, F.; Sbihi, M. Performance Evaluation of Web Application Security Scanners for Prevention and Protection against Vulnerabilities. *Int. J. Appl. Eng. Res.* **2017**, *12*, 4.
25. Koswara, K.J.; Asnar, Y.D.W. Improving Vulnerability Scanner Performance in Detecting AJAX Application Vulnerabilities. In Proceedings of the 2019 International Conference on Data and Software Engineering (ICoDSE): Gedung Konferensi Universitas Tanjungpura, Pontianak, Indonesia, 13–14 November 2019.
26. Qasaimeh, M.; Shamlawi, A.; Khairallah, T. Black Box Evaluation of Web Application Scanners: Standards Mapping Approach. *J. Theor. Appl. Inf. Technol.* **2018**, *31*, 4584–4596.
27. Jain, T.; Jain, N. Framework for Web Application Vulnerability Discovery and Mitigation by Customizing Rules through ModSecurity. In Proceedings of the 2019 6th International Conference on Signal Processing and Integrated Networks (SPIN), Noida, India, 7–8 March 2019.
28. Abdulghaffar, K.; Elmrabit, N.; Yousefi, M. Enhancing Web Application Security through Automated Penetration Testing with Multiple Vulnerability Scanners. *Computers* **2023**, *12*, 235. [CrossRef]
29. Albahar, M.; Alansari, D.; Jurcut, A. An Empirical Comparison of Pen-Testing Tools for Detecting Web App Vulnerabilities. *Electronics* **2022**, *11*, 2991. [CrossRef]
30. Mburano, B.; Si, W. Evaluation of Web Vulnerability Scanners Based on OWASP Benchmark. In Proceedings of the ICSEng 2018: 26th International Conference on Systems Engineering, Sydney, Australia, 18–20 December 2018; University of Technology: Sydney, Australia, 2018.
31. Team, Z.D. ZAP. 2023. Available online: https://www.zaproxy.org/ (accessed on 22 December 2024).
32. Portugal. Lei n.º 109-2009. *Diário da República. Série I de 2009-09-15*. 2009. Available online: https://diariodarepublica.pt/dr/detalhe/lei/109-2009-489693 (accessed on 22 December 2024).