Mika Riepponen

Selection of open-source web vulnerability scanner as testing tool in continuous software development

Master's Thesis in Information Technology

April 4, 2024

University of Jyväskylä

Faculty of Information Technology

Author: Mika Riepponen

Contact information: mika.riepponen@student.jyu.fi

Supervisor: Tapio Frantti

Title: Selection of open-source web vulnerability scanner as testing tool in continuous software development

Työn nimi: Avoimen lähdekoodin web sovelluksen haavoittuvuusskannerin valinta testaustyökaluksi jatkuvassa ohjelmistokehityksessä

Project: Master's Thesis

Study line: Software Engineering

Page count: 41+7

Abstract: Security is a critical part of web applications and vulnerabilities should be prevented or identified and fixed as early in the development process as possible. The purpose of this study is to determine how well open-source web vulnerability scanners suit for testing commercial web application in continuous software development. The need for this study came from Secapp Oy. Two open-source web vulnerability scanners, ZAP and Wapiti, were chosen to be evaluated. These two scanners were chosen because they were the only open-source web vulnerability scanners found from the latest studies that had command-line interface and were still in active development. Both scanners contributed to improving the security of the target web application. Neither of the scanners was so fast that it could be included in the integration pipeline as a test. Both scanners can be utilized as periodical automated scanner. ZAP offered more customization options for the scan, most importantly possibility to flag scan findings as false positive and skip crawling phase and only scan listed URLs. ZAP was also faster, more precise, found wider set of vulnerabilties and had better crawling coverage. Based on the results ZAP was chosen to scan the target web application in between the releases to test each major version for vulnerabilities.

Keywords: web vulnerability scanner, web scanner, dynamic application security testing, dast, development security operations, devsecops, continuous software development

Suomenkielinen tiivistelmä: Tietoturva on kriittinen osa web sovelluksia ja haavoittuvuudet tulisi ennaltaehkäistä tai tunnistaa sekä korjata mahdollisimman aikaisin ohjelmiston kehitysprosessissa. Tämän tutkimuksen tarkoitus on määrittää kuinka hyvin avoimen lähdekoodin web sovellusten haavoittuvuustestaustyökalut sopivat kaupallisen web sovelluksen testaukseen jatkuvassa ohjelmistokehitysprosessissa. Tarve tälle tutkimukselle tuli Secapp Oy yritykseltä. Arvioitavaksi valittiin kaksi avoimen lähdekoodin web haavoittuvuusskanneria, ZAP ja Wapiti. Nämä kaksi skanneria valittiin sen perusteella, että ne olivat ainoat viimeisimmistä tutkimuksista löytyneet avoimen lähdekoodin web haavoittuvuusskannerit, joissa oli komentorivi käyttöliittymä ja joita edelleen kehitettiin aktiivisesti. Kumpikin skanneri myötävaikutti kohteena olevan web sovelluksen tietoturvan parantamiseen. Kumpikaan skannereista ei tulosten perusteella sovellu integraatioputkessa ajettavaksi testiksi. Kumpaakin voidaan kuitenkin hyödyntää ajoittaisena automaattisena skannerina. ZAP tarjosi enemmän vaihtoehtoja mukauttaa skannausta, tärkeimpänä mahdollisuus luokitella skannauksen löydöksiä vääriksi positiiviksi ja kohdistaa skannaus vain ennalta määritettyyn listaan URL-osoitteita sen sijaan, että skanneri yrittäisi löytää niitä lisää. ZAP oli myös nopeampi, tarkempi löytämään oikeita haavoittuvuuksia, löysi enemmän eri haavoittuvuuksia ja oli parempi löytämään uusia sivuja crawler toiminnoillaan. Tulosten perusteella ZAP valittiin testaamaan kohteena oleva web sovellus pääversioiden julkaisujen välillä haavoittuvuuksien löytämiseksi.

Avainsanat: web haavoittuvuusskanneri, web skanneri, dynaaminen ohjelman tietoturvan testaus, dast, ohjelmistokehityksen tietoturvatoimet, devsecops, jatkuva ohjelmistotuotanto

Glossary

Acunetix	Commercial web vulnerability scanner.
Altoro Mutual	Vulnerable web site published by IBM to demontstrate the effectiveness of IBM products in detecting vulnerabilites in web applications.
AppSpider	Commercial web vulnerability scanner.
Arachni	Partly open-source web vulnerability scanner.
CWE	Common Weakness Enumeration is a category system for hardware and software weaknesses and vulnerabilities.
Burp Suite Professional	Commercial web vulnerability scanner.
CLI	Command-line Interface.
DAST	Dynamic Application Security Testing. Security testing an application while it is running.
DVNA	Damn Vulnerable Node Application. Node.js application that demonstrates OWASP top ten vulnerabilities
DVWA	Damn Vulnerable Web Application. Vulnerable PHP/MySQL web application.
false positive	Positively identified result that was actually negative.
F-measure	Measure of a test's accuracy. Calculated from the precision and recall of the test.
IBM	International Business Machines Corporation is an American multinational technology corporation.
IronWASP	Iron web-application advanced security testing platform. Open-source web vulnerability scanner.
OWASP	The Open Web Application Security Project. A nonprofit foundation that works to improve the security of software.
OWASP Benchmark	Java based test suite to evaluate software vulnerability detection tools.
OWASP Juice Shop	Vulnerable web application written in Node.js, Express and Angular. Demonstrates OWASP top ten and other security

	flaws.
OWASP NodeGoat	Node.js based web application that demonstrates OWASP top ten vulnerabilities.
OWASP WebGoat	Insecure application containing vulnerabilities commonly found from Java-based applications that use common and popular open source components.
ZAP	Zed Attack Proxy. Formerly known as OWASP ZAP. Open-source web vulnerability scanner.
JavaScript	Programming language that is one of the core technologies of the World Wide Web.
LDAP	Lightweight Directory Access Protocol. Protocol for serving directory information over Internet Protocol network.
LFI	Local File Inclusion. A type of attack.
MySQL	Open-source relational database management system.
NetSparker	Commercial web vulnerability scanner.
Node.js	JavaScript runtime environment that executes JavaScript outside browsers.
Precision	True positive results divided by the number of true positive and false positive results.
PHP	Scripting language intended to be used in web development.
Recall	True positive results divided by the all results which are supposed to identify as positive.
RFI	Remote File Inclusion. A type of attack.
Skipfish	Open-source web vulnerability scanner.
SQL	Structured Query Language used for managing data in relational database management system.
SQLi	SQL injection. A type of attack.
true positive	Positively identified result that was correctly identified.
URL	A Uniform Resource Locator. Commonly known as web address.
Vega	Open-source web vulnerability scanner.

WackoPicko	Website that contains common and known vulnerabilities.
Wapiti	Open-source web vulnerability scanner.
Watabo	Semi automated open-source web vulnerability scanner.
WAVSEP	The Web Application Vulnerability Scanner Evaluation Project. A vulnerable web application that can be used for evaluating web application vulnerability scanners.
WIVET	Web Input Vector Extractor Teaser. A benchmarking tool for web application vulnerability scanners.
WVS	Web Vulnerability Scanner.
W3AF	Web application attack and audit framework. Open-source web vulnerability scanner.
XSS	Cross Site Scripting. A type of injection attack.
ZAP	OWASP Zed Attack Proxy. Open-source web vulnerability scanner.

List of Figures

Figure 1. False positive finding in ZAP report. 23
Figure 2. Flagging the finding as false positive in Automation Framework YAML file..... 23

List of Tables

Table 1. Summary of important qualities of reviewed open-source scanners.......... 10
Table 2. Reported vulnerabilities by Wapiti. 20
Table 3. Reported vulnerabilities by ZAP. 21
Table 4. Scan times. 22
Table 5. Crawling coverage. 24
Table 6. An overall comparison based on results.......... 26

Contents

1 INTRODUCTION 1

2 WEB VULNERABILITY SCANNERS 3
- 2.1 Continuous integration and vulnerability testing 3
- 2.2 Web vulnerability scanner 3
- 2.3 OWASP top ten 5
- 2.4 Recent studies of web vulnerability scanners 6
- 2.5 Selecting web vulnerability scanners for this study 10
- 2.6 ZAP and Wapiti 11

3 RESEARCH QUESTIONS 12

4 RESEARCH METHOD 13
- 4.1 Design science method 13
- 4.2 Following the design science method 14

5 RESEARCH PROCESS 16
- 5.1 Setting up the Wapiti vulnerability scanner 16
- 5.2 Setting up the ZAP vulnerability scanner 16
- 5.3 Performing the vulnerability scans 18
- 5.4 Further configuring to handle false positive findings 19

6 RESULTS 20
- 6.1 Scan results 20
- 6.2 Handling false positive findings 22
- 6.3 Crawling coverage 24
- 6.4 Automation and integration features of the scanners 24
- 6.5 Suggestions for way of use for the scanners in continuous software development 25
- 6.6 Evaluation and criticism 26

7 CONCLUSIONS 28

BIBLIOGRAPHY 29

APPENDICES 34
- A Wapiti run commands 34
- B ZAP run command 34
- C ZAP Automation Framewok YAML 35

1 INTRODUCTION

Security is a critical part of web applications, but threats that compromise the security of applications have been steadily evolving (Baldassarre et al. 2020) and web applications have become increasingly vulnerable to malicious attacks (Althunayyan et al. 2022). Verizon reports from 2020 (Widup et al. 2020), 2021 (Widup et al. 2021), and 2022 (Widup et al. 2022) show a continuous rise in data breaches, as the number of confirmed breaches rose from 3950 out of 32002 analyzed incidents in 2020 to 5258 out of 29207 in 2021 and to 5212 out of 23896 in 2022. Web applications were the most frequent attack vectors in the breaches. McAfee report (Malekos Smith, Lostri, and Lewis 2020) estimated that the cost of cybercrime to the global economy was increased over 50% from 2018 to 2020.

Vulnerabilities should be prevented or identified and fixed early in the development process to maintain the security of applications. However, manually searching for vulnerabilities from web application is difficult and time consuming, and therefore there is a need to automate this process with web application vulnerability scanner (Makino and Klyuev 2015). The purpose of this study is to determine how well open-source web vulnerability scanners suit for testing commercial web application in continuous software development. The need for this study came from Secapp Oy. Automated vulnerability scanner as part of the development process would complement other security ensuring actions.

Open source was chosen as the scope for this study. Comparing studies between open-source and commercial WVSs (Web Vulnerability Scanners) show that open-source scanners can also be effective at detecting vulnerabilities (Amankwah et al. 2020; Qasaimeh, Shamlawi, and Khairallah 2018; El Idrissi et al. 2017) or perform similarly underwhelmingly (Anagandula and Zavarsky 2020; Althunayyan et al. 2022).

Recent comparing studies of open-source WVSs, for example from Al Anhar and Suryanto (2021), Althunayyan et al. (2022), Amankwah et al. (2020), Anagandula and Zavarsky (2020) and Zukran and Md Siraj (2021), focus on comparing WVSs against applications which have been built for testing purposes and have intentional vulnerabilities. This is suggested approach so the results can be reproduced and compared (Alazmi and Conte De Leon

2022). However, the studies in systematic literacy review from Alazmi and Conte De Leon (2022) show that the scanners perform differently when evaluated against different applications and even show disparate results in different studies. Thus, when the intention of using the scanner is against a specific application, it is reasonable to evaluate them against that specific application. In this study the scanners are evaluated against the commercial applications they are supposed to test if one of them is selected for that purpose as the result of this study. Another reason which supports new studies comparing the scanners is that some of the open-source scanners are updated often, so comparing studies need to be done again with the newer versions to determine the state they are in.

Studies that compare how different WVSs could be integrated into continuous software development as testing tools or evaluate their usability were not found. Studies researching the usability of the WVSs were encouraged in the latest systematic literacy review (Alazmi and Conte De Leon 2022) as they had not been performed yet. There are several factors which can affect how well the scanner suits for being used as a testing tool in continuous software development, for example are the authentication features able to handle login and maintain necessary session information of the web application, is the scanner able to crawl through the different parts of the application, how fast it performs the scan, how far the usage can be automated, and how precise are the scan findings, since worse precision rate leads to more false positive findings and confirming each of them takes precious working time from the software developers.

2 WEB VULNERABILITY SCANNERS

This chapter covers why vulnerability testing is done and the background theory of web vulnerability scanners needed for this study. Web vulnerability scanner as a concept is introduced. Then the most critical vulnerabilities in web applications, which the scanners are supposed to find, are explained. The end sections focus on comparing web vulnerability scanners that were found from the recent studies and on decision which scanners were chosen for this study.

2.1 Continuous integration and vulnerability testing

Continuous integration in software development is practice where developers continuously merge their code changes into central repository (AWS 2024), which is a storing place for a software project. This merging typically happens through a pipeline which consists of automated jobs which for example build the software and test it (Rehkopf 2024). The testing is important because each change to code presents a chance that something has gone wrong, and the software is no longer working as intended. Even when the software may seem to work as intended according to tests which evaluate the functionality of the software, it can have security flaws. The security flaws allow malicious actor to use software in a way which endangers the software users and their information. Vulnerability testing is a practice which intends to find these security flaws so they can be fixed.

2.2 Web vulnerability scanner

Software testing can be divided into two approaches, black box and white box testing. Black box testing considers software as a black box and observes the behavior of the software according to inputs. On the contrary, white box testing is interested in what happens inside the software during those inputs and traces them through the source code. It is more focused on the design and implementation of the block that is under the testing. (Hamza and Hammad 2019) Web application vulnerability scanners operate with the black box testing approach and automatically examine the web application for security vulnerabilities. They are popular because of the easy use, automation and because they do not depend on the specific technologies used in the web application. (Kagorora et al. 2015)

Web Application Security Consortium (Gaucher et al. 2009) defines web application security scanners as automated tools which are used to test web applications for common security problems that are for example Cross-Site Scripting (XSS), SQL injection, Directory Traversal, insecure configurations, and remote command execution vulnerabilities. This is done by crawling a web application and locating vulnerabilities from application layer either by manipulating HTTP messages or inspecting them for attributes that are suspicious.

Doupé, Cova, and Vigna (2010) described process of the web application scanners with three phases. The scanners first crawl through the web application to collect the reachable pages and input vectors that are associated to them. These are for example HTML forms, parameters of GET requests or features that allow one to upload files. Then they generate specially made input values and submit them to the application. These might be for example JavaScript code that could trigger XSS vulnerability or strings that have function in SQL language and trigger SQL injection vulnerability. Finally, they observe the behavior of the application and determine if the actions triggered vulnerabilities. Hence, it appears that they consist of three main modules that are crawler, attacker, and analysis module (Doupé, Cova, and Vigna 2010). Kagorora et al. (2015) also described the mechanics as a three-step process consisting of crawling, simulation of attacks and response analysis. The crawling phase was described to have different options, such as finding the web pages of the application with automatic web crawler, or with semi-automatic crawler which asks for operator's assistance, or by reading the records from a proxy.

National Institute of Standards and Technology (Black et al. 2008) defines minimum requirements for web application security scanners. They should be able to authenticate to the application and maintain a logged-in state. They should be able to identify OWASP top ten vulnerabilities and generate a text report that indicates, specifies and identifies an attack for each vulnerability that is identified. They should also have an acceptably low false positive rate.

2.3 OWASP top ten

National Institute of Standards and Technology minimum requirements (Black et al. 2008) for web application security scanners require that they are able to identify OWASP top ten vulnerabilities. OWASP top ten (OWASP 2021) is a regularly updated list of web application security risks that are deemed to be most critical as a broad consensus. OWASP recommends that companies make sure that their web applications minimize the risks that are included to them. The latest top ten list is from 2021 and consists of the following risks:

1. Broken access control. This happens when users can act outside of the permissions that are intended for them. For example, the user can modify URL or parameters and access a privileged page or resource.
2. Cryptographic failures, previously known as sensitive data exposure. This means failures related to cryptography that often lead to exposure of sensitive data. For example, sensitive data like passwords or credit card numbers are transmitted in clear text or encryption used is weak.
3. Injection. There are various injection types like SQL, NoSQL, OS Command, Object Relational Mapping (ORM) and Lightweight Directory Access Protocol (LDAP). Injections happen when user given data is not validated, filtered, or sanitized, and is then used in queries or search parameters.
4. Insecure design. Insecure design means design flaws that are different from implementation flaws and cannot be fixed with implementation. For example risks could have been profiled incorrectly and security controls are missing or weak. A practical example is using questions and answers for credential recovery, which is not secure since more than one person can know the answers.
5. Security misconfiguration. This includes various risks that are associated to configuration, for example cloud services have improperly configured permissions, unnecessary ports are open, default accounts are enabled and unchanged, error messages reveal too much information to users, security settings are not set to secure values or security features are disabled.
6. Vulnerable and outdated components. Software or components have not been updated and are outdated or have vulnerabilities. This includes operating system, server,

database management system, applications, APIs, components, runtime environments, and libraries.

7. Identification and authentication failures, previously known as broken authentication. For example, if brute force or other automated attacks are permitted, weak passwords are allowed, multi-factor authentication is missing or ineffective, credential recovery is "knowledge-based" like questions and answers, or if session identifier is exposed in URL, or reused, or not invalidated with logout or after certain time of inactivity.
8. Software and data integrity failures. This means that code and infrastructure do not protect against integrity violations. For example, plugins, libraries, or modules from untrusted sources, repositories, or content delivery networks, are used, or CI/CD pipeline is insecure, or updates are downloaded and applied without verification.
9. Security logging and monitoring failures. Logging and monitoring is insufficient and cannot detect security breaches. For example, when user acts suspiciously the suspicious actions are not logged, or the user cannot be traced from them, or when alerting and response escalation processes are not used or they do not work effectively.
10. Server-side request forgery. This happens when web application fetches remote resource without validating the user given URL. This allows the attacker to send tailored requests that identify as requests coming from the server authenticate as the server. For example the attacker can provide URL which points to local resource in server or metadata storage of could service.

2.4 Recent studies of web vulnerability scanners

Next is a comparison of open-source WVSs (web vulnerability scanners) that were found from the latest studies. The most popular and studied open-source WVSs in the recent studies presented here are ZAP (Zed Attack Proxy), Arachni, Skipfish, W3AF (web application attack and audit framework), Wapiti, Vega and IronWASP (Iron web-application advanced security testing platform). Across studies that were examined the performance of WVSs (Web Vulnerability Scanners) varies depending on the scanned target.

Amankwah et al. (2020) evaluated multiple WVSs (Acunetix, HP WebInspect, IBM AppScan, OWASP ZAP, Skipfish, Arachni, Vega and Iron WASP) against DVWA and Web-

Goat. While the included commercial scanners were considered effective, OWASP ZAP and Skipfish were considered equally efficient at detecting command execution, cross-site scripting, and SQL injection. Skipfish had precision score of 75% for both testing targets, while OWASP ZAP, Arachni and Vega had only 56%. Precision score of 56% means that 44% of the findings were false positive.

Study from Zukran and Md Siraj (2021) compared OWASP ZAP and Skipfish and found that OWASP ZAP outperformed Skipfish with precision rate by almost two times when evaluated against WAVSEP. The authors reported that OWASP ZAP performed better with small difference against DVWA. OWASP ZAP also had better coverage overall. In another study from Althunayyan et al. (2022) OWASP ZAP, Burp Suite Professional, Vega, Skipfish and Wapiti were evaluated against OWASP Juice Shop. OWASP ZAP and Burp Suite Professional found out vulnerabilities while Vega, Skipfish and Wapiti did not identify any correctly. However even ZAP and Burp Suite Professional only found out two out of the seven vulnerabilities. Researchers concluded that this was because of the lacking crawling abilities, and lacking abilities to detect all different types of injection vulnerabilities.

Another study from El Idrissi et al. (2017) compared commercial (Burp Suite, Acunetix, Netsparker, AppSpider, Arachni) and open-source scanners (Wapiti, Skipfish, W3AF, IronWASP, OWASP ZAP, Vega) against the WAVSEP, which benchmarks how the scanners find SQL injections, reflected XSS, RFI (remote file inclusion), LFI (local file inclusion) and path traversal. OWASP ZAP, Vega and IronWASP detected most true positive RFI out of all scanners, and ZAP and Vega detected also LFI better than other scanners excluding AppSpider, which was strongest at detecting LFI. All scanners except W3AF and AppSpider found all SQLi vulnerabilities. Also, all scanners except Wapiti, W3AF and IronWASP found more than 90% of the XSS vulnerabilities. W3AF found only 30% of the XSS, Wapiti 67% and IronWASP 79%.

OWASP ZAP was compared to Arachni with OWASP benchmark in a study from Mburano and Si (2018). They found that ZAP outperforms Arachni when detecting command injection, SQL injection and XSS, but Arachni outperformed ZAP by huge margin when detecting LDAP injection. This study also chose these two scanners out of OWASP ZAP, Arachni, W3AF, Wapiti and Watabo, because they considered them the most popular and well maintained.

Study from Al Anhar and Suryanto (2021) compared Burp Suite Professional, Arachni, OWASP ZAP and Wapiti against node applications DVNA and NodeGoat. Burp Suite Professional and OWASP ZAP found most vulnerabilities from DVNA, still finding only 57% (recall) out of all vulnerabilities in DVNA. Burp Suite Professional was best against NodeGoat with a recall score of 60%, Arachni came second and ZAP third. Wapiti performed worst, detecting zero vulnerabilities from DVNA and identifying vulnerabilities from NodeGoat with only 30% recall.

OWASP ZAP, Wapiti and Burp Suite Professional were also compared in another study from Anagandula and Zavarsky (2020) where commercial scanner Nessus Essential Edition was included too. The scanners were evaluated for stored XSS and stored SQL injection detection against WackoPicko and Scanit, which is custom testbed from Concordia University of Edmonton. In this study the scanners were anonymized, but the researchers concluded that there was not great difference in performance between the commercial and open-source scanners. Researchers recommended that the scanners need to be improved with correct attack vectors for detecting stored XSS and stored SQL injection. They performed better in detecting stored XSS which did not require login and needed single step, but did not handle well multistep stored XSS that required login.

Another study (Sagar et al. 2018) compared ZAP, W3AF and Skipfish against DVWA. In this study W3AF did not find any vulnerabilities and had a huge running time of five hours compared to the running time of ZAP and Skipfish, which were under three minutes. Researchers also noticed that W3AF configuration and usage was complex in comparison to ZAP and Skipfish. In this study ZAP correctly identified six vulnerabilities out of eight and Skipfish identified four. ZAP missed one SQL injection and blind SQL injection, which both Skipfish missed too.

Another study (Alsaleh et al. 2017) compared Skipfish, Wapiti and two different versions of Arachni (1.0.2 and 0.4.3) on multiple factors. Skipfish was fastest when scanning three test sites, taking at most 27 minutes while Arachni took at least 23 minutes and Wapiti took at least hour and half. Crawling coverage was assessed with WIVET scores. Arachni 1.0.2

was best with 94% coverage, Skipfish had 48%, Wapiti 44% and older Arachni 19%. Wapiti also had better crawling coverage with 44% versus older Arachni with 19% when evaluated against 140 websites trending in 2007-2013. Test site Altoro Mutual and WAVSEP were used to assess the accuracy of the four scanners. Arachni 1.0.2 found all 135 SQL vulnerabilities from WAVSEP, Arachni 0.4.3 found 134, Wapiti found 131 and Skipfish 104. Arachni 1.0.2 also found most XSS vulnerabilities from WAVSEP with 64 out of 73, Skipfish found 63, Arachni 0.4.3 found 47 and Wapiti 45. Surprisingly Arachni 0.4.3 found most vulnerabilities when evaluated against Altoro Mutual, Wapiti was second. Every scanner had over 95% accuracy and over 97% f-measure in identifying SQL attacks, except Skipfish, which had 79% accuracy and 87% f-measure. Arachni 1.0.2 had the best accuracy with 89% and f-measure with 93,4% in detecting XSS, Skipfish came very close with 88% accuracy and 92,7% f-measure and Wapiti was weakest with 61% accuracy and 74% f-measure.

In a study from Qasaimeh, Shamlawi, and Khairallah (2018) OWASP ZAP and commercial scanners (Acunetix, Burp Suite, NetSparker, Nessus) were evaluated against seven different applications designed to evaluate vulnerability scanners. OWASP ZAP found nearly the same number of vulnerabilities as the best commercial scanner Acunetix (758 vs 763) while surpassing the other commercial scanners Burp Suite, NetSparker and Nessus. ZAP had lower false positive rate than Burp Suite, but higher than the others. ZAP had 73% accuracy versus Burp Suite 50%. Acunetix and NetSparker had 91% accuracy.

In recent systematic literacy review from Alazmi and Conte De Leon (2022) it was concluded that the studies reviewed reported disparate and inconsistent efficacy from the scanners and most studies evaluated only SQLi and XSS vulnerability types with one or two scanners evaluated against one or two nonstandard web applications. It was also concluded that no published evaluations that assess the quality of use or usability of web vulnerability scanners were found. It was recommended that future studies should benchmark web applications against all OWASP top ten vulnerabilities, standard benchmark application should be created, inclusion or lack of commercial sponsors should be disclosed and the usability of WVSs should be evaluated.

2.5 Selecting web vulnerability scanners for this study

Scanner	CLI	Latest update	In active development
ZAP	yes	2024	yes
Wapiti	yes	2024	yes
Arachni	yes	2023	no
W3AF	yes	2020	no
Skipfish	yes	2012	no
Vega	no	2016	no
IronWasp	no	2013	no

Table 1. Summary of important qualities of reviewed open-source scanners.

Github has the most recent source code from the earlier mentioned open-source scanners (Bennetts 2022a) except IronWASP. IronWASP only has source code from 2013 (Kuppan 2013), but appears to have newer version according to the issues page, which is however no longer available since the website is down (Kuppan). When comparing repository activity of these other open-source scanners, ZAP and Wapiti are the ones with most recent commits within a week, Arachni has most recent commit from May 2023, W3AF from 2020, Vega from 2016 and Skipfish has the oldest most recent commit from 2012 (Bennetts 2022a).

Vega (Ahmad, Leidl, and McKinney) and IronWASP (Kuppan 2013) were excluded from the selection since they only offer graphical user interface. Fluent integration into software development process requires automation, so command-line interface is favored because it can be utilized in scripts. ZAP and Wapiti appear to be the only scanners which are still actively developed, since excluding Arachni the most recently updated other scanners have been updated two years ago, and Arachni readme file (Laskos 2022) states that the application is heading towards obsolescence.

The purpose of this study is to determine how well open-source web vulnerability scanners suit for testing commercial web application in continuous software development. Based on the qualities shown in the table 1, two of the open-source scanners were chosen to be evaluated: ZAP (Bennetts 2022b) and Wapiti (Surribas 2022). These scanners were selected because they are present in the latest studies, have command line interface which supports the automation purpose, they have been updated most recently and seem to be updated actively when compared to the other open-source web vulnerability scanners (Bennetts 2022a). Arachni was also considered, but unfortunately it is heading towards obsolescence (Laskos 2022), and an inquiry about the license for this use case was left unanswered. ZAP and Wapiti will be evaluated against commercial web application from Secapp Oy.

2.6 ZAP and Wapiti

Zed Attack Proxy (ZAP) is a popular automated web application scanner that is free and open source (ZAP 2022). It was maintained by Open Web Application Security Project (OWASP) foundation, that is nonprofit organization and works to improve security of software (OWASP 2022). ZAP was formerly called "OWASP ZAP" when it was maintained by OWASP, but in August 2023 ZAP joined Software Security Project (SSP) and is now called just "ZAP". ZAP was declared as a flagship project for OWASP and is now one of the founding projects in SSP. (ZAP 2024b)

ZAP is at its core a man-in-the-middle proxy that stands between browser and web application to intercept and inspect the messages between them. It modifies the intercepted content if needed and then forwards it to the destination. It can be used both as a standalone application and daemon process. ZAP is advertised to be world's most widely used web application scanner and an ideal tool to be used in automation. (ZAP 2022) To support automation ZAP has framework called Automation Framework which allows ZAP scan to be configured with one YAML file (ZAP 2024a). More of Automation Framework in section 5.2.

Wapiti is a free and open-source web application scanner that is intended to be used to audit the security of websites or web applications. It crawls through the deployed web application and looks for scripts and forms, so it can inject data to them. Once it has found URLs, forms, and inputs, it acts like a fuzzer and injects payloads to them to see if they are vulnerable. (Wapiti 2022) Wapiti has currently 921 stars, 150 forks and 28 contributors in GitHub (Wapiti 2024).

3 RESEARCH QUESTIONS

The research questions are as follows:

1. How does ZAP compare to Wapiti with precision, crawling coverage and speed when tested against commercial web application from Secapp Oy?
2. What options do ZAP and Wapiti offer for automated use?
3. How should ZAP and Wapiti be included as testing tools in continuous software development?

The first question gives an answer to how well each of these tools serve their purpose and find vulnerabilities from the designated web applications. In this study no vulnerabilities are known beforehand, so the performance is analyzed with how well they crawl across the applications, how precise they are about vulnerabilities they report and how fast they are. Precision is evaluated by comparing how much vulnerabilities the scanners find, and how much of the findings are true positives in comparison to false positives. Crawling coverage is evaluated by comparing the crawled paths under the site domain to the total number of paths under the site domain.

The second question gives an answer to what features both scanners offer for automated use. These are features that support the integration into continuous software development process. Command line interface offers usage through scripting, but it is also essential that the scan reports are given in a format that scripts can utilize.

The third question can be answered because of the first two questions. If the scanners are fast and offer enough features supporting automated use, they could be used in for example CI/CD pipeline, early testing process or spontaneous developer tests, otherwise they can be used in for example a process that runs at the night and does not take time away from the software development that happens in daytime. This question gives answer to how each of these web vulnerability scanners, according to their performance and usability, should be included as testing tools in continuous software development.

4 RESEARCH METHOD

The method for this research is design science research. The method was chosen because the study is of pragmatic nature, and the expected outcome will be an artifact in form of a suggested way of use for the chosen web vulnerability scanners as testing tools in continuous software development. This requires analyzing which way of applying the scanners into development process is effective and takes the least amount of time from the developers to use.

4.1 Design science method

Design science is a research strategy that is domain independent and tries to utilize opportunities in a field and point out problems in a field through understanding the actions, processes, and systems in the field (Van Aken, A., and J. 2016). The aim of the research strategy is to develop an artifact (Peffers, Tuunanen, and Niehaves 2018) that is created to address a problem (Hevner et al. 2004, 22), or otherwise to validate generic design in a pragmatic way, and to understand the function of the design (Van Aken, A., and J. 2016). Kuechler and Vaishnavi (2008) emphasize the importance of understanding the design, and propose that in addition to the relevant artifacts, the knowledge that is directly useful for construction of artifacts could be added to redefine the output of information systems design research. Hevner et al. (2004) define artifact as a construct, model or method in addition to an instantation.

Hevner et al. (2004) present the core concepts of the design science research through seven guidelines. The paper consolidates over 20 years of design science research (Kuechler and Vaishnavi 2008). First and foremost, an artifact must be produced as the outcome of the research. As mentioned, the artifact can be either construct, model, method, or an instantiation. The artifact must be a solution to important and relevant business problem. The design must be evaluated, and the utility, quality, and efficacy of the artifact must be well demonstrated.

For evaluation five methods are offered Hevner et al. (2004, 86): observation, analysis, experiment, testing and description. These methods are subcategorized into case study and field study (observation); static analysis, architecture analysis, optimization and dynamic analysis

(analysis); controlled experiment and simulation (experiment); functional testing and structural testing (testing); informed argument and scenarios that demonstrate the artifact's utility (description).

The construction and evaluation of the design artifact must be carried out with rigorous methods. Design science research must also contribute to the area of the design artifact in a clear and verifiable way. The design is also a search process that utilizes available means to reach an desirable end and find effective artifact. Finally, the research must be presented in a way that is effective for both technology-oriented and management oriented-audience. (Hevner et al. 2004)

Peffers et al. (2007) combined the elements from the other design science studies to propose a methodology to become a commonly accepted framework for how to conduct design science research. The framework consists of six different activities: defining a problem and justifying the need for a solution, defining objectives for a solution, designing and creating the artifact that serves the purpose as a solution, demonstrating the use of the artifact to validate that it solves an instance of the problem, evaluating the artifact as a solution and communicating the process effectively for the relevant audiences. The communication should be carried in a way that motivates towards solving the problem and reasons the importance of the artifact as a solution and the rigor of its design. The activities follow closely the guidelines from Hevner et al. (2004).

4.2 Following the design science method

In this study the design artifact answers to a business problem of ensuring the security of the products in the development process in a way that is automated as far as possible and takes the least amount of time from the developers to conduct. This is important to maintain the security of constantly changing application and to free human resources to tasks that cannot be automated.

The artifact in this study is a way of use for the web vulnerability scanners in continuous software development. The artifact explains the current state of the selected scanners to serve that purpose. The artifact is a method, and if plausible, leads to an instantiation for one of the scanners. The success of the artifact depends on the features and efficiency of the scanners and requires adapting to the state of those qualities to fulfill the purpose of using them in the role. If problems arise in this process, they contribute to the knowledge of the usability of these scanners.

A search is carried out for different options for the web vulnerability scanners, their features, and ways of use. The performance and ways of use are evaluated with observation and functional testing in the case of the Secapp Oy product. The research will contribute to the usability research of the web vulnerability scanners, and to the field of development security operations. The results are reported clearly to make sure the report is efficient for the technology-oriented and management-oriented audience. Though the aim is to apply the scanners into a specific use case, the results can be utilized in future research that also study the usability of web vulnerability scanners for similar purposes.

5 RESEARCH PROCESS

The design does not need to be based on a formal theory or formal process, the methodology is open to variety of processes (Peffers, Tuunanen, and Niehaves 2018). This research was done to strengthen the continuous vulnerability testing in Secapp Oy. Secapp is a SaaS platform for critical communication, alerting and documentation, which is built especially to cope with emergencies and ensure safety. It helps to broadcast mass notifications, alert individuals and teams, collect critical data and provides secure chats and videos.

5.1 Setting up the Wapiti vulnerability scanner

Wapiti was straightforward to download with pip and get ready to run in Python virtual environment. The options for the scanner were explored from the manual (Wapiti 2023a) page which also gave a hint of the wapiti-getcookie (Wapiti 2023b) program, which could perform the authentication and save the session cookie into a file, which could be then used with the Wapiti scanner. This method was chosen for authentication as it did not need username and password to be given with the shell command when launching the scanner. Alternatively, the arguments "–form-user", "–form-password" and "–form-url" could have been used for the authentication.

5.2 Setting up the ZAP vulnerability scanner

ZAP was more complicated to get started with than Wapiti. At first the ZAP was explored with the graphic user interface version, because there was much to learn before just jumping into automating the scanner. Firstly I came accross the need to set up a context, which was set of rules applied to the scan. It included set of URLs to be scanned, set of excluded URLs, structure for GET and POST requests, technologies used, authentication method, user credentials, session management, authorization procedure when receiving sending unauthorized request, AJAX spider configuration, custom page list for custom error conditions and alert filters. Out of these I configured the included and excluded URLs, authentication method, session management and user credentials needed to be configured and rest could be left as default at the start.

Form-based authentication was chosen, but it took trying out different settings and combinations to get this to actually work. Debugging why the scanner would not scan authenticated pages properly was tedious because in addition to the scanner authenticating through login page it also tried to attack the login page. ZAP would automatically extract the login page POST data from the data that was gathered from manually exploring the application, but somehow CSRF token handling did not work. After adjusting the parameters and checking Anti CSRF token settings from the settings this was fixed. Some settings had to be disabled from the scanned web application because otherwise the scanner constant login request attack tries would have led to a situation where the scanner requests would just get blocked. Finally, the authentication worked and ZAP would also include authenticated requests into the report.

After learning about setting up the context for the scan and "jobs" involved in the scanning process, including traditional spider which crawled the web application, more sophisticated AJAX spider (ZAP 2023c) called Crawlax (Crawljax 2023), passive scan which automatically scans the HTTP requests and responses (ZAP 2023g), active scanner which attacks the target web application (ZAP 2023b) and report generation options, it was time to move on to exploring how this could all be automated without using the graphical user interface version of ZAP.

ZAP provides Automation Framework (ZAP 2023e) which is a framework that can automate ZAP flexibly by defining context and different jobs to run. There was no example template found for the YAML file, even though ZAP documentation offered templates for individual jobs and environment or context options. Luckily ZAP graphical user interface version included a feature which would create automation plan from selected context and jobs. Through that feature it was easy to create the Automation Framework YAML file and then configure it further. ZAP offers ready docker images which have ZAP preinstalled. The Automation Framework YAML file could be used together with the ready docker image containing ZAP to launch automated scan. An example of the used YAML file template can be found from appendix C.

5.3 Performing the vulnerability scans

The scans were run on laptop with Intel Core i7-10750H processor, NVIDIA Quadro P620 graphics card, 32GB DDR4 RAM, 512 GB SSD hard drive, and Ubuntu 20.04 LTS operating system. The target web application Secapp was at first run locally inside docker container and then after exploring the scanners the actual results were collected from running it on test a server. Several scans were run with both scanners to explore their options and behavior.

Wapiti web application vulnerability scanner was run from a BASH shell with a command-line interface. Wapiti version was 3.1.8. Python virtual environment was used to contain Wapiti which was installed as a Python package through pip, a package installer for Python. Python version used was 3.9.16.

With Wapiti the authentication was done with a cookie that was created before the scan with another command line tool called wapiti-getcookie which came with wapiti. That tool was run with a command presented in appendix A. With that command Wapiti automatically identified username and password fields from the web application login form and asked inputs for those. It then saved the session cookie to a file which could be used to authenticate during the scan.

The Wapiti scan was run with the next command presented in appendix A. The scan was targeted to whole domain with option "–scope domain", the verbosity level was defined to be normal with "-v 1", report format was defined as html with "-f html", report output folder was set with "-o path" and earlier scans were not taken into account with command "–flush-session". Finally, a long list of different URLs were give to the scanner to target the scan at least to those URLs.

ZAP was run from a docker container that was created from ZAP docker image which has Zed Attack Proxy preinstalled. ZAP version was 2.13.0. ZAP scan was controlled by a plan defined in YAML file with specific ZAP exclusive Automation Framework. The docker container was also created with BASH shell command which also passed command line options to start the ZAP scanner inside the container and finally destroy the container after the scan finished. The command to run the scan with OWASP ZAP is presented in appendix B. The example of Automation Framework YAML file that was used is presented in appendix C.

More definitive YAML file cannot be presented as it would expose details about the scanned web application, which is Secapp property.

5.4 Further configuring to handle false positive findings

To dismiss or filter out the false positive results presented by ZAP the Automation Framework offered passive scan rules to turn off some of the rules (ZAP 2023f). Additionally, the Automation Framework also has a feature to change alert filters (ZAP 2023d), which defines how should different findings be interpreted and should alerts be raised of them. The scan report presented findings in a way which would often have some or all the fields URL, parameter, attack and evidence. These alert filters were matched against these fields and then the findings could be recategorized as 'False Positive', 'Info', 'Low', 'Medium' or 'High'. In this case the false positive findings were defined as 'False Positive' to dismiss them. Unfortunately, similar features were not found from Wapiti.

6 RESULTS

The scan results are presented in this section. Scanners are compared by how precise their findings are, how well they crawl the application and how fast they are. Their automation and integration features are also assessed to determine how they could be utilized in continuous software development to strengthen the application security in development phase. All the scan findings were handled and mitigated appropriately by Secapp Oy internal processes.

6.1 Scan results

Category	Count
Content Security Policy Configuration	1
HTTP Secure Headers	2
HttpOnly Flag cookie	1
Internal Server Error	226

Table 2. Reported vulnerabilities by Wapiti.

Wapiti reported vulnerabilities from four different vulnerability categories, as shown in table 2. They were analyzed and proven true or false positive. Findings from Content Security Policy Configuration, HTTP Secure Headers and HttpOnly Flag cookie were proven true positive, but the majority of findings, which were reported as Internal Server Error, were false positive. Wapiti had categorized responses with 403 forbidden status as internal server error. The precision of this scan was approximaterly 0.17 from 4/230. Here it has to be highlighted that without the reported internal server error findings the other findings from the three other categories had precision of 1. The scan took 172 minutes.

ZAP reported vulnerabilities from 15 different vulnerability categories, as shown in table 3. This does not include findings which ZAP reported with informational risk level. Findings were analyzed and proven true or false positive. True positive findings were from Absence of Anti-CSRF Tokens, Content Security Policy (CSP) Header Not Set, Missing

Anti-clickjacking Header, Application Error Disclosure, Cookie No HttpOnly Flag, Cross-Domain JavaScript Source File Inclusion, Information Disclosure - Debug Error Messages, Server Leaks Version Information via "Server" HTTP Response Header Field, Strict-Transport-Security Header Not Set, X-Content-Type-Options Header Missing and Timestamp Disclosure - Unix. In contrast to Wapiti ZAP reported three internal server errors with Application Error Disclosure and Information Disclosure - Debug Error Messages, and they were correct findings. High risk level alerts reported by ZAP were all false positive. The precision of this scan was approximately 0.98 from 408/418. The scan took 18 minutes when AJAX spider crawler job was set to have maximum time of three minutes.

Category	Risk level	Count
SQL Injection	High	4
SQL Injection - SQLite	High	2
SQL Injection - Authentication Bypass	High	3
Absence of Anti-CSRF Tokens	Medium	4
.htaccess Information Leak	Medium	1
Missing Anti-clickjacking Header	Medium	1
Content Security Policy (CSP) Header Not Set	Medium	91
Cookie No HttpOnly Flag	Low	19
Application Error Disclosure	Low	2
Information Disclosure - Debug Error Messages	Low	1
Cross-Domain JavaScript Source File Inclusion	Low	3
Strict-Transport-Security Header Not Set	Low	100
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	100
X-Content-Type-Options Header Missing	Low	42
Timestamp Disclosure - Unix	Low	45

Table 3. Reported vulnerabilities by ZAP.

When comparing scan times ZAP seems significantly faster. Comparison of scan times is shown in table 4.

	ZAP	Wapiti
Scan time (min)	18	172

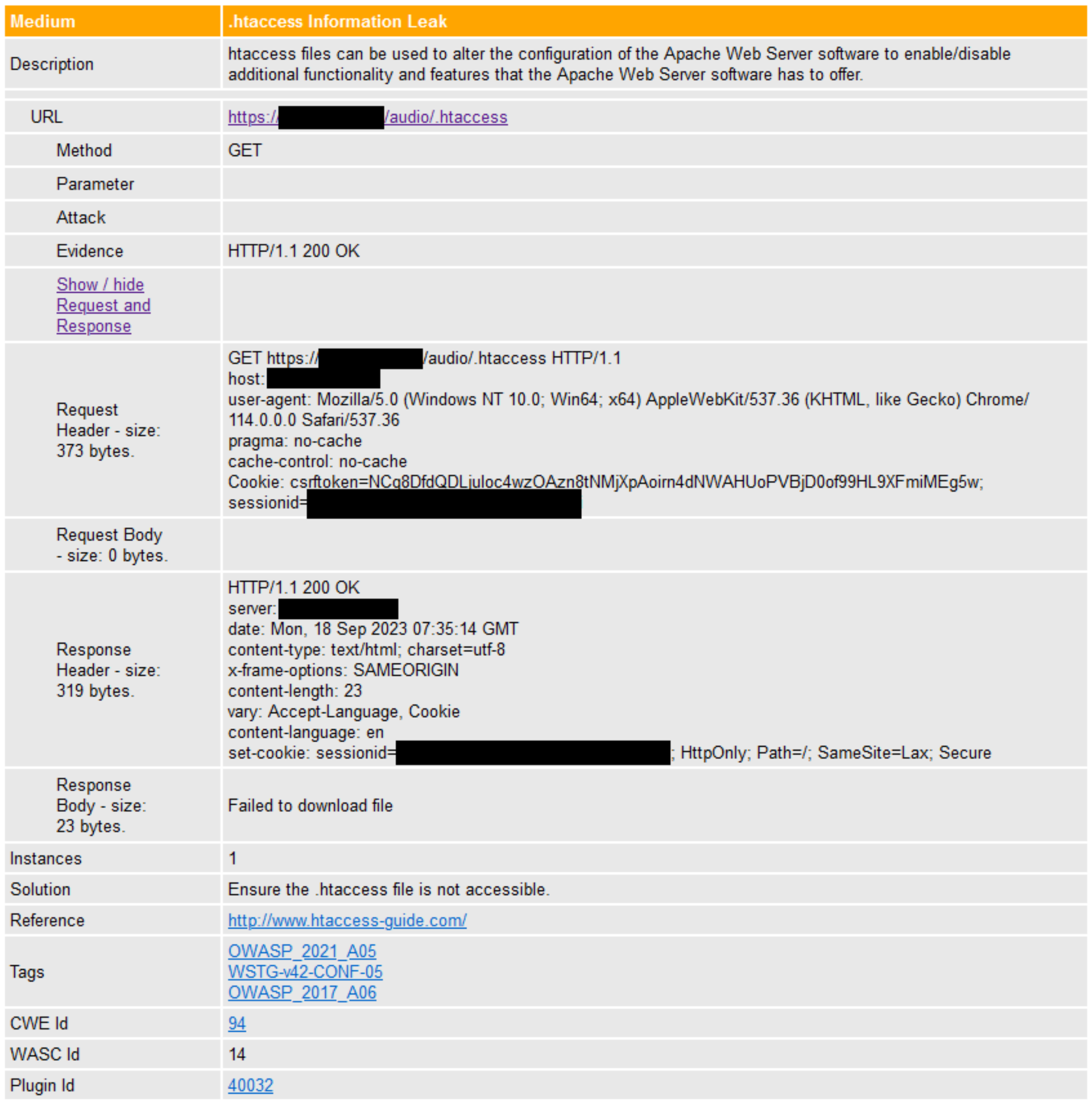
Table 4. Scan times.

6.2 Handling false positive findings

ZAP Automation Framework makes it possible to disable scan rules or flag findings false positive. Similar feature was not found from Wapiti. Several rules and alert filters were added to the Automation Framework YAML file, to dismiss or filter out false positive or only informational findings from ZAP.

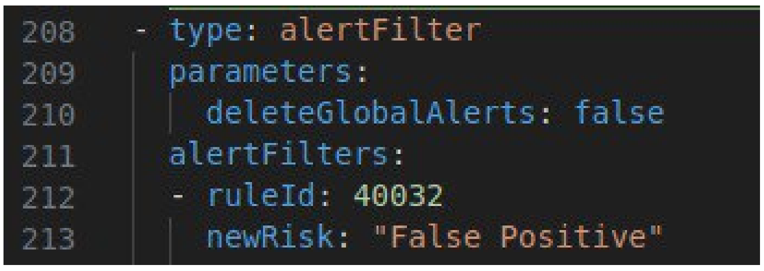
In the figure 1 a false positive ZAP finding is identified by plugin id and then flagged false positive in Automation Framework YAML file presented in figure 2. The example is simple one where whole vulnerability category ".ht access Information Leak" is flagged false positive, but the filtering could be made more definitive by adding additional "url", "parameter" or "attack" identifiers to the filter, which can be regular expression and are used to match the finding in ZAP report. In this example the ZAP thought this was correct finding because the server responded with 200 status and response contained "Failed to download file".

Multiple following scans and iterations were made to adjust these rules and filters. Each iteration reduced the number of false positive findings and did not bring new true positive findings. The scan times ranged from 18 to 21 minutes.



Medium	.htaccess Information Leak
Description	htaccess files can be used to alter the configuration of the Apache Web Server software to enable/disable additional functionality and features that the Apache Web Server software has to offer.
URL	https://[redacted]/audio/.htaccess
Method	GET
Parameter	
Attack	
Evidence	HTTP/1.1 200 OK
Show / hide Request and Response	
Request Header - size: 373 bytes.	GET https://[redacted]/audio/.htaccess HTTP/1.1 host: [redacted] user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36 pragma: no-cache cache-control: no-cache Cookie: csrftoken=NCq8DfdQDLjuloc4wzOAzn8tNMjXpAoirn4dNWAHUoPVBjD0of99HL9XFmiMEg5w; sessionid=[redacted]
Request Body - size: 0 bytes.	
Response Header - size: 319 bytes.	HTTP/1.1 200 OK server: [redacted] date: Mon, 18 Sep 2023 07:35:14 GMT content-type: text/html; charset=utf-8 x-frame-options: SAMEORIGIN content-length: 23 vary: Accept-Language, Cookie content-language: en set-cookie: sessionid=[redacted]; HttpOnly; Path=/; SameSite=Lax; Secure
Response Body - size: 23 bytes.	Failed to download file
Instances	1
Solution	Ensure the .htaccess file is not accessible.
Reference	http://www.htaccess-guide.com/
Tags	OWASP_2021_A05 WSTG-v42-CONF-05 OWASP_2017_A06
CWE Id	94
WASC Id	14
Plugin Id	40032

Figure 1. False positive finding in ZAP report.



```yaml
  - type: alertFilter
    parameters:
      deleteGlobalAlerts: false
    alertFilters:
    - ruleId: 40032
      newRisk: "False Positive"
```

Figure 2. Flagging the finding as false positive in Automation Framework YAML file.

6.3 Crawling coverage

Both scanners were able to scan the given URLs in the previous scans, but when comparing crawling coverage, both scanners were tested without giving them other URLs than base URL, login URL and one of the URLs which requires authentication. The earlier list of URLs given to the scanners was not fully crawlable, so in this test the crawled URLs were compared against a more constricted set of URLs which could be crawled from the given three URLs. Additionally, URLs with fragment part were excluded from the comparable set of URLs as the fragment part is not sent to the web application server and could not be recorded. The crawlers could have found at least 22 URLs with these restrictions.

	ZAP	Wapiti	Test URLs
Count	12	10	22
Coverage	55%	45%	100%

Table 5. Crawling coverage.

Crawling coverage results are presented in table 5. Wapiti had no option to only perform crawl, so another full scan was done with only those three URLs given. Wapiti found 10 of the 22 URLs and the scan took 31 minutes. Crawling coverage is hence approximately 45%. ZAP had option to perform only spider and AJAX spider jobs which are the crawlers of ZAP. When given 10 minutes time limit for AJAX spider the ZAP found 12 of the 22 URLs, which makes the crawling coverate approximately 55%. When given time to run with unlimited time and stopped at 200 minutes, the ZAP found no more URLs but started trying new API requests, which were not in the scope of this test. Both scanners found this specific API endpoint. ZAP found three URLs which Wapiti did not find and Wapiti found one URL which ZAP did not find.

6.4 Automation and integration features of the scanners

Both scanners can be used with command-line interface, which makes it possible to automate their use in various environments. ZAP comes as docker image and Wapiti could also be contained in docker container if needed. To support different scopes for the scans, both scanners can be limited to scan only the given URLs. Both scanners can have a maximum scan time and also use defined strength level for the attacks.

Both scanners have options to customize how crawler is used to find additional URLs within the defined scope. Wapiti has option to limit what depth the crawler can reach from any given URL. Default depth is 40. It also has options to limit the maximum number of links per page or files per directory which are scanned. ZAP has options to limit the depth and time for both crawler jobs, the traditional spider and AJAX spider. The traditional spider has depth of five by default and AJAX spider has ten. ZAP has also numerous other settings for the crawlers, for example the AJAX spider has setting which defines how many browser instances the crawler uses, or should some HTML elements be excluded. ZAP also has possibility to skip crawling phase, which Wapiti does not have.

Both scanners offer various formats for the report generated from the scan. Wapiti offers HTML, JSON, txt and XML formatted reports. ZAP offers HTML, JSON, XML, Markdown and PDF formats. JSON and XML formats can both be fluently processed programmatically and hence support automated and integrated use. Both scanners support JSON and XML formatted reports.

6.5 Suggestions for way of use for the scanners in continuous software development

Even though it depends on the scanned application, it seems that both scanners take too long to be included in continuous integration pipelines. However if defined to only scan restricted set of URLs and take certain time at maximum, they could be utilized in pipeline too. In continuous delivery or deployment pipeline they could serve as a required test, depending on how often the releases happen, although false positive findings serving as a blocker in a pipeline would need to be able to be bypassed manually. With ZAP the findings can be classified false positive so they are automatically categorized false positive in future scans, but with Wapiti all findings would need to be classified each time either true or false positive.

Both scanners can be used as developer tools to test certain parts of the application during development, although they cannot replace manual vulnerability testing as they are able to find only basic vulnerabilities. As the ZAP penetration test page suggests, the application should be manually tested to find more vulnerabilities (ZAP 2023a). Both scanners can also be used to automatically test the software less often than in continuous integration pipeline. This can be done with for example scheduling the scanner to test the target software on a timely basis.

6.6 Evaluation and criticism

In the table 6 there is an overall comparison of most important findings about the two scanners tested in this study. Based on the results of this study the ZAP was chosen to scan the target web application in between the releases to test each major version for vulnerabilities.

	ZAP	Wapiti
Reported true positive vulnerability categories	11	3
Precision	0.98	0.17
Scan time (min)	18	172
False positive findings can be flagged	yes	no
Crawling coverage	55%	45%

Table 6. An overall comparison based on results.

Most time consuming parts of this research were the theory part of gathering the information about current reseach on open-source web vulnerability scanners and secondly configuring and testing the ZAP to get authentication working automatically with it. Otherwise the process of exploring and testing the two selected scanners went well.

In the studies reviewed the web vulnerability scanners were tested with specific web sites meant for testing web vulnerability scanners which make the reliability of those results good. In this test the reliability cannot be validated outside of the Secapp Oy. The tests were still performed meticulously, as it is in the best interest of the company to get reliable results. Same applies to the validity of the results. How the true and false positive findings were identified and then how the following rules to filter out the false positive findings were added based on the identification works for the best interest of the company, even when the validity cannot be reaffirmed from the outside. The scan findings could not be presented in detail due to the sensitivity of the subject since the scanned application is commercial.

7 CONCLUSIONS

Many open-source web vulnerability scanners were found from the recent research. However, it was surprising that only two of those scanners were still updated and developed. New vulnerabilities are often reported, so scanners which are used in practice need to be kept up to date to be able to find those new vulnerabilities.

ZAP and Wapiti scanners were tested and compared. Both scanners were able to find existing vulnerabilities from the target web application and contributed towards fixing them. Both scanners offer features which can be utilized to either scan the whole application or parts of it. ZAP offers more configuration options for how and if the crawler is utilized to map the web application for the scan. The time taken for full scan varied as Wapiti took almost three hours while ZAP took about twenty minutes. Anticipated use cases for both scanners were using them in integration pipeline as test job for each new code push, in developer testing and in periodical automated test. The pipeline use case is not realistic according to results as the scans take so much time. Periodical automated test seems the most useful form for the use according to the results, although developer testing can also be performed.

Based on the results ZAP was more precise with vulnerability findings than Wapiti. ZAP also found more wide set of vulnerabilties, had better crawling coverage and was faster with how long the scan took. ZAP also offered more customization options with Automation Framework and most importantly an option to flag findings false positive. Based on the results ZAP was chosen to scan the target web application in between the releases to test each major version for vulnerabilities.

The results can vary depending on the scanned web application, and the results presented in this study can prove to be different when the scanners are tested against other commercial web applications. More studies are needed to evaluate how the open-source web vulnerability scanners perform against commercial web applications.

Bibliography

Ahmad, D., B. Leidl, and D. McKinney. *Vega vulnerability scanner.* https://subgraph.com/vega/index.en.html. Accessed: 29-08-2022.

Al Anhar, A., and Y. Suryanto. 2021. "Evaluation of Web Application Vulnerability Scanner for Modern Web Application". In *2021 International Conference on Artificial Intelligence and Computer Science Technology (ICAICST),* 200–204. https://doi.org/10.1109/ICAICST53116.2021.9497831.

Alazmi, S., and D. Conte De Leon. 2022. "A Systematic Literature Review on the Characteristics and Effectiveness of Web Application Vulnerability Scanners". *IEEE Access* 10:33200–33219. https://doi.org/10.1109/ACCESS.2022.3161522.

Alsaleh, M., N. Alomar, M. Alshreef, A. Alarifi, and A. Al-Salman. 2017. "Performance-Based Comparative Assessment of Open Source Web Vulnerability Scanners". *Security and Communication Networks* 2017:1–14. https://doi.org/10.1155/2017/6158107.

Althunayyan, M., N. Saxena, S. Li, and P. Gope. 2022. "Evaluation of Black-Box Web Application Security Scanners in Detecting Injection Vulnerabilities". *Electronics (Switzerland)* 11 (13). https://doi.org/10.3390/electronics11132049.

Amankwah, R., J. Chen, P. Kwaku Kudjo, and D. Towey. 2020. "An empirical comparison of commercial and open-source web vulnerability scanners". *Software: Practice and Experience* 50 (9): 1842–1857. https://doi.org/https://doi.org/10.1002/spe.2870.

Anagandula, K., and P. Zavarsky. 2020. "An Analysis of Effectiveness of Black-Box Web Application Scanners in Detection of Stored SQL Injection and Stored XSS Vulnerabilities". In *2020 3rd International Conference on Data Intelligence and Security (ICDIS),* 40–48. https://doi.org/10.1109/ICDIS50059.2020.00012.

AWS, Amazon. 2024. *What is Continuous Integration?* https://aws.amazon.com/devops/continuous-integration/. Accessed: 04-02-2024.

Baldassarre, M.T., V.S. Barletta, D. Caivano, and M. Scalera. 2020. "Integrating security and privacy in software development". *Software Quality Journal* 28 (3): 987–1018.

Bennetts, S. 2022a. *open-source-web-scanners.* https://github.com/psiinon/open-source-web-scanners. Accessed: 03-10-2022.

———. 2022b. *OWASP ZAP.* https://github.com/zaproxy/zaproxy. Accessed: 03-10-2022.

Black, P., E. Fong, V. Okun, and R. Gaucher. 2008. *Software Assurance Tools: Web Application Security Scanner Functional Specification Version 1.0.* https://doi.org/https://doi.org/10.6028/NIST.SP.500-269.

Crawljax. 2023. *Crawljax.* https://github.com/crawljax/crawljax. Accessed: 01-10-2023.

Doupé, A., M. Cova, and G. Vigna. 2010. "Why Johnny Can't Pentest: An Analysis of Black-Box Web Vulnerability Scanners", 6201:111–131. ISBN: 978-3-642-14214-7. https://doi.org/10.1007/978-3-642-14215-4_7.

El Idrissi, S., N. Berbiche, F. Guerouate, and M Sbihi. 2017. "Performance Evaluation of Web Application Security Scanners for Prevention and Protection against Vulnerabilities", 12:11068–11076. 21.

Gaucher, R., R. Auger, Barnett., R., S. Gordeychik, S. Koussa, O. Shezaf, and B. Shura. 2009. *Web Application Security Scanner Evaluation Criteria.* http://projects.webappsec.org/w/page/13246986/WebApplicationSecurityScannerEvaluationCriteria.

Hamza, Z.A., and M. Hammad. 2019. "Web and mobile applications' testing using black and white box approaches". In *2nd Smart Cities Symposium (SCS 2019),* 1–4. https://doi.org/10.1049/cp.2019.0210.

Hevner, Al., S.T. March, J. Park, and S. Ram. 2004. "Design Science in Information Systems Research". *MIS Q.* 28 (1): 75–105. ISSN: 0276-7783.

Kagorora, F., J. Li, D. Hanyurwimfura, and L. Camara. 2015. "Effectiveness of Web Application Security Scanners at Detecting Vulnerabilities behind AJAX/JSON". *International Journal of Innovative Research in Science, Engineering and Technology* 4:4179–4188.

Kuechler, W., and V. Vaishnavi. 2008. "The emergence of design research in information systems in North America". *Journal of Design Research* 7 (1): 1–16.

Kuppan, L. 2013. *IronWASP.* https://github.com/Lavakumar/IronWASP. Accessed: 03-10-2022.

———. *IronWASP website.* https://ironwasp.org/. Accessed: 03-10-2022.

Laskos, A. 2022. *Arachni.* https://github.com/Arachni/arachni. Accessed: 03-10-2022.

Makino, Y., and V. Klyuev. 2015. “Evaluation of Web Vulnerability Scanners”. In *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS),* 399–402. Warsaw, Poland: IEEE Press. ISBN: 978-1-4673-8359-2. https://doi.org/10.1109/IDAACS.2015.7340766.

Malekos Smith, Z., E. Lostri, and J. Lewis. 2020. *The Hidden Costs of Cybercrime.* https://www.mcafee.com/enterprise/en-us/assets/reports/rp-hidden-costs-of-cybercrime.pdf.

Mburano, B., and W. Si. 2018. “Evaluation of Web Vulnerability Scanners Based on OWASP Benchmark”, 1–6. https://doi.org/10.1109/ICSENG.2018.8638176.

OWASP. 2021. *OWASP Top Ten.* https://owasp.org/Top10/. Accessed: 18-10-2022.

———. 2022. *About the OWASP Foundation.* https://owasp.org/about/. Accessed: 18-10-2022.

Peffers, K., T. Tuunanen, and B. Niehaves. 2018. “Design science research genres: introduction to the special issue on exemplars and criteria for applicable design science research”. *European Journal of Information Systems* 27 (2): 129–139. https://doi.org/10.1080/0960085X.2018.1458066.

Peffers, K., T. Tuunanen, M.A. Rothenberger, and S. Chatterjee. 2007. “A Design Science Research Methodology for Information Systems Research”. *Journal of Management Information Systems* 24 (3): 45–77. https://doi.org/10.2753/MIS0742-1222240302.

Qasaimeh, M., A. Shamlawi, and T. Khairallah. 2018. “Black box evaluation of web application scanners: standards mapping approach”. *Journal of Theoretical and Applied Information Technology* 22.

Rehkopf, Max. 2024. *What is Continuous Integration?* https://www.atlassian.com/continuous-delivery/continuous-integration. Accessed: 04-02-2024.

Sagar, D., S. Kukreja, J. Brahma, S. Tyagi, and P. Jain. 2018. “Studying open source vulnerability scanners for vulnerabilities in web applications”. *IIOAB Journal* 9:43–49.

Surribas, N. 2022. *Wapiti.* https://github.com/wapiti-scanner/wapiti. Accessed: 03-10-2022.

Van Aken, J., Chandrasekaran A., and Halman J. 2016. “Conducting and publishing design science research: Inaugural essay of the design science department of the Journal of Operations Management”. *Journal of Operations Management* 47-48:1–8. ISSN: 0272-6963. https://doi.org/https://doi.org/10.1016/j.jom.2016.06.004.

Wapiti. 2022. *The web-application vulnerability scanner.* https://wapiti-scanner.github.io/. Accessed: 18-10-2022.

———. 2023a. *wapiti-getcookie(1) – A Wapiti utility to fetch cookies from a webpage and store them in the Wapiti JSON format.* https://github.com/wapiti-scanner/wapiti/blob/master/doc/wapiti-getcookie.ronn. Accessed: 01-10-2023.

———. 2023b. *wapiti(1) – A web application vulnerability scanner in Python.* https://github.com/wapiti-scanner/wapiti/blob/master/doc/wapiti.ronn. Accessed: 01-10-2023.

———. 2024. *Wapiti - Web Vulnerability Scanner.* https://github.com/wapiti-scanner/wapiti. Accessed: 14-02-2024.

Widup, S., D. Hylender, G. Bassett, P. Langlois, and A. Pinto. 2020. *2020 Verizon Data Breach Investigations Report.* https://doi.org/10.13140/RG.2.2.21300.48008.

Widup, S., A. Pinto, D. Hylender, G. Bassett, and P. Langlois. 2021. *2021 Verizon Data Breach Investigations Report.*

———. 2022. *2022 Verizon Data Breach Investigations Report.* https://doi.org/10.13140/RG.2.2.28833.89447.

ZAP. 2022. *ZAP - Getting Started.* https://www.zaproxy.org/getting-started/. Accessed: 18-10-2022.

———. 2023a. *A Basic Penetration Test.* https://www.zaproxy.org/docs/desktop/start/pentest/. Accessed: 01-10-2023.

ZAP. 2023b. *Active scan.* https://www.zaproxy.org/docs/desktop/start/features/ascan/. Accessed: 01-10-2023.

———. 2023c. *AJAX Spider.* https://www.zaproxy.org/docs/desktop/addons/ajax-spider/. Accessed: 01-10-2023.

———. 2023d. *Alert Filter Automation Framework Support.* https://www.zaproxy.org/docs/desktop/addons/alert-filters/automation/. Accessed: 01-10-2023.

———. 2023e. *Automation Framework.* https://www.zaproxy.org/docs/desktop/addons/automation-framework/. Accessed: 01-10-2023.

———. 2023f. *Automation Framework - passiveScan-config Job.* https://www.zaproxy.org/docs/desktop/addons/automation-framework/job-pscanconf/. Accessed: 01-10-2023.

———. 2023g. *Passive scan.* https://www.zaproxy.org/docs/desktop/start/features/pscan/. Accessed: 01-10-2023.

———. 2024a. *Automation Framework.* https://www.zaproxy.org/docs/automate/automation-framework/. Accessed: 14-02-2024.

———. 2024b. *ZAP is Joining the Software Security Project.* https://www.zaproxy.org/blog/2023-08-01-zap-is-joining-the-software-security-project/. Accessed: 14-02-2024.

Zukran, B., and M. Md Siraj. 2021. “Performance Comparison on SQL Injection and XSS Detection using Open Source Vulnerability Scanners”. In *2021 International Conference on Data Science and Its Applications (ICoDSA),* 61–65. https://doi.org/10.1109/ICoDSA53588.2021.9617484.

Appendices

A Wapiti run commands

Command to extract session cookie:

```
wapiti-getcookie \
-u https://test_server_address \
-c session_cookie.json
```

Command to run Wapiti and record the run time:

```
time wapiti \
-u https://test_server_address \
-c session_cookie.json --scope folder \
-v 1 -f html \
-o /path/to/output/folder --flush-session \
-s "https://test_server_address/subdirectory" \
-s "https://test_server_address/subdirectory_2" \
...
-s "https://test_server_address/subdirectory_n"
```

B ZAP run command

Command to run ZAP from the docker container and record the run time:

```
time docker run --rm --network="host" \
-v /path/to/scan/plan/folder:/zap/wrk/:rw \
-v /path/to/report/folder/:/zap/reports \
-t owasp/zap2docker-stable \
zap.sh -cmd autorun /zap/work/plan.yaml
```

C ZAP Automation Framewok YAML

```
---
env:
  contexts:
  - name: "name of the context"
    # A mandatory list of top level urls
    urls:
    - "included url"
    # An optional list of regexes to include (not used)
    includePaths:
    # An optional list of regexes to exclude
    excludePaths:
    - "excluded url"
    authentication:
      # One of ’manual’, ’http’, ’form’, ’json’ or ’script’
      method: "form"
      parameters:
        loginPageUrl: "login page url"
        loginRequestUrl: "login request url"
        # String, the login request body
        loginRequestBody: "login request body"
      verification:
        # One of ’response’, ’request’, ’both’, ’poll’
        method: "response"
        # Pattern for determining if logged in
        loggedInRegex: "regex pattern"
        # Pattern for determining if logged out
        loggedOutRegex: "regex pattern"
    sessionManagement:
      # One of ’cookie’, ’http’, ’script’
      method: "cookie"
      # List of 0 or more parameters (not used)
      parameters: {}
```

```
        technology:
        # https://www.zaproxy.org/techtags/
        # not used
          exclude: []
        users:
        # user credentials for the context
        - name: "some name"
          credentials:
            username: "username"
            password: "password"
      parameters:
        # If set exit on an error
        failOnError: true
        # If set exit on a warning
        failOnWarning: false
        # If set will write job progress to stdout
        progressToStdout: true
        # Custom variables to be used throughout the config file.
        # Not used
      vars: {}
    jobs:
# https://www.zaproxy.org/docs/desktop/addons/
automation-framework/test-stats/
    - name: "passiveScan-config"
# https://www.zaproxy.org/docs/desktop/addons/
automation-framework/job-pscanconf/
      type: "passiveScan-config"
      parameters:
        # Enable scanning only in scope
        scanOnlyInScope: true
        # Bool: Enable passive scan tags
        enableTags: false
        # Disable all rules before applying the settings
```

```
          # in the rules section
          disableAllRules: false
        rules:
        # Used this to disable alerts that
        # only bring informational value
# https://www.zaproxy.org/docs/alerts/ (Type: Passive)
        - id: 1234
          # The Alert Threshold for this rule,
          # one of Off, Low, Medium, High
          threshold: "Off"
# https://www.zaproxy.org/docs/desktop/addons/
automation-framework/job-spider/
      # The traditional spider
      - type: "spider"
        parameters:
          user: "the user defined earlier"
        tests:
        - name: "At least 100 URLs found"
          # Type of test, only 'stats' is supported for now
          type: "stats"
          # Name of an integer / long statistic,
          # currently supported:  'automation.spider.urls.added'
          statistic: "automation.spider.urls.added"
          # Operator used for testing
          operator: ">="
          # Number of URLs you expect to find
          value: 100
          # 'warn' / 'error' / 'info'
          onFail: "INFO"
# https://www.zaproxy.org/docs/desktop/addons/
ajax-spider/automation/
      - type: "spiderAjax"
        parameters:
```

```
        # Max time in minutes
        maxDuration: 3
        # Max depth that the crawler can reach
        maxCrawlDepth: 10
        # Number of browsers to be used
        numberOfBrowsers: 24
        # Discard out of scope urls
        inScopeOnly: true
        user: "the user defined earlier"
      tests:
      - name: "At least 100 URLs found"
        # Type of test, only ’stats’ is supported for now
        type: "stats"
        # Name of an integer / long statistic,
        # currently supported: ’spiderAjax.urls.added’
        statistic: "spiderAjax.urls.added"
        # Operator used for testing
        operator: ">="
        # Number of URLs you expect to find
        value: 100
        # ’warn’ / ’error’ / ’info’
        onFail: "INFO"
# https://www.zaproxy.org/docs/desktop/addons/
automation-framework/job-pscanwait/
    # Wait for the passive scanner to finish
    - type: "passiveScan-wait"
      parameters: {}
# https://www.zaproxy.org/docs/desktop/addons/
alert-filters/automation/
    # Used to change the risk levels of alerts
    - type: alertFilter
      parameters:
        # Do not delete all existing global alerts
        deleteGlobalAlerts: false
      alertFilters:
        # Alert filters are used to change risk level of alerts.
        # alert ids: https://www.zaproxy.org/docs/alerts/
        - ruleId: 1234
          # New risk level, one of ’False Positive’, ’Info’,
          # ’Low’, ’Medium’, ’High’
          newRisk: "False Positive"
          # Optional string to match against the alert url field
          url: "regex of url"
          # Is url regex
          urlRegex: True
          # Optional string to match against the alert
          # parameter field
          parameter: "parameter name"
          # Is parameter regex
          parameterRegex: False
          # Optional string to match against the alert
          # attack field
          attack: "regex"
          # Is attack regex
          attackRegex: True
          # Optional string to match against the alert
          # evidence field
          evidence: "evidence finding"
          # Is evidence regex
          evidenceRegex: True
# https://www.zaproxy.org/docs/desktop/addons/
automation-framework/job-ascan/
    # The active scanner - this actively attacks the target
    - type: "activeScan"
      parameters:
        # Automatically handle anti CSRF tokens
```

```
          handleAntiCSRFTokens: true
          user: "the user defined earlier"
        policyDefinition:
          # The default Attack Strength for all rules, one of Low,
          # Medium, High, Insane (not recommended)
          defaultStrength: "Medium"
          # The default Alert Threshold for all rules, one of Off,
          # Low, Medium, High, default: Medium
          defaultThreshold:
          rules:
          # Disable alerts for rules that
          # only bring informational value
          # https://www.zaproxy.org/docs/alerts/
          - id: 1234
            # The Alert Threshold for this rule,
            # one of Off, Low, Medium, High
            threshold: "Off"
# https://www.zaproxy.org/docs/desktop/addons/
report-generation/automation/
    # Report generation
    - type: "report"
      parameters:
        # The template id, default : traditional-html
        template: "traditional-html-plus"
        # Where the report will be written
        reportDir: "/zap/reports"
        # The report file name pattern,
        # default: {{yyyy-MM-dd}}-ZAP-Report-[[site]]
        reportFile: "{{yyyyMMdd-hhmmss}}-ZAP-Report-[[site]]"
        # The report title
        reportTitle: "ZAP Scanning Report"
        # The report description
        reportDescription: ""
```