# Why Johnny Still Can't Pentest: A Comparative Analysis of Open-source Black-box Web Vulnerability Scanners

by

**Rana Fouad Khalil**

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the Master degree in
Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

# Abstract

Black-box web application vulnerability scanners are automated tools that are used to crawl a web application to look for vulnerabilities. These tools are often used in one of two ways. In the first approach, scanners are used as Point-and-Shoot tools where a scanner is only given the root URL of an application and asked to scan the site. Whereas, in the second approach, scanners are first configured to maximize the crawling coverage and vulnerability detection accuracy. Although the performance of leading commercial scanners has been thoroughly studied, very little research has been done to evaluate open-source scanners.

This paper presents a feature and performance evaluation of five open-source scanners. We analyze the crawling coverage, vulnerability detection accuracy, scanning speed, reporting and usability features. The scanners are tested against two well known benchmarks: WIVET and WAVSEP. Additionally, the scanners are tested against a realistic web application called WackoPicko. The chosen benchmarks are composed of a wide range of vulnerabilities and crawling challenges. Each scanner is tested in two modes: default and configured. Lastly, the scanners are compared with the state of the art commercial scanner Burp Suite Professional.

Our results show that being able to properly crawl a web application is a critical task in detecting vulnerabilities. Unfortunately, the majority of the scanners evaluated had difficulty crawling through common web technologies such as dynamically generated JavaScript content and Flash applications. We also identified several classes of vulnerabilities that are not being detected by the scanners. Furthermore, our results show that scanners displayed considerable improvement when run in configured mode.

# Acknowledgements

My sincere thanks and praise, that always comes first and foremost, to Allah (God) for the countless blessings He has bestowed upon me.

To my parents, your infinite struggles and sacrifices have not gone unnoticed. I pray that one day I am able to give you back at least a fraction of the unconditional love and support you have given me.

To the Palestinian people, who are living under oppression and those who are forced to live in exile. We have not forgotten the systematic oppression of Palestinian civilians, nor will our children forget. Your previous and ongoing struggle, is a constant reminder of why as Palestinians we need to excel in every field we pursue.

To my supervisor, Dr. Carlisle Adams, for his unwavering support and genuine guidance throughout my studies at the University of Ottawa. I am honored to have had you as a professor, an undergraduate honors project supervisor and specially as a thesis supervisor. Your support through the many academic and personal circumstances I went through is appreciated. It goes without saying, that this thesis would not have been possible without you.

To my family and friends, who are too many to individually name, thank you for your constant support and belief in me.

To all the developers and individuals that generously shared their tools and expertise with the open-source community, thank you. To PortSwigger, thank you for providing me with a free license to Burp Suite Professional scanner to include in my research.

*Dedicated to the free and open software security community.*

# Contents

# List of Acronyms

| | |
|---|---|
| **BWA** | Broken Web Applications |
| **CLI** | Command Line Interface |
| **CSRF** | Cross-site Request Forgery |
| **FP** | False Positive |
| **GUI** | Graphical User Interface |
| **HTML** | Hypertext Markup Language |
| **HTTP** | Hypertext Transfer Protocol |
| **IP** | Internet Protocol |
| **JS** | JavaScript |
| **LFI** | Local File Inclusion |
| **OWASP** | Open Web Application Security Project |
| **PaS** | Point-and-Shoot |
| **Pro** | Professional |
| **RFI** | Remote File Inclusion |
| **RIA** | Rich Internet Application |
| **SQL** | Structured Query Language |
| **SQLi** | SQL Injection |
| **SSL** | Secure Sockets Layer |
| **SWF** | Shockwave Flash |

| | |
|---|---|
| **TLS** | Transport Layer Security |
| **TP** | True Positive |
| **URL** | Uniform Resource Locator |
| **VM** | Virtual Machine |
| **VWAD** | Vulnerable Web Applications Directory Project |
| **WAF** | Web Application Firewall |
| **WASC** | Web Application Security Consortium |
| **WASSEC** | Web Application Security Scanner Evaluation Criteria |
| **WAVS** | Web Application Vulnerability Scanner |
| **WAVSEP** | Web Application Vulnerability Scanner Evaluation Project |
| **WIVET** | Web Input Vector Extractor Teaser |
| **XCS** | Cross Channel Scripting |
| **XML** | Extensible Markup Language |
| **XSS** | Cross-site Scripting |
| **ZAP** | Zed Attack Proxy |

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

The number of internet users has seen a dramatic increase in the past 10 years with an estimated number of 3.9 billion users worldwide [1]. Therefore, it comes as no surprise that the number of websites has also seen a significant increase with an estimated total of over 1.8 billion websites on the world wide web [2]. Websites are being used for various applications, ranging from e-commerce and online banking to social networking and social media. Regardless of the services a website performs, web security has become a major concern for businesses. An insecure website not only can pose a major threat to the business itself, but also to its customers. A recent example is the Equifax breach that exposed the personal information of 143 million US users and an estimated 100,000 Canadian users [3]. The breach was caused by exploiting a web application vulnerability in a two month old bug that Equifax failed to patch. Unfortunately, the occurrence of such breaches is not surprising. According to Trustwave's 2018 Global Security Report [4], 100% of the web applications scanned by Trustwave displayed at least one vulnerability, with a median number of 11 vulnerabilities detected per application.

In order to ensure a web application is secure, a combination of techniques are used. These techniques include but are not limited to, secure coding practices, web application firewalls (WAF), static code analysis and black-box web application vulnerability scanners (WAVS). In this thesis, we focus on the study of open-source web application vulnerability scanners.

A black-box web application vulnerability scanner is an automated tool that is used to crawl an application to look for vulnerabilities. Since the WAVS is a black-box testing tool, it is independent of the technology used to develop the web application and therefore can be used without access to the application's source code. This unique property allows the WAVS to mimic external attacks used by hackers. Traditionally, WAVS were used only by security professionals to conduct penetration tests on web applications. However, with the recent shift towards integrating security in the software development life cycle, WAVS are being used by individuals with diverse professional backgrounds such as quality assurance specialists and software developers. Therefore, generally speaking, scanners are being used in two ways:

1. In the first approach, users use the scanners in a Point-and-Shoot (PaS) manner where the scanner is just given the root URL of the application to test. The user relies on the default configuration of the scanner to crawl and scan the site's functionality with little to no human assistance. This approach is usually used by individuals who (1) lack the expertise and time to learn how to use a scanner and, (2) have the expertise but lack the time to configure and train the scanner.

2. In the second approach, users train and configure the scanner to maximize the crawling coverage and vulnerability detection accuracy. This may involve (1) changing the default crawling and vulnerability detection configuration of the scanner, (2) individually visiting every page of the application while the scanner is in proxy mode so that the scanner is aware of all the pages in the application and, (3) using external plugins to improve the scanner performance.

Unfortunately, previous research shows that despite the numerous advantages offered by these scanners, there are significant limitations that users should be aware of. The most critical limitation is the poor crawling capabilities of the scanners. As web application architecture becomes more complex, it poses a greater challenge for the crawler component of a WAVS that is in charge of parsing and crawling through the entire application. Several studies have been done to measure the performance of WAVS, however, most of the research was conducted on commercial scanners [5] [6] [7] [8] and does not provide much insight on the capabilities and performance evaluation of open-source web application vulnerability scanners.

## 1.2 Thesis Contributions

In this thesis, we evaluate the performance of five open-source web application vulnerability scanners. The study involves:

- *Running the scanners against three well known benchmarks: WIVET, WAVSEP and WackoPicko.* The crawling coverage, vulnerability detection accuracy and scanning speed are measured. The study shows that majority of the scanners evaluated had difficulty crawling through common web technologies such as dynamically generated JavaScript content and Flash applications. Furthermore, our results indicate several classes of vulnerabilities are not being detected by the scanners.

- *Running the scanners in two modes: default and configured.* The study shows that the majority of the scanners showed considerable improvement in crawling coverage and vulnerability detection when run in configured mode. However, this came at the cost of increased running time and added human interference.

- *Evaluating the reporting and usability features of each scanner.* The study shows that all scanners contain the necessary reporting components. Similarly, all scanners

are fairly easy to use and have user manuals / tutorials explaining how to use the scanners. However, several of the scanners lack online support.

- *Comparing the performance of the five chosen open-source scanners to the state of the art commercial scanner Burp Suite Professional (Burp Suite Pro).* The study shows that there is no strong correlation between the cost and the performance of the scanner. Although Burp Suite Pro ranked highest in our evaluation, the open-source scanner ZAP came at a close second with a difference of only three points. The final ranking was done on the performance of the WackoPicko application. However, several of the open-source scanners scored much higher than Burp Suite Pro on the other benchmarks.

## 1.3 Thesis Organization

The remaining chapters are organized as follows.

**Chapter 2:** We explain the main background knowledge required to understand the remainder of this thesis. Section 2.1 reviews the common vulnerabilities found in web applications. Section 2.2 discusses the abstract structure of a WAVS and the different modules that a WAVS consists of. Section 2.3 describes the Web Application Security Consortium's (WASC) criteria for evaluating web application vulnerability scanners.

**Chapter 3** : We present a literature review of previous research that has been performed in the area of evaluating and comparing WAVS. We also compare our work to previous research to demonstrate the added contributions of our research.

**Chapter 4** : We explain the methodology used to conduct the research. This includes tool selection, benchmark selection, environment setup, feature and metric selection and result analysis.

**Chapter 5** : We present the results obtained from running our experiments and we analyze the results for correlations between the different features tested. We also provide a final ranking of the scanners.

**Chapter 6** : We summarize and conclude our work. We also discuss the limitations of our research methodology and provide suggestions for future research on WAVS evaluation methods.

# Chapter 2

# Web Application Security

Before discussing the methodology of our research, we present the concepts required for the reader to properly understand the remaining sections of this work. First, we briefly discuss several of the most common web application vulnerabilities. Second, we describe the abstract architecture of web application vulnerability scanners. Last, we discuss the Web Application Security Consortium's criteria for evaluating web application vulnerability scanners.

## 2.1 Web Application Vulnerabilities

As web application technology becomes more sophisticated, new ranges of security vulnerabilities are introduced and different ways of exploiting existing vulnerabilities are found. The Open Web Application Security Project (OWASP) [9], maintains an updated list of the top 10 most critical web application vulnerabilities found in the wild [10]. Here, we briefly describe several of the most common vulnerabilities presented in the OWASP Top 10 - 2017 report [11].

- **SQL Injection (SQLi)**: This vulnerability allows an attacker to submit maliciously crafted SQL queries to interact with the application's back end database. This could permit an attacker to view, tamper with and destroy existing data.

- **Command Injection**: This vulnerability allows an attacker to run arbitrary system commands on the host operating system of the application. This may allow an attacker to compromise the server of the hosting application and gain complete control over the application.

- **Broken Authentication**: This vulnerability occurs when there are flaws in the authentication and session management functions of the web application. This vulnerability may allow an attacker to compromise user passwords and session tokens to temporarily or permanently gain access to restricted areas of the application.

- **Cross-site Scripting (XSS)**: This vulnerability allows an attacker to inject malicious client side scripts on web pages accessible by other users of the application. This may permit an attacker to execute code on the user's client side on behalf of the application.

- **Sensitive Information Leakage**: This vulnerability usually occurs when the application unintentionally leaks sensitive information about its configuration and internal workings. This may allow an attacker to learn about the application and craft a more suitable attack.

As stated above, the OWASP Top 10 Project only includes the top 10 most critical vulnerabilities. Below, we describe several other vulnerabilities that are found in the wild but are not included in the OWASP Top 10 list. However, depending on the context in which these vulnerabilities are exploited, they can be categorized as part of the OWASP Top 10. For example, a file inclusion vulnerability that allows direct access to objects based on user supplied input, falls under Broken Access Control in the OWASP Top 10 list.

- **File Inclusion**: This vulnerability allows an attacker to view / upload a file, either locally present on the server (LFI) or remotely supplied by the attacker (RFI), in the targeted application. This could result in sensitive data exposure and code execution on the application.

- **Path Traversal / Directory Traversal**: This vulnerability allows an attacker to access restricted resources outside of the web server's root directory. This may allow an attacker to gain access to sensitive information.

- **Unvalidated Redirect**: This vulnerability may permit an attacker to redirect a user from a legitimate web application to a malicious site. This may allow an attacker to successfully carry out phishing scams.

- **Logic Flaw**: This vulnerability results from a flaw in the business logic used by the application to make decisions. A simple example of this vulnerability is an online shopping website that allows a user to enter a coupon number in order to receive a discount, however does not limit the number of times the user can apply the coupon number on the same product. Due to the nature of this type of vulnerability, it is difficult for a WAVS to discover, and therefore usually requires professional human assistance.

The chosen set of scanners were tested against a diverse list of vulnerabilities, including the ones mentioned above. These vulnerabilities cover six of the OWASP Top 10 security risks, namely, Injection, Broken Authentication, Sensitive Data Exposure, Broken Access Control, Security Misconfiguration and Cross-Site Scripting. The security risks XML External Entities (XXE), Insecure Deserialization, Insufficient Logging and Monitoring and Using Components with Known Vulnerabilities were not included in our study due to the lack of intentionally vulnerable applications that contain these vulnerabilities and can withstand aggressive automated scans.

Refer to Section 4.2 for an exhaustive list of the used test cases.

## 2.2 Web Application Vulnerability Scanners

At a high level overview, a WAVS consists of three main components: a crawler module, an attacker module and an analysis module. Below, we briefly describe each module.

- **Crawler**: The crawler module maps the application by taking in a set of URLs that are treated as the roots of the application. The crawler starts at the root URLs and follows any reachable links and redirects in the application. This involves indexing all the discovered pages and the associated input vectors such as HTML forms, GET and POST parameters and cookies.

- **Attacker**: The attacker module analyzes the discovered URLs and input vectors in the crawling phase. The attacker module then generates attack patterns for each entry point to test for any potential vulnerabilities. The crafted attack patterns are either predefined values that are already known to exploit a specific vulnerability such as a predefined XSS Vectors Cheat Sheet [12], or input values that are generated using heuristic algorithms. This module is also referred to as the fuzzing module.

- **Analysis**: The analysis module analyzes the pages returned in response to the attack patterns from the attacker module to detect the presence of a possible vulnerability.

## 2.3 Web Application Security Scanner Evaluation Criteria

The Web Application Security Scanner Evaluation Criteria (WASSEC) project provides a set of detailed evaluation criteria and a framework for formally evaluating web application vulnerability scanners [13]. The project has two main goals:

1. Provide scanner users with the necessary information to conduct a detailed evaluation and make an educated decision on which scanner to choose.

2. Provide scanner developers with the list of features to compare their scanners against and plan for any necessary future enhancements.

The WASSEC project outlines eight categories of features that an ideal scanner should satisfy. Here, we will briefly describe the categories (for further details, the reader should refer to [13]).

- **Protocol Support**: A scanner should support all communication protocols commonly used by web applications. This includes but is not limited to, HTTP, SSL/TLS and SOCKS.
- **Authentication**: A scanner should support all the commonly used authentication methods to authenticate a user in a web application. This includes but is not limited to, basic, HTML form-based, Single Sign On and Client SSL Certificates authentication techniques.
- **Session Management**: A scanner should maintain a valid session with the application during a security scan.
- **Crawling**: A scanner should have a crawler component that is able to thoroughly (depending on user defined configuration) and efficiently crawl a web application.
- **Parsing**: A scanner should be able to parse through the most commonly used web technologies in order to extract information about the web application's functionality and structure. This includes but is not limited to, HTML (up to HTML5), JavaScript, XML and Flash technologies.
- **Testing**: A scanner should detect the vulnerabilities and architectural weaknesses present in a web application. Moreover, a scanner should provide configuration options to the user in order to customize a scan.

- **Command and Control**: A scanner should have command and control features that will improve user experience. Examples include being able to schedule scans, pause and resume scans and schedule multiple scans simultaneously.

- **Reporting**: A scanner should be able to generate a customized report after each scan.

Although an ideal scanner would be able to satisfy all the requirements specified in the WASSEC guidelines, a user generally does not require all the features in order to run an effective scan. Therefore, as mentioned in the WASSEC guidelines, before conducting an evaluation, the user should determine which of the listed features are most important to the context in which the scanner will be used.

In Section 4.4, we present the subset of features that the chosen scanners will be evaluated against.

# Chapter 3

# Literature Review

In both academia and industry, there has been a growing interest in evaluating web application vulnerability scanners. In 2007, Suto wrote a case study that evaluated three scanners to measure (1) the effectiveness of the crawler component, (2) the code coverage, and (3) the number of vulnerabilities detected and the number of vulnerabilities missed [5]. The study showed that scanners failed to thoroughly crawl the tested web applications and missed a significant number of vulnerabilities. Suto's work was later criticized by security professionals due to several reasons with the main reason being that scanners are not usually used as Point-and-Shoot (PaS) tools where the default configuration is not altered. Instead, the architecture and web technologies used in the application is first studied and then the scanner is properly configured to maximize the crawling coverage and vulnerability detection accuracy. This compelled Suto to write a follow-up paper, published in 2010, focusing on the accuracy and time cost metrics for measuring the effectiveness of web application vulnerability scanners [6]. In his follow-up paper, Suto tested seven scanners with each scanner run in two modes of configuration: (1) A PaS mode where the scanner is only given the root URL of the web application and the default configuration is not altered, and (2) a Trained mode where the scanner is made aware of all the pages it is supposed to test. The follow-up study focused on measuring the effectiveness of the following four

areas: (1) The number of vulnerabilities found in PaS mode, (2) the number of vulnerabilities found in Trained mode, (3) the accuracy of the reported vulnerabilities, and (4) the extra human time required to configure a scanner. Despite the criticisms, Suto's follow-up study results were generally consistent with the results found in the 2007 study. Most scanners only showed moderate improvements when run in Trained mode with the exception of only one scanner where the detection accuracy was significantly higher. Moreover, the study observed that the scanners that were more likely to miss vulnerabilities, were more likely to report a high false positive number. Suto also noted that the human time required to configure a scanner considerably varied across the different scanners.

In an attempt to unify and standardize the applications that are used to test scanners, several applications were developed during the past ten years. Two of the most widely used benchmarks are the Web Input Vector Extractor Teaser (WIVET) [14] and the Web Application Vulnerability Scanner Evaluation Project (WAVSEP) [15]. The WIVET benchmark was first created in 2009 by Tatli et al. to measure how well a crawler is able to discover and follow links. It is an open-source stand-alone web-based application that contains 56 test cases that utilize both Web 1.0 and Web 2.0 technologies. The application has since been updated with the latest version released in 2014. Similarly, the WAVSEP application was first created in 2010 by Chen, to help evaluate the quality and accuracy of scanners. It is an open-source intentionally vulnerable web application that consists of a total of 1220 true positive (TP) test cases and 40 false positive (FP) test cases covering several vulnerability categories. The WAVSEP application has also been updated with the latest version released in 2014. These benchmarks were used in several of the subsequent studies performed [16] [17] [18] [7] [19] and have also been included as part of our research.

In [8], Bau et al. published a paper on the evaluation of eight commercial black-box web application vulnerability scanners. The study aimed to answer three questions: (1) what type of vulnerabilities can be tested by the scanners, (2) how representative are these vulnerabilities to ones found in the wild, and (3) how effective are the scanners. The scanners

evaluated included Acunetix WVS, HailStorm Pro, WebInspect, IBM AppScan, McAfee SECURE, N-Stacker, QualysGuard PCI and Rapid7 NeXpose. The authors state that at the time of the research, they were unable to find competitive open-source scanners and therefore, the study was limited to evaluating commercial scanners.

The study was divided into two phases. In the first phase, the scanners were evaluated against previous versions of real world vulnerable applications. This included previous versions of Drupal, phpBB, and WordPress that were released in 2006. The chosen versions of the applications had well documented XSS, SQLi, cross channel scripting (XCS) - that allowed a user to inject code onto a web server across a non-web channel, session, cross-site request forgery (CSRF) and information leakage vulnerabilities. Bau et al. reported that the scanners did exceptionally well in detecting information disclosure and session management vulnerabilities. The scanners also did moderately well in detecting XSS and SQLi vulnerabilities with a 50% detection rate for both categories. However, CSRF and XCS vulnerabilities had a very low detection rate. In the second phase, the authors constructed a custom intentionally vulnerable web application to further measure the vulnerability accuracy and crawling ability of the evaluated scanners. The authors conclude that most scanners are not effective in crawling web technologies such as Java applets, SilverLight and Flash. They also deduced that vulnerability detection rates on average were below 50%.

In [7], Doupé et al. evaluated 11 web application vulnerability scanners; eight commercial and three open-source. The authors used the WIVET application and developed a custom web application, called WackoPicko, that is an open-source intentionally vulnerable realistic web application. In the WackoPicko application, a user is able to comment on photos, upload photos, register as a user, log in into an account and buy the rights to a high-quality version of a photo. WackoPicko consists of sixteen common and known vulnerabilities and contains several crawling challenges. In their work, Doupé et al. mentioned that scanners are being advertised as "point-and-click" pentesting tools that require little to no human

intervention. Therefore, the scanners were run in three different configuration modes; INITIAL, CONFIG and MANUAL. In the INITIAL mode, the scanner is directed to the root URL of the application. In CONFIG mode, the scanner is given a valid username and password combination. In MANUAL mode, the scanner was put in "proxy" mode while the authors browsed every page of the application. INITIAL and CONFIG mode represent the PaS mode that was used by Suto in [6], with the exception that in this application, some of the vulnerabilities are only accessible to authenticated users and therefore a scanner has to either exploit the vulnerable authentication mechanism or be provided with valid credentials in order to detect these vulnerabilities. Likewise, the MANUAL mode is equivalent to the Trained mode that was used by Suto.

The results found in their study were consistent with the studies performed by Bau et al. and Suto. The authors found that crawling modern web applications poses a serious challenge for scanners. Several of the vulnerabilities were only found in MANUAL mode. The web technologies used in the application were partly responsible for the low crawling coverage. Doupé et al. found that scanners are also incapable of handling complex architectures of web applications such as being able to upload a file, handling authentication forms and correctly dealing with muli-step processes. However, unlike the authors of [8], Doupé reported that the commercial scanners did not significantly score better than the open-source scanners.

Several other comparisons have been done in the past few years testing open-source and commercial scanners against several different applications [18] [16]. These studies provide valuable information on the type of vulnerabilities that scanners are able to detect and the crawling coverage of the scanners. However, in all the above mentioned studies, the scanners were run in default mode, with the exception of the work performed by Suto and Doupé et al. in 2010. Moreover, the attack component of several of the scanners is configurable, however, the previous studies make no mention of configuring the attack strength. Lastly, previous research largely focuses only on the crawling coverage and vulnerability

detection accuracy of the scanners and does not consider other features such as reporting and usability.

In our research, we duplicate and update the work done by Doupé et al. in [7] in addition to tackling the above mentioned points. We use the most current available versions of the scanners and the latest versions available for the web application benchmarks. Also, in addition to using the benchmarks WIVET and WackoPicko, we include a third well known benchmark: the WAVSEP application. Similar to the work done by Doupé et al., scanners are run in two modes, default and configured. However, in comparison to Doupé's work, our configured mode involves additionally increasing the attack strength configuration in the scanners. Our work also only focuses on open-source scanners and compares these open-source scanners with the Burp Suite commercial scanner.

Since the majority of the previous research is dedicated to evaluating commercial scanners, we aim to provide a detailed evaluation of the performance of open-source scanners to benefit the free and open software community. We have two objectives for this study. The first objective is to learn the crawling and vulnerability detection challenges that scanners have and analyze the reasoning behind these challenges. The results of this study can later be examined by scanner developers to improve the quality of their scanners. The second objective is to provide a comprehensive performance and feature comparison study for individuals interested in adopting a scanner as part of their software development life cycle.

# Chapter 4

# Methodology

This chapter explains in detail the research methodology we used to select and evaluate the chosen web application vulnerability scanners. Figure 4.1 lists the steps used in the methodology process.

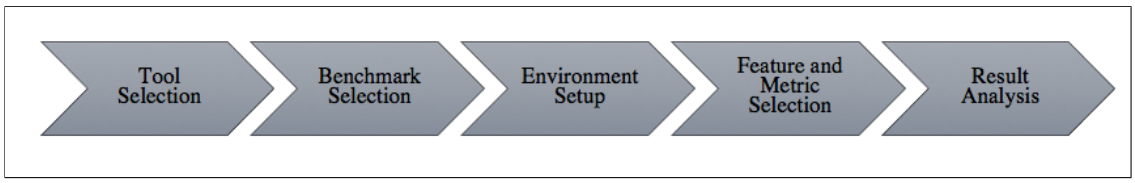



Figure 4.1: Methodology Process

The remaining sections of this chapter will discuss each step in detail.

## 4.1 Tool Selection

The first step of our research was to conduct an initial review of all the open-source web application vulnerability scanners. The review was conducted by examining the feature comparison of open-source scanners presented by Chen in 2016 [20]. In his work, Chen evaluated both commercial and open-source scanners based on price and feature comparison. The comparison evaluated the authentication, control, connection and audit features of the scanners. Most importantly, he also evaluated the crawling coverage and vulnerability

detection accuracy using the WIVET and WAVSEP applications. In our initial review, we used these two metrics to choose the best performing open-source scanners.

Based on the performance results, we selected the scanners Arachni [21], IronWASP [22], ZAP [23], Skipfish [24], Wapiti [25] and Vega [26]. The list was later narrowed down to Arachni, ZAP, Skipfish, Wapiti and Vega after consulting professional ethical hackers on the open-source scanners used in the industry. Additionally, we were recommended to compare our chosen open-source scanners with the most commonly used commercial scanners Burp Suite Professional (Burp Suite Pro) [27] and IBM AppScan [28] (A. Aleali, personal communication, Dec. 2, 2017) (T. Janca, personal communication, January 14, 2018).

As a result, we asked PortSwigger and IBM for free licenses to use their scanners in our research study. However, we were only able to acquire a free license to Burp Suite Pro.

Table 4.1 lists the characteristics of the scanners evaluated. Note that the most recent version of each scanner was used.

| **Name** | **Version** | **License** | **Type** | **Price** | **Last Update** |
|---|---|---|---|---|---|
| Arachni | 1.5.1-0.5.12 | Arachni Public Source v1.0 | Proxy | N/A | 2017-03-29 |
| Burp Pro | 1.7.35 | Commercial | Proxy | $349/year | 2018-06-29 |
| Skipfish | 2.10b | Apache v2.0 | Standalone | N/A | 2012-12-04 |
| Vega | 1.0 | MIT | Proxy | N/A | 2016-06-29 |
| Wapiti | 3.0.1 | GNU GPL v2 | Standalone | N/A | 2018-05-11 |
| ZAP | 2.7.0 | Apache v2.0 | Proxy | N/A | 2017-11-28 |

Table 4.1: Characteristics of the Scanners Evaluated

As can be seen in Table 4.1, the chosen scanners are divided into two types: proxy and standalone. Scanners of type proxy have both a scanner component and a proxy component. The proxy component is usually used to perform manual security tests along side automated scanner runs.

## 4.2 Benchmark Selection

The second step of our research was to choose web applications that will allow us to test the vulnerability detection accuracy and crawling coverage of the chosen scanners.

In academia and industry, there are two well known benchmarks that are used to evaluate scanners: the WAVSEP [15] and WIVET [14] applications. The WAVSEP application is used to evaluate the vulnerability detection accuracy and the WIVET application is used to evaluate the crawling coverage. We include WIVET v4 and WAVSEP v1.5 as part of our comparative study.

The WAVSEP application is representative of vulnerabilities found in the wild but does not model the complex architecture and design of web applications. On the other hand, the WIVET application is representative of the complex architecture and design of current web applications but does not contain any vulnerability test cases. Therefore, we have decided to include a third benchmark to our comparative study that is representative of a real world vulnerable web application in terms of functionality, technology used and common vulnerabilities found in the wild.

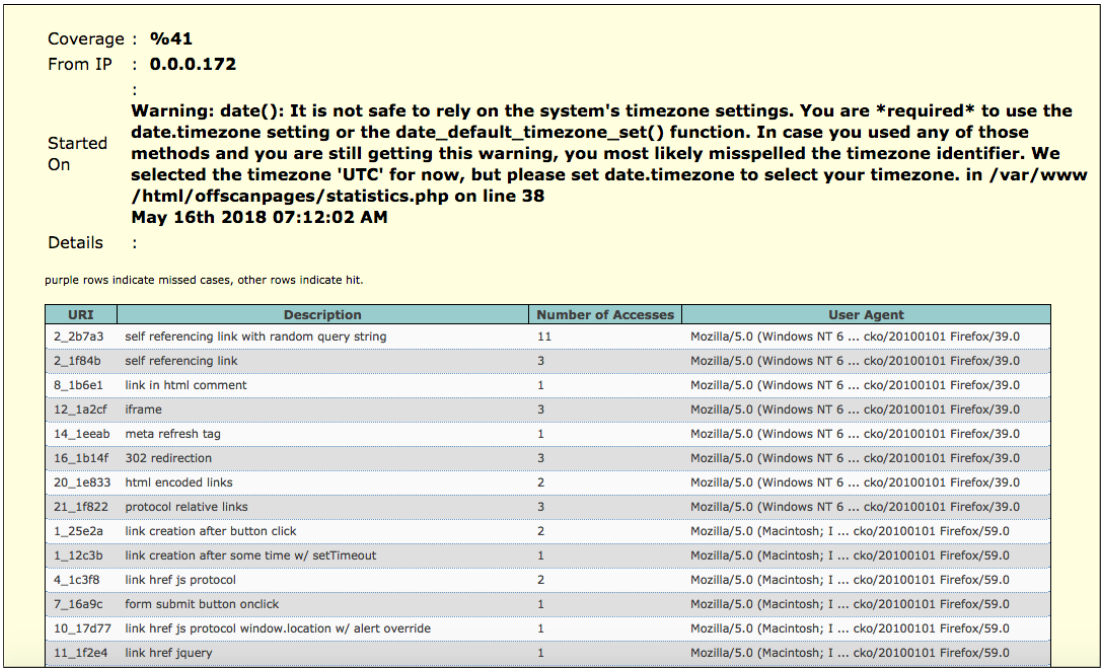
The OWASP Vulnerable Web Applications Directory (VWAD) project is a comprehensive registry of all the known intentionally vulnerable web applications created for security learning and testing purposes [29]. We evaluated the available realistic web applications and decided to include WackoPicko as our third benchmark. The decision was made based on three factors: (1) the type of vulnerabilities included in the application, (2) the realistic architecture of the application and the web technologies used, and (3) the ability of the application to withstand aggressive automated scans.

In the remaining subsections, we describe each chosen benchmark in detail.

### 4.2.1 WIVET

The Web Input Vector Extractor Teaser (WIVET) is an open-source stand-alone web-based application that measures how well a crawler is able to discover and follow links [14]. WIVET contains 56 test cases that utilize both Web 1.0 and Web 2.0 technologies. The test cases include having to analyze standard anchor links, links created dynamically using JavaScript, multi-page forms, links in comments, links embedded in Flash objects and target links within AJAX requests.

Since the WIVET application was design to measure the crawling coverage, it has a built-in feature that keeps track of the test cases that have been successfully visited by the scanner. This can be seen in Figure 4.2.



Coverage : **%41**
From IP : **0.0.0.172**
Started On : **Warning: date(): It is not safe to rely on the system's timezone settings. You are *required* to use the date.timezone setting or the date_default_timezone_set() function. In case you used any of those methods and you are still getting this warning, you most likely misspelled the timezone identifier. We selected the timezone 'UTC' for now, but please set date.timezone to select your timezone. in /var/www/html/offscanpages/statistics.php on line 38**
**May 16th 2018 07:12:02 AM**
Details :

purple rows indicate missed cases, other rows indicate hit.

| URI | Description | Number of Accesses | User Agent |
|---|---|---|---|
| 2_2b7a3 | self referencing link with random query string | 11 | Mozilla/5.0 (Windows NT 6 ... cko/20100101 Firefox/39.0 |
| 2_1f84b | self referencing link | 3 | Mozilla/5.0 (Windows NT 6 ... cko/20100101 Firefox/39.0 |
| 8_1b6e1 | link in html comment | 1 | Mozilla/5.0 (Windows NT 6 ... cko/20100101 Firefox/39.0 |
| 12_1a2cf | iframe | 3 | Mozilla/5.0 (Windows NT 6 ... cko/20100101 Firefox/39.0 |
| 14_1eeab | meta refresh tag | 1 | Mozilla/5.0 (Windows NT 6 ... cko/20100101 Firefox/39.0 |
| 16_1b14f | 302 redirection | 3 | Mozilla/5.0 (Windows NT 6 ... cko/20100101 Firefox/39.0 |
| 20_1e833 | html encoded links | 2 | Mozilla/5.0 (Windows NT 6 ... cko/20100101 Firefox/39.0 |
| 21_1f822 | protocol relative links | 3 | Mozilla/5.0 (Windows NT 6 ... cko/20100101 Firefox/39.0 |
| 1_25e2a | link creation after button click | 2 | Mozilla/5.0 (Macintosh; I ... cko/20100101 Firefox/59.0 |
| 1_12c3b | link creation after some time w/ setTimeout | 1 | Mozilla/5.0 (Macintosh; I ... cko/20100101 Firefox/59.0 |
| 4_1c3f8 | link href js protocol | 2 | Mozilla/5.0 (Macintosh; I ... cko/20100101 Firefox/59.0 |
| 7_16a9c | form submit button onclick | 1 | Mozilla/5.0 (Macintosh; I ... cko/20100101 Firefox/59.0 |
| 10_17d77 | link href js protocol window.location w/ alert override | 1 | Mozilla/5.0 (Macintosh; I ... cko/20100101 Firefox/59.0 |
| 11_1f2e4 | link href jquery | 1 | Mozilla/5.0 (Macintosh; I ... cko/20100101 Firefox/59.0 |

Figure 4.2: Sample Current Run WIVET Results

In order for the WIVET application to calculate the crawling coverage of a scanner, the scanner must maintain a single valid session during the scan. Therefore, to avoid the scan-

ner accidentally terminating and starting a new session, the user should configure the scanner to avoid clicking on the Logout link (100.php / logout.php).

### 4.2.2 WAVSEP

The Web Application Vulnerability Scanner Evaluation Project (WAVSEP) is an open-source intentionally vulnerable web application designed to help evaluate the quality and accuracy of scanners [15]. WAVSEP consists of a total of 1220 true positive (TP) test cases and 40 false positive (FP) test cases. A TP test case is a test case that contains a vulnerability. Whereas, FP test cases are created by Chen in order to determine the vulnerability detection behaviour of the scanners without having to view the source code. For example, one of the FP SQL injection test cases is a login page that responds with erroneous 500 HTTP responses to any input validation failures. Therefore, scanners that base their injection identification exclusively on exceptions (such as an HTTP 500 response) that derive from SQL characters might falsely report this as a vulnerability.

The vulnerability categories in the WAVSEP application include SQL injection, reflected XSS, LFI, RFI, unvalidated redirect, DOM XSS and passive test cases such as information disclosure. Table 4.2 lists the number of TP and FP test cases for each vulnerability category.

| **Vulnerability Category** | **Number of TP Test Cases** | **Number of FP Test Cases** |
|---|---|---|
| SQL Injection | 138 | 10 |
| Reflected XSS | 89 | 7 |
| Path Traversal / LFI | 816 | 8 |
| RFI | 108 | 6 |
| Unvalidated Redirect | 60 | 9 |
| DOM XSS | 4 | 0 |
| Passive | 5 | 0 |

Table 4.2: WAVSEP Vulnerability Categories and Test Cases

### 4.2.3 WackoPicko

WackoPicko is an open-source intentionally vulnerable realistic web application that is a photo sharing and purchasing site. In this application, a user is able to comment on photos, upload photos, browse other user's photos, and buy the rights to a high-quality version of a photo. The site was originally created by Doupé et al. in 2010 with the purpose of evaluating web application vulnerability scanners [30].

WackoPicko consists of sixteen common and known vulnerabilities. Since WackoPicko models a realistic web application, a user has to log in in order to access restricted functionality (refer to Table 4.4 for a sample list of predefined valid user credentials). Therefore, as can be seen in Table 4.3, the vulnerabilities are divided into two groups; ones that are publicly accessible and others that require authentication.

| Vulnerability Category | Publicly Accessible | Requires Authentication |
|---|---|---|
| Reflected XSS | ✓ | |
| Stored XSS | ✓ | |
| Session ID | ✓ | |
| Reflected SQL Injection | ✓ | |
| Command Line Injection | ✓ | |
| File Inclusion | ✓ | |
| File Exposure | ✓ | |
| Reflected XSS Behind JS | ✓ | |
| Weak Password | ✓ | |
| Parameter Manipulation | ✓ | |
| Stored SQL Injection | | ✓ |
| Directory Traversal | | ✓ |
| Multi-Step Stored XSS | | ✓ |
| Forceful Browsing | | ✓ |
| Logic Flaws - Coupon | | ✓ |
| Reflected XSS behind Flash | | ✓ |

Table 4.3: WackoPicko Test Cases

| Privilege | Username | Password |
|---|---|---|
| Standard | scanner1 | scanner1 |
| Standard | scanner2 | scanner2 |
| Standard | bryce | bryce |
| Admin | admin | admin |
| Admin | adamd | adamd |

Table 4.4: WackoPicko Valid Credentials

WackoPicko also contains several crawling challenges [7]:

- **HTML Parsing**: The WackoPicko application contains several features that require a scanner to properly parse through HTML frames. An instance of this challenge is testing the ability of a scanner to upload a file on the application.
- **Multi-Step Process**: In order to post a comment on an image, the user has to go through multiple steps that first require previewing the comment and then posting it. Unless the scanner is able to handle multi-step processes, the stored XSS vulnerability will not be detected.
- **Infinite Web Site**: WackoPicko contains a calendar that displays the agenda for a given day and links to the agenda for the following day. If a scanner does not limit the number of links it follows this will create an infinite sequence of pages that will cause the scanner to run indefinitely.
- **Authentication**: Since most websites require a user to create an account in order to access additional functionality, scanners must be able to handle authentication techniques. WackPicko contains registration and login features.
- **Client-side Code**: WackoPicko includes vulnerabilities that require a scanner to properly understand and parse through JavaScript and Flash applications.

## 4.3 Environment Setup

The environment setup consisted of two virtual machines (VM); a Kali Linux VM and the OWASP BWA Project VM [31]. The Kali Linux VM contained: (1) the evaluated scanners, (2) a docker container image of the WIVET application [32] and (3) a docker container image of the WAVSEP application [33]. The OWASP BWA Project VM was used to run the WackoPicko application with the network setting configured to Host-Only Adapter to grant access to the WackoPicko server IP address on the Kali Linux VM. To ensure that all tests are performed in an identical environment, both VMs were restored to their initial state before every test run.

Each scanner was run in two modes: default and configured. In the default mode, the default configuration settings (such as the crawl depth, wait time and elements to crawl) were not changed before running a scan. In the configured mode, the default settings were changed in order to maximize the crawling coverage and vulnerability detection. This involved three steps:

1. The architecture of the application and the web technologies used were studied to maximize the crawling coverage. An example configuration would be to increase the crawl depth or to ensure that the crawler clicks on a specific HTML element that is not specified in the default configuration. This step is equivalent to the mapping phase in web application penetration testing.

2. If the configuration in step 1 is still not able to map the entire application and the scanner has a proxy component, we manually visit every missed page of the application while the scanner is in proxy mode. This step helped us identify whether a scanner missed a vulnerability because it doesn't detect that type of vulnerability or because it wasn't able to reach it due to its poorly implemented crawler component.

3. The attack strength was increased to a maximum so that the scanner tests for a broader scope of vulnerabilities. In this step, we just maximized the vulnerability

attack strength instead of specifically choosing the attack vectors for the vulnerabilities present in our applications. This is because, unlike mapping an application, an ethical hacker will not have prior knowledge of the vulnerabilities present in the application.

| **Application** | **Configured Scan** | | |
|---|---|---|---|
| | Step 1 | Step 2 | Step 3 |
| WIVET | ✓ | | |
| WAVSEP | ✓ | | ✓ |
| WackoPicko | ✓ | ✓ | ✓ |

Table 4.5: Steps Included in Configuration Scan

Table 4.5 lists the different configuration steps performed for each application when running a configured scan. As can be seen in the table, a configured WIVET scan only involves step 1 since the purpose of the WIVET application is to measure the crawling coverage. Whereas, in the WAVSEP application, step 2 was not performed because the application does not contain any crawling challenges and therefore performing step 1 was enough to crawl the whole application.

Since the WackoPicko application requires user authentication (which is vulnerable and can be bypassed) in order to access several of the vulnerable pages, the test scans were further divided into two subcategories: INITIAL and CONFIG. In the INITIAL scan the scanner is just given the root URL of the application. Whereas, in the CONFIG scan the scanner is given a valid username/password combination.

Therefore, in total, each scanner was run eight times (twice on the WIVET application, twice on the WAVSEP application and four times on the WackoPicko application). Appendix A and Appendix B contain detailed instructions of the steps performed to run each scanner in default and configured modes against the chosen applications. Moreover, Appendix C explains the structure and content of the compiled results and associated files that we made publicly available on GitHub for further research.

## 4.4 Feature and Metric Selection

The fourth step of our research was to choose a subset of the features specified in the WASSEC guidelines that we consider to be essential to have in a scanner. Additionally, we chose a metrics system to measure the availability and performance of each feature.

Figure 4.3 provides a summary of the features and metrics selected.

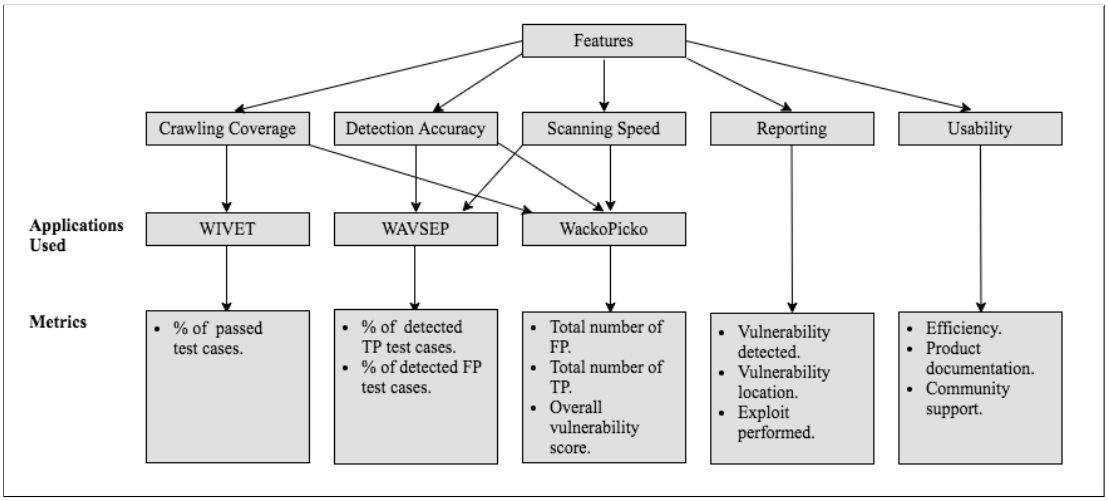



Figure 4.3: Summary of Feature and Metrics Selection

- **Crawling Coverage**: The crawling coverage was measured using the WIVET and WackoPicko applications. For the WIVET application, an overall percentage of the number of test cases passed is calculated. For the WackoPicko application, the list of crawling challenges that the scanner was not able to navigate is stated. Since the WAVSEP application does not contain any crawling challenges, it was not included in the applications used to measure this feature.

- **Vulnerability Detection Accuracy**: The vulnerability detection accuracy was measured using the WAVSEP and WackoPicko applications. For the WAVSEP application, a percentage of the number of TP and FP test cases detected is calculated. For the WackoPicko application, a list of the detected vulnerabilities is recorded with the

scanning mode it was detected in (INITIAL or CONFIG) and a total number of FP is reported. Since the WIVET application does not contain any vulnerabilities, it was not included in the applications used to measure this feature.

- **Scanning Speed**: The time it takes for a scanner to crawl and scan the WAVSEP and WackoPicko applications is measured. Since several of the chosen scanners have separate crawler and scanning components, we opted not to include the WIVET application to test this feature.

- **Reporting**: A report must include the vulnerabilities detected, the location of each vulnerability and the exploit performed to find the vulnerability.

- **Usability**: This feature is divided into three separate criteria:

  - Efficiency: Does the scanner have a GUI? Is the GUI user friendly? Is the software easy to obtain and install? Is prior technical knowledge required to operate the software?

  - Product Documentation: Does the scanner come with a user manual? Is the scanner well documented?

  - Community Support: Does the scanner have an online forum for support? How active is the online forum?

As mentioned in Section 2.3, there is no standardized metric system for measuring the qualitative and quantitative performance of a scanner. Therefore, depending on the needs and requirements of the user or organization a measuring system should be defined. For the purpose of our study, we adopt the metrics system used by Doupé et al. in [7] to measure the performance of scanners based on the crawling coverage and vulnerability detection on the WackoPicko application. Doupé et al. assign "Detection" scores for each vulnerability as shown in Table 4.6. Each vulnerability also has a "Reachability" score depending on how difficult it is to reach the vulnerability when the scanner is run in INITIAL or CONFIG

mode. Doupé et al. acknowledge that these vulnerability scores are subjective and they were chosen to try and evaluate the crawling and detection difficulty of each vulnerability. The final score for each scanner is calculated by summing up the detection score and the corresponding reachability score for each vulnerability detected. In our research, we use this scoring system to rank the evaluated scanners.

| **Name** | **Detection** | **INITIAL Reachability** | **CONFIG Reachability** |
|---|---|---|---|
| XSS Reflected | 1 | 0 | 0 |
| XSS Stored | 2 | 0 | 0 |
| SessionID | 4 | 0 | 0 |
| SQL Injection Reflected | 1 | 0 | 0 |
| Commandline Injection | 4 | 0 | 0 |
| File Inclusion | 3 | 0 | 0 |
| File Exposure | 3 | 0 | 0 |
| XSS Reflected behind JavaScript | 1 | 3 | 3 |
| Parameter Manipulation | 8 | 0 | 0 |
| Weak Password | 3 | 0 | 0 |
| SQL Injection Stored Login | 7 | 7 | 3 |
| Directory Traversal Login | 8 | 8 | 6 |
| XSS Stored Login | 2 | 8 | 7 |
| Forceful Browsing Login | 8 | 7 | 6 |
| Logic Flaws - Coupon | 9 | 9 | 8 |
| XSS Reflected behind Flash | 1 | 9 | 7 |

Table 4.6: Vulnerability Scores

## 4.5 Result Analysis

The last step of our methodology is to analyze the results obtained from running the test scans. We aim to answer the following questions:

- What is the overall crawling coverage of each scanner?
- Are the evaluated scanners capable of crawling modern web applications? If not, which technologies do the scanners lack support for?
- What is the overall vulnerability detection accuracy of each scanner?

- Are the evaluated scanners capable of detecting vulnerabilities commonly found in the wild? If not, which vulnerability categories are scanners most likely to miss?
- Is there a significant difference in terms of the crawling coverage, vulnerability detection accuracy and scanning speed between a default scan and a configured scan?
- What is the overall performance of the open-source scanners compared to the commercial scanner tested?

The next chapter discusses the results we have obtained in pursuing answers to the above questions.

# Chapter 5

# Experimental Evaluation and Results

In this chapter, we present the results obtained from running our experiments on the WIVET, WAVSEP and WackoPicko applications. We also provide a feature comparison on the usability and reporting quality of the scanners. Lastly, we rank the scanners based on crawling coverage and vulnerability detection.

## 5.1 Vulnerability Detection Accuracy

To determine the vulnerability detection accuracy, the scanners were run against the WAVSEP and WackoPicko applications. We analyzed the results and calculated the number of true positives, false positives and false negatives produced by each scanner.

### 5.1.1 False Negatives

The results obtained from running each scanner on the WackoPicko application are presented in Tables 5.1 and 5.2 for the default scan mode and Tables 5.3 and 5.4 for the configured scan mode. As can be seen, the following test cases were not detected by any scanner.

**Weak Password:** WackoPicko contains a separate admin interface that can be logged into with the username/password credentials admin/admin. Regardless of the easily guessable choice in authentication credentials, none of the scanners were able to figure out the username/password combination and log in as an admin.

**Session ID:** As mentioned above, none of the scanners were able to log into the admin interface and therefore none of the scanners detected the session id vulnerability present in the admin interface. As was done in the research methodology of Doupé et al., we decided not to give the scanners valid credentials for the admin interface since that would be inconsistent with a real world scenario where a user has to pentest a web application.

**Parameter Manipulation:** The WackoPicko home page contains a link that can be used by an unauthenticated user to view a sample user profile (whose id is 1). This link is vulnerable to parameter manipulation. If a user changes the id parameter in the URL to another valid id, the user is able to view that individual's page without logging in first and therefore bypass authentication. None of the scanners were able to detect this vulnerability. Burp, Wapiti, Vega and ZAP did not attempt to manipulate the parameter field and therefore did not detect the vulnerability. Arachni manipulated the parameter field but failed to enter a valid number. Skipfish was the only scanner to successfully manipulate the parameter field but did not realize that the returned response is an authenticated page and therefore did not report the parameter manipulation as a possible vulnerability.

**Stored SQL Injection:** In order to discover the stored SQL injection, the scanner has to successfully create a user account first. As can be seen in Table 5.6 all the scanners, with the exception of Wapiti, were able to create multiple user accounts. However, none of the scanners detected this vulnerability.

**Directory Traversal:** In order to discover the directory traversal vulnerability, the scanner has to be able to upload a picture. Arachni, Skipfish, Vega and Wapiti were not able to upload a photo and therefore did not detect the vulnerability. Burp and ZAP were also

not able to upload a photo when the scan was run in default mode, however when we proxied through the application (in configured scan mode) and manually showed Burp and ZAP how to upload a photo, they were able to upload pictures when the scan was run. Regardless, both scanners were still not able to detect the vulnerability.

**Multi-Step Stored XSS:** The multi-step vulnerability that required a user to first confirm a comment before submitting it was also not detected by any of the scanners. Arachni, Skipfish, Vega and Wapiti did not attempt to add a comment and therefore did not detect the vulnerability. Burp and ZAP behaved the same way when the scan was run in default mode, however, when we proxied through the application (in configured scan mode) and manually showed Burp and ZAP how to confirm and post a comment, they were able to perform this functionality. Regardless, both scanners were still not able to detect the vulnerability.

**Forceful Browsing:** WackoPicko contains a vulnerability where access to a link that contains a high quality version of a picture is not checked. Therefore, if a user obtains the link, the user can access the high quality version of the picture without logging in. None of the scanners discovered this vulnerability which is expected since this is an application specific vulnerability.

**Logic Flaw:** None of the scanners discovered the logic flaw in the coupon management functionality. This is also not surprising since such a flaw requires understanding the business logic used by the application to make decisions.

**Note**: Due to the large number of test cases in the WAVSEP application, we don't individually list each missed test case. The interested reader should refer to [34] for a complete list of passed and failed test cases.

### 5.1.2 True Positives

Most of the scanners detected less than 50% of the vulnerabilities available in the WackoPicko application. Tables 5.1 - 5.4 contain the detection results for the default and configured scans. For each scanner, the simplest configuration that detected the vulnerability was listed. For example, if the Stored XSS vulnerability was detected in both INITIAL and CONFIG modes, the configuration listed will be INITIAL. Refer to Section 4.3 to review the configuration setting of the different scanning modes.

As can be seen in the tables, the reflected XSS and stored XSS vulnerabilities were detected by all scanners. Similarly, the reflected SQL injection, command line injection and file inclusion vulnerabilities were detected by most scanners. However, several of the vulnerabilities that required crawling through dynamic JavaScript and Flash forms were not detected by nearly all the scanners. In fact, the XSS Reflected vulnerability behind Flash was only detected by the Burp and ZAP scanners in configured scan mode after manually proxying through the application. We analyze the crawling capabilities of the scanners in Section 5.2.

In terms of the WAVSEP application, the results were considerably better for most scanners. As can be seen in Figure 5.1, ZAP achieved the highest overall true positive detection rate covering a little under 80% of the test cases. Vega came in second, followed by Skipfish and Arachni. Figures 5.2 - 5.8 further breakdown the detection rate based on vulnerability category. As can be seen, Arachni was able to detect all the SQLi, unvalidated redirect and DOM XSS vulnerabilities. Moreover, the scanner scored really well in RFI and reflected XSS vulnerabilities. Similarly, ZAP scored really well in reflected XSS, SQLi, LFI, RFI and unvalidated redirect vulnerabilities. However, it performed poorly on the passive vulnerabilities and didn't detect any of the DOM XSS test cases. Burp Suite Pro performed well on the SQLi, reflected XSS and unvalidated redirect vulnerabilities. However, it performed poorly with the DOM XSS, passive, LFI and RFI test cases. Vega performed really well on most vulnerability categories, except for unvalidated redirect, DOM XSS

| Name | XSS Reflected | XSS Stored | SessionID | SQL Injection Reflected | Command line Injection | File Inclusion | File Exposure | XSS Reflected behind JavaScript |
|---|---|---|---|---|---|---|---|---|
| Arachni | INITIAL | INITIAL | | INITIAL | INITIAL | INITIAL | | INITIAL |
| Burp Pro | INITIAL | | | INITIAL | INITIAL | INITIAL | INITIAL | |
| Skipfish | INITIAL | INITIAL | | INITIAL | | INITIAL | INITIAL | |
| Vega | INITIAL | INITIAL | | INITIAL | | | | |
| Wapiti | INITIAL | INITIAL | | | INITIAL | INITIAL | | |
| ZAP | INITIAL | INITIAL | | INITIAL | INITIAL | INITIAL | | INITIAL |

Table 5.1: WackoPicko Default Scan Detection Results - Part 1/2

| Name | Parameter Manipulation | Weak Password | SQL Injection Stored Login | Directory Traversal Login | XSS Stored Login | Forceful Browsing Login | Logic Flaws - Coupon | XSS Reflected behind Flash |
|---|---|---|---|---|---|---|---|---|
| Arachni | | | | | | | | |
| Burp Pro | | | | | | | | |
| Skipfish | | | | | | | | |
| Vega | | | | | | | | |
| Wapiti | | | | | | | | |
| ZAP | | | | | | | | |

Table 5.2: WackoPicko Default Scan Detection Results - Part 2/2

| Name | XSS Reflected | XSS Stored | SessionID | SQL Injection Reflected | Command line Injection | File Inclusion | File Exposure | XSS Reflected behind JavaScript |
|---|---|---|---|---|---|---|---|---|
| Arachni | INITIAL | INITIAL | | INITIAL | INITIAL | INITIAL | | INITIAL |
| Burp Pro | INITIAL | INITIAL | | INITIAL | INITIAL | INITIAL | INITIAL | INITIAL |
| Skipfish | INITIAL | INITIAL | | INITIAL | | INITIAL | INITIAL | |
| Vega | INTIAL | INITIAL | | INITIAL | INITIAL | | | |
| Wapiti | INITIAL | INITIAL | | | INITIAL | INITIAL | | |
| ZAP | INITIAL | INITIAL | | INITIAL | INITIAL | INITIAL | | INITIAL |

Table 5.3: WackoPicko Configured Scan Detection Results - Part 1/2

| Name | Parameter Manipulation | Weak Password | SQL Injection Stored Login | Directory Traversal Login | XSS Stored Login | Forceful Browsing Login | Logic Flaws - Coupon | XSS Reflected behind Flash |
|---|---|---|---|---|---|---|---|---|
| Arachni | | | | | | | | |
| Burp Pro | | | | | | | | CONFIG |
| Skipfish | | | | | | | | |
| Vega | | | | | | | | |
| Wapiti | | | | | | | | |
| ZAP | | | | | | | | CONFIG |

Table 5.4: WackoPicko Configured Scan Detection Results - Part 2/2

and passive test cases. On the other hand, Wapiti received a poor detection rate on most vulnerability categories. Lastly, Skipfish performed really well in SQLi, reflected XSS and LFI vulnerabilities.

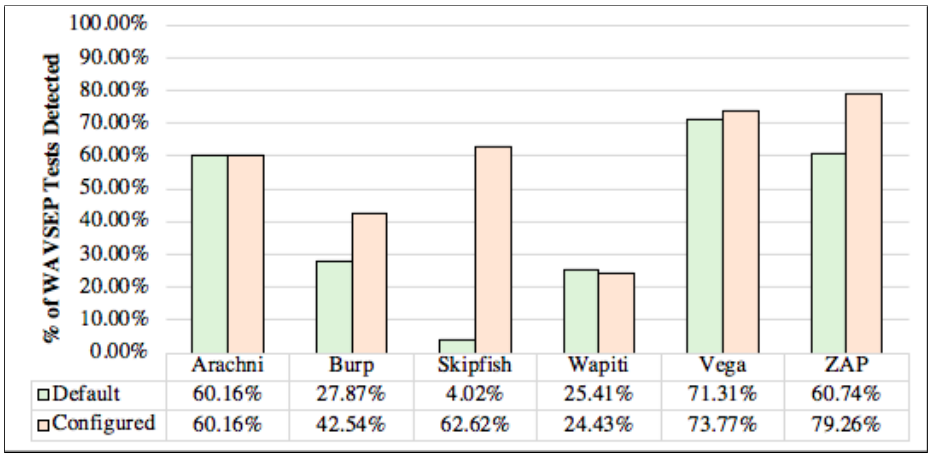



Figure 5.1: WAVSEP Overall TP Detection

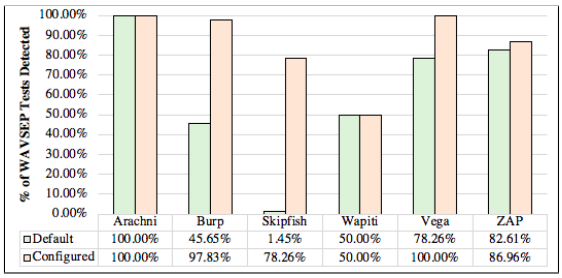



Figure 5.2: WAVSEP SQL Injection TP Detection

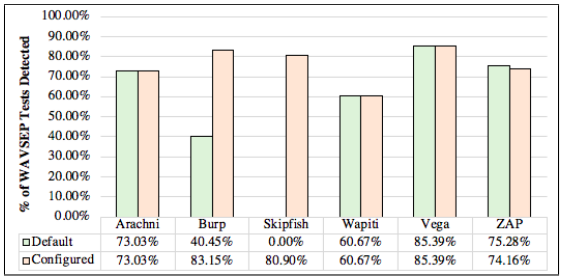



Figure 5.3: WAVSEP Reflected XSS TP Detection

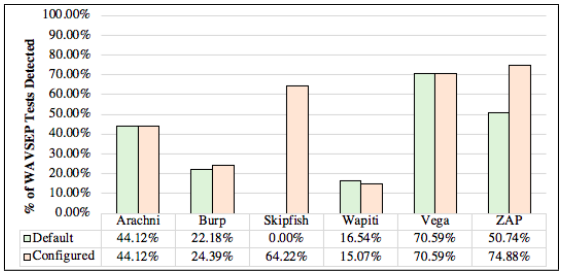



Figure 5.4: WAVSEP LFI TP Detection

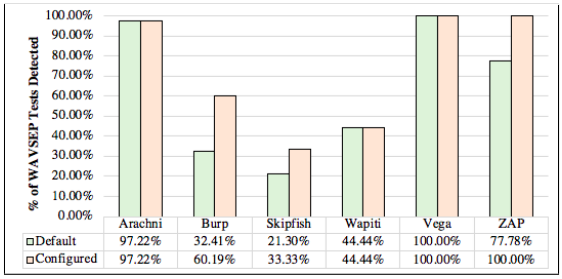



Figure 5.5: WAVSEP RFI TP Detection

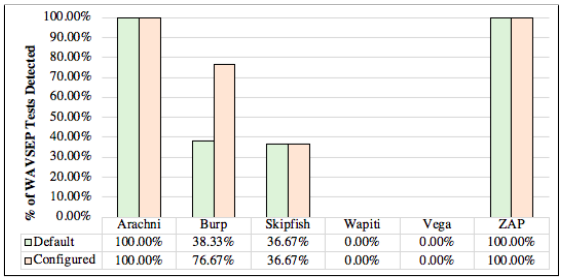



Figure 5.6: WAVSEP Unvalidated Redirect TP Detection

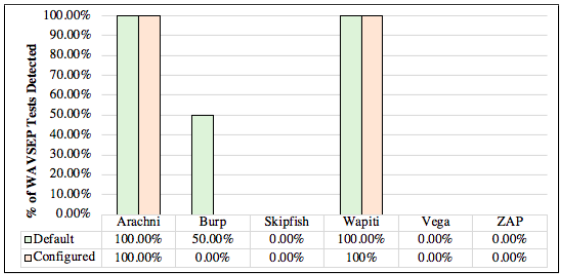



Figure 5.7: WAVSEP DOM XSS TP Detection

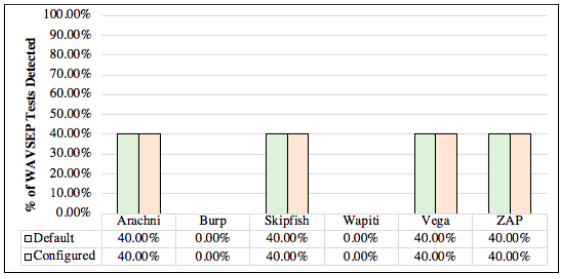



Figure 5.8: WAVSEP Passive TP Detection

### 5.1.3 False Positives

Table 5.5 lists the number of false positives reported by the scanners for both default and configured scan modes on the WackoPicko application. There seems to be a correlation between the number of true positives detected and the number of false positives reported. Burp, Arachni and ZAP were among the scanners that detected the largest number of vulnerabilities and therefore achieved the highest number of true positives. In comparison, they also have the highest number of false positives.

| | **Default Scan** | | **Configured Scan** | |
|---|---|---|---|---|
| **Name** | INITIAL | CONFIG | INITIAL | CONFIG |
| Arachni | 5 | 5 | 5 | 5 |
| Burp Pro | 4 | 4 | 4 | 5 |
| Skipfish | 2 | 0 | 0 | 1 |
| Vega | 3 | 2 | 2 | 1 |
| Wapiti | 0 | 0 | 0 | 0 |
| ZAP | 1 | 2 | 3 | 8 |

Table 5.5: Number of False Positives

This correlation is not consistent with the results obtained from running the scanners on the WAVSEP application. ZAP followed by Vega, Skipfish and Arachni had the highest true positive detection rate, however in Figure 5.9, Burp Suite Pro received the highest overall false positive detection rate. It is worth noting that the false positives were calculated differently for the WackoPicko and WAVSEP applications. In the WackoPicko application, we analyzed all the vulnerabilities reported and any high critical vulnerability incorrectly reported was recorded as a false positive. However, in the WAVSEP application, there is a predefined set of false positive test cases, that if detected by the scanners were reported as a false positive. Any other generated false positives that did not fall in the predefined test cases list were not reported. Therefore, it is not surprising that the correlations made using the WackoPicko application are inconsistent with those made using the WAVSEP application.

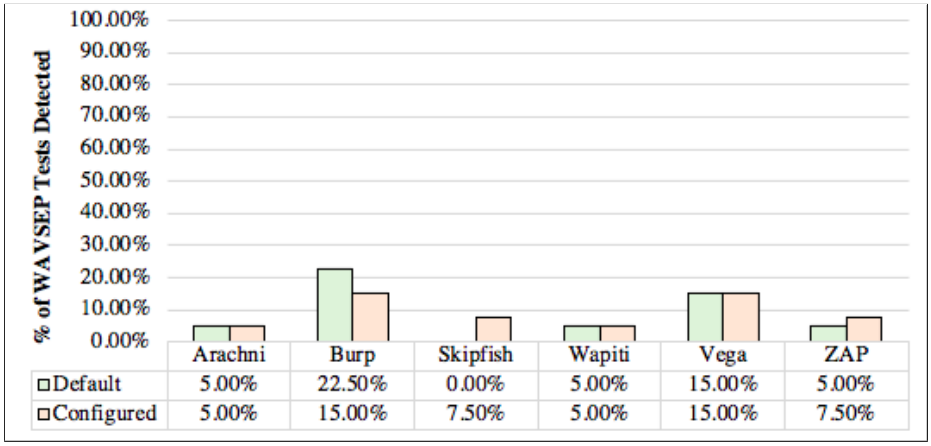



Figure 5.9: WAVSEP Overall FP Detection

Figure 5.10 - 5.14 further break down the number of reported false positives based on vulnerability categories. As can be seen in Figure 5.11 none of the scanners reported any RXSS false positives. Whereas, the majority of the scanners reported LFI false positives. For the WAVSEP application, there seems to be no correlation between the number of TP and FP reported by the scanners.

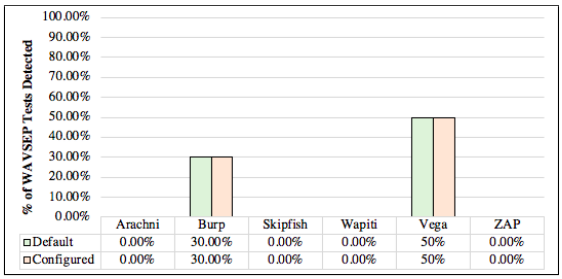



Figure 5.10: WAVSEP SQL Injection FP Detection

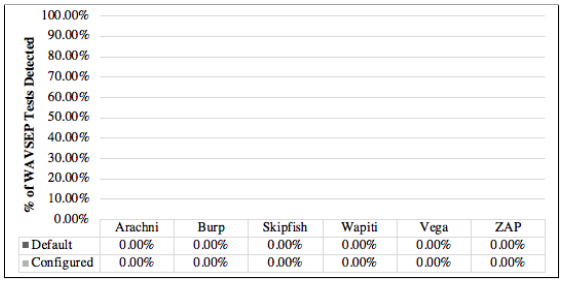



Figure 5.11: WAVSEP Reflected XSS FP Detection

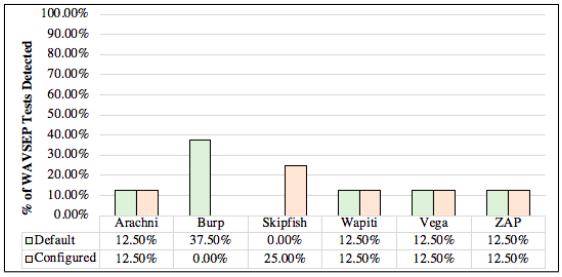



Figure 5.12: WAVSEP LFI FP Detection

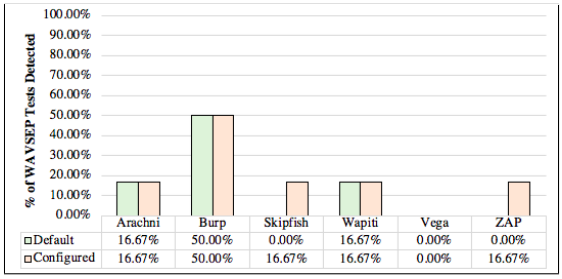



Figure 5.13: WAVSEP RFI FP Detection

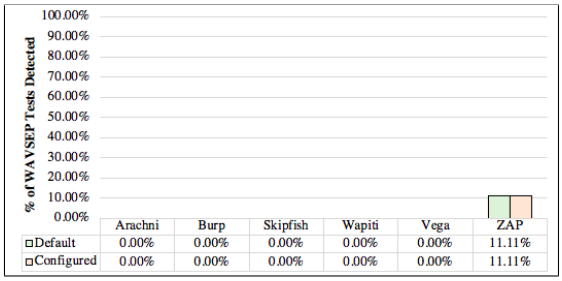



Figure 5.14: WAVSEP Unvalidated Redirect FP Detection

## 5.2 Crawling Coverage

To determine the crawling coverage, the scanners were run against the WIVET and WackoPicko applications. For the WIVET application, an overall percentage of the number of test cases passed is calculated. For the WackoPicko application, the list of crawling challenges that the scanner was not able to navigate is discussed.

Below we describe the various features that scanners found difficult to crawl in the WackoPicko application.

**Uploading a Picture.** All the scanners were not able to upload a picture in default scan mode. When we proxied through the application (in configured scan mode) and manually showed Burp and ZAP how to upload a photo, they were able to perform this functionality.

**Authentication.** All the scanners, with the exception of Wapiti, were able to register an account. Table 5.6 lists the number of accounts created when the scanner is run in default mode. Wapiti did not attempt to create an account in either the default or the configured scan. Since most web applications require a user to register an account, it is very important that a scanner supports this functionality. However, it is worth noting that the scanners that created the accounts, did not use the accounts to log in into an authenticated session. Therefore, this leads us to infer that the scanners were merely passing data to the form and unintentionally ended up creating accounts.

| Name | Number of Accounts |
|---|---|
| Arachni | 202 |
| Burp Pro | 113 |
| Skipfish | 364 |
| Vega | 117 |
| Wapiti | 0 |
| ZAP | 111 |

Table 5.6: Account Creation

**Multi-step Processes.** All the scanners did not attempt to add a comment on any of the pictures. This leads us to believe that there is a lack of support for crawling the HTML `<textarea>` tag where the comment is input. When we proxied through the application (in configured scan mode) and manually showed Burp and ZAP that input can be entered in the comment field, they were able to complete this multi-step functionality. Although Vega also has a proxy component, it was still not able to post a comment which leads us to believe that this is a limitation in the proxy component.

**Infinite Websites.** All the scanners, except for Arachni, were able to figure out that the calendar page can lead to an infinite sequence of pages and therefore the scanners stopped crawling the page after several tries. When we first ran Arachni using the default settings,

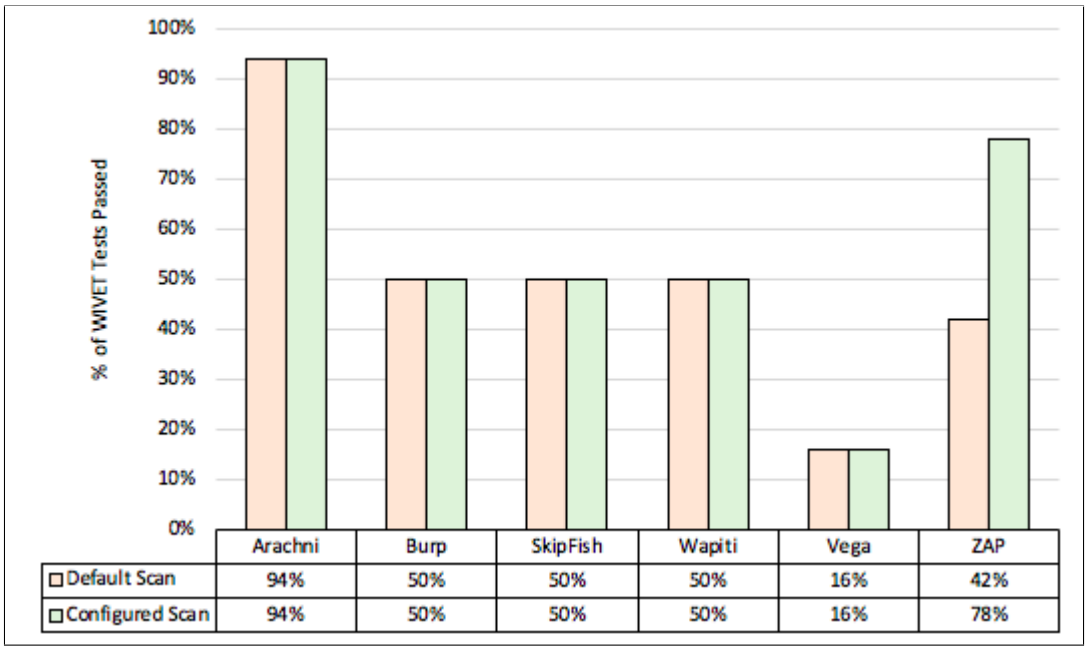
the scanner resulted in an infinite crawl and had to be terminated. Therefore, we added a built in redundancy filter in the script to limit the amount of times a URL in the calendar page is followed.

**Client-side Code.** The results obtained from the scanners on the WIVET application are shown in Figure 5.15. As can be seen, Arachni achieved the highest score among all scanners and was able to crawl 53 out of the 56 test cases present in the WIVET application. Two of the missed test cases were due to the lack of support for SWF, an Adobe Flash file format. The other missed test case was because Arachni did not follow a link provided in an unattached JS function. However, in all the other test cases where JS functions were called, Arachni was able to detect them. Therefore, we conclude that Arachni is able to properly parse through websites that contain dynamic JavaScript. This outcome is also supported by Arachni discovering the reflected XSS vulnerability in the WackoPicko application that is only accessible by crawling a form that is dynamically generated using JS.

The second highest score was achieved by the ZAP scanner that was able to detect 44 out of the 56 test cases. This was largely due to ZAP not being able to handle the `onmouseout` and `onmouseover` events. However, since ZAP was able to successfully handle the `onmousedown` and `onmouseup` events, we consider this to be a bug in ZAP's JavaScript parsing as opposed to a general limitation in the crawler component of ZAP. This outcome is also supported by ZAP discovering the reflected XSS vulnerability in the WackoPicko application hidden behind JavaScript.

Burp, Skipfish and Wapiti achieved the third highest score by detecting 28 out of the 56 test cases. This was largely due to ineffective parsing of dynamic JavaScript. Vega came in last detecting only 9 out of the 56 test cases. Most of the detected test cases required little to no JavaScript parsing and therefore, we conclude Vega is generally not able to handle content dynamically generated by JavaScript. The above two conclusions are also supported by the scanners (with the exception of Burp) not being able to discover the reflected XSS vulnerability in the WackoPicko application hidden behind JavaScript.

The last result to note is that all of the scanners were not able to detect the two SWF test cases present in the WIVET application, except for Wapiti which was able to detect only one of the test cases. This leads us to the conclusion that all the scanners have little to no support for Flash applications. This conclusion can also be supported by the results seen in the WackoPicko application. None of the scanners were able to detect the vulnerability behind a Flash application in the default scan. ZAP and Burp were only able to detect it after we individually visited that page while the scanner was in proxy mode. This shows that although ZAP and Burp have the ability to detect that type of vulnerability, they were not able to reach it and therefore missed it in the default scan test run.



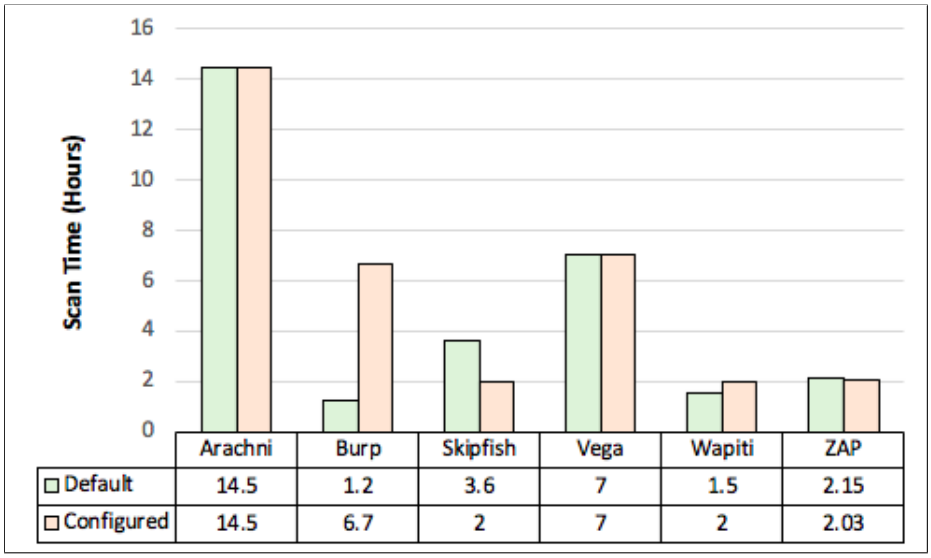

| | Arachni | Burp | SkipFish | Wapiti | Vega | ZAP |
|---|---|---|---|---|---|---|
| Default Scan | 94% | 50% | 50% | 50% | 16% | 42% |
| Configured Scan | 94% | 50% | 50% | 50% | 16% | 78% |

Figure 5.15: WIVET Results

## 5.3 Scanning Speed

We measured the time it takes to scan the WAVSEP and WackoPicko applications. As can be seen in Figure 5.16, the scanning speed varies across the different scanners. In Section 5.1 Figure 5.1 Arachni, ZAP, Vega and Skipfish when tested in configured mode

achieved a comparable vulnerability detection score. However, Arachni took 14.5 hours to run, whereas ZAP and Skipfish took only a little over 2 hours.




| | Arachni | Burp | Skipfish | Vega | Wapiti | ZAP |
|---|---|---|---|---|---|---|
| Default | 14.5 | 1.2 | 3.6 | 7 | 1.5 | 2.15 |
| Configured | 14.5 | 6.7 | 2 | 7 | 2 | 2.03 |

Figure 5.16: WAVSEP Scanning Speed

The WackoPicko scanning speed outcome is similar to that of WAVSEP. Figures 5.17 and 5.18 show that there is a considerable variation between the scanning speed of the scanners. The one correlation we can arrive at, is that configuring a scanner generally increases the time it takes to scan an application. This makes sense since in a configured scan, the attack vector and crawling coverage is increased.

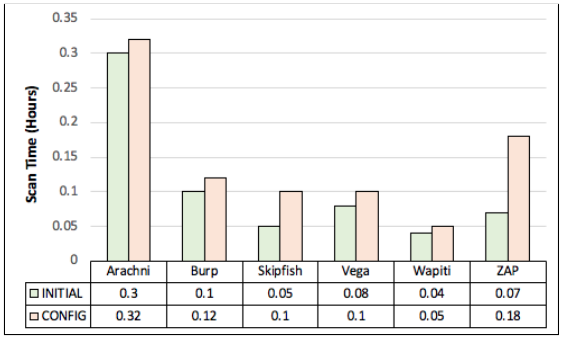



Figure 5.17: WackoPicko Default Scanning Speed

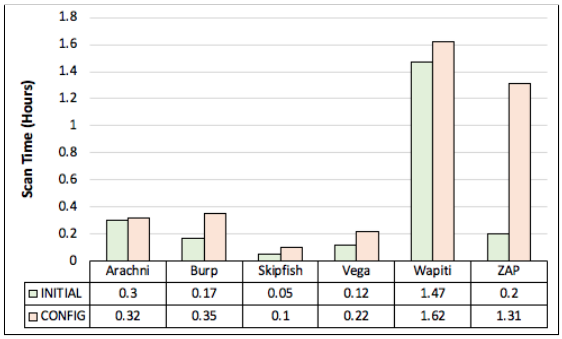



Figure 5.18: WackoPicko Configured Scanning Speed

## 5.4 Reporting Features

Each scanner has its own unique reporting features. Examples of such features include allowing a user to customize report content, supporting multiple export formats and color coding vulnerabilities based on severity. We consider that the most important features a report must have are (1) a list of all the vulnerabilities detected, (2) the locations of all the detected vulnerabilities, and (3) the exploits performed to detect these vulnerabilities. We reviewed the reports generated by the chosen scanners and all six scanners implement the above three features.

## 5.5 Usability Features

The usability features we evaluated were divided into three separate criteria: (1) efficiency, (2) product documentation and (3) community support. Table 5.7 summarize the results of our analysis.

**Efficiency.** Arachni, Burp, Vega and ZAP have a user friendly GUI that a user can interact with in order to run test scans. On the other hand, Skipfish and Wapiti require users to

interact with a command line interface. In general, prior technical knowledge was required to operate all the scanners. Each scanner either has different GUI components or command-line options that a user needs to learn before operating the software. In terms of software installation, all scanners were easy to obtain and install. Several of the scanners required dependencies that were not originally installed on the used operating system; however, that is standard procedure for installing software.

**Product Documentation.** All scanners came with user manuals that were well documented. In terms of the scanners that had both a GUI and a CLI, the GUI came with a user manual, whereas the CLI came with a `man` page.

**Community Support.** Arachni, Burp, Vega, Wapiti and ZAP have multiple online forums such as Google Groups, SourceForge projects and GitHub repositories that a user can use to ask questions and report issues. However, only the Arachni, Burp, Wapiti and ZAP forums have been active in the past three months. Note that the issues we discovered with the scanners (Appendix D) have been reported to the scanner developers across the past several months, however, we didn't receive a response except from the ZAP team.

| | **Efficiency** | | **Product Documentation** | | **Community Support** | |
|---|---|---|---|---|---|---|
| **Name** | GUI | CLI | User Manual | Well Documented | Support | Active |
| Arachni | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Burp Pro | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| Skipfish | x | ✓ | ✓ | ✓ | x | x |
| Vega | ✓ | x | ✓ | ✓ | ✓ | x |
| Wapiti | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| ZAP | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Table 5.7: Usability Features

## 5.6 Final Ranking

As mentioned earlier, there is no standardized metric system for measuring the performance of a scanner. Therefore, depending on the needs and requirements of the user or organi-

zation a measuring system should be defined. We rank the scanners based on crawling coverage and vulnerability detection using the scoring system described in Section 4.4. Table 5.8 presents the overall final ranking of the six evaluated scanners. As can be seen, Burp Suite Pro achieved the highest score followed by ZAP, Arachni, Wapiti, Skipfish and Vega.

| Name | Score |
|---|---|
| Burp Pro | 26 |
| ZAP | 23 |
| Arachni | 15 |
| Wapiti | 10 |
| Skipfish | 10 |
| Vega | 8 |

Table 5.8: Final Ranking

## 5.7 Further Discussion of Results

In the above sections, we provided a comparative technical analysis on six scanners by measuring the vulnerability detection accuracy, crawling coverage, speed, reporting and usability features. In this section, we take a step back from the technical details and provide a high level discussion on the performance of the scanners based on the identified limitations.

The first thing to note is that web application vulnerability scanners are trying to solve a very difficult problem. In computability theory, it is known that having a program determine the properties of another program is undecidable and therefore cannot be done. An example of that is the halting problem where Turing proved that given a computer program and arbitrary input to that program, it is impossible to determine whether the program will eventually halt or run forever. Vulnerability scanners whose function is to determine all the possible attack vectors and identify all vulnerabilities in an application can be reduced to the halting problem. Therefore, scanners are in principal, trying to perform a task that is

impossible to do completely. As a consequence of that, scanners are bound to miss vulnerabilities. This begs the question, how many vulnerabilities is a scanner likely to miss? It is clear from the results of our research and the current state of technology that scanners are going to miss more vulnerabilities than they detect. We showed that detecting a vulnerability essentially depends on two components: (1) effective crawling of the application, and (2) cleverly crafting an attack pattern to detect a vulnerability.

Let's discuss the first component. The scanners we tested had varying crawling capabilities. In terms of web technologies, Arachni and ZAP had the highest ability to crawl dynamic JavaScript. In terms of different web architectures / designs, all scanners had difficulty crawling through common web functionality such as uploading a file and completing a multi-step process without any human interference. This brings us to the proxy component of a scanner. In our study, scanners that had a properly working proxy component such as Burp and ZAP, achieved a higher crawling coverage and as a byproduct, a higher vulnerability detection rate. This was due to the fact that we were able to train the scanners to identify links in the application that they would not have otherwise detected in a default automated scan. Moreover, we showed that scanners have difficulty recognizing and maintaining an authenticated state. Although all of the tested scanners, with the exception of Wapiti, discovered the SQL injection in the log in form, none of the scanners were able to maintain an authenticated page to discover the vulnerabilities that require authentication. This issue was only solved when we forced the scanners to stay in an authenticated state by giving the scanners user credentials and removing the log out link from the scanner scope.

In academia, extensive research has been done to improve web crawlers. This research addresses the limitations of crawling Web 2.0 technologies such as dynamic JavaScript, AJAX applications and Rich Internet Applications (RIA) [35] [36] [37] [38] [39] [40] [41] by proposing new algorithms and techniques. It is interesting to see that many of the scanners displayed poor crawling capabilities regardless of the comprehensive research on this subject. As we discuss in the next section, one way to improve the scanners would be

to look at the body of literature and attempt to integrate these algorithms and techniques to increase the crawling capability of the scanners on modern web applications.

The second component to detecting a vulnerability is to cleverly craft an attack pattern. The crafted attack patterns are either predefined values that are already known to exploit a specific vulnerability such as a predefined xss vectors cheat sheet, or input values that are generated using heuristic algorithms. The scanners we tested showed varying vulnerability detection capabilities. In terms of the WAVSEP application, if a combination of the scanners tested was used, all the vulnerabilities would have been detected. This is an important result for individuals looking to integrate a scanner into the SDLC of their application. Instead of integrating one scanner, a combination of scanners can provide a broader coverage of the application and test for a wider range of vulnerability categories. In terms of the WackoPicko application, half the vulnerabilities went undetected by all the scanners. This was due to largely two reasons: (1) the web functionality and architecture used - which we discussed in the above few paragraphs, and (2) understanding the business logic of the application. An example of the second reason is the forceful browsing vulnerability and logic flaw in the coupon management system. Similarly, the parameter manipulation vulnerability that required understanding how a session id is allocated to a user. It is worth noting that the individuals who created the WackoPicko application [7] had a group of students with average security skills test the application for any vulnerabilities. The students discovered all the vulnerabilities with the exception of the forceful browsing vulnerability. This goes to show that an individual with average security skills is likely to discover a considerably higher number of vulnerabilities than a scanner would.

There is no doubt that scanners are becoming more effective in crawling web applications and detecting known vulnerabilities, however that progress is happening very slowly. Above, we referred to a body of literature that can be used to improve these scanners. However, with the rate at which web technologies are becoming more complex, the coverage gap has widened across the years and put scanners at a constant battle of trying to keep up

with these technologies. And if our research shows anything, it is that this is a battle that scanners are currently losing.

### 5.7.1 Comparison with Previous Work

Providing a concrete comparison across the years is difficult for a topic such as the one presented in this thesis. This is because previous research evaluated different tools on different benchmarks with different configuration settings. In this section, we attempt to compare our research with the previous work discussed in Section 3.

Suto's 2007 [5] and 2010 [6] case studies tested scanners in two modes of configuration: (1) PaS mode where the scanner is only given the root URL of the web application and the default configuration is not altered, and (2) Trained mode where the scanner is made aware of all the pages it is supposed to test. Suto's work showed that scanners missed a significant number of vulnerabilities regardless of mode of configuration. Most scanners only showed moderate improvements when run in Trained mode with the exception of only one scanner where the detection accuracy was significantly higher. Moreover, the study observed that the scanners that were more likely to miss vulnerabilities, were more likely to report a high false positive number. Our results don't agree with the conclusions presented by Suto. We showed that configuring a scanner can have a significant increase on the vulnerability detection rate of the scanner. In the WackoPicko application, Burp, Vega and ZAP showed an overall higher detection rate when run in Configured mode. Burp had the highest increase from 37.5% to 50%. Similarly, the same scanners showed significant increase in the WAVSEP application with Skipfish having the highest overall increase from 4% to 62.6%. Our results also show that unlike Suto's outcome, scanners that detected the largest number of vulnerabilities have the highest number of false positives. The inconsistencies between our results and Suto's results is not surprising since there is no overlap between the scanners tested and the benchmark applications used in this research with the ones used by Suto.

In [20], Chen provides a feature and performance comparison of open-source and commercial scanners on the WIVET and WAVSEP applications. This study is updated every few years with the latest study performed in 2016. For the WIVET application, our results are generally consistent with the results presented by Chen. Arachni and Burp achieved the same score on Chen's evaluation. Wapiti and ZAP achieved slightly higher scores in our study, which is not surprising since we're using more up to date versions of the scanners. In our study Skipfish achieved a score of 50% whereas in Chen's evaluation it achieved a score of 48%. This is unexpected since Skipfish hasn't been updated in the past 8 years and both studies are using the same version. Similarly, Vega v1.0 achieved a score of 16% in our study, whereas it achieved a score of 50% in Chen's study. The configuration steps are not listed in Chen's evaluation so it is difficult to deduce whether this was a mistake in Chen's evaluation or incorrect configuration on our part. It is worth noting that other than changing the crawl depth, the Vega scanner as very limited crawling configuration and we have iteratively tested all the configuration settings in order to achieve the best possible score. For the WAVSEP application, our results were generally significantly lower than the results presented in Chen's evaluation. This could be due to several reasons: (1) difference in configuration settings, (2) use of external plugins in order to increase scanner performance, which we did not include in order to have a level playing field in which the scanners are tested, and (3) how the detected vulnerabilities were interpreted. We attribute the difference in results mainly due to the last two reasons, with specific focus on the third reason. When we analyzed the vulnerabilities detected by the scanners, we noticed that the scanners detected a large portion of the vulnerabilities, if not all the vulnerabilities. However, these vulnerabilities were classified in incorrect vulnerability categories. For example, an SQL injection being classified as an XSS vulnerability. In this research, if a vulnerability was incorrectly classified, the vulnerability was listed as undetected.

In [8], Bau et al. evaluated scanners against previous versions of real world vulnerable applications. The chosen versions of the applications had well documented XSS, SQLi,

XCS, session, CSRF and information leakage vulnerabilities. Bau et al. reported that the scanners did exceptionally well in detecting information disclosure and session management vulnerabilities. The scanners also did moderately well in detecting XSS and SQLi vulnerabilities with a 50% detection rate for both categories. However, CSRF and XCS vulnerabilities had a very low detection rate. In our study we didn't evaluate any CSRF and XCS vulnerabilities. However, we included several test cases of XSS, SQLi and information disclosure vulnerabilities. Our results show that scanners generally did exceptionally well detecting XSS and SQLi vulnerabilities, whereas passive test cases that included information disclosure vulnerabilities had a very low detection rate. As mentioned earlier, it is difficult to compare results since there is no overlap between the scanners tested and the benchmark applications used in this research with the ones used by Bau et al..

In [7], Doupé et al. evaluated the scanners against a benchmark application they developed, called WackoPicko. The scanners were run in three different configuration modes; INITIAL, CONFIG and MANUAL. In the INITIAL mode, the scanner is directed to the root URL of the application. In CONFIG mode, the scanner is given a valid username and password combination. In MANUAL mode, the scanner was put in "proxy" mode while the authors browsed every page of the application. The authors found that crawling modern web applications poses a serious challenge for scanners. Several of the vulnerabilities were only found in MANUAL mode. The web technologies used in the application were partly responsible for the low crawling coverage. Doupé et al. found that scanners are also incapable of handling complex architectures of web applications such as being able to upload a file, handling authentication forms and correctly dealing with muli-step processes. Moreover, Doupé et al. reported that the commercial scanners did not significantly score better than the open-source scanners. In our study we used the WackoPicko application as one of our benchmarks and therefore we can compare our results to the ones presented by Doupé et al.. Although we used different scanners, our conclusions are consistent with the conclusion presented in Doupé et al.'s study. The open-source scanners evaluated in our study also

had difficulty crawling through common web technologies such as dynamic JavaScript and were incapable of handling complex architectures of web applications such as uploading a file and correctly dealing with muli-step processes. Similarly, the scanners did not detect several categories of vulnerabilities and missed at least 50% of the vulnerabilities present in the WackoPicko application. Lastly, we also compared the open-source scanners with the commercial scanner Burp Suite Pro and showed that the commercial scanner does not have a significant added value compared to the open-source scanners.

# Chapter 6

# Conclusion

In this thesis, we evaluated six black-box web application vulnerability scanners: five open-source and one commercial. We measured the vulnerability detection accuracy, crawling coverage, scanning speed, reporting features and usability features of each scanner. The vulnerability detection accuracy and crawling coverage were evaluated by running the scanners in a controlled environment against three benchmark applications: WIVET, WAVSEP and WackoPicko. The WIVET application was used to measure the crawling capabilities of the scanners and the WAVSEP application was used to measure the vulnerability detection accuracy. WackoPicko, on the other hand, contained both crawling challenges that represent the current architecture of web applications and web technologies used, and included several categories of common and known vulnerabilities. Since WackoPicko represents a realistic web application, we used its crawling coverage and detection accuracy to determine a final ranking of the tested scanners.

Our results show that being able to properly crawl a web application is a critical task in detecting vulnerabilities. Unfortunately, the majority of the scanners evaluated had difficulty crawling and properly parsing through common web technologies such as dynamically generated JavaScript content and Flash applications. Several vulnerabilities were overlooked because the scanners were not able to properly crawl through the application. On the other

hand, our work shows that several classes of vulnerabilities still go undetected due to (1) the logic used by the application to find the vulnerability, and (2) the lack of rules and detection support for such vulnerabilities.

For each benchmark application, the scanners were run in a point-and-shoot manner where the scanner is just given the root URL of the application to test, and in a trained mode where the scanner is configured to maximize the crawling coverage and vulnerability detection accuracy. Comparing the output of both modes has yielded interesting results. Several of the scanners showed considerable improvements when run in trained mode, whereas others displayed little to no improvement. This was largely due to the predefined default configuration that the scanners used and the presence of a proxy component. This improvement, however, came at a cost of increased running time and added human interference.

We have also found that the performance of several of the open-source scanners was similar to the performance of the state-of-the-art Burp Suite commercial scanner. Limited to the scope of this study, this shows that commercial scanners do not necessarily have a significant added value compared to open-source scanners. Additional research should be conducted to comparatively evaluate the performance of the remaining commonly used scanners not considered in this study.

## 6.1 Limitations and Future Work

The limitations and future work can be divided into three categories: (1) limited choice of web application vulnerability scanners tested, (2) analysis of the crawling and vulnerability detection challenges discovered, and (3) benchmark selection.

**Limited choice of web application vulnerability scanners.** In our study, we evaluated five open-source scanners and one commercial scanner. The scanners were chosen based on crawling coverage and vulnerability detection accuracy conducted in previous literature and on recommendations from professional ethical hackers. However, there is a growing

list of scanners being used in the industry. Therefore, future research can be done to apply our evaluation method to include more of the most commonly used scanners.

**Analysis of crawling and vulnerability detection challenges discovered.** In our research we analyzed the reasons behind several of the crawling challenges experienced by the scanners. Since five of the scanners evaluated were open-source, this provides the researcher a unique opportunity to view the scanner's source code and more concretely explain the limitations in the crawling and attack components of a scanner. Therefore, future research can be done for a more in depth approach in analyzing the design issues and limitations of the scanners. In the previous section we also referred to an extensive body of literature that can be studied in order to integrate the proposed algorithms and techniques to increase the crawling capability and vulnerability detection rate of the scanners on modern web applications.

**Benchmark selection.** In our study we chose the benchmark applications based on the vulnerability categories and crawling challenges they contain. It is obviously impossible to test the scanners against every vulnerability found in the wild and all the web technologies and architectural implementations found in web applications. We believe that the benchmarks selected contained a comprehensive list of common vulnerabilities and crawling challenges. However, future research can be done to include more test cases to the existing benchmarks as new technologies and vulnerabilities arise.

# References


[1] Internet Live Stats, "Internet Users." http://www.internetlivestats.com/internet-users/, 2018. Accessed Aug. 4, 2018.

[2] Internet Live Stats, "Total number of Websites." http://www.internetlivestats.com/total-number-of-websites/, 2018. Accessed Aug. 4, 2018.

[3] Braga, M., "100,000 Canadian victims: What we know about the Equifax breach - and what we don't." https://www.cbc.ca/news/technology/equifax-canada-breach-sin-cybersecurity-what-we-know-1.4297532, Sept. 2017. Accessed Aug. 4, 2018.

[4] Trustwave, "2018 Trustwave Global Security Report." https://www2.trustwave.com/GlobalSecurityReport.html, 2018. Accessed Aug. 4, 2018.

[5] L. Suto, "Analyzing the Effectiveness and Coverage of Web Application Security Scanners," Oct. 2007.

[6] L. Suto, "Analyzing the Accuracy and Time Costs of Web Application Security Scanners," Feb. 2010.

[7] A. Doupé, M. Cova, and G. Vigna, "Why Johnny Can't Pentest: An Analysis of Black-Box Web Vulnerability Scanners," in *Detection of Intrusions and Malware, and Vulnerability Assessment* (C. Kreibich and M. Jahnke, eds.), (Berlin, Heidelberg), pp. 111–131, Springer Berlin Heidelberg, 2010.

[8] D. Gupta, J. Bau, J. Mitchell, and E. Bursztein, "State of the Art: Automated Black-Box Web Application Vulnerability Testing," in *2010 IEEE Symposium on Security and Privacy (SP)*, pp. 332–345, May 2010.

[9] "The OWASP Foundation." https://www.owasp.org/index.php/Main_Page, 2018. Accessed Aug. 3, 2018.

[10] "Category:OWASP Top Ten Project." https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=Main, 2018. Accessed Aug. 3, 2018.

[11] "OWASP Top 10 - 2017: The Ten Most Critical Web Application Security Risks." https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf, 2018. Accessed Aug. 3, 2018.

[12] J. Jive, "XSS Vectors Cheat Sheet." https://gist.github.com/kurobeats/9a613c9ab68914312cbb415134795b45, 2017. Accessed Aug. 3, 2018.

[13] B. Shura, R. Auger, and R. Gaucher, "Web Application Security Scanner Evaluation Criteria." http://projects.webappsec.org/w/page/13246986/Web%20Application%20Security%20Scanner%20Evaluation%20Criteria, 2009. Accessed Aug. 4, 2018.

[14] E. I. Tatli and B. Urgun, "Web Input Vector Extractor Teaser." https://github.com/bedirhan/wivet, 2014. Accessed Aug. 1, 2018.

[15] S. Chen, "The Web Application Vulnerability Scanner Evaluation Project." https://github.com/sectooladdict/wavsep, 2014. Accessed Aug. 2, 2018.

[16] S. ElIdressi, N. Berbiche, F. Guerouate, and M. Sbihi, "Performance Evaluation of Web Application Security Scanners for Prevention and Protection against Vulnerabilities," in *International Journal of Applied Engineering Research*, vol. 12, pp. 11068–11076, 2017.

[17] Y. Makino and V. Klyuev, "Evaluation of Web Vulnerability Scanners," in *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, vol. 1, pp. 399–402, Sept. 2015.

[18] K. McQuade, "Open Source Web Vulnerability Scanners: The Cost Effective Choice?," in *Conference for Information Systems Applied Research*, 2014.

[19] M. Alsaleh, N. Alomar, M. Alshreef, A. Alarifi, and A. Al-Salman, "Performance-Based Comparative Assessment of Open Source Web Vulnerability Scanners," *Security and Communication Networks*, vol. 2017, pp. 1–14, May 2017.

[20] S. Chen, "The Prices vs. Features of Web Application Vulnerability Scanners." `http://sectoolmarket.com/price-and-feature-comparison-of-web-application-scanners-opensource-list.html`, 2016. Accessed Aug. 1, 2018.

[21] "Home - Arachni - Web Application Security Scanner Framework." `http://www.arachni-scanner.com/`, 2017. Accessed Aug. 4, 2018.

[22] L. Kuppan, "IronWASP - Iron Web application Advanced Security testing Platform." `https://ironwasp.org/index.html`, 2014. Accessed Aug. 4, 2018.

[23] "OWASP Zed Attack Proxy Project." `https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project`, 2018. Accessed Aug. 4, 2018.

[24] M. Zalewski, "skipfish(1) - Linux man page." `https://linux.die.net/man/1/skipfish`. Accessed Aug. 4, 2018.

[25] N. Surribas, "Wapiti : a Free and Open-source Web-application Vulnerability Scanner." `http://wapiti.sourceforge.net/`, 2018. Accessed Aug. 4, 2018.

[26] "Vega Vulnerability Scanner - Subgraph OS." `https://subgraph.com/vega/`, 2014. Accessed Aug. 4, 2018.

[27] "Burp Suite Scanner | PortSwigger." https://portswigger.net/burp, 2018. Accessed Aug. 4, 2018.

[28] "AppScan - Application Security | IBM." https://www.ibm.com/security/application-security/appscan, 2018. Accessed Aug. 3, 2018.

[29] R. Siles and S. Bennetts, "OWASP Vulnerable Web Applications Directory Project." https://www.owasp.org/index.php/OWASP_Vulnerable_Web_Applications_Directory_Project, 2018. Accessed Aug. 1, 2018.

[30] A. Doupe, "WackoPicko Vulnerable Website." https://github.com/adamdoupe/WackoPicko, 2018. Accessed Aug. 2, 2018.

[31] C. Willis, "OWASP Broken Web Applications Project." https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project, 2016. Accessed Aug. 3, 2018.

[32] A. Riancho, "WIVET - Web Input Vector Extractor Teaser." https://hub.docker.com/r/andresriancho/wivet/, 2015. Accessed Aug. 2, 2018.

[33] "The Web Application Vulnerability Scanner Evaluation Project." https://hub.docker.com/r/owaspvwad/wavsep/, 2016. Accessed Aug. 2, 2018.

[34] R. Khalil, "Thesis-Test-Results." https://github.com/rkhal101/Thesis-Test-Results, 2018.

[35] Y. Li, P. Han, C. Liu, and B. Fang, "Automatically Crawling Dynamic Web Applications via Proxy-Based Javascript Injection and Runtime Analysis," in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, pp. 242–249, June 2018.

[36] D. Amalfitano, A. R. Fasolino, and P. Tramontana, "Techniques and Tools for Rich Internet Applications Testing," in *2010 12th IEEE International Symposium on Web Systems Evolution (WSE)*, pp. 63–72, Sept. 2010.

[37] G. C. P. Suganthan, "Ajax Crawler," in *2012 International Conference on Data Science Engineering (ICDSE)*, pp. 27–30, July 2012.

[38] S. Choudhary, M. E. Dincturk, G. V. Bochmann, G. Jourdan, I. V. Onut, and P. Ionescu, "Solving Some Modeling Challenges when Testing Rich Internet Applications for Security," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pp. 850–857, April 2012.

[39] S. M. Mirtaheri, D. Zou, G. V. Bochmann, G. Jourdan, and I. V. Onut, "Dist-RIA Crawler: A Distributed Crawler for Rich Internet Applications," in *2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pp. 105–112, Oct. 2013.

[40] K. Ben Hafaiedh, G. Bochmann, G.-V. Jourdan, and I.-V. Onut, *A Scalable P2P RIA Crawling System with Partial Knowledge*, pp. 185–199. Jan. 2014.

[41] S. M. Mirtaheri, G. V. Bochmann, G.-V. Jourdan, and I. V. Onut, "PDist-RIA Crawler: A Peer-to-Peer Distributed Crawler for Rich Internet Applications," in *Web Information Systems Engineering – WISE 2014* (B. Benatallah, A. Bestavros, Y. Manolopoulos, A. Vakali, and Y. Zhang, eds.), (Cham), pp. 365–380, Springer International Publishing, 2014.

[42] "Crawler Coverage and Vulnerability Detection." `http://www.arachni-scanner.com/features/framework/crawl-coverage-vulnerability-detection/`. Accessed Aug. 4, 2018.

[43] R. Khalil, "Arachni does not maintain session across scan #986." `https://github.com/Arachni/arachni/issues/986`, 2018. Accessed Aug. 4, 2018.

[44] R. Khalil, “Sitemap does not contain all crawled links #987.” https://github.com/Arachni/arachni/issues/987, 2018. Accessed Aug. 4, 2018.

[45] R. Khalil, “Run Vega on WIVET #157.” https://github.com/subgraph/Vega/issues/157, 2018. Accessed Aug. 4, 2018.

[46] R. Khalil, “"Alerts for this node" does not display high alerts of risk type High #4899.” https://github.com/zaproxy/zaproxy/issues/4899, 2018. Accessed Aug. 4, 2018.

# Appendix A

# Default Scan

This section contains detailed instructions of the steps performed to run a default scan using each scanner against the chosen applications.

## A.1 WIVET Configuration

The WIVET application was run on 127.0.0.1 (localhost) port 8090 for all the scanners, with the exception of Arachni (that does not allow loopback interfaces), where the application was run on 192.168.X.X port 8090.

## Arachni

The following script was used to run a default scan on the WIVET application.

```
1 #!bin
2 ./arachni http://192.168.0.18:8090/ --checks trainer --audit-links
     --audit-forms \
3 --scope-include-pattern 'http://192.168.0.18:8090/' \
4 --scope-exclude-pattern 'http://192.168.0.18:8090/offscanpages.*' \
5 --scope-exclude-pattern 'http://192.168.0.18:8090/logout.php' \
```

```
6 --scope-exclude-pattern 'http://192.168.0.18:8090/pages/100.php' \
7 --http-cookie-string="PHPSESSID=77d4ad6bbe505bba989152390e4e9e25" \
8 --report-save-path=wivet-arachni-report.afr
```

To convert the file format from .afr to zip, use the following command:

```
1 ./arachni_reporter wivet-arachni-report.afr
    --reporter=html:outfile=wivet-arachni-report.html.zip
```

## Burp Suite Professional

The following steps were performed on the GUI of Burp to run a default scan on the WIVET application:

1. Configure Burp to work with your browser.
2. Navigate to the `Proxy` tab and turn the intercept off by clicking the `Intercept is on` button.
3. Load the WIVET application on the configured browser so that the website URL appears under the `Site map` sub-tab of the `Target` tab.
4. In the `Scope` sub-tab under the `Target` tab, include the website URL in the scope by clicking on the `Add` button under the `Include in scope` section. Then add the URL: `http://127.0.0.1:8090/`.
5. Under the `Exclude from scope` section add the following URLs using the `Add` button: `http://127.0.0.1:8090/offscanpages.*`, `http://127.0.0.1:8090/logout.php`, `http://127.0.0.1:8090/100.php`.
6. Navigate to the `Site map` sub-tab and right click on the website URL $>$ select `Spider this host` $>$ click `No`.

7. Navigate to the `Spider` tab and click the `Spider is paused` button under the `Control` sub-tab.

## Skipfish

The following script was used to run a default scan on the WIVET application.

```
1 skipfish -o /root/Desktop/skipfishoutput -N -C
    PHPSESSID=628e4286a3e8b1fd2cc1296a97718c9d -X logout.php -X
    /offscanpages/ -X 100.php http://127.0.0.1:8090/
```

## Vega

The following steps were performed on the GUI of Vega to run a default scan on the WIVET application:

1. Click on the `Scan` option on the top menu in Vega > select `Start New Scan`.
2. Select the option `Choose a target scope for scan` and click on the `Edit Scopes` button.
3. Under the `Base Path` option add the URL `http://127.0.0.1:8090/`. Under the `Exclude (URL or pattern)` option add the following URLs: `http://127.0.0.1:8090/offscanpages.*`, `http://127.0.0.1:8090/logout.php` and `http://127.0.0.1:8090/pages/100.php`.
4. Click `OK` and then click `Next`.
5. In the `Select Modules` window, keep the default selected modules with the exception that the option "`Bash Environment Variable Blind OS Injection`" under `Injection modules` should be deselected. After trial and error, we noticed that this

option starts a new session (even if a cookie is set in the `Authentication` option) and does not add to the crawling coverage on WIVET and therefore we decided to not include it.

6. Click `Next`.

7. In the `Authentication Options` window, add the session cookie under the `Set-Cookie or Set-Cookie2 value` option.

8. Click `Next` and then click `Finish`.

## Wapiti

The following script was used to run a default scan on the WIVET application.

```
1 #!bin
2 wapiti --flush-session -u http://127.0.0.1:8090/ \
3 -x http://127.0.0.1:8090/logout.php \
4 -x http://127.0.0.1:8090/offscanpages.* \
5 -x http://127.0.0.1:8090/pages/100.php
```

## ZAP

The following steps were performed on the GUI of ZAP to run a default scan on the WIVET application.

1. Configure ZAP to work with your browser.

2. Load the WIVET application on the configured browser so that the website URL appears under the `Sites` menu of the ZAP GUI.

3. Include the website URL in the context by right clicking on the URL > select `Include in Context` > then select `Default Context`.

4. In the `Session Properties` window, click on the `Exclude from Context` option and add the following URLs using the `Add` button: `http://127.0.0.1/offscanpages.*`, `http://127.0.0.1/logout.php`, `http://127.0.0.1/100.php`.

5. Click `OK`.

6. Click on the + sign in the bottom tab of the ZAP GUI and select `HTTP Sessions`.

7. In the `HTTP Sessions` tab, right click on the listed session and select `Set as Active`. This ensures that the current session is maintained.

8. Under the `Sites` tab, right click on the website URL > select `Attack` > select `Spider`.

9. In the `Spider` window, make sure that the `Context` is set to `Default Context`, then click `Start Scan`.

10. Under the `Sites` tab, right click on the website URL > select `Attack` > select `AJAX Spider`. In the `AJAX Spider` window, make sure that the `Context` is set to `Default Context`, then click `Start Scan`.

## A.2 WAVSEP Configuration

The WAVSEP application was run on 127.0.0.1 (localhost) port 8090 for all the scanners, with the exception of Arachni (that does not allow loopback interfaces), where the application was run on 192.168.X.X port 8090.

## Arachni

The following script was used to run a default scan on the WAVSEP application.

```
1 #!bin
2 ./arachni http://192.168.0.18:8090/wavsep/ \
3 --scope-include-pattern 'http://192.168.0.18:8090/wavsep' \
4 --scope-extend-paths 'paths.txt' \
5 --report-save-path=wavsep-arachni-report.afr
```

The content of the paths.txt file is as follows.

```
1 http://192.168.0.18:8090/wavsep/index-active.jsp
2 http://192.168.0.18:8090/wavsep/index-passive.jsp
```

To convert the file format from .afr to zip, use the following command:

```
1 ./arachni_reporter wavsep-arachni-report.afr
     --reporter=html:outfile=wavsep-arachni-report.html.zip
```

## Burp Suite Professional

The following steps were performed on the GUI of Burp to run a default scan on the WAVSEP application.

1. Configure Burp to work with your browser.
2. Navigate to the `Proxy` tab and turn the intercept off by clicking the `Intercept is on` button.
3. Load the WAVSEP application on the configured browser so that the website URL appears under the `Site map` sub-tab of the `Target` tab. Similarly, load the URLs for the `index-active` and `index-passive` pages.

4. In the Scope sub-tab under the Target tab, include the website URL in the scope by clicking on the Add button under the Include in scope section. Then add the URL: http://127.0.0.1:8090/wavsep.

5. Navigate to the Site map sub-tab and right click on the website WAVSEP root URL > select Spider this host > click No.

6. Navigate to the Spider tab and click the Spider is paused button under the Control sub-tab.

7. Navigate back to the Site map sub-tab and right click on the website WAVSEP root URL > then select Actively scan this host.

## Skipfish

The following script was used to run a default scan on the WAVSEP application.

```
1 #!bin
2 skipfish -o /root/Desktop/skipfishwavsepoutput -I
    http://127.0.0.1:8090/wavsep
    http://127.0.0.1:8090/wavsep/index-passive.jsp
    http://127.0.0.1:8090/wavsep/index-active.jsp
```

## Vega

The following steps were performed on the GUI of Vega to run a default scan on the WAVSEP application.

1. Click on Scan on the top menu in Vega > select Start New Scan.

2. Select the option `Choose a target scope for scan` and click on the `Edit Scopes` button.

3. Under the `Base Path` option add the URLs `http://127.0.0.1:8090/wavsep/`, `http://127.0.0.1:8090/wavsep/index-passive.jsp` and `http://127.0.0.1:8090/wavsep/index-active.jsp`.

4. Click `OK` and then click `Next`.

5. In the `Select Modules` window, keep the default selected modules.

6. Click `Next`.

7. Click `Next` and then click `Finish`.

## Wapiti

The following script was used to run a default scan on the WAVSEP application.

```
1 #!bin
2 wapiti --flush-session -u http://127.0.0.1:8090/wavsep/ \
3 --start "http://127.0.0.1:8090/wavsep/index-active.jsp" \
4 --start "http://127.0.0.1:8090/wavsep/index-passive.jsp" \
5 --scope "domain" \
6 --output "wapiti-wavsep-result-official" -f txt
```

## ZAP

The following steps were performed on the GUI of ZAP to run a default scan on the WAVSEP application.

1. Configure ZAP to work with your browser.

2. Load the WAVSEP application on the configured browser so that the website URL appears under the `Sites` menu of the ZAP GUI. Similarly, load the URLs for the `index-active` and `index-passive` pages.

3. Include the root website URL in the `Context` by right clicking on the root URL > select `Include in Context` > then select `Default Context`.

4. Under the `Sites` tab, right click on the `index-passive` URL > select `Attack` > select `Spider`. In the `Spider` window, make sure that the `Context` is set to `Default Context`, then click `Start Scan`.

5. Repeat step 4 for the `index-active` URL.

6. Under the `Sites` tab, right click on the `index-passive` URL > select `Attack` > select `AJAX Spider`. In the `AJAX Spider` window, make sure that the `Context` is set to `Default Context`, then click `Start Scan`.

7. Repeat step 6 for the `index-active` URL.

8. Under the `Sites` tab, right click on the WAVSEP root URL > select `Attack` > select `Active Scan`. Click on `Start Scan`.

## A.3 WackoPicko Configuration

The WackoPicko application was run using the OWASP BWA Project virtual machine on 192.168.X.X.

## Arachni

The following script was used to run a default INITIAL scan on the WackoPicko application.

```
1 #!bin
2 ./arachni http://192.168.0.26/WackoPicko/ \
3 --scope-include-pattern 'http://192.168.0.26/WackoPicko/' \
4 --scope-redundant-path-pattern='calendar.php:1' \
5 --report-save-path=wackopicko-default--initial-arachni-report.afr
```

To convert the file format from .afr to zip, use the following command:

```
1 ./arachni_reporter wackopicko-default--initial-arachni-report.afr
    --reporter=html:outfile=wackopicko-default-initial-arachni-report.html.zip
```

The following script was used to run a default CONFIG scan on the WackoPicko application.

```
1 #!bin
2 ./arachni --http-authentication-username "bryce"
    --http-authentication-password "bryce" http://192.168.0.26/WackoPicko/ \
3 --scope-include-pattern 'http://192.168.0.26/WackoPicko/' \
4 --scope-include-pattern 'http://192.168.0.26/WackoPicko/' \
5 --scope-redundant-path-pattern='calendar.php:1' \
6 --scope-exclude-pattern 'http://192.168.0.26/WackoPicko/users/logout.php' \
7 --report-save-path=wackopicko-default-config-arachni-report.afr
```

## Burp Suite Professional

The following steps were performed on the GUI of Burp to run a default INITIAL scan on the WackoPicko application:

1. Configure Burp to work with your browser.

2. Navigate to the Proxy tab and turn the intercept off by clicking the Intercept is on button.

3. Load the WackoPicko application on the configured browser so that the website URL appears under the Site map sub-tab of the Target tab.

4. In the Scope sub-tab under the Target tab, include the website URL in the scope by clicking on the Add button under the Include in scope section. Then add the URL: http://192.168.0.26/WackoPicko/.

5. Navigate to the Spider tab and click on the Options sub-tab. Scroll down to Application Logic and select the Handle as ordinary forms option.

6. Navigate to the Site map sub-tab of the Target tab and right click on the website URL > select Spider this host > click No.

7. Navigate to the Spider tab and click the Spider is paused button under the Control sub-tab.

8. Navigate back to the Site map sub-tab of the Target tab and right click on the website URL > then select Actively scan this branch.

To run a default scan on the WackoPicko application in CONFIG mode, follow the same steps listed for INITIAL mode with the exception of adding a user before scanning the application and removing the logout link from the context. This can be done as follows:

1. In the Scope sub-tab under the Target tab, exclude the website URL from the scope by clicking on the Add button under the Exclude from scope section. Then add the URL: http://192.168.0.26/WackoPicko/users/logout.php.

2. Navigate to the Options sub-tab of the Spider tab and add the username/password bryce/bryce under Application Login.

## SkipFish

The following script was used to run a default INITIAL scan on the WackoPicko application.

```
1 skipfish -o /root/Desktop/skipfish-wackopicko-default-initial-output -I
    http://192.168.0.26/WackoPicko/ http://192.168.0.26/WackoPicko/
```

The following script was used to run a default CONFIG scan on the WackoPicko application.

```
1 skipfish -o /root/Desktop/skipfish-wackopicko-default-initial-config -I
    http://192.168.0.26/WackoPicko/ -A bryce:bryce -X
    http://192.168.0.26/WackoPicko/users/logout.php
    http://192.168.0.26/WackoPicko/
```

## Vega

The following steps were used to run a default scan on the WackoPicko application in INITIAL mode :

1. Click on `Scan` on the top menu in Vega > select `Start New Scan`.
2. Select the option `Choose a target scope for scan` and click on the `Edit Scopes` button.
3. Under the `Base Path` option add the URL `http://192.168.0.26/WackoPicko/`.
4. Click `OK` and then click `Next`.
5. In the `Select Modules` window, keep the default selected modules > click `Next`.
6. Click `Next` and then click `Finish`.

To run a default scan on the WackoPicko application in CONFIG mode, follow the same steps listed for INITIAL mode with the exception of adding a user before scanning the application and removing the logout link from the context. This can be done as follows:

1. Configure your browser to work with the Vega proxy. Navigate to the `Proxy` tab and click on the `Start HTTP Proxy` button.

2. Enter the username/password information bryce/bryce on the Login page. You should now be able to see the POST login request in the `Requests` sub-tab.

3. Navigate to the `Scanner` tab > click on the `Create Identity` icon.

4. In the `Create an Identity` window add the name bryce and choose `macro` as the `Authentication type` > then click on `Create macro`.

5. In the `Macro Editor` set the `Macro Name` to bryce and click on the `add item` button > select the POST login request and click `OK`.

6. Click `OK` > click `Finish`.

7. When running the scan using steps listed for the INITIAL mode, make sure to exclude the logout link from the Target Scope and set the identity as `bryce` in the `Authentication Options` window.

## Wapiti

The following script was used to run a default scan on the WackoPicko application in INITIAL mode.

```
1 #!bin
2 wapiti --flush-session -u http://192.168.0.26/WackoPicko/ \
3 --scope "folder" \
4 --output "wapiti-wackopicko-default-initial-result.txt" -f txt
```

The following steps were used to run a default scan on the WackoPicko application in CONFIG mode.

– First, use `wapiti-getcookie` to login to the restricted area and save the cookie in `cookies.json`.

```
1    wapiti-getcookie -u http://192.168.0.26/WackoPicko/users/login.php
        -c cookies.json -d "username=bryce&password=bryce"
```

– Then, run the following script.

```
1 #!bin
2 wapiti --flush-session -u http://192.168.0.26/WackoPicko/ \
3 --scope "folder" \
4 -x http://192.168.0.26/WackoPicko/logout.php -c cookies.json \
5 --output "wapiti-wackopicko-default-config-result.txt" -f txt
```

# ZAP

The following steps were performed on the GUI of ZAP to run a default scan on the WackoPicko application in INITIAL mode :

1. Configure ZAP to work with your browser.
2. Load the WackoPicko application on the configured browser so that the website URL appears under the `Sites` menu of the ZAP GUI.
3. Include the website URL in the context by right clicking on the URL $>$ select `Include in Context` $>$ then select `Default Context`.
4. Click `OK`.

5. Under the Sites tab, right click on the website URL > select `Attack` > select `Spider`.

6. In the `Spider` window, make sure that the `Context` is set to `Default Context`, then click `Start Scan`.

7. Under the `Sites` tab, right click on the website URL > select `Attack` > select `AJAX Spider`.

8. Under the `Sites` tab, right click on the website URL > select `Attack` > select `Active Scan`.

To run a default scan on the WackoPicko application in CONFIG mode, follow the same steps listed for INITIAL mode with the exception of adding a user before scanning the application and removing the logout link from the context. This can be done as follows:

1. Enter the username/password information bryce/bryce on the Login page.

2. Under the `Sites` tab, right click the `POST` request for authentication and select `Flag as Context` > select `Default Context: Form-based Auth Login Request`.

3. Click `OK`.

4. In the `Session Properties` window, select `Exclude from Context` > click on the `Add` button > add the logout link `http://192.168.0.26/WackoPicko/users/logout.php`.

5. When you run the spider / ajax spider / scan, ensure that the user "bryce" is selected in the scope.

# Appendix B

# Configured Scan

This section contains detailed instructions of the steps performed to run a configured scan using each scanner against the chosen applications.

## B.1 WIVET Configuration

We were not able to find configuration settings for the scanners Arachni, Burp Suite Pro, Skipfish, Vega and Wapiti that increased the crawling coverage achieved by the Default scan. Therefore, this section only includes the steps performed to run a configured scan on the WIVET application for the ZAP scanner. The WIVET application was run on 127.0.0.1 (localhost) port 8090.

### ZAP

The following steps were performed on the GUI of ZAP to run a configured scan on the WIVET application.

1. Configure ZAP to work with your browser.

2. Load the WIVET application on the configured browser so that the website URL appears under the `Sites` menu of the ZAP GUI.

3. Include the website URL in the context by right clicking on the URL > select `Include in Context` > then select `Default Context`.

4. In the `Session Properties` window, click on the `Exclude from Context` option and add the following URLs using the `Add` button: `http://127.0.0.1/offscanpages.*`, `http://127.0.0.1/logout.php`, `http://127.0.0.1/100.php`.

5. Click `OK`.

6. Click on the + sign in the bottom tab of the ZAP GUI and select `HTTP Sessions`.

7. In the `HTTP Sessions` tab, right click on the listed session and select `Set as Active`. This ensures that the current session is maintained.

8. Click on the `options` icon > select `AJAX Spider` > increase `Maximum duration time` to 120 and deselect `Click default elements only (a, button, input)` > click `OK`.

9. Under the `Sites` tab, right click on the website URL > select `Attack` > select `Spider`. In the `Spider` window, make sure that the `Context` is set to `Default Context`, then click `Start Scan`.

10. Under the `Sites` tab, right click on the website URL > select `Attack` > select `AJAX Spider`. In the `AJAX Spider` window, make sure that the `Context` is set to `Default Context`, then click `Start Scan`.

## B.2 WAVSEP Configuration

The WAVSEP application was run on 127.0.0.1 (localhost) port 8090 for all the scanners, with the exception of Arachni (that does not allow loopback interfaces), where the application was run on 192.168.X.X port 8090.

### Arachni

By default, Arachni will load all checks, the plugins under `/plugins/defaults` and audit all links, forms and cookies. Therefore, we cannot further configure Arachni to achieve a better score.

### Burp Suite Professional

The following steps were performed on the GUI of Burp to run a configured scan on the WAVSEP application.

1. Configure Burp to work with your browser.
2. Navigate to the `Proxy` tab and turn the intercept off by clicking the `Intercept is on` button.
3. Load the WAVSEP application on the configured browser so that the website URL appears under the `Site map` sub-tab of the `Target` tab. Similarly, load the URLs for the `index-active` and `index-passive` pages.
4. In the `Scope` sub-tab under the `Target` tab, include the website URL in the scope by clicking on the `Add` button under the `Include in scope` section. Then add the URL: `http://127.0.0.1:8090/wavsep`.

5. Navigate to the Options sub-tab of the Scanner tab. In the Active Scanning Optimization section, choose the Thorough option for Scan speed and deselect the option Use intelligent attack selection.

6. Navigate to the Options sub-tab of the Spider tab. In the Application Logic section, choose the option Handle as ordinary forms. In the Spider Engine section, set the Number of threads option to 1.

7. Navigate to the Site map sub-tab of the Target tab and right click on the website root URL > select Spider this host > click No.

8. Navigate to the Spider tab and click the Spider is paused button under the Control sub-tab.

9. Navigate back to the Site map sub-tab and right click on the website root URL > then select Actively scan this host.

## Skipfish

The following script was used to run a configured scan on the WAVSEP application.

```
1 skipfish -o /folder/location -I http://localhost:8090/wavsep/[directory]/
    http://localhost:8090/wavsep/[directory]/file-name.jsp
```

## Vega

The following steps were performed on the GUI of Vega to run a configured scan on the WAVSEP application.

1. Click on Scan on the top menu in Vega > select Start New Scan.

2. Select the option `Choose a target scope for scan` and click on the `Edit Scopes` button.
3. Under the `Base Path` option add the URLs `http://127.0.0.1:8090/wavsep/`, `http://127.0.0.1:8090/wavsep/index-passive.jsp` and `http://127.0.0.1:8090/wavsep/index-active.jsp`.
4. Click `OK` and then click `Next`.
5. In the `Select Modules` window, select all the modules listed.
6. Click `Next`.
7. Click `Next` and then click `Finish`.

## Wapiti

The following script was used to run a configured scan on the WAVSEP application.

```
1 #!bin
2 wapiti --flush-session -u http://127.0.0.1:8090/wavsep/ \
3 --start "http://127.0.0.1:8090/wavsep/index-active.jsp" \
4 --start "http://127.0.0.1:8090/wavsep/index-passive.jsp" \
5 --scope "domain" \
6 --module "backup,blindsql,buster,crlf,delay,exec,file,htaccess,
7    methods,nikto,permanentxss,shellshock,sql,ssrf,xss" \
8 --output "wapiti-wavsep-configured" -f txt
```

## ZAP

The following steps were performed on the GUI of ZAP to run a configured scan on the WAVSEP application:

1. Configure ZAP to work with your browser.

2. Load the WAVSEP application on the configured browser so that the website URL appears under the `Sites` menu of the ZAP GUI. Similarly, load the URLs for the `index-active` and `index-passive` pages.

3. Include the website URLs in the `Context` by right clicking on each of the three website URLs > select `Include in Context` > then select `Default Context`.

4. Click on the `options` icon > select `Ajax Spider` > change the `Maximum crawl depth` to 20, `Maximum duration` to 700, and deselect `Click default elements only`.

5. Similarly, in the `Options` window, select `Spider` > change the `Maximum depth to crawl` to 19.

6. Click OK.

7. Under the `Sites` tab, right click on the `index-passive` URL > select `Attack` > select `Spider`. In the `Spider` window, make sure that the `Context` is set to `Default Context`, then click `Start Scan`.

8. Repeat step 7 for the `index-active` URL.

9. Under the `Sites` tab, right click on the `index-passive` URL > select `Attack` > select `AJAX Spider`. In the `AJAX Spider` window, make sure that the `Context` is set to `Default Context`, then click `Start Scan`.

10. Repeat step 9 for the `index-active` URL.

11. Under the `Sites` tab, right click on the root URL > select `Attack` > select `Active Scan`. Navigate to the `Policy` tab and change `Default Attack Strength` to `Insane`. Then click on `Start Scan`.

## B.3 WackoPicko Configuration

The WackoPicko application was run using the OWASP BWA Project virtual machine on 192.168.X.X.

### Arachni

By default, Arachni will load all checks, the plugins under `/plugins/defaults` and audit all links, forms and cookies. Therefore, we cannot further configure Arachni to achieve a better score.

### Burp

Burp does not offer any extra configuration options that will achieve a better score. Therefore, to run a configured INITIAL scan on the WackoPicko application visit every page in the application that does not require authentication using the configured browser, then apply the steps listed in Appendix A.3. Similarly, to run a configured CONFIG scan on the WackoPicko application visit every page in the application using the configured browser, then apply the steps listed in Appendix A.3.

### SkipFish

We were not able to configure the scanner to achieve a higher score and since SkipFish does not have a proxy component we will regard the results for the default scan as the results for the configured scan.

# Vega

The following was used to run a configured scan on the WackoPicko application in INITIAL mode :

1. Configure your browser to work with the Vega proxy. Navigate to the `Proxy` tab and click on the `Start HTTP Proxy` button.

2. In the configured browser, visit all the pages that do not require authentication.

3. Click on `Scan` on the top menu in Vega > select `Start New Scan`.

4. Select the option `Choose a target scope for scan` and click on the `Edit Scopes` button.

5. Under the `Base Path` option add the URL `http://192.168.0.26/WackoPicko/`.

6. Click `OK` and then click `Next`.

7. In the `Select Modules` window, select all the available modules > click `Next`.

8. Click `Next` > deselect the option `Exclude listed parameters from scan` and then click `Finish`.

To run a configured scan on the WackoPicko application in CONFIG mode, follow the same steps listed for INITIAL mode with the exception of adding a user before scanning the application and removing the logout link from the context. This can be done as follows:

1. Enter the username/password information bryce/bryce on the Login page. You should now be able to see the POST login request in the `Requests` sub-tab.

2. Navigate to the `Scanner` tab > click on the `Create Identity` icon.

3. In the `Create an Identity` window add the name bryce and choose `macro` as the `Authentication type` > then click on `Create macro`.

4. In the Macro Editor set the Macro Name to bryce and click on the add item button > select the POST login request and click OK.

5. Click OK > click Finish.

6. When running the scan using steps listed for the INITIAL mode, make sure to exclude the logout link from the Target Scope and set the identity as bryce in the Authentication Options window.

## Wapiti

The following script was used to run a configured scan on the WackoPicko application in INITIAL mode.

```
1 #!bin
2 wapiti --flush-session -u http://192.168.0.26/WackoPicko/ \
3 --scope "folder" \
4 --module
    "backup,blindsql,buster,crlf,delay,exec,file,htaccess,methods,nikto,
5       permanentxss,shellshock,sql,ssrf,xss" \
6 --output "wapiti-wackopicko-configured-initial-result.txt" -f txt
```

The following steps were used to run a configured scan on the WackoPicko application in CONFIG mode.

– First, use wapiti-getcookie to login to the restricted area and save the cookie in cookies.json.

```
1    wapiti-getcookie -u http://192.168.0.26/WackoPicko/users/login.php
        -c cookies.json -d "username=bryce&password=bryce"
```

– Then run the following script.

```
#!bin
wapiti --flush-session -u http://192.168.0.26/WackoPicko/ \
--scope "folder" \
--module
    "backup,blindsql,buster,crlf,delay,exec,file,htaccess,methods,nikto,
         permanentxss,shellshock,sql,ssrf,xss" \
-x http://192.168.0.26/WackoPicko/logout.php -c cookies.json \
--output "wapiti-wackopicko-configured-config-result.txt" -f txt
```

# ZAP

The following steps were performed on the GUI of ZAP to run a configured scan on the WackoPicko application in INITIAL mode :

1. Configure ZAP to work with your browser.
2. Load the WackoPicko application on the configured browser so that the website URL appears under the `Sites` menu of the ZAP GUI.
3. Include the website URL in the context by right clicking on the URL > select `Include in Context` > then select `Default Context`.
4. Click `OK`.
5. Visit every page that does not require authentication in the application on the configured browser.
6. Click on the `Options` icon > select AJAX Spider > deselect `Click default elements only (a, button, input)` > click `OK`.

7. Under the Sites tab, right click on the website URL > select `Attack` > select `Spider`.
8. In the `Spider` window, make sure that the `Context` is set to `Default Context`, then click `Start Scan`.
9. Under the `Sites` tab, right click on the website URL > select `Attack` > select `AJAX Spider`.
10. Under the `Sites` tab, right click on the website URL > select `Attack` > select `Active Scan`.
11. In the `Active Scan` window, select `Show advanced options`. Navigate to the `Policy` tab and change the `Defaut Attack Strength` to `Insane`.
12. Click `Start Scan`.

To run a configured scan on the WackoPicko application in CONFIG mode, follow the same steps listed for INITIAL mode with the exception of adding a user before scanning the application, removing the logout link from the context and proxying the pages in the application that require authentication. This can be done as follows:

1. Enter the username/password information bryce/bryce on the Login page. Then visit all the pages in the application.
2. Under the `Sites` tab, right click the `POST` request for authentication and select `Flag as Context` > select `Default Context: Form-based Auth Login Request`.
3. Click `OK`.
4. In the `Session Properties` window, select `Exclude from Context` > click on the `Add` button > add the logout link `http://192.168.0.26/WackoPicko/users/logout.php`.
5. When you run the spider / ajax spider / scan, ensure that the user "bryce" is selected in the scope.

# Appendix C

# Sample Detailed Results

For transparency purposes and to maximize the benefit of this work to the security community, we have added a GitHub repository [34] that includes the detailed results we achieved for each scanner. The repository also includes the database file / session snapshot / report for each scan so that the interested user can independently verify the results we obtained. In this section, we provide a sample of the results achieved from running the ZAP scanner on the chosen applications in order to give the reader an illustration of the structure and content of the compiled results.

## C.1 Sample Detailed WIVET Results

For each scanner, we have included an excel document that contains the numbers and names of the 56 WIVET test cases. A test case is colored green if it was detected by the scanner, otherwise, the test case is left uncolored. In addition, the number and overall percentage of passed test cases is included in the excel document as can be seen in Figure C.1.

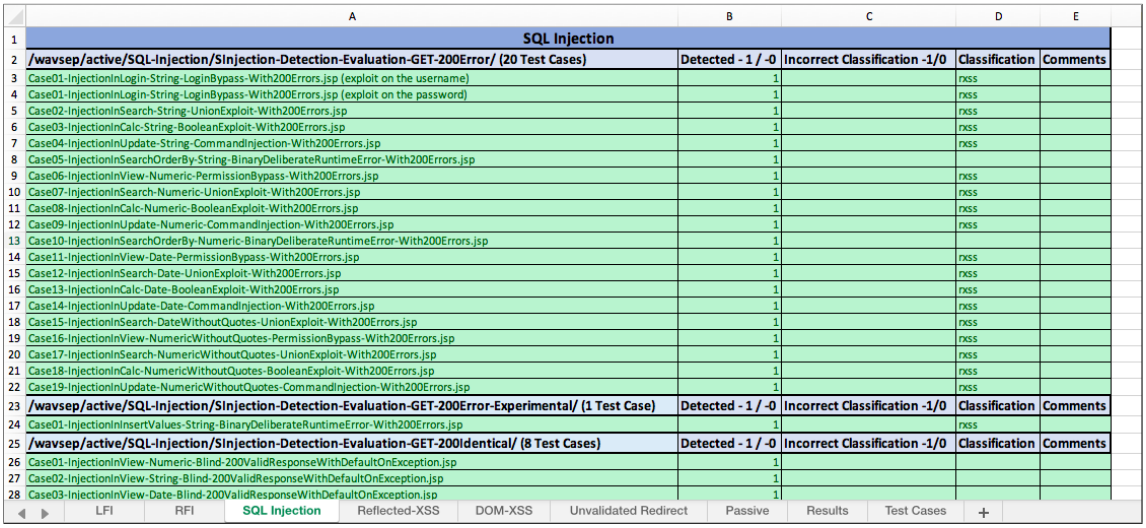
| A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|
| 1_12c3b | link creation after some time w/ setTimeout | | WIVET Score: 44/56*100 = 78% | | | |
| 1_25e2a | link creation after button click | | | | | |
| 2_1f84b | self referencing link | | | | | |
| 2_2b7a3 | self referencing link with random query string | | | | | |
| 3_45589 | multi-page form with a single path to final destination | | | | | |
| 4_1c3f8 | link href js protocol | | | | | |
| 5_1e4d2 | div onmouseover window.open | | | | | |
| 6_14b3c | form submit thru select onchange w/ simple alert | | | | | |
| 7_16a9c | form submit button onclick | | | | | |
| 8_1b6e1 | link in html comment | | | | | |
| 8_2b6f1 | relative link in html comment | | | | | |
| 9_1a1b2 | span onclick window.location | | | | | |
| 9_2ff21 | span onmouseout window.location.href | | | | | |
| 9_3a2b7 | span onmousedown document.location.href | | | | | |
| 9_4b82d | span onmouseup document.location | | | | | |
| 9_5ee31 | p onclick window.location.href | | | | | |
| 9_6ee31 | p onmouseout window.location.href | | | | | |
| 9_7ee31 | p onmousedown window.location.href | | | | | |
| 9_8ee31 | p onmouseup window.location.href | | | | | |
| 9_9ee31 | div onclick window.location.href | | | | | |
| 9_10ee31 | div onmouseout window.location.href | | | | | |
| 9_11ee31 | div onmousedown window.location.href | | | | | |
| 9_12ee31 | div onmouseup window.location.href | | | | | |
| 9_13ee31 | td onclick window.location.href | | | | | |
| 9_14ee31 | td onmouseout window.location.href | | | | | |

Figure C.1: Sample ZAP Detailed WIVET Results Layout

An excel document is included for each of the default and configured scans on the WIVET application.

## C.2 Sample Detailed WAVSEP Results

For each scanner, we have included an excel document that contains the list of test cases found in the WAVSEP application. The document is split into several tabs where each tab is dedicated to a vulnerability category. A test case is either colored green if it was detected by the scanner, colored yellow if it was detected by the scanner but classified in the wrong category, or left uncolored if it was not detected. In Figure C.2, all the test cases shown have been detected by the ZAP scanner and therefore are colored green. Several of the test cases displayed have also been classified incorrectly (in addition to being detected with the correct classification) and therefore, the incorrect classification is added under the "Classification" column.

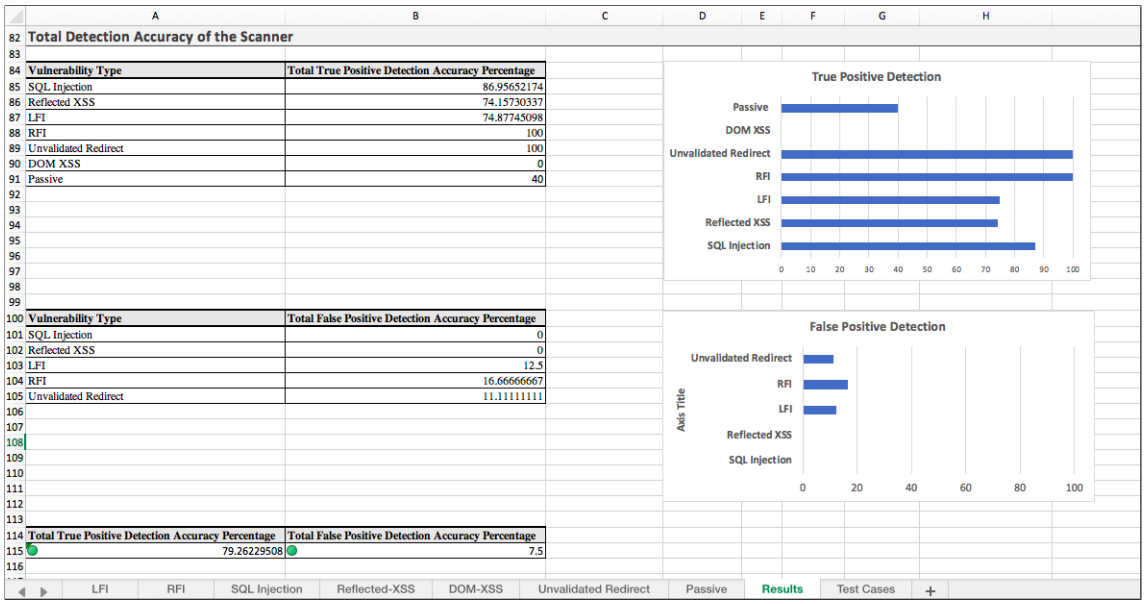


| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | **SQL Injection** | | | | |
| 2 | **/wavsep/active/SQL-Injection/SInjection-Detection-Evaluation-GET-200Error/ (20 Test Cases)** | **Detected - 1 / -0** | **Incorrect Classification -1/0** | **Classification** | **Comments** |
| 3 | Case01-InjectionInLogin-String-LoginBypass-With200Errors.jsp (exploit on the username) | 1 | | rxss | |
| 4 | Case01-InjectionInLogin-String-LoginBypass-With200Errors.jsp (exploit on the password) | 1 | | rxss | |
| 5 | Case02-InjectionInSearch-String-UnionExploit-With200Errors.jsp | 1 | | rxss | |
| 6 | Case03-InjectionInCalc-String-BooleanExploit-With200Errors.jsp | 1 | | rxss | |
| 7 | Case04-InjectionInUpdate-String-CommandInjection-With200Errors.jsp | 1 | | rxss | |
| 8 | Case05-InjectionInSearchOrderBy-String-BinaryDeliberateRuntimeError-With200Errors.jsp | 1 | | | |
| 9 | Case06-InjectionInView-Numeric-PermissionBypass-With200Errors.jsp | 1 | | rxss | |
| 10 | Case07-InjectionInSearch-Numeric-UnionExploit-With200Errors.jsp | 1 | | rxss | |
| 11 | Case08-InjectionInCalc-Numeric-BooleanExploit-With200Errors.jsp | 1 | | rxss | |
| 12 | Case09-InjectionInUpdate-Numeric-CommandInjection-With200Errors.jsp | 1 | | rxss | |
| 13 | Case10-InjectionInSearchOrderBy-Numeric-BinaryDeliberateRuntimeError-With200Errors.jsp | 1 | | | |
| 14 | Case11-InjectionInView-Date-PermissionBypass-With200Errors.jsp | 1 | | rxss | |
| 15 | Case12-InjectionInSearch-Date-UnionExploit-With200Errors.jsp | 1 | | rxss | |
| 16 | Case13-InjectionInCalc-Date-BooleanExploit-With200Errors.jsp | 1 | | rxss | |
| 17 | Case14-InjectionInUpdate-Date-CommandInjection-With200Errors.jsp | 1 | | rxss | |
| 18 | Case15-InjectionInSearch-DateWithoutQuotes-UnionExploit-With200Errors.jsp | 1 | | rxss | |
| 19 | Case16-InjectionInView-NumericWithoutQuotes-PermissionBypass-With200Errors.jsp | 1 | | rxss | |
| 20 | Case17-InjectionInSearch-NumericWithoutQuotes-UnionExploit-With200Errors.jsp | 1 | | rxss | |
| 21 | Case18-InjectionInCalc-NumericWithoutQuotes-BooleanExploit-With200Errors.jsp | 1 | | rxss | |
| 22 | Case19-InjectionInUpdate-NumericWithoutQuotes-CommandInjection-With200Errors.jsp | 1 | | rxss | |
| 23 | **/wavsep/active/SQL-Injection/SInjection-Detection-Evaluation-GET-200Error-Experimental/ (1 Test Case)** | **Detected - 1 / -0** | **Incorrect Classification -1/0** | **Classification** | **Comments** |
| 24 | Case01-InjectionInInsertValues-String-BinaryDeliberateRuntimeError-With200Errors.jsp | 1 | | rxss | |
| 25 | **/wavsep/active/SQL-Injection/SInjection-Detection-Evaluation-GET-200Identical/ (8 Test Cases)** | **Detected - 1 / -0** | **Incorrect Classification -1/0** | **Classification** | **Comments** |
| 26 | Case01-InjectionInView-Numeric-Blind-200ValidResponseWithDefaultOnException.jsp | 1 | | | |
| 27 | Case02-InjectionInView-String-Blind-200ValidResponseWithDefaultOnException.jsp | 1 | | | |
| 28 | Case03-InjectionInView-Date-Blind-200ValidResponseWithDefaultOnException.jsp | 1 | | | |

LFI | RFI | SQL Injection | Reflected-XSS | DOM-XSS | Unvalidated Redirect | Passive | Results | Test Cases

Figure C.2: Sample ZAP Detailed WAVSEP Results Layout - Part 1

The overall results of the scan are calculated and presented in the "Results" tab as shown in Figure C.3.



**Total Detection Accuracy of the Scanner**

| Vulnerability Type | Total True Positive Detection Accuracy Percentage |
|---|---|
| SQL Injection | 86.95652174 |
| Reflected XSS | 74.15730337 |
| LFI | 74.87745098 |
| RFI | 100 |
| Unvalidated Redirect | 100 |
| DOM XSS | 0 |
| Passive | 40 |



| Vulnerability Type | Total False Positive Detection Accuracy Percentage |
|---|---|
| SQL Injection | 0 |
| Reflected XSS | 0 |
| LFI | 12.5 |
| RFI | 16.66666667 |
| Unvalidated Redirect | 11.11111111 |



| Total True Positive Detection Accuracy Percentage | Total False Positive Detection Accuracy Percentage |
|---|---|
| 79.26229508 | 7.5 |

LFI | RFI | SQL Injection | Reflected-XSS | DOM-XSS | Unvalidated Redirect | Passive | Results | Test Cases

Figure C.3: Sample ZAP Detailed WAVSEP Results Layout - Part 2

An excel document is included for each of the default and configured scans on the WAVSEP application.

## C.3 Sample Detailed WackoPicko Results

For each scanner, we have included an excel document that contains a list of all the vulnerabilities found in the WackoPicko application. The "Results" tab contains the vulnerability detection results. The remaining tabs contain the list of false positives detected for each test case mode. This can be seen in Figure C.4 and Figure C.5.

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | | Default Scan | | Configured Scan | | |
| 2 | Vulnerability | INITIAL | CONFIG | INITIAL | CONFIG | |
| 3 | Reflected XSS | 1 | 1 | 1 | 1 | |
| 4 | Stored XSS | 1 | 1 | 1 | 1 | |
| 5 | Session ID | | | | | |
| 6 | Reflected SQL Injection | 1 | 1 | 1 | 1 | |
| 7 | Command Line Injection | 1 | 1 | 1 | 1 | |
| 8 | File Inclusion | 1 | 1 | 1 | 1 | |
| 9 | File Exposure | | | | | |
| 10 | Reflected XSS Behind JavaScript | 1 | 1 | 1 | 1 | |
| 11 | Parameter Manipulation | | | | | |
| 12 | Weak Password | | | | | |
| 13 | Stored SQL Injection | | | | | |
| 14 | Directory Traversal | | | | | |
| 15 | Multi-Step Stored XSS | | | | | |
| 16 | Forceful Browsing | | | | | |
| 17 | Logic Flaws - Coupon | | | | | |
| 18 | Reflected XSS behind Flash | | | | 1 | |
| 19 | Total Detection | 6 | 6 | 6 | 7 | |
| 20 | Total Detection Percentage | 37.5 | 37.5 | 37.5 | 43.75 | |
| 21 | | | | | | |
| 22 | | | | | | |
| 23 | | | | | | |

Results | FP-default-INITIAL | FP-default-CONFIG | FP-configured-INITIAL | FP-configured-CONFIG | +

Figure C.4: Sample ZAP Detailed WackoPicko Results Layout - Part 1

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Persistent XSS | POST: http://192.168.0.31/WackoPicko/guestbook.php | name field | | |
| 2 | Persistent XSS | GET: http://192.168.0.26/WackoPicko/pictures/recent.php | name | | |
| 3 | Persistent XSS | GET: http://192.168.0.26/WackoPicko/pictures/recent.php | tag | | |
| 4 | Reflected XSS | POST: http://192.168.0.26/WackoPicko/pictures/upload.php | name | | |
| 5 | Reflected XSS | POST: http://192.168.0.26/WackoPicko/pictures/upload.php | tag | | |
| 6 | SQL Injection | http://192.168.0.26/WackoPicko/calendar.php?query=query%25 | query | | |
| 7 | SQL Injection | GET: http://192.168.0.26/WackoPicko/guestbook.php?query=query+ASC++--+ | query | | |
| 8 | SQL Injection | GET: http://192.168.0.26/WackoPicko/pictures/view.php?picid=14%27+AND+%271%27%3D%271%27+--+ | picid | | |
| 9 | | | | | |
| 10 | | | | | |
| 11 | | | | | |
| 12 | | | | | |
| 13 | | | | | |
| 14 | | | | | |
| 15 | | | | | |
| 16 | | | | | |
| 17 | | | | | |
| 18 | | | | | |
| 19 | | | | | |
| 20 | | | | | |
| 21 | | | | | |
| 22 | | | | | |
| 23 | | | | | |

Results | FP-default-INITIAL | FP-default-CONFIG | FP-configured-INITIAL | FP-configured-CONFIG | +

Figure C.5: Sample ZAP Detailed WackoPicko Results Layout - Part 2

# Appendix D

# Scanner Issues Discovered

This section discusses the scanner bugs / issues we have found during our research.

**#1 Arachni** The Arachni scanner cannot maintain a single living session across a default scan. According to the documentation, the `-http-cookie-string` parameter forces the scanner to maintain a single session [42]. However, we observed that this is not true when running a default scan. In a default scan all forms, links and cookies are audited and therefore we hypothesize that the reason a single session is not maintained is because the cookie value is being attacked and changed during a scan to determine if it has any associated vulnerabilities. As a work-around, the user can specify the checks, audits and plugins that the scanner should use while avoiding the ones that change the cookie value. We reported this issue on the official Arachni GitHub page [43].

**#2 Arachni:** The report produced by the Arachni scanner has a `Sitemap` section that lists all the visited URLs. However, during our study we found that not all the URLs visited by the scanner are included in the `Sitemap`. We reported this issue on the official Arachni GitHub page [44].

**#3 Arachni:** Using the default settings, the Arachni scanner results in an infinite crawl / loop when presented with a set of URLs that always link to other URLs such as a

calendar page. This issue was not reported because it has already been brought to the attention of the Arachni developers. A work-around would be either to exclude the problematic link from the scope or to use the redundancy filters to limit the amount of times a URL is followed.

**#4 SkipFish:** The Skipfish scanner cannot maintain a single living session across a scan. According to the Skipfish man page, the `-C` parameter appends a custom cookie to all requests and the `-N` parameter informs the scanner to not accept any new cookies [24]. We used both parameters when testing the scanner on WIVET, however, the scanner was not able to maintain a single valid cookie and created multiple valid sessions during the scan run. We have informed the skipfish developers about this issue by email.

**#5 Vega:** The Vega scanner cannot maintain a single living session across a scan. During testing, we set the `Set-Cookie or Set-Cookie2` parameter to a valid session cookie, however, multiple sessions were still created. We reported this issue on the official Vega GitHub page [45].

**#6 ZAP:** The option `Alerts for this Node` in the `Sitemap` tree does not show high risk alerts. This error is trivial since the high risk alerts can still be seen in the `Alerts` tab. We reported this issue on the official ZAP GitHub page [46].