Evaluation of Web Vulnerability Scanners Based on OWASP Benchmark

Balume Mburano

Master of Research (ICT)

Supervisor: Dr. Weisheng Si

School of Computing, Engineering, and Mathematics
Western Sydney University
Sydney, Australia
June 2018

DECLARATION

The work contained in this thesis has not been previously submitted to meet requirements for an award at this or any other higher education institution. To the best of my knowledge and belief, this thesis contains no material previously published or written by another person except where due reference is made.

Balume Mburano

[redacted]

09th June 2018

ACKNOWLEDGEMENTS

I would like to thank my principal supervisor Dr. Weisheng Si for offering me the opportunity to evaluate the effectiveness of web vulnerability scanners. I appreciate his patience, enthusiasm, encouragement and broad knowledge. I was continuously advised and supported by his guidance in all the time of research timeframe and writing of this thesis.

Also, I appreciate my wife Aline Pendeza Balume for her patience and support, our friends Gary Atkinson and Suzan Oley for their support during the time of this study.

Lastly, thanks to the research service team of SCEM and HDR, Western Sydney University for providing administrative and technical support.

ABSTRACT

Web applications have become an integral part of everyday life, but many of these applications are deployed with critical vulnerabilities that can be fatally exploited. Web Vulnerability scanners have been widely adopted for the detection of vulnerabilities in web applications by checking through the applications with the attackers' perspectives. However, studies have shown that vulnerability scanners perform differently on detection of vulnerabilities. Furthermore, the effectiveness of some of these scanners has become questionable due to the ever-growing cyber-attacks that have been exploiting undetected vulnerabilities in some web applications.

To evaluate the effectiveness of these scanners, people often run these scanners against a benchmark web application with known vulnerabilities. This thesis first presents our results on the effectiveness of two popular web vulnerability scanners based on the OWASP benchmark, which is a benchmark developed by OWASP (Open Web Application Security Project), a prestigious non-profit web security organization. The two scanners chosen in this thesis are OWASP Zed Attack Proxy (OWASP ZAP) and Arachni. As there are many categories of web vulnerabilities and we cannot evaluate the scanner performance on all of them due to time limitation, we pick the following four major vulnerability categories in our thesis: Command Injection, Cross-Site Scripting (XSS), Light Weight Access Protocol (LDAP) Injection, and SQL Injection. Moreover, we compare our results on scanner effectiveness from the OWASP benchmark with the existing results from Web Application Vulnerability Security Evaluation Project (WAVSEP) benchmark, another popular benchmark used to evaluate scanner effectiveness. We are the first to make this comparison between these two benchmarks in literature.

The results mainly show that:

- Scanners perform differently in different vulnerability categories. That is, no scanner can serve as the all-rounder in scanning web vulnerabilities.
- The benchmarks also demonstrate different capabilities in reflecting the effectiveness of scanners in different vulnerability categories. It is recommended to combine the results from different benchmarks to determine the effectiveness of a scanner.
- Regarding scanner effectiveness, OWASP ZAP performs the best in CMDI, SQLI, and XSS; Arachni performs the best in LDAP.

- Regarding benchmark capability, OWASP benchmark outperforms WAVSEP benchmark in all the examined categories.

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

While this project covers different aspects that determine the effectiveness of two web vulnerability scanners, this chapter will present a general introduction to the project including project motivation and the project background. Furthermore, it will give an outline of the thesis structure.

Web vulnerability scanners are applications that investigate the presence of exploitable flaws in web applications with the objective of preventing or minimizing attacks [1]. Today, web application vulnerability scanners are widely available on both free open source and commercial basis. While there is easy access to these scanners on the market today, there is need to determine their effectiveness in unveiling vulnerabilities in web applications.

The primary aim of this study is to examine the effectiveness of two open source web vulnerability scanners using OWASP benchmark based on True Positive, True Negative, False Positive, and False Negative metrics. These metrics are used by OWASP benchmark to draw the performance results of a scanner and will allow us to give a detailed analysis of the results and draw a conclusion for each scanner.

1.1 Research Background

The security evaluation of Information Technology infrastructures by lawfully trying to exploit vulnerabilities is known as Penetration Testing or Ethical Hacking[2]. Penetration Testing was first used in the 1970's by the USA Department of Defence with the aim of unveiling security issues in computer systems to defend against unauthorized access and others security breaches in the systems so that these flaws can be fixed before their possible unauthorized exploitation.

As computers gained popularity and their ability to share and exchange information across communication lines rose, so was the challenge to protect the transferred data against attacks. To that end, in early 1965 computer security experts issued a warning about the inevitable attempt to compromise data transported across communication lines. Around 15000 governments, business analysts and computers security experts, therefore, discussed these concerns to come up with the term "Penetration Testing" and the identification of what we can qualify as one of the significant challenges of web applications today[3]. A task force of experts from NASA, CIA, computer security and academia was formed. This team effort demonstrated the usefulness of Penetration Testing as one of the tools to evaluate system's security[3].

Today, hackers' techniques have become more sophisticated; moreover, there is an increase in the complexity of technology used to develop web applications, penetration testing has become therefore an essential technique used to assess the security of computer systems using vulnerability scanners. However, the need to assess the effectiveness of these scanners is essential for their improvement, better scanner choice and ultimately assuring a better web application security.

1.2 Basic Concepts

Most applications vulnerability scanners comprise three main components. These include crawling, attacking element known as fuzzing (Fuzzing-consists of injecting semi malformed and malformed data in an automated way to find bugs in an application) [4] and analysis component (scraping- which is the process of collecting accessible and or processed data from an application)[5]. Developers and application testers have at their disposal some technologies that can be utilized to detect application flaws before or after an application is released. These include Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), and Interactive Application Security Testing (IAST).

Static Application Security Testing (SAST): is a code-based web application testing which can be done manually, or with the use of code analysis tool to find bugs in the application's source code, this can also be referred to as 'White Box Testing' [6]. However, it is difficult to find all the security flaws with source code analysis method, especially with complex application codes. Additionally, knowing the internal structure, design, and implementation of the application by the tester may become a hindrance in finding flaws in the application.

Dynamic Application Security Testing (DAST): DAST is a process of finding application vulnerabilities without prior knowledge of the structure, design, and implementation of the application. This method is also known as 'Black-box Testing' and 'Penetration Testing'. Fuzzing, scraping and crawling over web requests are some of the techniques used in this method to find vulnerabilities in the target applications [7].
Considering their features, Static Application Security Testing and Dynamic Application Security Testing methods have both some weaknesses and strengths. While it is evident that SAST uses a different approach as compared to DAST, both techniques complement each other.

Studies have shown that DAST method may perform well in detecting Cross Site Scripting(XSS) vulnerabilities because of its client-side execution (scanning) ability instead of a simple reflection in the SAST method. Nevertheless, it has been argued that DAST method has not been very successful in detecting vulnerabilities related to the password entered in the form of clear text into back-end log file [8], whereas this kind of vulnerability can be well managed by SAST method [9]. Despite its fallibilities, SAST is preferred by developers as it enables the development team to make needed changes to the code as flaws are detected at the conceptual level while reducing the cost that may incur if the defect is detected at the end of the project [10].

Interactive Application Security Testing (IAST) – IAST is a combination of DAST and SAST. Designed to complement the two methods (SAST and DAST), IAST exploits the strengths of both approaches and therefore helps in the minimization of the fundamental weaknesses of each of the process. It lessens false positive detection rates in both methods (SAST and DAST) by confirming each other. IAST does this by placing an agent within the target application for real-time monitoring and analysis [10].

In this study, we examine OWASP ZAP and Arachni as some of the scanners used for Dynamic Application Security Testing (DAST).

1.3 Vulnerabilities

It is crucial to get a basic understanding of application vulnerabilities before exploring different application vulnerability scanners. Vulnerability in a web application security is known as an unintended weakness or a flaw that can be exploited by an intruder for malicious purposes. Application vulnerability has mostly three aspects: the application flaw or susceptibility, unauthorized access by hackers to the application defect, and hacker being capable of exploiting the weakness [11].

In this study, we consider OWASP benchmark 2017 release which implements the most critical web application vulnerability test cases. These include different type of injections, session management and broken authentication, cross-site scripting, sensitive data exposure, broken access control, cross-site request forgery, security misconfiguration, under-protected APIs, insufficient attack protection and using components with known vulnerabilities [12]. We highlight some of the significant vulnerabilities below:

- **Cross-Site Scripting (XSS):**

XSS is an injection attack in which a malicious script is injected into an application. This type of attacks occurs when a hacker in the form of browser-side script sends a malicious script to several users [13]. If successful, the attacker will get the access privileges of the victim who has executed the script. Consequently, if the victim has the privilege to get access to sensitive data in the application, then this constitutes a severe vulnerability. Unfortunately, vulnerabilities that allow this kind of attacks to succeed are said to be widespread. Although vulnerability scanners can automatically detect some cross-site scripting issues, different web applications' build their output differently and make use of diverse interpreters such as Flash, JavaScript, Silverlight and ActiveX making the automatic detection hard [14]. Cross-Site Scripting attack can be performed in three different ways including ***Reflected or Non-Persistent XSS-*** which occurs when an exploit is supplied to a web application and then reflected back to the target browser to be executed. Including malicious content as a parameter in the URL is one of the most common mechanisms of delivering this attack.

Persistent or Stored XSS- in this attack, on the other hand, the application store the malicious data into its logs, database, message forum or other data store and the malicious data is then read back and included into the applications active content. ***DOM Based XSS***- while in the other type of this attack the injection is performed by the server, in DOM-based XSS the injection is performed by the client.

- ***SQL Injection:***

This is an attack method that is used to inject an SQL query as an input from the client side into the application. If successful, it allows the attacker to disrupt the predefined or standard execution of the applications SQL commands. This vulnerability occurs when data is kept in a database in an unsafe manner, and often, an organization that falls victim to this attack are unaware of the attack[15]. Although it is argued that programmatic interfaces such as ASP.NET and J2EE applications are resistant to this kind of attack, in general, SQL Injection attacks have high severity impact if successful [16]. SQL Injection issues have become common in database driven web applications as they can be easily detected and exploited.

- ***XPath (XML Path) injection:***

XPath is a query language that describes how different elements can be located in an XML document without access control restrictions, XPath injection attack may give the attacker unauthorized access to XML documents. Applications that insert supplied data in an insecure way may succumb to blind XPath injection attack that can be used to get unauthorized access to the application data [14].

- ***File Inclusion:***

This is an attack that exploits the dynamic file inclusion mechanism of a web application. When a user input data into the application and passes them into file include commands, this attack tricks the application by incorporating a file with malicious code. This, therefore, gives the attacker unauthorized access to sensitive data on the file server and web server [17].

- ***Lightweight Directory Access Protocol (LDAP):***

Directory information services are maintained and accessed using Lightweight Directory Access protocol. Single Sign-On (SSO) service is one of the most uses of LDAP which allow users access to the application with the assumption that the credentials have been verified and accepted by the LDAP provider. LDAP Injection happens when untrusted or malicious data is used by hackers to query the LDAP directory without prior authentication[18].

- ***Command Injection:***

This is an attack executed via a web interface with the objective of running Operating System command. Attackers might use for example the command nslookup for the user to supply their hostname which may then be used as an argument by placing a command separator from the hostname and make it possible to execute a malicious program after the nslookup command. If successful, this attack allows the intruder to upload the malicious program into the system and even get illegal access to passwords.

- ***Cross-Site Request Forgery(CSRF):***

Most web applications today require users to submit forms which can perform sensitive operations. These forms are also used by application administration to for example grant new users access to the application. CSRF attack occurs when an application administrator is tricked to click on a malicious link to log into the application and submit the login form without additional interaction. The following things are required for CSRF to occur:

- The target form must be used to perform a sensitive action, i.e., Admin login form
- The target session must be active

- The parameters must be guessable or known, i.e., Username, Password, and Role

1.4 Web Vulnerability Scanners

Web applications often contain vulnerabilities; therefore, vulnerability scanners are used to unveil exploitable flaws in the applications for their minimization or elimination. However, scanners accuracy and effectiveness are not always perfect, and not all scanners are easy to use[19]. Some of the vulnerability scanners include:

- *Burp Suite:* created by PortSwigger, Burp suite is a Java-based web security framework used by information security professionals and penetration testers to discover vulnerabilities and attack vectors in web applications[20]. As an intercepting proxy, this scanner can capture and analyze requests and responses from the target application. It allows manual setting of specific injection points. The main vulnerabilities targeted by burp suite are: Cross-site Scripting, SQL injection, OS Command Injection, and File path traversal[15].

- *Web Application Attack and Audit Framework(W3af):* is a free open source scanner that help discover vulnerabilities in web applications. Based on Python, this scanner offers command line interface as well as Graphical user interface. W3af architecture is divided into two parts which include plugins and core. The core provides features used by plugins to detect vulnerabilities in them using a knowledge base to share information. These plugins are categorised into Audit, Grep, Discovery, attack Mangle, Brute force and Output[21, 22].

- *Wapiti*: is a command line free open source web application vulnerability scanner that perform black-box scans of the target application. It crawls the web pages looking for forms and scripts for payloads injection to check whether the script is vulnerable[23]. Wapiti general features include an easy way to add a payload to a scanning process as a simple line to a text file, color coding to distinguish the severity of detected vulnerabilities and multiformat report generation.

- *Watabo* is a semi-automated open source scanner used to audit web applications. Based on ruby, this scanner has session management capabilities, smart filter functions and can act as a transparent proxy.

The evaluation of the effectiveness of various web vulnerability scanners has been done before. Nevertheless, the review and examination in contrast of OWASP Zed Attack Proxy (ZAP) and Arachni based on OWSP Benchmark have never been done before.

This study evaluates the effectiveness of these two open- source and cross-platform scanners using OWASP benchmark. This is particularly important because these scanners appeal to all type of testers and developers no matter their level of knowledge as they are easy to use. Hence, enabling better choice of scanners and development of more secure web applications.

1.5 . Research Motivation

Web applications have become an indispensable part of our lives today for the crucial roles that they play in our social, financial and other regular daily activities. Meanwhile, hackers' exploitation of web application vulnerabilities is increasing and the damages caused are devastating. The ever-changing and more sophisticated techniques used by hackers to exploit web applications is making it difficult to develop an utterly secure web application. However, ensuring the security of information is an essential aspect of any organization that deals with sensitive information. Therefore, web application security testing is performed to check for vulnerabilities. Nevertheless, manual testing of application vulnerabilities has proven to be demanding, costly, time consuming and error-prone. While automated application vulnerability scanners have been considered to remediate this situation, there is need to consider the efficiency of the chosen application vulnerability scanner. Some functions can determine a scanner's efficiency. These functions include Fuzzing, Web Crawling, Web Scraping and should be able to test application vulnerabilities such as Command Injection, Cross-Site Scripting, Insecure cookie, Light Weight Access Protocol (LDAP) Injection, Path Traversal, SQL Injection, Weak Encryption and Hash Algorithm just to name a few.

Fuzzing is an automated application testing technique that involves inputting invalid, random or unexpected data to an application to detect vulnerabilities [24].

Crawling is a phase during which the application automatically searches the world wide web for indexing of all web pages. Crawling coverage is essential in web application security testing because a high crawling coverage means that the scanner can thoroughly audit all resources without missing any.

Web scraping is a process used to extract information from web applications using a piece of code called scraper[25]. The code (scraper) sends "GET" requests to the target application then parses a document in HTML format on the received results, searches for needed data in the record and presents it in a specified form. It should be noted, however, that Crawling is the main component of web scraping.

1.6 Benchmarking and Metrics

It is difficult to understand and compare the weaknesses and strengths of application vulnerability scanners if we are not able to measure them. Benchmarking is one of the techniques used to do so. Benchmarking is a process of running a few standard tests against a set of applications to evaluate their relative performance [26]. Different benchmarks are used to assess vulnerability scanners including web Input Vector Extractor Teaser (WIVET), Web Application Vulnerability Scanner Evaluation Project (WAVSEP), Acunetix, AltoroMutual and OWASP Benchmark to name a few. In this study, OWASP benchmark has been chosen to evaluate the effectiveness of the selected application security scanners by comparing their accuracy and speed.

A lot can be learned about a web vulnerability scanner using True positive, False Positive, True Negative and False negative metrics. It is, however, essential to understanding these metrics before they can help us learn how effective a web vulnerability scanner is.

True Positive (TP): True positive is the number of cases that are positive and are detected as positive.

False Positive (FP): this is the number of cases that are negative but are detected as positive. In other words, this is the number of false alarms.

True Negative (TN): this is the number of cases that are negative and are detected as negative.

False Negative (FN): this is the number of cases that are positive but are detected as negative.

True Positive Rate (TPR): this is the rate at which a scanner correctly identifies and detects real vulnerabilities (positive cases) in an application [27, 28]. It is obtained by taking the number of true positives divided by a total number of positive tests.

$$TPR = \frac{TP(numbe\ of\ true\ positives)}{P(total\ number\ of\ positive\ tests)}$$

False Positive Rate (FPR): This is the rate at which a scanner reports non-existing conditions as existing. It fails to ignore and bypass false alarms [27]. In other words, it is the percentage at which a scanner wrongly gives positive decisions when checking some conditions given that events were not present.

$$FPR = \frac{FP(numbe\ of\ false\ positives)}{N(total\ number\ of\ negative\ tests)}$$

True Negative Rate(TNR): is the rate at which a scanner correctly ignores false alarms [27, 28]. Meaning that a scanner report that an event does not exist given the conditions.

$$TNR = \frac{TN\ (Number\ of\ true\ negative)}{N\ (total\ number\ of\ negatuve\ tests)}$$

False Negative Rate(FNR): It is the rate at which a scanner fails to identify and detect real vulnerabilities in an application [27, 28]. False Negative Rate can also be said to be the percentage at which a scanner reports that some conditions do not hold when in reality they do.

$$FNR = \frac{FN(number\ offalse\ negatives)}{P(total\ number\ of\ positive\ tests)}$$

Let us illustrate these metrics in an example for better understanding:

Consider a security vulnerability scanner that is subjected to 100 test cases. Seventy (70) of the test cases represent no vulnerabilities (Negative conditions), and thirty (30) represent vulnerabilities (Positive conditions).

- When applied to the negative tests, the scanner detects fifty- five (55) as negative and fifteen (15) of the negative tests as positive.
- As for the positive tests, the scanner detects twenty (20) of them as positive and ten (10) as negative

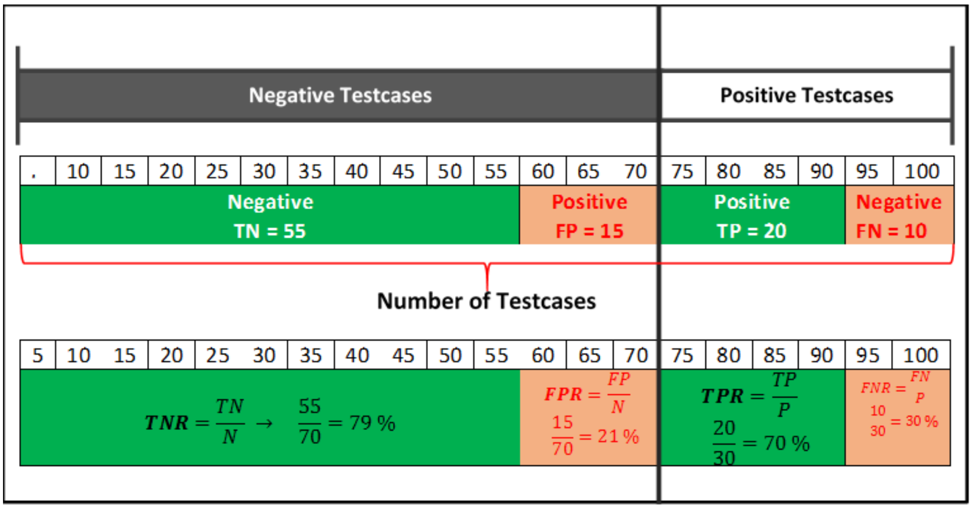


Figure 1: Benchmarking Metrics summary

As per the above illustration, the following data can be collected:

P - Total number of positive cases = 30

N - Total number of negative cases = 70

True positive (TP) = 20 - this is the number of correctly reported positive tests.

True negative (TN) = 55 - this is the number of correctly reported negative tests.

False Negative (FN) = 10 - is the number of positive tests that are incorrectly reported

False Positive(FP) =15 - this is the number of positive tests that are incorrectly reported.

$$\boldsymbol{True\ Positive\ Rate}\ (\boldsymbol{TPR}) \ = \ \frac{20}{30} = 0.7$$

$$\boldsymbol{False\ Negative\ Rate}\ (\boldsymbol{FPR}) = \frac{15}{30} = 0.3$$

$$\boldsymbol{True\ Negative\ Rate}\ (\boldsymbol{TNR}) = \frac{55}{70} = 0.79$$

$$\boldsymbol{False\ Positive\ Rate}\ (\boldsymbol{FNR}) = \frac{15}{70} = 0.21$$

Another metric that needs to be understood is "Accuracy." Accuracy is the ability of a scanner to correctly detected both positive and negative cases[28, 29]. Accuracy is calculated as follows:

$$\boldsymbol{Accuracy} = \frac{TP + TN}{P + N}$$

OWASP Benchmark scoring logic is based on the above-discussed metrics (Tue Positive, True Negative, False Positive and False Negative). To compute the individual score, OWASP Benchmark uses the Youden Index in order to avoid misclassifications by putting equal weights on the scanners' performance on both negative case and positive cases. Youden Index is calculated by subtracting one from the total number of test's specificity and Sensitivity. Sensitivity equals True Positive Rate(TPR) and Specificity equal to one minus False Positive Rate(FPR) [28, 30].

$$\boldsymbol{Youden's\ Index} = Sensitivity - (1 - Specificity)$$

A higher Youden Index value indicates a good performance of the scanner.

These metrics have historically been used in the evaluation of technology in the medical and military sectors and the calculation of accuracy.

1.7 Significance and Scope

First, the selection, which application vulnerability scanners should be analyzed and the benchmark to utilize to examine the selected vulnerability scanners was made by reviewing the popularity, benchmarking history and how often the programs are updated. Arachni and OWASP ZAP are opensource scanners that have become some of the most used application security scanners. OWASP Benchmark, on the other hand, has been getting regular updates and had enough contributors. However, it has not yet been used to benchmark these scanners against each other before. Therefore, these application vulnerability scanners and benchmarks were taken into consideration in the evaluation of the effectiveness of vulnerability scanners in this thesis.

The scope of the benchmarking process was set for the evaluation of the effectiveness of two vulnerability scanners using OWASP benchmark; hence, use of numerous benchmarks and other vulnerability scanners were skipped but not ignored by considering previous studies in this area. Firstly, as one of the aims of this study, we acquired a sound understanding of OWASP ZAP, Arachni and OWASP Benchmark functionalities and techniques. Then proceeded to identify the differences between these scanners by subjecting them to different test cases available in OWASP Benchmark that produced the overall performance as per OWASP Benchmark metrics. These metrics include True Positive (TP), False Negative (FN), True Negative (TN), False Positive (FP) and their corresponding Rates. These metrics are calculated on each vulnerability type, such as SQL-Injection, Cross-Site Scripting, command Injection, among others.

1.8 Thesis outline

This thesis is structured as follows: Chapter 1 briefly introduces the background information and discusses the basic concepts of web application vulnerabilities and web vulnerability scanners and objectives. Chapter 2 gives a brief survey of some previous studies that aimed to evaluate the effectiveness of web vulnerability scanners and point out the research gap. Chapter 3 describes how the experimental environment suitable for this study was developed and explains the significant steps involved. Furthermore, it highlights the properties and features of scanners and benchmark chosen for the evaluation. Chapter 4 presents a detailed summary of each scanner results in the selected categories, a comparison of the scanners benchmark results, followed by a comparison of our OWASP benchmark results with WAVSEP benchmark results from a previous study. Finally, Chapter 5 gives conclusions drawn from the experiments, possible recommendations, and future research direction.

CHAPTER 2 LITERATURE SURVEY

2.1 Web Vulnerability Scanning

The importance of using vulnerability scanners to unveil flaws in web applications before they are deployed has been realized by many organizations today. This has been highlighted in Daud, Abu Bakar, and Hassan's study[31]. Due to the ever-growing cybercrime, this study has examined some scanners that can be used to detect vulnerabilities that can be easily be missed by manual testing. This study has claimed that although it is essential to use web vulnerability scanners for web testing, these scanners perform differently and give different results depending on the configuration setting and how often the scanner's plugins are updated. It has added that the efficiency of a scanner can be evaluated by the number of vulnerabilities that it can detect and that this depends on the number of plugins that exist in the scanners knowledge base. However, the following questions can be raised regarding scanners effectiveness:

Can a scanner's effectiveness be determined by only the number of plugins in its knowledge base? Are there other ways and methods that can be used to determine the effectiveness of a web vulnerability scanner?

In regards to scanners selection, the study has suggested that for better results, commercial canners are preferable because of their regular updates as compared to open source canners. However, Shay Chen study has vigorously argued that some open source scanners such as Arachni are becoming as effective as commercial scanners[32]. On the other hand, the study has suggested that to get better scanning results, perform scanning with different scanners, use different policy settings and perform the scanning at a different time to take advantage of the scanners updates. Additionally, it has suggested that organizations need to identify the appropriate scanner for them based on cost, scanner's support, easy to use, and purpose of the scanning. The study has, however, recognized that more research is needed in the evaluation of scanning scanners effectiveness for better and easy choice.

Acknowledging difficulties that are faced by companies in the choice of vulnerability scanners, a study by Alzahrani, Alquazzaz, Fu, Almashfi and Zhu entitled " Web Application Security Tools Analysis[33]" has explored the ways that can be used to address this issue. This study proceeded by first identifying factors that cause insecurity in web applications and some reasons why it is hard to eliminate vulnerabilities in web applications as well as the most widespread vulnerabilities. For each discussed vulnerability, the study has suggested the best vulnerability scanner that can be used to identify its related loopholes.

For information disclosure vulnerabilities such as XSS and SQL-Injection, the study has suggested:

1. Netcraft as a scanner that can be used to unveil this as this scanner is said to be able to gather useful footprinting information related to a target domain.
2. Cross-Site Scripter(XSSer), an open source framework that can be used to detect Cascading Style Sheet related vulnerabilities in a web application.
3. OWASP Xenotic XSS supports both manual and automated mode for CrossSite Scripting and exploitation detection.
4. Scanners for SQL injection vulnerabilities include: SQL Inject Me, SQLninja, and Havij.

Although this study has attempted a different way of addressing problem-related to vulnerability scanning by suggesting the use of different scanners for different vulnerabilities as opposed to using one scanner, it has concluded that determining the effectiveness of a scanner for a better choice remains a challenge.

Therefore, there is a compelling need to address the research gap identified in the examined studies[31, 33] above, i.e., finding a method that can determine the effectiveness of web vulnerability scanners. Several other studies[19, 25, 32, 34], however, have highlighted the importance of benchmarking in determining the effectiveness of web vulnerability scanners.

2.2 Benchmarking

Benchmarking has been used in earlier studies as one of the approaches to evaluating the effectiveness of different vulnerability scanners. Such is the case of EL Malaka study entitled "Benchmarking Vulnerability Scanners: An Experiment on SCADA Devices and Scientific Instruments[19]." Focusing on Burp and Nessus accuracy in finding vulnerabilities in Scientific instruments and SCADA devices, the study used WAVSEP to obtain the benchmarking results based on vulnerabilities described in table 1 below:

Vulnerabilities	Test Cases	Description
Local File Inclusion (LFI)	816	Includes files on a server through a web browser, capable of allowing for directory traversal characters to be injected.
SQL Injection	130	Used to attack data-driven applications by inserting SQL statements into an entry field for execution
Remote File Inclusion (RFI)	108	Enables attacker to run malicious code on the server
Cross Site Scripting (XSS)	64	Enables attackers to inject client-side scripts into web pages
Open Redirection	60	A security flaw that enables a web page to fail correctly authenticating URL's
Unreferenced Files	22	Grant intruder access to inner workings, backdoors, administrative interfaces by accessing these files to gain knowledge about the infrastructure or credentials

Table 1: Description of Vulnerability categories used to benchmark SCADA devices

The study outlined attributes that are to be satisfied by benchmarks to produce accurate results, as follow:

- ***Benchmarks Applicability or relevance*** – benchmarks suitability of providing meaningful performance measure of the target scanners.
- ***Metrics*** – are known as proper measuring standards (benchmark test cases based on good metrics)
- ***Scalability*** – relevance of the benchmark to different scanners based on cost and performance.
- ***Acceptability*** - produce results that comply with the industry standards

The study results have shown that Burp outperformed Nessus in its accuracy to find vulnerabilities and false-positive detection. It has indicated that Burp found 78% of the vulnerabilities while Nessus found around 33.3% of the vulnerabilities. Burp took 12 hours to scan 1,182 SCADA IPS and Nessus 8 hours. On the other hand, to examine 184 scientific devices, the scanners spent 3 and 6 hours respectively as indicated in the study [29]. Considering the time taken by each scanner and the results produced, it may be argued that the time that a scanner takes to scan the target system may also be considered as one of the factors that may influence the results of a scanner's effectiveness in vulnerability detection. Although the study does not outline the above argument, it indicates that the scanner that performed better(Burp) took twice the time of the scanner that underperformed(Nessus).

The importance of benchmarking web application vulnerability scanners and the use of different benchmarks can also be verified in a recent study by AL Saleh M., Alomar N., Alshreef A. and Al-Salman A. entitled "Performance-Based Comparative Assessment of Open Source Web Vulnerability Scanners" [35]. This study has evaluated different vulnerability scanners including Skipfish, Arachni, Wapiti, Iron WASP, Vega and W3af and compared the performance results of these scanners based on different benchmarks. The study has confirmed that there are variations in results from various benchmarks. It has shown that all the evaluated vulnerability scanners have detected 77% SQL test cases of WAVSEP benchmark whereas most of the scanners were only able to identify 38% SQL test cases of Altoro-Mutual. In consideration of variations in the benchmarks results, the study has recommended the use of different benchmarks to evaluate the effectiveness of web vulnerability scanners. While this study examined a few scanners and different benchmarks were used to obtain the results, it is clear that OWASP ZAP and OWASP Benchmark were not included in the study.

However, the above arguments do not entail that OWASP ZAP has never been evaluated and compared with other scanners. Mariko Y. and Klyuev V.'s study has evaluated

OWASP ZAP effectiveness against Skipfish by scanning Damn Vulnerability Web Application (DVWA), a web application with known vulnerabilities against the two scanners and used WAVSEP benchmark to evaluate the effectiveness of the scanners [25]. The results of the study found that OWASP ZAP performed better than skipfish as it has detected around 107 vulnerabilities as compared to 13 detected by skipfish. Their benchmarking precision results are shown in table2 below:

	OWASP ZAP	skipfish
RXXS	100%	8.2%
SQLI	100%	9.5%
LFI	43.2%	1.0%
RFI	0.0%	0.0%

Table 2: Benchmarking Results of ZAP vs. Skipfish based on WAVSEP

Though the results found have shown that a scanner performance can be determined by comparing their scanning results, according to El Malaka study, different benchmarks need to be used to measure scanners effectiveness [19]. However, most earlier studies that have evaluated the effectiveness of web application vulnerability scanners have used WAVSEP benchmark only as the benchmark platform [35].

It is also the case in Shay Chen's study in which more than sixty open source and commercial web vulnerability scanners are evaluated however using one benchmark. While it may be argued that commercial scanners might be considered better than open source scanners, Chen's study results have indicated that open source scanners are becoming very popular and their performance is becoming as good as some of the commercial scanners [32]. His comments that "Arachni features such as load sharing and Crystal Report (RPT) interface have the potential of making it a must-have scanner in Software as a Service(SAAS) multi-product environment. Additionally, regardless of the size of the application scanned, number of threads and even against an easy target, Arachni appears to produce consistent results" [32]. Chen's claims strongly argue for the usefulness of open source scanners. Nevertheless, his claims are based on results gotten from one benchmark, WAVSEP benchmark only [35].

2.3 Current research summary and challenges

In this section, significant research efforts in the evaluation of web vulnerability scanners and the importance of benchmarking have been highlighted. Researchers have been devoted in finding best scanners and justify the importance of benchmarking, in the effort to find the best security solution to the ever-growing web security bridges, however, hacking activities are still on the rise.

Although the examined studies have demonstrated that it is necessary to benchmark scanners against each other in term or their scalability, accuracy and their overall performance, not all studies included a great variety of scanners and the results were obtained using one benchmark, WAVSEP benchmark [35]. Therefore, considerable research challenges and research gaps still exist. First, web vulnerability scanners are memory thirsty and require much time. Additionally, not all scanners can be accessed for free, and not all benchmarks have scripts for testing the available web vulnerability scanners.

Considering the rapid changes in ways and techniques used by hackers to get illegal access to web applications, it is necessary to consider the evaluation of a wider variety of commercial and popular open source vulnerability scanners such as OWASP ZAP and Arachni and different benchmarks. In the choice of benchmarks, it worth considering how often the benchmark and the scanner are updated and the number of contributors.

Therefore, this study has considered benchmarking two scanners, OWASP ZAP and Arachni with the latest version of OWASP Benchmark. This is particularly important because OWASP Benchmark is regularly updated and has a large number of contributors [23] and OWASP ZAP and Arachni have never been benchmarked against each other using OWASP Benchmark.

CHAPTER 3 EXPERIMENTAL ENVIRONMENT, SCANNERS AND METHODS

3.1 Methodology

An appropriate method was required to evaluate the chosen web vulnerability scanners for this study. The preferred method process is shown in figure 2 below:

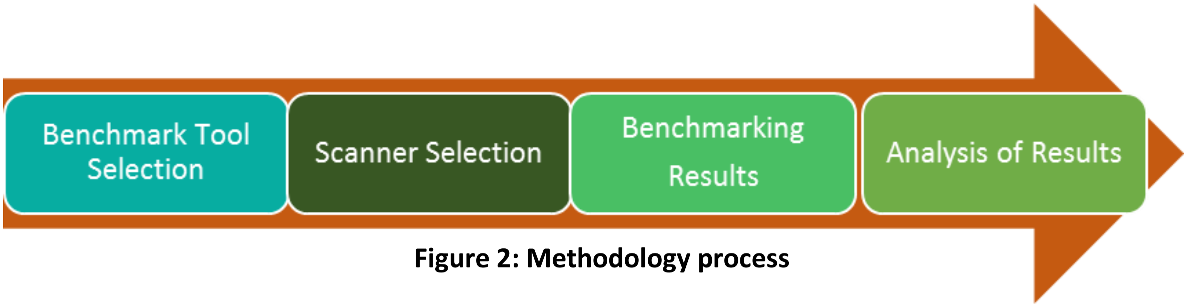


Figure 2: Methodology process

The study was divided into different steps:

A. Benchmark selection:

To evaluate and test web application vulnerability scanners, an application that has the needed list of vulnerable test cases was needed so that true positive, true negatives, false positives, and false negatives. A better decision in this selection required us to examine previous studies with the objective of getting an understanding of application benchmarking process as well as the existing benchmarks. Moreover, use the knowledge acquired from the examined studies for a better selection of scanners to be tested as well as the benchmark to be used for the testing.

There are several benchmarks such as OWASP Benchmark and Web Application Vulnerability Scanner Evaluation Project (WAVSEP). For our experiment, we are using OWASP Benchmark.

B. Scanners Selection

While many previous studies have evaluated both commercial and free open source scanners, this study focuses on two free open source scanners including Arachni and OWASP ZAP.

A testing environment was created consisting of a local area network of two computers with one acting as a target and the other as the attacking computer. All useful applications necessary to perform the benchmarking which include OWASP Benchmark, OWASP ZAP, and Arachni, were installed. These programs were explored, and their different functionalities understood.

C. Benchmarking Results

The benchmarking results are obtained by first executing the scanners against OWASP Benchmark test suite. The scanners results are then used to generate an XML file that is then fed back into OWASP benchmark to create scorecards that are then examined to draw conclusions on the performance of the scanners.

D. Analysis of Results

The benchmarking results of each scanner are discussed and compared to each other. Then, both scanners results are compared to results from the previous study by Shay Chen that have used WAVSEP benchmark to evaluate the scanners.

3.2 Scanners Overview

3.2.1 OWASP Zed Attack Proxy (ZAP)

OWASP ZAP is an easy to use scanner for finding vulnerabilities in web applications. It is one of the OWASP flagship projects that is recommended by OWASP for web applications vulnerability testing. ZAP is widely used by people ranging from security professionals, developers, and functional testers for automated security tests that can be incorporated into the continuous development environment. Additionally, ZAP is a free Open Source cross-platform scanner that is becoming a framework for advanced web application vulnerability testing[36].

Some of the ZAP features include:

Intercepting proxy, meaning that the browser can be configured to proxy through ZAP so that it can see all the request and responses which can also be changed.

ZAP provides both Passive and Active scanners. The passive scanner does not perform any attacks thus is safe to use on any web application. It runs all the time and examines the requests and responses but can still detect certain types of problems on that basis. The Active scanner, on the other hand, performs a wide range of attacks therefore formal permission is required for this to be used[36].

Spider, this is a ZAP feature that can be used to crawl the target application for missed and hidden pages and links.

Brute Force, Brute force is a trial and error method used to obtain information such as passwords and personal identification. By using OWASP DirBuster code, ZAP can find files that do not have links to them using the brute force component which is based on the dirbuster tool.

Fuzzing, ZAP can fuzz parameters and includes fuzzing libraries from the jbrofuzz and fuzz DB tools. This feature can be used to find more subtle vulnerabilities that the automated scanners might not detect.

Auto-tagging, this feature tags messages in ZAP to show which pages have hidden fields for example. This feature can be changed to tag anything of interest to the tester.

Dynamic SSL Certificate, this is a feature that allows users to generate unique root certificate authority that can tell the browser to trust it, therefore allowing ZAP to intercept secure hypertext transfer protocol (https) traffic seamlessly[36].

Report Generation, ZAP can generate reports on the detected issues including information about the problems and suggestions on how to solve them.

3.2.2 Arachni

Arachni is a high-performance free Open Source ruby based framework that is aimed to help administrators and penetration testers evaluate the security of web applications. Arachni supports multiple platforms including Windows, Linux, and Mac OS X and can be instantly deployed using its portable packages[37]. Arachni deployment options include: *Command Line Interface*(CLI) for quick scans, *Web User Interface*(WebUI) for multi-user, multi-scan and multi-dispatcher management and distributed system with remote agents[37]. Some of the most important features of Arachni include:

Intelligence, a feature that enables Arachni to adapt to each web application on the fly, individual analysis of application resources which allows Arachni to customize requests to the used technologies.

To be able to handle complicated workflows and identify new input points, Arachni continuously self-trains by learning from the HTTP request throughout the scanning process[37].

3.2.3 OWASP Benchmark

OWASP (Open Web Application Security Project) Benchmark was launched in the year 2015 with the aim of evaluating the accuracy, coverage, and speed of web-application vulnerability scanners. As an open source program, organizations and researchers may use this framework to evaluate web vulnerability scanners using thousands of test cases provided by OWASP Benchmark across eleven distinct categories of vulnerabilities. These categories include Command Injection (CMDI), Cross Site Scripting (XSS), Insecure Cookie, Lightweight Directory Access Protocol (LDAP) Injection, Path Traversal, Structured Query Language (SQL) Injection, Trust Boundary Violation, Weak Encryption Algorithm, Weak Hash Algorithm, Weak Random Number and XPath Injection. Implemented by Java, OWASP Benchmark can be used to evaluate different types of Static Application Security Tools(SAST), Dynamic Application Security Tools (DAST) such as Arachni and Zed Attack Proxy (ZAP) and Interactive Application Security Tools(IAST). It also uses codes that seem vulnerable but are not, for false alarms detection. Although OWASP Benchmark is a free open source program, it remains state-of-the-art as it has a significant number of contributors and it is regularly updated. Therefore, OWASP Benchmark may be considered one of the benchmark choices for measuring the effectiveness of vulnerability scanners[38]. It gives the score of a tested scanner based on true positive rate, false positive rate, true negative rate and false negative rate. This is particularly important because time and ability needed to discover true and false metrics of a scanner make them incredibly important and a clear understanding of these is required for the choice of vulnerability scanner.

The score produced by OWASP Benchmark is a ***Youden index*** which is a standard method that summarises test set accuracy[29]. OWASP Benchmark computes individual scores for each test case category called Benchmark Accuracy Score ranging between 0 and 100[29].

The following example gives an overview of how OWASP Benchmark calculates a scanner's accuracy score.

Assume that a scanner has returned a True Positive Rate (TPR) of 88% and False Positive Rate (FPR) of 15%; This means that, its Sensitivity = TPR (0.88) and its Specificity = 1-FPR (0.85). Therefore, the ***Youden Index*** is (0.88+0.85) - 1 = 0.73 and OWASP Benchmark Score is 73 since it normalizes the results to the range of 0 to 100.

The visual representation of a scanner performance for both True Positive and False positive results is shown in figure 3 below:

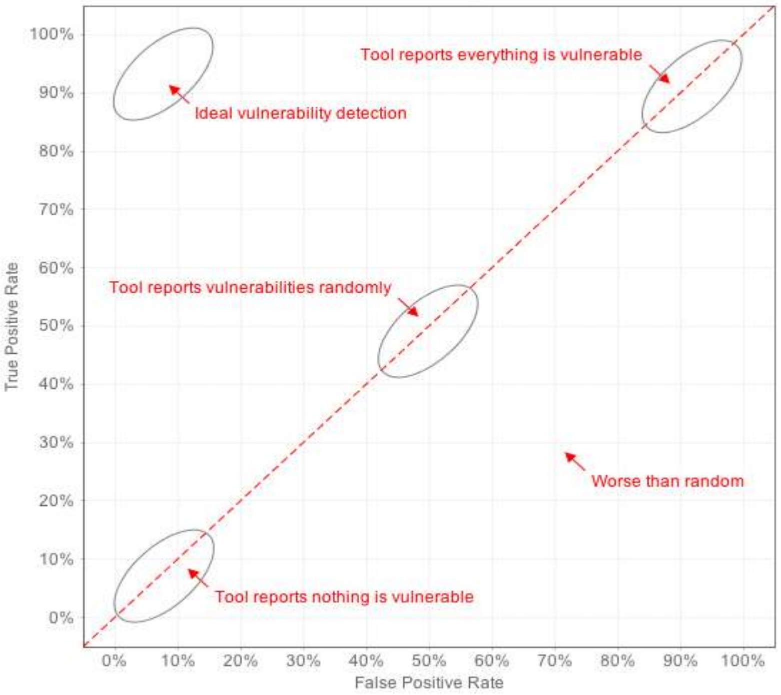


Figure 3: OWASP Benchmark Results Interpretation Guide

As it can be seen in figure 3 above, OWASP Benchmark produces positive and negative scores. The points above the diagonal line are positive scores, meaning that the True Positive Rate is higher than the False Positive Rate, and the points under the diagonal line are negative scores indicating that the results of False Positive Rate are higher than True Positive Rate.

The version of OWASP Benchmark used in this study has 2740 test cases (positive and negative cases) that have been created based on these metrics.

The Vulnerability areas, number of cases and the expected results for each are shown in table 3 below:

VULNERABILITY AREA	NUMBER OF TEST CASES	POSITIVE CASES	NEGATIVE CASES
Command Injection	251	126	125
Weak Cryptography	246	130	116
Weak Hash	236	129	107
LDAP Injection	59	27	32
Path Traversal	268	133	135
Secure Cookie Flag	67	36	31
SQL Injection	504	272	232
Trust Boundary Violation	126	83	43
Weak randomness	493	218	275
XPATH Injection	35	15	20
XSS (Cross-Site Scripting)	455	246	209
Total number of Cases	**2740**		

Table 3: Number of OWASP Benchmark Test Cases per Category

These test cases derive from real applications coding pattern but are not to be considered as real applications[29].

3.3 Experimental Environment

For the evaluation of the effectiveness of web application vulnerability scanners, there is a need for vulnerable test applications. To obtain the true positives, true negatives, false positives and false negatives, the application must have the exact tests.

While there are several benchmarks, Open Web Application Security Project Benchmark (OWASP Benchmark) is the chosen evaluation platform for this study.

As we aim to evaluate the effectiveness of OWASP ZAP and Arachni based on OWASP benchmark, our testbed has two significant components. The first component comprises of the web Application vulnerability scanners (Arachni and OWASP ZAP) and the second part of our testbed contains the benchmark (OWASP Benchmark).

Figure 4 below demonstrates the lab environment set up for benchmarking tests for ZAP and Arachni using OWASP Benchmark.

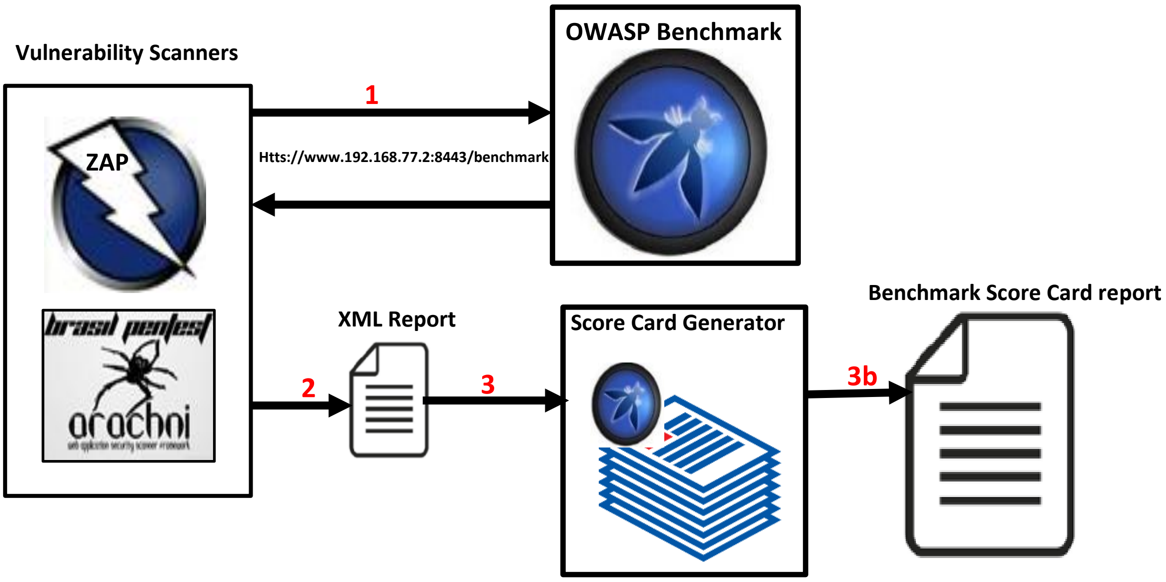


Figure 4: Lab Environment and Experimental Steps

The benchmarking process comprised of three significant steps:

Step1. Setting the chosen scanner (Arachni and ZAP) to attack OWASP Benchmark. This is an essential step as it subjects the scanner to the existing Vulnerability tests within benchmark and generates a report that will be used to measure the scanners' performance using true positive and false positive, true negative and false negative metrics.

For ZAP, first, we run a spider on the target (OWASP Benchmark) to discover all the resources (URLs) that are available in the target before launching the attack, then launch the attack using the 'Active Scan.'

For Arachni, using the command line interface, we navigate to the bin folder into the Arachni then execute run Arachni while specifying the target URL, the checks to be executed and specify the report name as follow: ***https://192.168.77.2:8443/benchmark/ --checks=*,-code_injection_timing,-os_cmd_injection_timing,-sql_injection_timing--http-request-queue-size 300--report-save-path=C:\Tools\Arachni\arachni-2.0dev-1.0dev-windows--86_64\bin\benchmark_Notiming_Report.afr***

Step 2. For Arachni, the command line interface generates a dot **Afr** (Arachni Framework Report) report. This report is then used to produce other reports in different formats including HTML and XML. For the purpose of this study, the XML report was the most needed to generate benchmark scorecards. On the other hand, if Arachni or ZAP Web interface is used, at the end of a successful scan, the scanners automatically generate reports in different formats that can be downloaded from provided links.

Step 3. The XML report is then copied back into results folder in OWASP Benchmark, then the command ***createScoreCards.bat*** (for Windows) or ***createscorecards.sh*** (for Linux) is executed to generate benchmark results known as Scorecards.

It should be noted that to determine the accuracy of the obtained results during verifications; the scan was run multiple time, first as a whole, then, each category. This method has been applied more in obtaining the Arachni results.

CHAPTER 4 RESULTS

Our results are organized into four sections. The first section will examine the results of Arachni and OWASP ZAP individually with OWASP Benchmark as the target. The second section will compare the OWASP Benchmark results for each scanner in each vulnerability category. And the third section will give a summarised comparison our benchmarking results for both scanners with previous benchmarking study results based on WAVSEP benchmark.

1.1 Results of Individual Scanners

This section discusses each experimental scanner results using OWASP Benchmark as a target. A brief description of the findings will be given as well as a suggestion of how the detected flaw may be fixed.

Although OWASP Benchmark has eleven categories of vulnerabilities, four of the critical vulnerability categories will be examined. These include Command Injection, LDAP Injection, SQL Injection, and XSS. These categories were chosen in consideration of their criticality in addition to how favorable they are to both scanners.

Arachni results will be examined first followed by OWASP ZAP results.

1.1.1 Arachni Results

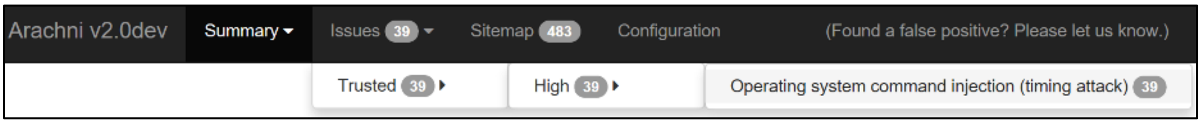
To achieve successful scan, for each category we pointed Arachni to attack the OWASP benchmark corresponding URLs and specifying the security checks that are supposed to be done while instructing Arachni to overlook the checks that are not relevant to that specific target category. This decision was reached at based on our realization that OWASP benchmark does not have test cases that correspond to some Arachni checks. Our decision was approved by Dave Wrenchers, the project leader of OWASP benchmark who confirmed that some timing checks should be overlooked as OWASP benchmark does not provide their corresponding test cases. The overlooked checks include Blind SQL injection Timing and Code Injection Timing.

After a successful scan, Arachni categorized its returned results by their **severity** with the ***red color*** standing for ‘***High Severity*,**’ the bright ***orange color*** standing for ‘***Medium Severity,'*** the ***faded orange*** standing for ‘***Low Severity'*** and ***Blue*** standing for ‘***Informational.'*** The same color code was used to highlight the severity rate of vulnerable HTML elements detected in the scanning of each category. Green color has been used to report HTML elements without any significant or reportable issues.

The average time taken by Arachni to scan most of the categories was 10 hours and 30 minutes, but Command Injection category took exceptionally longer time than other categories.

a. *Command Injection(cmdi)*

To test for Command Injection vulnerabilities, we pointed Arachni to attack the OWASP Benchmark Command Injection URLs while instructing it to overlook other URLs. It took Arachni 3 days, 5hours and 40 minutes to examine this category. Arachni scanned through all the benchmark test cases in this category, discovered 483 command injection-related URLs with 39 of this returned as positive cases of Command Injection attack as shown in figure 5 below.



Arachni v2.0dev | Summary ▾ | Issues 39 ▾ | Sitemap 483 | Configuration | (Found a false positive? Please let us know.)

Trusted 39 ▸ | High 39 ▸ | Operating system command injection (timing attack) 39

Figure 5: Arachni Command Injection URLs Discovery summary

The discovered positive cases have been classified as 100% high severity command injection attack cases. The charts below show Arachni classification of the detected cases as per severity level and HTML elements with issues by type.

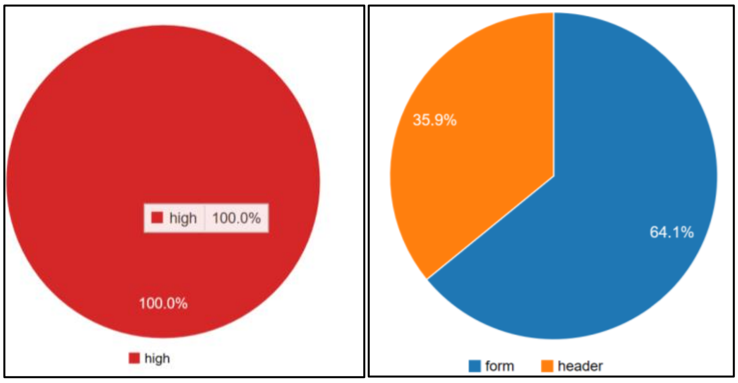
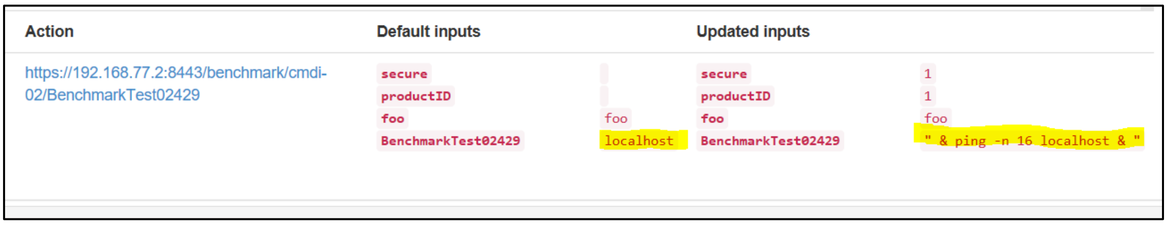


Figure 6: Arachni results of OWASP Benchmark Command Injection Tests Category: on the left - the severity of detected cases and the right - HTML elements with issues by type

As it can be seen in the charts, all positive cases detected were of high severity. The HTML header element representing 35.9 % of all elements in this category had the medium severity issues. On the other hand, Form element which represents 64.1% of all detected HTML elements were reported as having informational issues. However, although the results show a high rate of high severity issues (100%), it is important to note that this represents 39 successfully detected cases out of 126 Command Injection positive test cases. Meaning that only 31% command injection cases were detected with 100% of them being positive and classified as high severity cases.

To achieve the above results, Arachni used a ping command to map the network in the attempt to get full control of the server. Figure7 below show how Arachni performed this attack by injecting control operators such as (**&**, **&&**, |, ||, \, **#**) into the supplied command.



Action	Default inputs		Updated inputs	
https://192.168.77.2:8443/benchmark/cmdi-02/BenchmarkTest02429	secure productID foo BenchmarkTest02429	 foo localhost	secure productID foo BenchmarkTest02429	1 1 foo " & ping -n 16 localhost & "

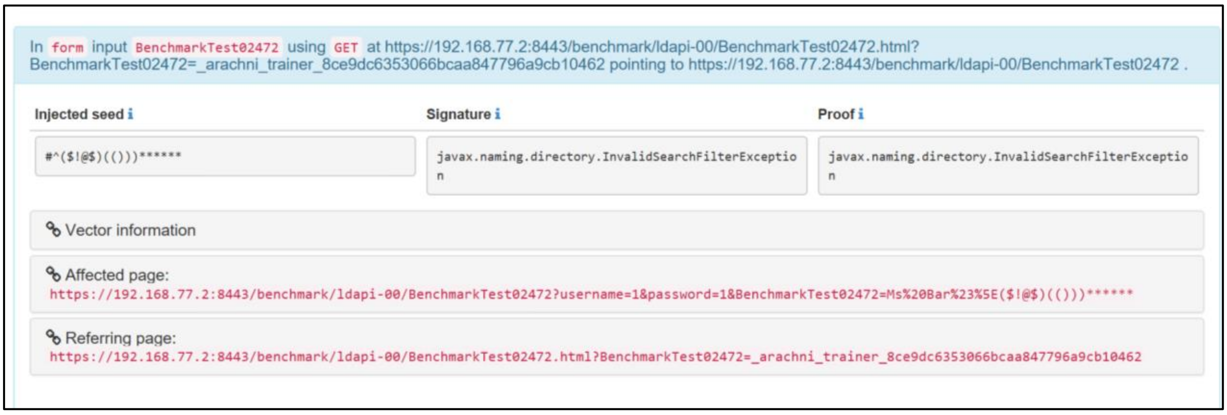
Figure 7: Arachni use of ping command to attack bench test case number 02429

As shown in figure 7 above, it is recommended that all control operators such as (**&, &&**, |, ||, **$, \, #**) should be explicitly denied and never accepted as valid input by the server. However, Arachni successfully supplied these operators to a ping command and got a positive server response, therefore, returning this case as a positive command injection case.

b. Lightweight Directory Access Protocol (LDAP) Injection

To test for LDAP Vulnerabilities, we applied a filter for Arachni to only scan the OWSP Benchmark LDAP test URLs and overlook the rest. The observation of the scanning process revealed that positive cases in this category detected between Cross-Site Request Forgery, LDAP, Backup file and Strict Transport Security header. Their severity, however, moved from high to low between LDAP injection and Backup file and moved further down from Medium to informational between Strict transport Security header and Insecure cookie.

To detect LDAP Injection issues, Arachni supplied a series of characters such as **#^ ($! @$) (())** ****** to perform the attack. This is because it is recommended that untrusted character or data should not be used to form an LDAP query. Therefore, correct validation should be applied to the supplied data to ensure that only required actions are to be performed by the supplied character values. The figure 8 and nine below is an example of how Arachni has successfully performed the attack to detect LDAP Injection issues.



In form input BenchmarkTest02472 using GET at https://192.168.77.2:8443/benchmark/ldapi-00/BenchmarkTest02472.html?BenchmarkTest02472=_arachni_trainer_8ce9dc6353066bcaa847796a9cb10462 pointing to https://192.168.77.2:8443/benchmark/ldapi-00/BenchmarkTest02472 .

Injected seed i	Signature i	Proof i
`#^($!@$)(()))******`	`javax.naming.directory.InvalidSearchFilterException`	`javax.naming.directory.InvalidSearchFilterException`

Vector information

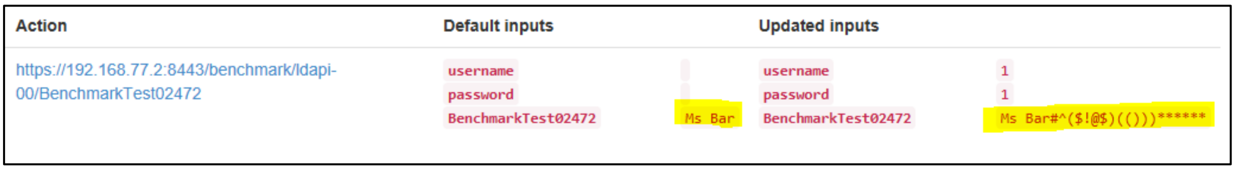
Affected page:

```
https://192.168.77.2:8443/benchmark/ldapi-00/BenchmarkTest02472?username=1&password=1&BenchmarkTest02472=Ms%20Bar%23%5E($!@$)(()))******
```

Referring page:

```
https://192.168.77.2:8443/benchmark/ldapi-00/BenchmarkTest02472.html?BenchmarkTest02472=_arachni_trainer_8ce9dc6353066bcaa847796a9cb10462
```

Figure 8: Arachni LDAP Injection Method and Proof



Action	Default inputs		Updated inputs	
https://192.168.77.2:8443/benchmark/ldapi-00/BenchmarkTest02472	username		username	1
	password		password	1
	BenchmarkTest02472	Ms Bar	BenchmarkTest02472	Ms Bar#^($!@$)(()))******

Figure 9: Arachni Successful LDAP Injection in OWASP Benchmark Test case number 02472

Injected seed in the above figure 6 represent the characters or seed used by Arachni to uncover the vulnerable vector during the audit; the ***signature*** is the signature used to detect the issue and ***proof*** is the string used to verify the existence of the issue. Figure 7, on the other hand, shows how the injected seed in figure 6 was successfully applied in the OWASP benchmark test case number 2472. The above-shown test method was then applied to all the relevant OWASP Benchmark Test cases. The returned results were found under two different categories. Including Cross-Site Request Forgery - which is an attack that forces users to perform unsolicited actions on a web application in which they are currently authenticated with the intent to change the state of the HTTP request and LDAP injection – which is an attack that targets web applications that construct LDAP statements based on user inputs.

The figure 10 below represents the overall results for LDAP Injection cases by severity and the vulnerability category:

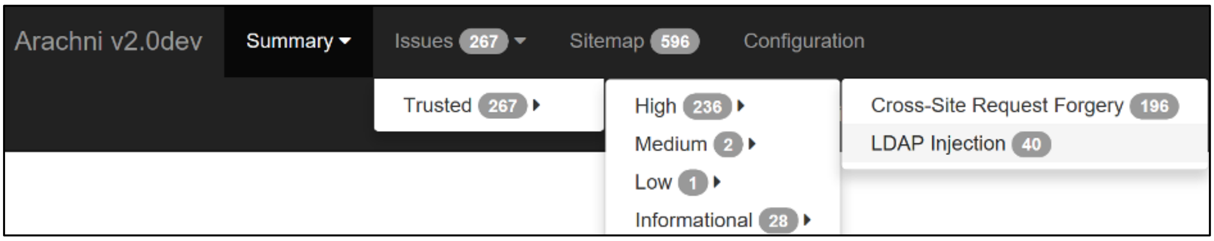


Figure 10: Arachni LDAP number of test cases detected, severity and category

As it can be seen in figure 10 above, a total number of 596 cases were detected in this category among which 257 cases were reported as having issues or positive cases. Out of the 236 high severity cases, 196 were cross-site request forgery, and 40 were LDAP Injection. Medium severity, low severity and information cases were 2,1 and 28 categorized insecure cookies, backup file and missing strict transport header respectively. Although it is noticeable that there are fewer LDAP issues detected under this category as compared to CSRF, it should be noted that these issues have been rated higher than most of the other issues in other examined vulnerability categories as we will discuss in later parts in this document.

Figure 11 below give a summary rate of the detected cases per their severity levels and the type of elements identified as having issues.

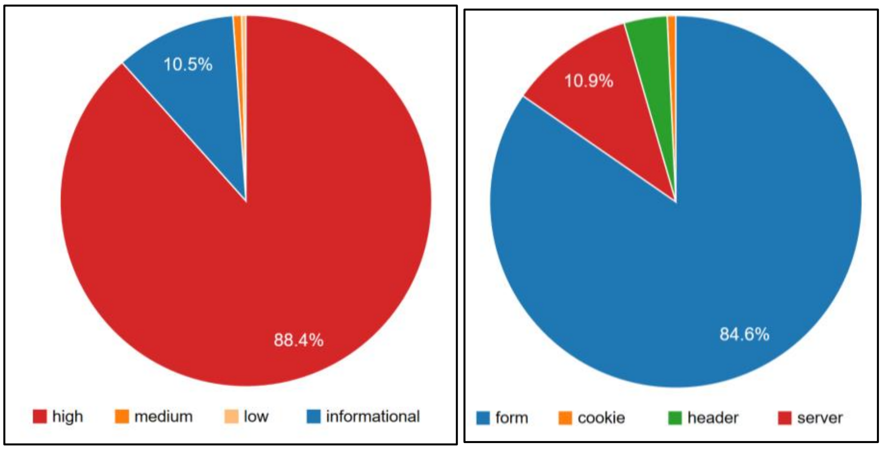


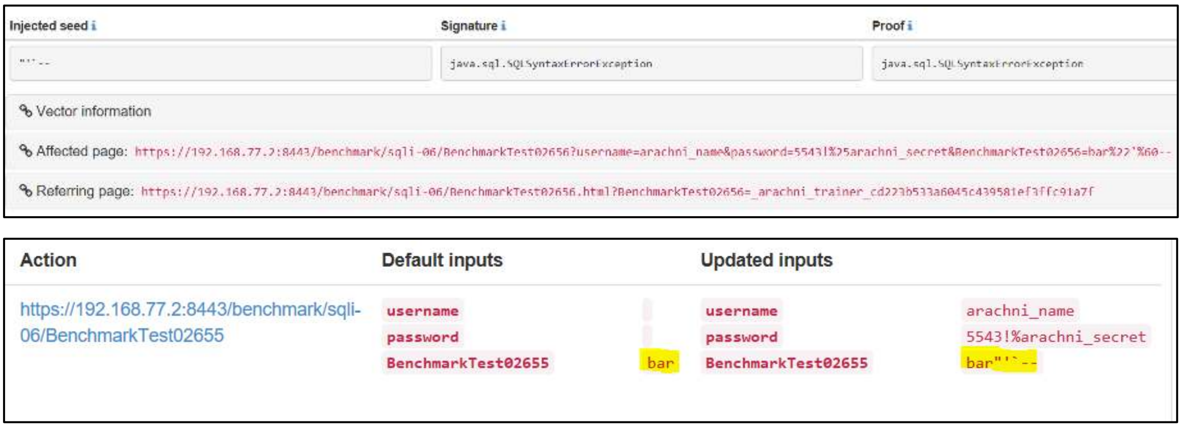
Figure 11: Arachni results of OWASP Benchmark LDAP Injection Tests Category: On the left- severity rate and on the right - infected elements

In figure 11 above, 88.4% of all cases were reported as high severity among which 10.9% were server related issues. 0.7% of all reported issues had medium severity level, 0.4 % low severity and 10.5% reported as informational. Most of the detected issues were forms related based on either known or predictable parameters and known error messages. Therefore, these test cases were reported as vulnerable to CSRF and LDAP Injection.

c. *SQL Injection*

Just like in previous tests, SQL Injection test was accomplished by pointing Arachni to OWASP benchmark corresponding URLs while preventing it from scanning other non-SQL Injection-related URLs. As discussed in chapter one, an SQL injection attack occurs when a value from the client request is used within an SQL query without prior verification. This could allow cyber-criminals to execute random SQL code and potentially get unauthorized access to sensitive data or use additional functionality available on the database server to control different server components.

In this category, Arachni was able to detect the issues by deceiving the server to respond to its requests with database related errors as shown in figure 12 below.



Injected seed	Signature	Proof
"'`--	java.sql.SQLSyntaxErrorException	java.sql.SQLSyntaxErrorException

Vector information

Affected page: https://192.168.77.2:8443/benchmark/sqli-06/BenchmarkTest02656?username=arachni_name&password=5543!%25arachni_secret&BenchmarkTest02656=bar%22'%60--

Referring page: https://192.168.77.2:8443/benchmark/sqli-06/BenchmarkTest02656.html?BenchmarkTest02656=_arachni_trainer_cd223b533a6045c439581ef3ffc91a7f

Action	Default inputs		Updated inputs	
https://192.168.77.2:8443/benchmark/sqli-06/BenchmarkTest02655	username		username	arachni_name
	password		password	5543!%arachni_secret
	BenchmarkTest02655	bar	BenchmarkTest02655	bar"'`--

Figure 12: Arachni SQL Injection Method

As a result of the above attacking method, Arachni was able to successfully detect some tests cases under three different SQL Injection categories including SQL Injection, Blind NoSQL Injection, and Blind SQL Injection. SQL Injection attack is consists of inserting an SQL query through the input data from the client or user side into the application. Blind SQL injection, on the other hand, is an attack designed to send true and false queries to the application database and determine the answer based on the receive database responses. The difference however of Blind SQL injection and Blind NoSQL injection is that the NoSQL does not involve any Structured Query Language.

Figure 13 below shows the returned number of URLs, issues, and their respective categories

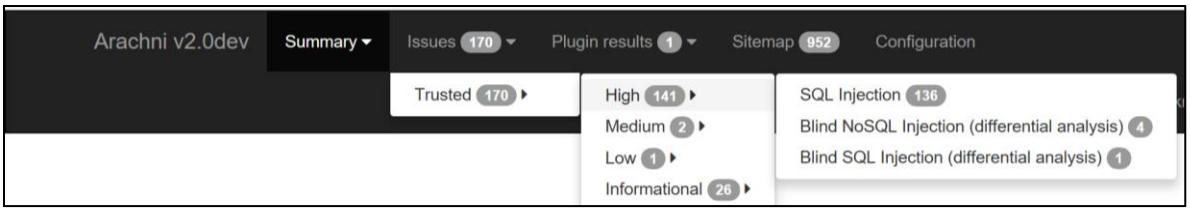


Figure 13: Arachni SQL Injection, number of Test cases detected, their severity and category

In figure 13 above, it can be seen that Arachni was able to discover 952 URLs while examining this category and successfully detected 170 positive cases among which 141 were classified as high severity. The highest number of these cases were SQL injection category with 136 cases, followed by Blind NoSQL injection with 4 cases and Blind SQL injection with just 1 case. Other positive reported cases include Missing Strict Transport Security header and Missing X-Frame Option header. The categorization of the cases severity rate as well as the HTML elements with issues was also done.

The charts below show the detected SQL Injection cases by severity on the left and HTML infected elements on the right:

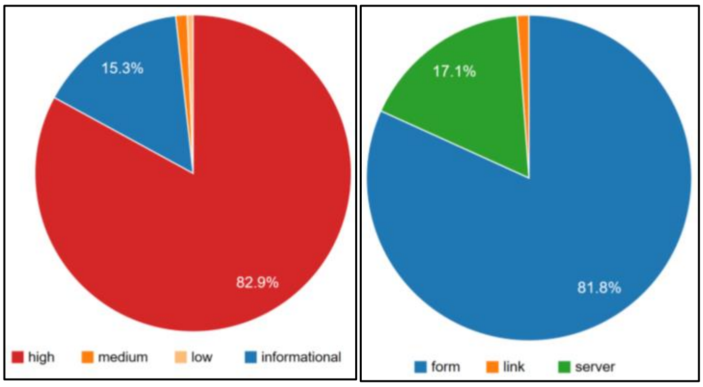
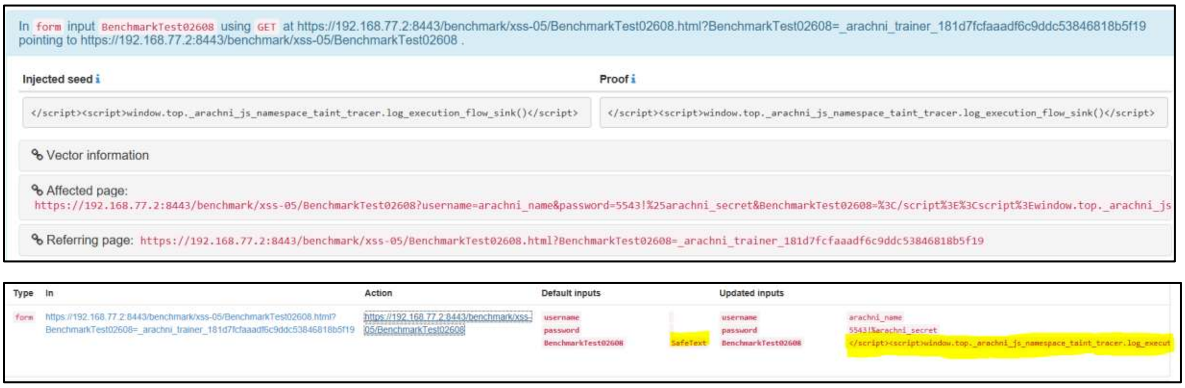


Figure 14: Arachni results of OWASP Benchmark SQL Injection Tests Category: on the left – severity rates and on the right – affected elements

In figure 14 above, 84.1% of cases have been classified as high severity with 99% of it reported as SQL injection issues. Medium, low and informational issues were 1.2%, 0.6%, and 14.8% respectively. 81.8% of all issues were detected in the HTML form elements, 1.2% from HTML links and 17.1% were application Server related issues.

d. Cross Site Scripting(XSS)

Many modern web applications use client-side scripts that can perform simple functions as well as complex ones that can interact with the operating system. When an application tolerates the use of client-side injected scripts without validation, then there is the possibility of an attacker to deceive the user to execute a custom script that can successfully return some results from the user's computer. Arachni has used the same method to discover Cross Site Scripting issues in the OWASP Benchmark test cases. It has therefore reported that it is possible to trick the browser to execute tailored JavaScript code. Figure 15 below shows how Arachni has attempted to execute this attack on OWASP Benchmark.



In form input BenchmarkTest02608 using GET at https://192.168.77.2:8443/benchmark/xss-05/BenchmarkTest02608.html?BenchmarkTest02608=_arachni_trainer_181d7fcfaadf6c9ddc53846818b5f19 pointing to https://192.168.77.2:8443/benchmark/xss-05/BenchmarkTest02608 .

Injected seed	Proof
`</script><script>window.top._arachni_js_namespace_taint_tracer.log_execution_flow_sink()</script>`	`</script><script>window.top._arachni_js_namespace_taint_tracer.log_execution_flow_sink()</script>`

Vector information

Affected page:
https://192.168.77.2:8443/benchmark/xss-05/BenchmarkTest02608?username=arachni_name&password=5543!%25arachni_secret&BenchmarkTest02608=%3C/script%3E%3Cscript%3Ewindow.top._arachni_js

Referring page: https://192.168.77.2:8443/benchmark/xss-05/BenchmarkTest02608.html?BenchmarkTest02608=_arachni_trainer_181d7fcfaadf6c9ddc53846818b5f19

Type	In	Action	Default inputs		Updated inputs	
form	https://192.168.77.2:8443/benchmark/xss-05/BenchmarkTest02608.html?BenchmarkTest02608=_arachni_trainer_181d7fcfaadf6c9ddc53846818b5f19	https://192.168.77.2:8443/benchmark/xss-05/BenchmarkTest02608	username password BenchmarkTest02608	 SafeText	username password BenchmarkTest02608	arachni_name 5543!%arachni_secret </script><script>window.top._arachni_js_namespace_taint_tracer.log_execut

Figure 15: Arachni XSS attack on OWASP Benchmark

The execution of the above attack produced the results seen in figure 16 below.

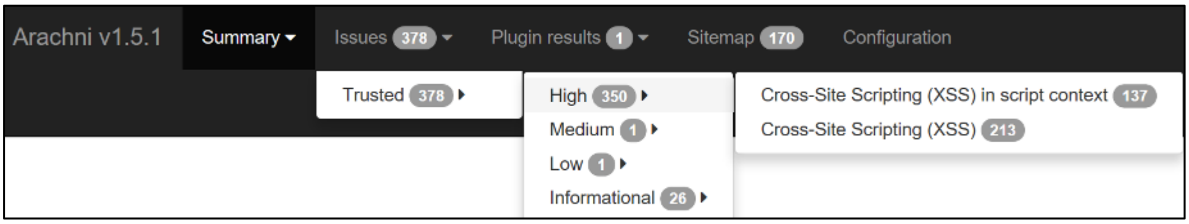


Figure 16: Arachni XSS tests detection and grouping by severity and category

Figure 16 shows that the number of issues or positive XSS cases were 350 out of a total of 378 high severity cases detected in this category. The high severity cases were subdivided into two distinct categories comprising 213 Cross Site Scripting and 137 Cross Site Scripting in Script contest. This is particularly important because XSS attack can be executed in different ways including forcing the page to execute a custom JavaScript code or inserting a tailored script content directly into an HTML element content.

The charts in figure 17 below show the rating of the detected cases by their severity and HTML elements affected by Cross-Site Scripting issues:

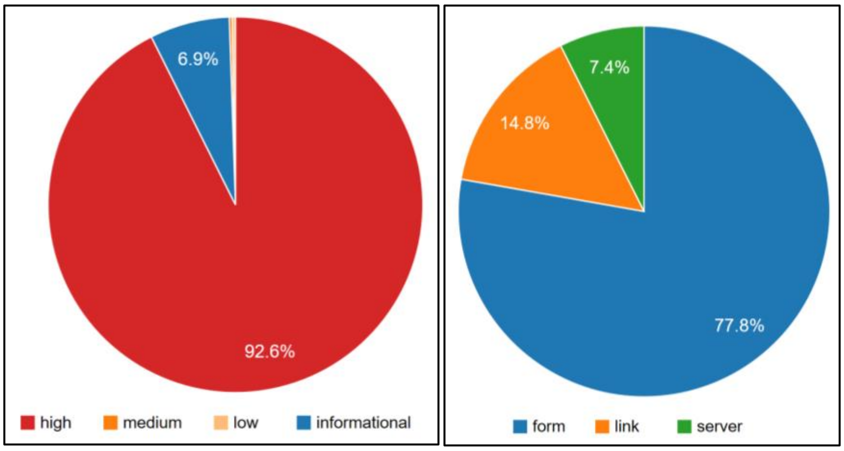


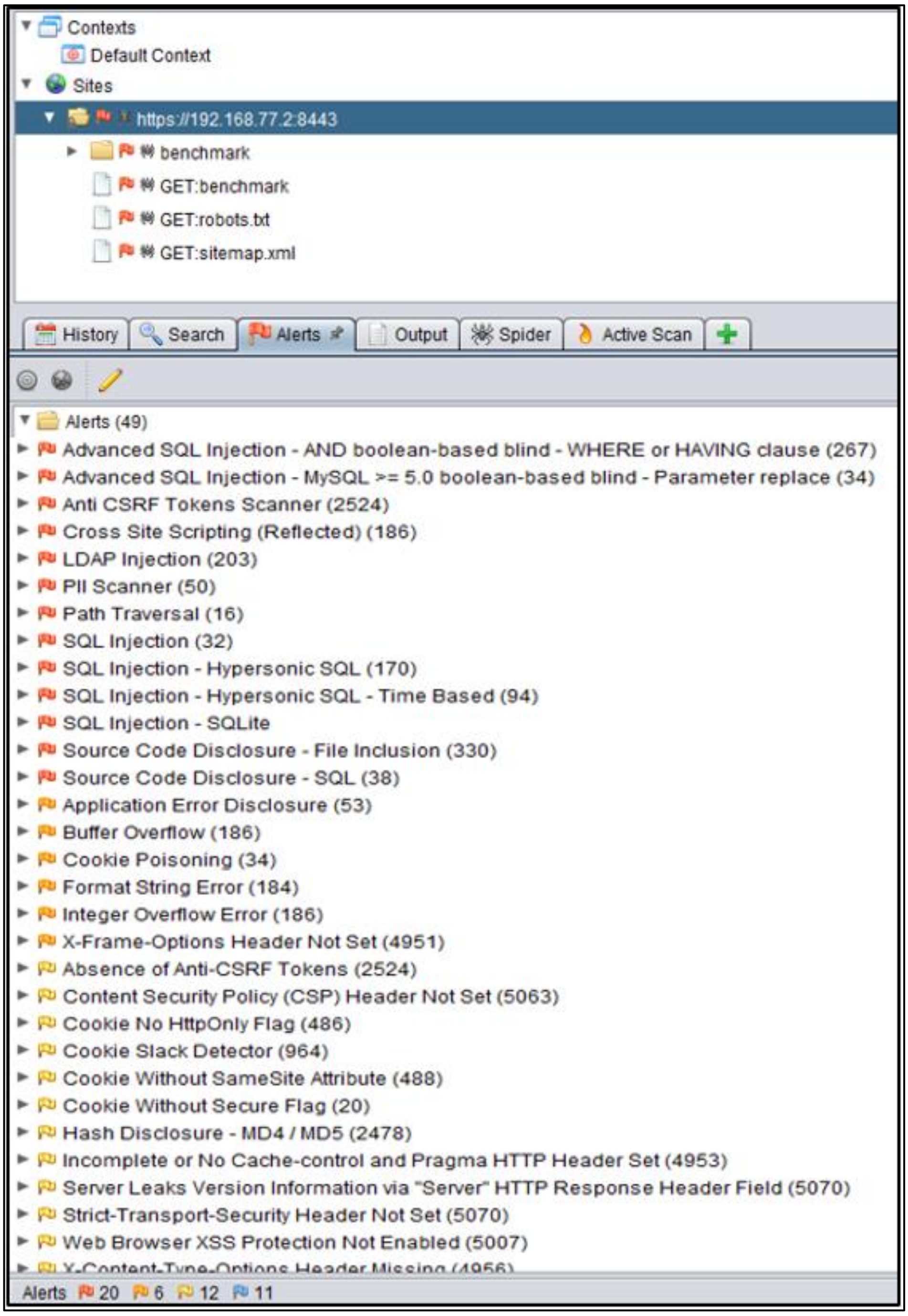
Figure 17: Arachni results of OWASP Benchmark Cross Site Scripting Tests Category: On the left Severity rate and the right – affected HTML elements

Figure 17 above depicts Arachni detection rate in Cross Site Scripting Category. 96.2% of the detected positive cases were classified as high severity, 0.3% medium severity and 0.3% low severity and 6.9% informational. As it can be seen in the left chart in figure 17, it is clear that a higher rate of positive cases was detected in this category as compared to previously discussed categories. The interesting part is that all the reported 92.6% high severity cases were Cross-Site Scripting Cases as it could be seen in figure 16. Three-quarter of the informational issues were HTML form related, and 14.8% of medium severity issues were from HTML links as shown in the right chart in figure17 above.

1.1.2 OWASP Zed Attack Proxy (ZAP) Results

This section discusses the results from OWASP ZAP on OWASP Benchmark. While we will attempt to give a detailed discussion of ZAP scan report, this discussion will not be as detailed as Arachni because the produced ZAP report has provided less detailed data as compared to Arachni which produced more detailed data including some graphical representations. Nevertheless, four critical vulnerability categories will be discussed including LDAP Injection, SQL Injection, XSS and Insecure Cookies. For a better comparison, however, Command Injection results will also be examined. On the other hand, while with Arachni it is possible to scan vulnerability categories separately, ZAP does not provide this flexibility. Therefore, with ZAP, one scan has covered all categories. To ensure that the obtained results are accurate, we conducted multiple scans with different operating systems including Kali Linux, Ubuntu, and Windows. Then, the best results from these scans have therefore been considered for discussion in this study.

Similar to Arachni, ZAP uses colored flags, red, orange, yellow and blue to categorize the severity of detected cases on the identified vulnerability categories. The ***Red flag*** signifies ***high severity*** issues; ***orange*** signifies ***medium severity*** cases, ***yellow*** signifies ***low severity*** cases, and ***blue*** signifies ***informational*** cases. Before going through the details of the results, let us have a general look at ZAP returned results in figure 18 below.



```
▼ Contexts
    Default Context
▼ Sites
  ▼ https://192.168.77.2:8443
    ► benchmark
      GET:benchmark
      GET:robots.txt
      GET:sitemap.xml
```

```
▼ Alerts (49)
► Advanced SQL Injection - AND boolean-based blind - WHERE or HAVING clause (267)
► Advanced SQL Injection - MySQL >= 5.0 boolean-based blind - Parameter replace (34)
► Anti CSRF Tokens Scanner (2524)
► Cross Site Scripting (Reflected) (186)
► LDAP Injection (203)
► PII Scanner (50)
► Path Traversal (16)
► SQL Injection (32)
► SQL Injection - Hypersonic SQL (170)
► SQL Injection - Hypersonic SQL - Time Based (94)
► SQL Injection - SQLite
► Source Code Disclosure - File Inclusion (330)
► Source Code Disclosure - SQL (38)
► Application Error Disclosure (53)
► Buffer Overflow (186)
► Cookie Poisoning (34)
► Format String Error (184)
► Integer Overflow Error (186)
► X-Frame-Options Header Not Set (4951)
► Absence of Anti-CSRF Tokens (2524)
► Content Security Policy (CSP) Header Not Set (5063)
► Cookie No HttpOnly Flag (486)
► Cookie Slack Detector (964)
► Cookie Without SameSite Attribute (488)
► Cookie Without Secure Flag (20)
► Hash Disclosure - MD4 / MD5 (2478)
► Incomplete or No Cache-control and Pragma HTTP Header Set (4953)
► Server Leaks Version Information via "Server" HTTP Response Header Field (5070)
► Strict-Transport-Security Header Not Set (5070)
► Web Browser XSS Protection Not Enabled (5007)
► X-Content-Type-Options Header Missing (4956)
Alerts 20 6 12 11
```

Figure 18: OWASP ZAP summary scan results of OWASP Benchmark

A close examination of the results in figure 18 shows that ZAP detected 20 high severity cases, six medium severity cases, 12 low severity cases and 11 informational cases. The high severity cases include SQL Injection, Cross Site Scripting, Anti CSRF Token Scanner and LDAP injection. Anti CSRF Token Scanner had a most significant number of high severity cases followed by SQL Injection, LDAP Injection Cross Site Scripting and Path Traversal. Cookie related cases were reported as medium and low severity cases (under orange and yellow flags). However, the number of severity alerts should not be confused with the number of issues detected per category as it can be seen in figure 18 above which shows that a category can have multiple issues, but all these issues will be put under one flag.

Table 4 below gives a summary of the number of detected issues detected under each category:

Category	Number of Alerts per Category
Anti CSRF Token	2524
SQL Injection	597
LDAP Injection	203
Cross Site Scripting	186
Insecure Cookies	1958

Table 4: Number of OWASP Benchmark Test Cases detected by ZAP in the listed categories

The categorization of all the detected cases as per risk level and the number of alert per level is shown in table 5 below:

Risk Level	Number of Alerts
High	20
Medium	6
Low	12
Informational	11

Table 5: Number Alerts per Severity Level

The data shown in the above table 5 indicates that more alerts were classified as high-risk cases as compared to medium and low. Nevertheless, despite SQL Injection having a less number of detected cases as compare to Anti CSRF Token and Insecure Cookies as shown in table 4 above, more than a quarter (7/20) of the high-risk level cases are SQL Injections cases as it can be seen in figure 18.

Figure 19 below shows the severity percentage rate of ZAP scan results:

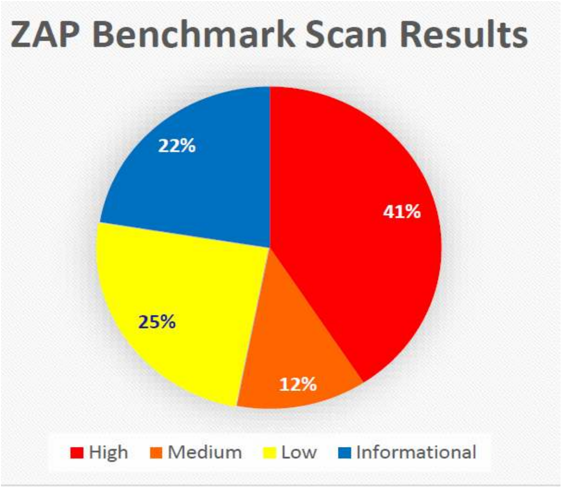


Figure 19: Percentage rate of the scan results as per their severity

It can be seen that high severity cases were 41% of all detected cases followed by low severity case with 25 %, informational cases were 22% and medium severity cases 12%.

a. Lightweight Directory Access Protocol (LDAP) Injection

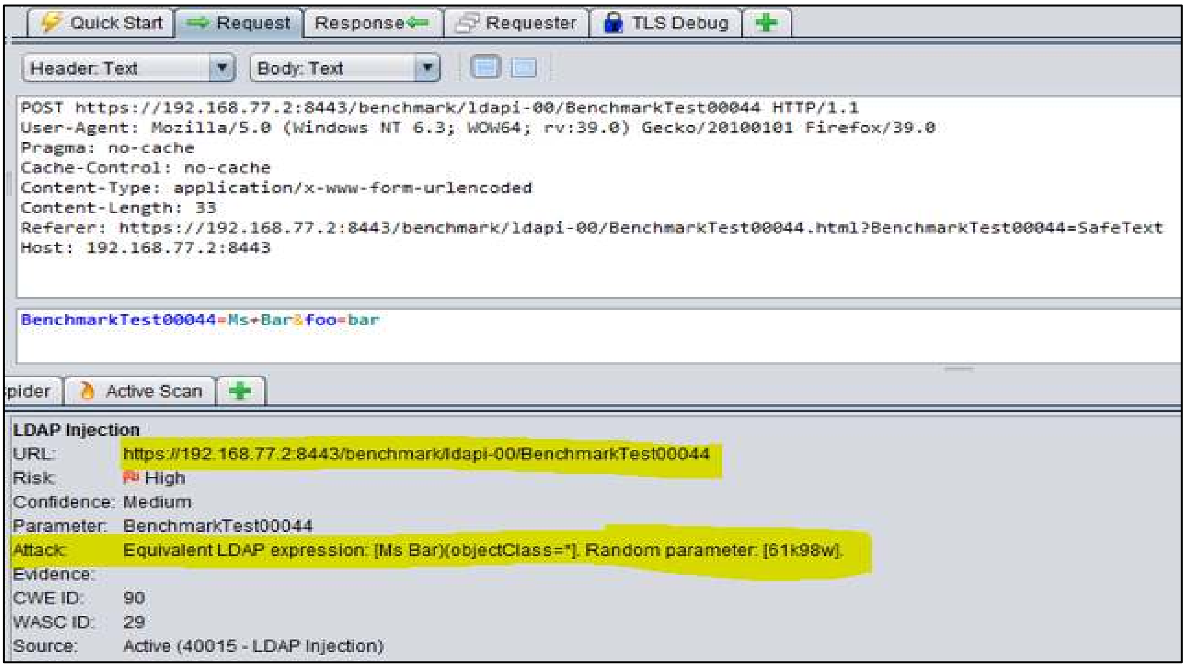
LDAP Injection attack usually occurs in situations where an application has a form that requires the user to enter some data such as username. Moreover, the underlying code behind that executes the request will take the search query information and produce an LDAP query that will be acceptable for searching the LDAP database.

OWASP ZAP has used a similar method to achieve a successful LDAP Injection on OWASP benchmark related test cases. ZAP has used logically equivalent expressions to the target URL or test case to achieve the attack.

For benchmark test 0044 for example, ZAP used the following parameter to bypass any possible authentication controls and give the attacker the ability to view and modify arbitrary data in the LDAP directory:

Parameter = [BenchmarkTest00044] on [POST] *https://192.168.77.2:8443/benchmark/ldapi-00/BenchmarkTest00044*]

Equivalent expression: [Ms Bar) (objectClass=*], and FALSE expression [61k98w]. Figure 20 below shows how ZAP executed LDAP Injection on the OWASP Benchmark test Cases:



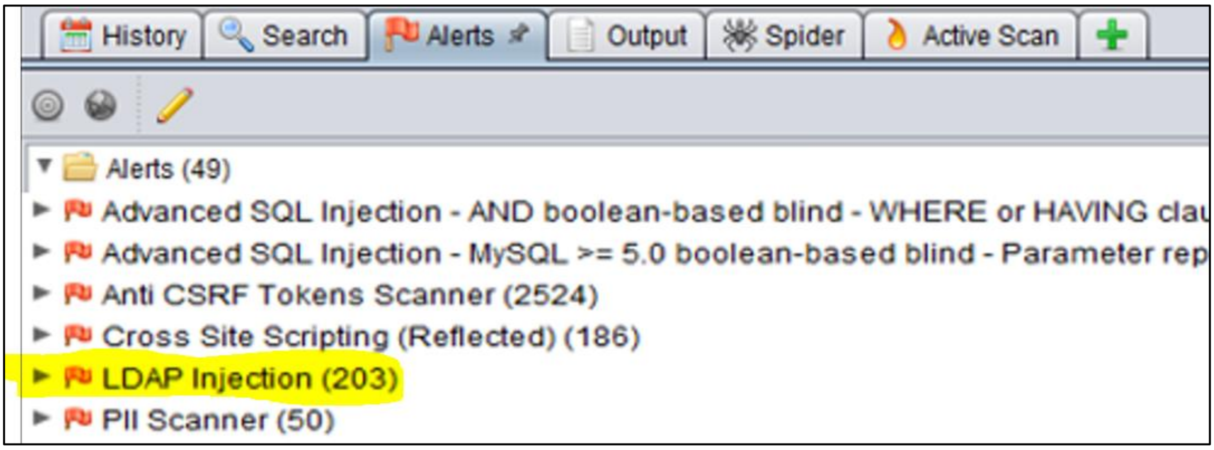
```
POST https://192.168.77.2:8443/benchmark/ldapi-00/BenchmarkTest00044 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:39.0) Gecko/20100101 Firefox/39.0
Pragma: no-cache
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded
Content-Length: 33
Referer: https://192.168.77.2:8443/benchmark/ldapi-00/BenchmarkTest00044.html?BenchmarkTest00044=SafeText
Host: 192.168.77.2:8443

BenchmarkTest00044=Ms+Bar&foo=bar
```

```
LDAP Injection
URL:        https://192.168.77.2:8443/benchmark/ldapi-00/BenchmarkTest00044
Risk:       High
Confidence: Medium
Parameter:  BenchmarkTest00044
Attack:     Equivalent LDAP expression: [Ms Bar)(objectClass=*]. Random parameter: [61k98w].
Evidence:
CWE ID:     90
WASC ID:    29
Source:     Active (40015 - LDAP Injection)
```

Figure 20: ZAP LDAP Injection attack

As it can be seen in figure 20 above, the use of similar expressions by ZAP as its input request in the place of the target URL for LDAP database searches returned a positive response. Therefore, this indicates that LDAP Injection was possible. This technique was applied to all the other test cases and reported that 203 positive LDAP Injection test cases as shown in figure21 below:

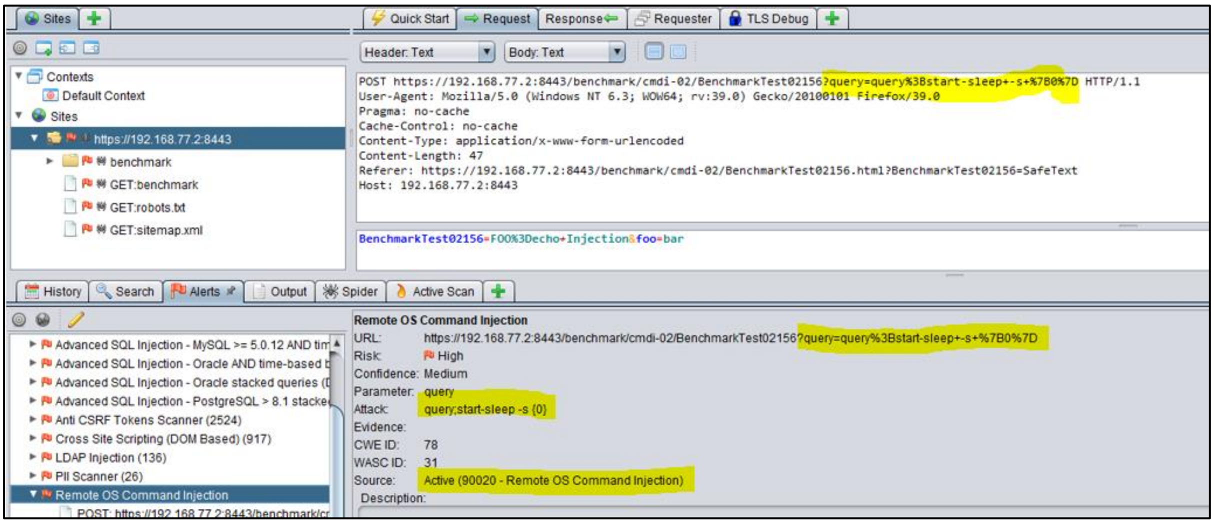


```
▼ Alerts (49)
► Advanced SQL Injection - AND boolean-based blind - WHERE or HAVING clau
► Advanced SQL Injection - MySQL >= 5.0 boolean-based blind - Parameter rep
► Anti CSRF Tokens Scanner (2524)
► Cross Site Scripting (Reflected) (186)
► LDAP Injection (203)
► PII Scanner (50)
```

Figure 21: Number of positive LDAP Injection Cases

b. *Command Injection (CMDI)*

In this category, OWASP ZAP attempted to perform unauthorized execution of operating system commands to check whether this attack is possible in the discovered cases. Figure 21 below shows how ZAP attempted to execute this attack on OWASP Benchmark test number 2156.



```
POST https://192.168.77.2:8443/benchmark/cmdi-02/BenchmarkTest02156?query=query%3Bstart-sleep+-s+%7B0%7D HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:39.0) Gecko/20100101 Firefox/39.0
Pragma: no-cache
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded
Content-Length: 47
Referer: https://192.168.77.2:8443/benchmark/cmdi-02/BenchmarkTest02156.html?BenchmarkTest02156=SafeText
Host: 192.168.77.2:8443

BenchmarkTest02156=FOO%3Decho+Injection&foo=bar
```

- Advanced SQL Injection - MySQL >= 5.0.12 AND tim
- Advanced SQL Injection - Oracle AND time-based b
- Advanced SQL Injection - Oracle stacked queries (D
- Advanced SQL Injection - PostgreSQL > 8.1 stacke
- Anti CSRF Tokens Scanner (2524)
- Cross Site Scripting (DOM Based) (917)
- LDAP Injection (136)
- PII Scanner (26)
- Remote OS Command Injection
  - POST: https://192.168.77.2:8443/benchmark/cr

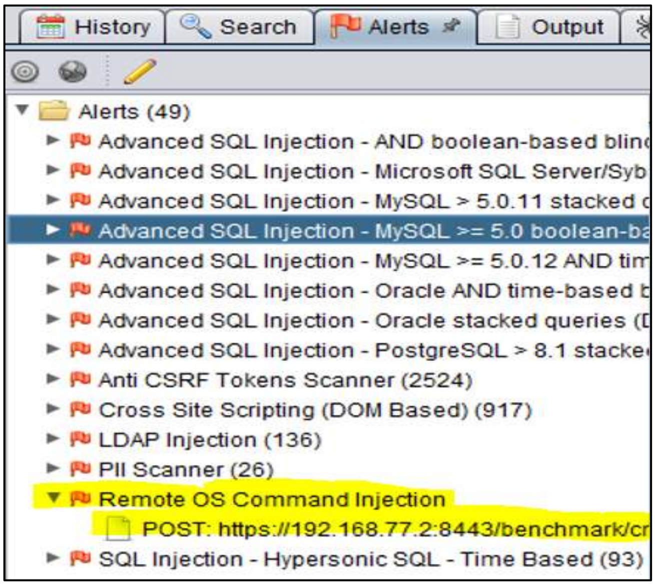
Remote OS Command Injection

URL:	https://192.168.77.2:8443/benchmark/cmdi-02/BenchmarkTest02156?query=query%3Bstart-sleep+-s+%7B0%7D
Risk:	High
Confidence:	Medium
Parameter:	query
Attack:	query;start-sleep -s {0}
Evidence:	
CWE ID:	78
WASC ID:	31
Source:	Active (90020 - Remote OS Command Injection)
Description:	

Figure 22: ZAP Command Injection attack on OWASP benchmark Test 2156

As it can be seen in figure 22 above, command injection attack has been flagged red signifying high severity. The examination of the highlighted details indicates that this attack can only be possible when an application accepts some untrusted input in the building of the operating system commands in an insecure manner or improper external program calling. Therefore, if possible, the use of library calls rather than external processes in the creation of the desired functionality is recommended. Avoidance of this attack may be feasible by keeping data that may be used to generate an executable command out of external control as much as possible. In a web application, for example, this may require storing the command locally in the session's state instead of sending it in a hidden file to the user or client.

Interestingly, in this category, ZAP was only able to discover one high severity case as shown in figure 23 below:



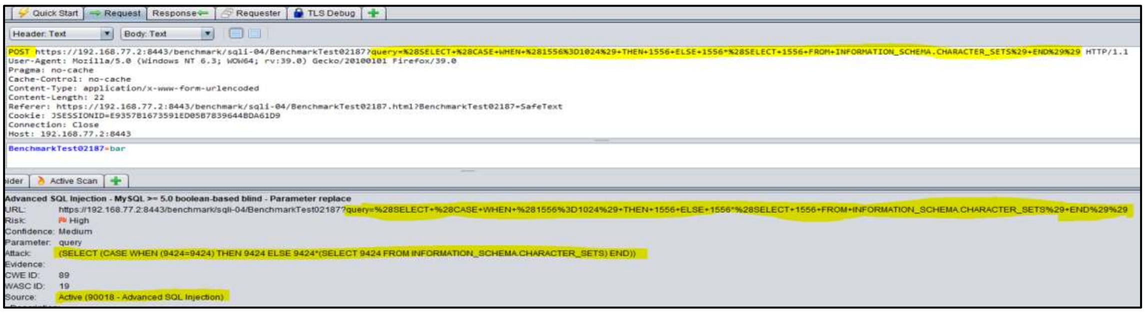
```
▼ Alerts (49)
  ► Advanced SQL Injection - AND boolean-based blin
  ► Advanced SQL Injection - Microsoft SQL Server/Syb
  ► Advanced SQL Injection - MySQL > 5.0.11 stacked 
  ► Advanced SQL Injection - MySQL >= 5.0 boolean-ba
  ► Advanced SQL Injection - MySQL >= 5.0.12 AND tir
  ► Advanced SQL Injection - Oracle AND time-based b
  ► Advanced SQL Injection - Oracle stacked queries (
  ► Advanced SQL Injection - PostgreSQL > 8.1 stacke
  ► Anti CSRF Tokens Scanner (2524)
  ► Cross Site Scripting (DOM Based) (917)
  ► LDAP Injection (136)
  ► PII Scanner (26)
  ▼ Remote OS Command Injection
      POST: https://192.168.77.2:8443/benchmark/cr
  ► SQL Injection - Hypersonic SQL - Time Based (93)
```

Figure 23: Number of Command Injection Cases detected

c. *SQL Injection*

In this category, ZAP discovered that SQL Injection is possible under the detected test cases using a payload. A Payload is a malicious piece of code that is run in the attacker's box, which is then translated by the application exploit and generate a GET and POST requests combinations to be sent to the remote Web server[39].

Figure 24 below demonstrates how ZAP successfully used boolean conditions in a SQL select statement to achieve an SQL injection attack in OWASP Benchmark test case number 2187



```
POST https://192.168.77.2:8443/benchmark/sqli-04/BenchmarkTest02187?query=%28SELECT+%28CASE+WHEN+%281556%3D1024%29+THEN+1556+ELSE+1556*%28SELECT+1556+FROM+INFORMATION_SCHEMA.CHARACTER_SETS%29+END%29%29 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:39.0) Gecko/20100101 Firefox/39.0
Pragma: no-cache
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded
Content-Length: 22
Referer: https://192.168.77.2:8443/benchmark/sqli-04/BenchmarkTest02187.html?BenchmarkTest02187=SafeText
Cookie: JSESSIONID=E9357B1673591ED05B7839644BDA61D9
Connection: Close
Host: 192.168.77.2:8443

BenchmarkTest02187=bar
```

```
Advanced SQL Injection - MySQL >= 5.0 boolean-based blind - Parameter replace
URL:         https://192.168.77.2:8443/benchmark/sqli-04/BenchmarkTest02187?query=%28SELECT+%28CASE+WHEN+%281556%3D1024%29+THEN+1556+ELSE+1556*%28SELECT+1556+FROM+INFORMATION_SCHEMA.CHARACTER_SETS%29+END%29%29
Risk:        High
Confidence:  Medium
Parameter:   query
Attack:      (SELECT (CASE WHEN (9424=9424) THEN 9424 ELSE 9424*(SELECT 9424 FROM INFORMATION_SCHEMA.CHARACTER_SETS) END))
Evidence:
CWE ID:      89
WASC ID:     19
Source:      Active (90018 - Advanced SQL Injection)
```

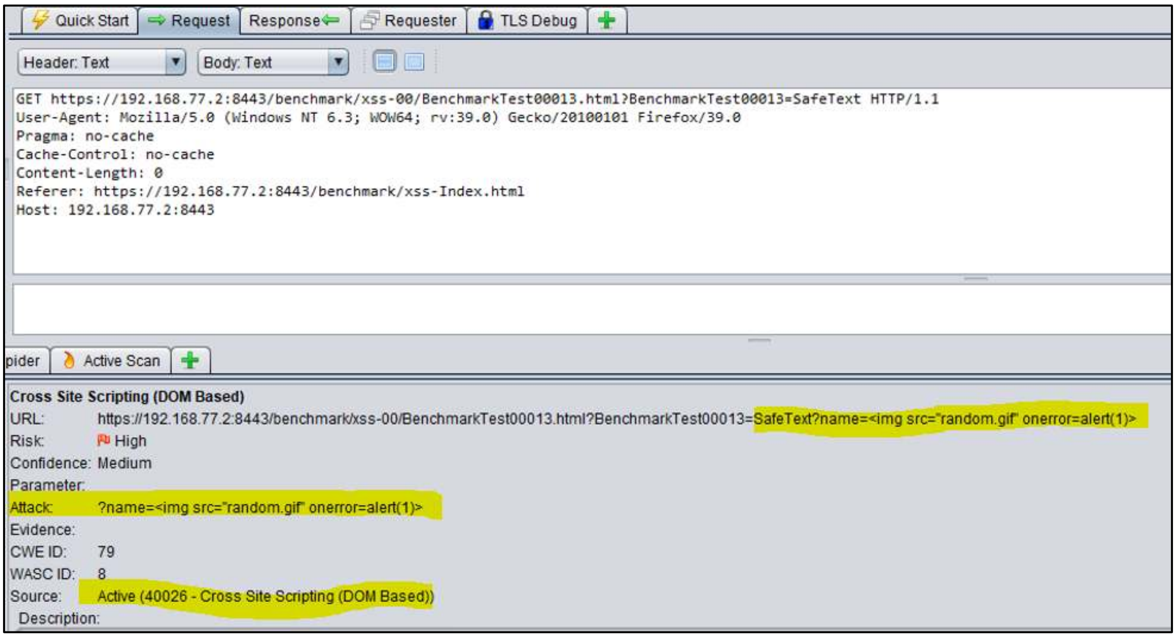
Figure 24: ZAP successful SQL injection on OWASP Benchmark test number 2187

It is evident in figure 24 above, ZAP used the following Boolean conditions: [(select (case when (9424=9424) then 9424 else 9424*(select 9424 from information_schema.character_sets) end))] and [(select (case when (1556=1024) then 1556 else 1556*(select 1556 from information_schema.character_sets) end))] to successfully manipulate the page results. By doing this, ZAP, therefore, restricted the data originally returned by manipulating the parameter.

The modified values were stripped from the returned HTML output for comparison purposes. Using this method, ZAP was able to detect 597 positive SQL Injection OWASP Benchmark test cases. The above results show that in web application security the client-side input is not always to be trusted even if client-side validation is in place. Therefore, server-side validation of all data is necessary to avoid such attacks.

d. *Cross Site Scripting(XSS)*

In this category, ZAP discovered over 186 OWASP Benchmark URLs that were vulnerable to Cross Site Scripting attacks. Figure 25 below shows how ZAP executed the attack to discover the vulnerable test cases.



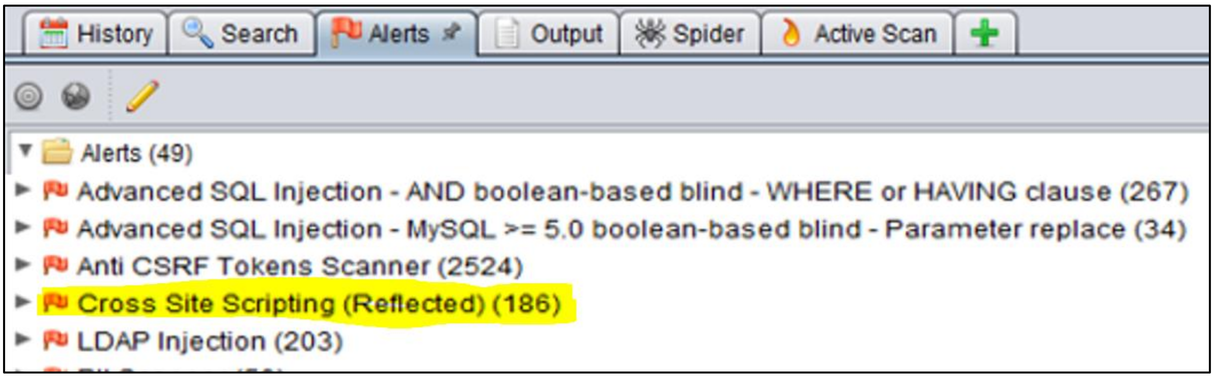
```
GET https://192.168.77.2:8443/benchmark/xss-00/BenchmarkTest00013.html?BenchmarkTest00013=SafeText HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:39.0) Gecko/20100101 Firefox/39.0
Pragma: no-cache
Cache-Control: no-cache
Content-Length: 0
Referer: https://192.168.77.2:8443/benchmark/xss-Index.html
Host: 192.168.77.2:8443
```

```
Cross Site Scripting (DOM Based)
URL:        https://192.168.77.2:8443/benchmark/xss-00/BenchmarkTest00013.html?BenchmarkTest00013=SafeText?name=<img src="random.gif" onerror=alert(1)>
Risk:       High
Confidence: Medium
Parameter:
Attack:     ?name=<img src="random.gif" onerror=alert(1)>
Evidence:
CWE ID:     79
WASC ID:    8
Source:     Active (40026 - Cross Site Scripting (DOM Based))
 Description:
```

Figure 25: ZAP Cross Site Scripting attack on OWASP Benchmark Test Case Number 0013

ZAP applied an attack technique that involves echoing a code into the browser instance as shown in the highlights in figure 25 above. Similarly, in real life, when an attacker succeeds to get the target browser to execute his or her code, the code will run within the security context of the hosting website, therefore making the attack possible.

To avoid this type of attack, it is essential to get an understanding of the context in which the application's data will be used, and the expected encoding. This is especially vital when data is transmitted between different devices, or when outputs that contain multiple encoding is generated simultaneously such as multi-part mail messages or web pages. Figure 26 below shows the number of positive XSS cases detected:



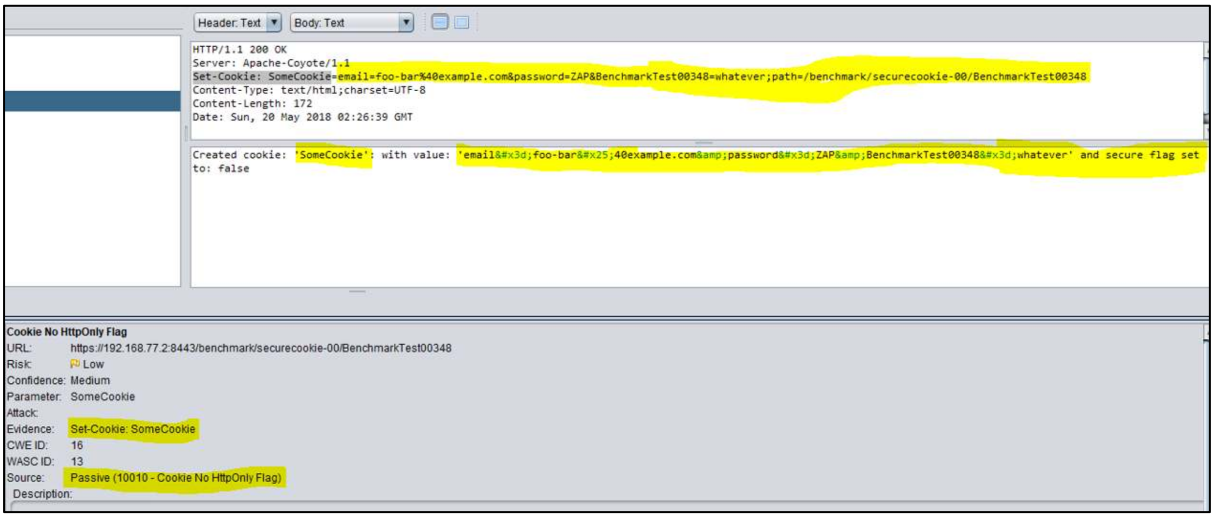
```
▼ Alerts (49)
► Advanced SQL Injection - AND boolean-based blind - WHERE or HAVING clause (267)
► Advanced SQL Injection - MySQL >= 5.0 boolean-based blind - Parameter replace (34)
► Anti CSRF Tokens Scanner (2524)
► Cross Site Scripting (Reflected) (186)
► LDAP Injection (203)
```

Figure 26: The number of Detected Cross Site Scripting Cases

e. Insecure Cookies

In this category, ZAP has identified that in the discovered URLs cookies have been set without a secure flag, meaning that these cookies can be accessed through unencrypted connections. When a cookie is set without an HTTPOnly flag, JavaScript can be used to access it. Meaning that a malicious script can be run on the page and the cookie can be accessed and can be transmitted to another site which can result in session hijacking if this is a session cookie.

Figure 27 below shows how ZAP discovered some insecure cookies in some the discovered test cases under this category.



```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: SomeCookie=email=foo-bar%40example.com&password=ZAP&BenchmarkTest00348=whatever;path=/benchmark/securecookie-00/BenchmarkTest00348
Content-Type: text/html;charset=UTF-8
Content-Length: 172
Date: Sun, 20 May 2018 02:26:39 GMT
```

```
Created cookie: 'SomeCookie': with value: 'email&#x3d;foo-bar&#x25;40example.com&amp;password&#x3d;ZAP&amp;BenchmarkTest00348&#x3d;whatever' and secure flag set
to: false
```

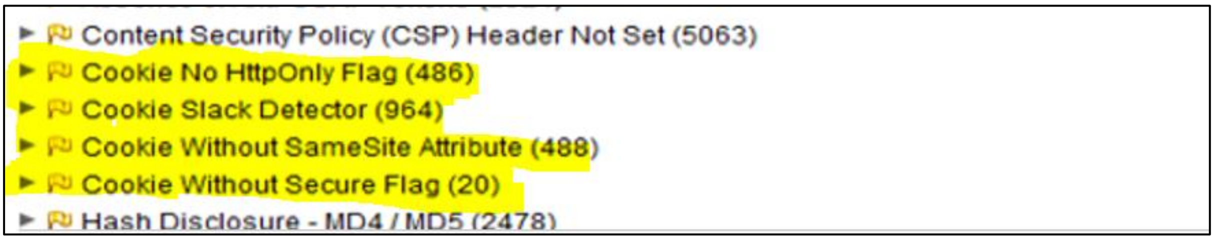
Cookie No HttpOnly Flag

URL:	https://192.168.77.2:8443/benchmark/securecookie-00/BenchmarkTest00348
Risk:	Low
Confidence:	Medium
Parameter:	SomeCookie
Attack:	
Evidence:	Set-Cookie: SomeCookie
CWE ID:	16
WASC ID:	13
Source:	Passive (10010 - Cookie No HttpOnly Flag)

Description:

Figure 27: ZAP detection of insecure cookies in OWASP Benchmark test cases

As shown in figure 27, ZAP successfully discovered insecure cookie stored in the variable some cookie set without HTTP Only flag. Therefore, the page is susceptible to malicious exploitation. Therefore, to avoid this kind of vulnerability, all cookie should be set with HTTP only flag. The number of detected insecure cookies by ZAP in the OWASP Benchmark test cases are listed in figure 28 below:

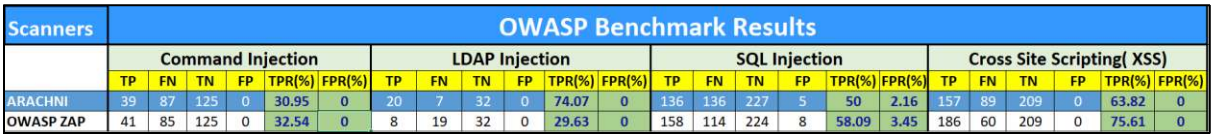


- Content Security Policy (CSP) Header Not Set (5063)
- Cookie No HttpOnly Flag (486)
- Cookie Slack Detector (964)
- Cookie Without SameSite Attribute (488)
- Cookie Without Secure Flag (20)
- Hash Disclosure - MD4 / MD5 (2478)

Figure 28: Number of positive insecure cookies test cases detected by ZAP

4.2 Comparison of Arachni and ZAP

The results of the scanners are executed against OWASP Benchmark. Table 6 below shows the benchmark detection results. For each web vulnerability scanner and vulnerability types, some metrics including TP, FN, TN, FP, TPR, and FNR were calculated. The table 6 below shows a summary of the detected results. The values in bold type with a light green background indicate the detection rate of each scanner in each category, the others are the values of the TP, FN, TN, and FP found.



Scanners	OWASP Benchmark Results																							
	Command Injection						LDAP Injection						SQL Injection						Cross Site Scripting(XSS)					
	TP	FN	TN	FP	TPR(%)	FPR(%)	TP	FN	TN	FP	TPR(%)	FPR(%)	TP	FN	TN	FP	TPR(%)	FPR(%)	TP	FN	TN	FP	TPR(%)	FPR(%)
ARACHNI	39	87	125	0	**30.95**	**0**	20	7	32	0	**74.07**	**0**	136	136	227	5	**50**	**2.16**	157	89	209	0	**63.82**	**0**
OWASP ZAP	41	85	125	0	**32.54**	**0**	8	19	32	0	**29.63**	**0**	158	114	224	8	**58.09**	**3.45**	186	60	209	0	**75.61**	**0**

Table 6: Arachni and ZAP Benchmark detection results in four selected categories

On each category in table 6 above, OWASP Benchmark applied the previously discussed metrics to obtain the most appropriate measures to score each scanner to promote a reasonable interpretation of results and draw sound conclusions. OWASP Benchmark, therefore, produces scorecards that highlight the overall performance of each scanner in every category. OWASP Benchmark score is the normalized distance from the random guess line which the difference between a scanner's TPR and FPR (**Score = TPR-FPR**).

4.2.1 Command Injection

Figure 29 below shows Arachni and ZAP scores in the Command Injection category.

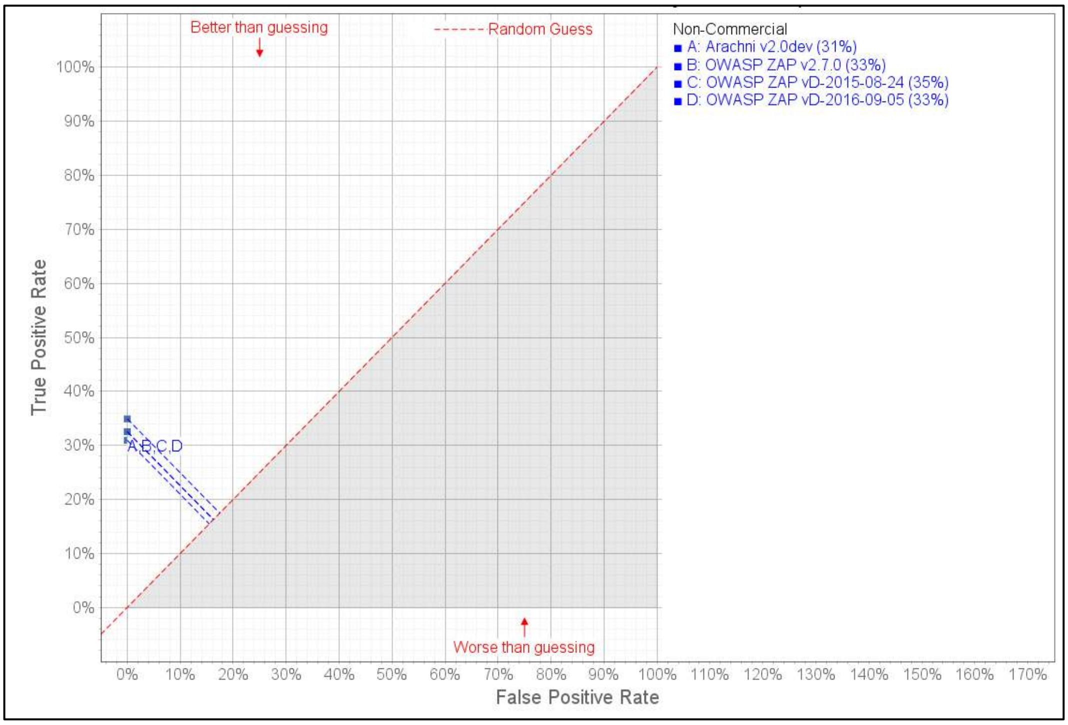


Figure 29: OWASP Benchmark Comparison Scores for Command Injection

As it can be seen in figure 29 above, OWASP ZAP outperformed Arachni with 33% score. Nonetheless, the previous version of ZAP(2.5) performed better in this category with 35% as compared to the other versions. The performance of the latest version of ZAP in this category has not been as expected as it can be seen in figure 29 above. Consequently, we have taken the initiative to submit these results to both Dave Wrenchers the Project Leader of OWASP Benchmark as well as Simon Bennetts Project Leader of OWASP ZAP for their insight on what might be the cause of ZAP underperforming in this category.

4.2.2 LDAP Injection

The figure 30 below shows OWASP Benchmark LDAP Injection scores for both scanners

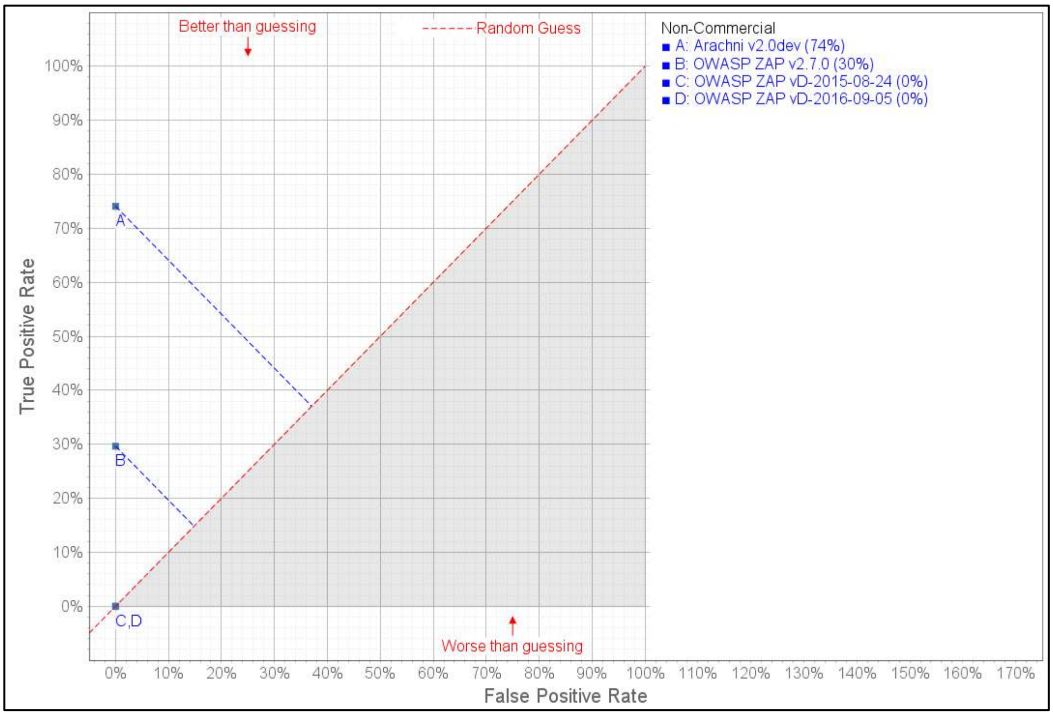


Figure 30: OWASP Benchmark LDAP Injection Comparison

It is evident that as shown in figure 20 above that Arachni has the highest score of 74% as compared to 30 % score of ZAP. However, it is noticeable that there has been a significant improvement in ZAP performance in this category considering 0% score of its previous versions.

4.2.3 SQL Injection

The next figure highlights the OWASP Benchmark scores in the SQL Injection vulnerability category for ZAP and Arachni:

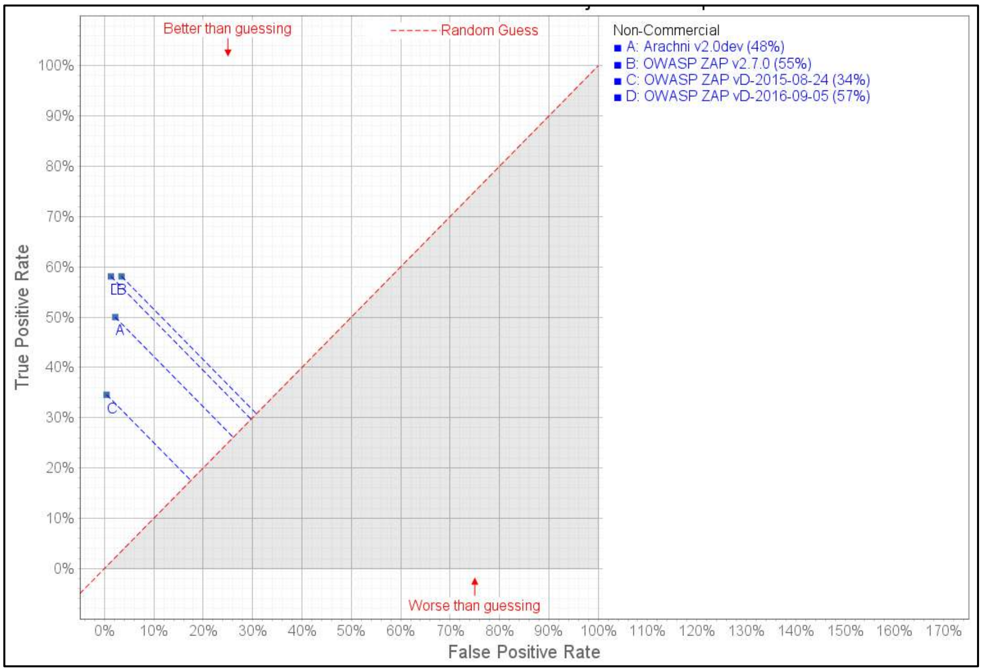


Figure 31: OWASP Benchmark Comparison Scores of Arachni and ZAP for SQL Injection

As it can be seen in the above figure 31, ZAP has performed better than Arachni in this category with 55% and 48% detection score respectively.

4.2.4 Cross Site Scripting (XSS)

The performance results of the scanners In Cross Site Scripting category is highlighted in the figure below:

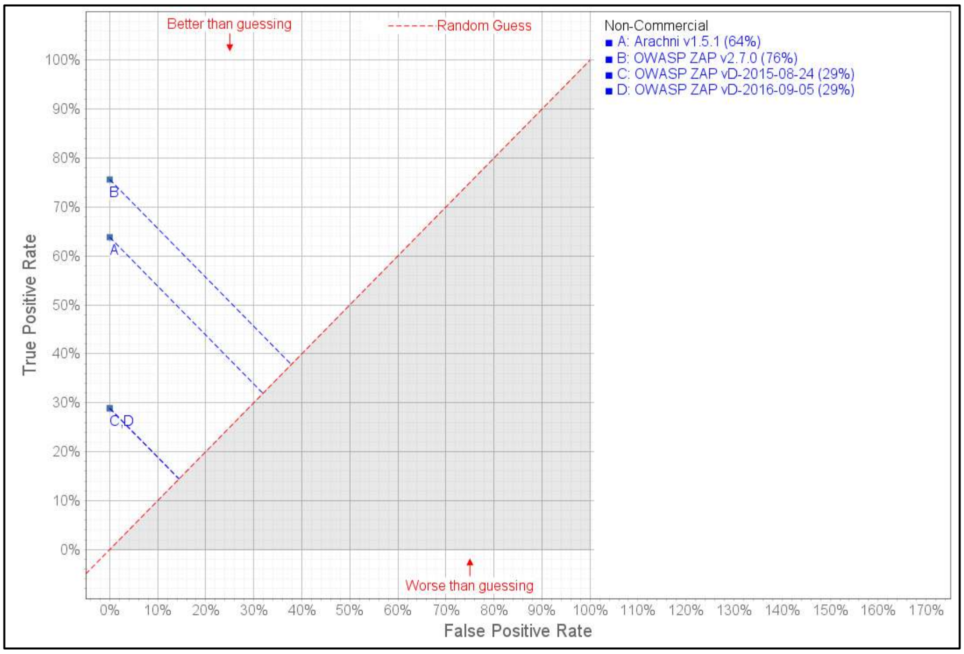


Figure 32: OWASP Benchmark Comparison Scores of Arachni and ZAP for Cross-Site Scripting

It is evident in figure 32 above that ZAP has performed better than Arachni in Cross-Site Scripting category with a detection accuracy score of 76% as compared to 64% detection accuracy score of Arachni. Once more, there has been a significant improvement in ZAP performance in this category as compared to the results of the previous Version of ZAP with the score of 29% detection accuracy rate for both 2.5 and 2.6.

The analysis of the obtained experimental results above has allowed us to get an overview of the performances of Arachni and ZAP related to Command Injection, LDAP Injection, SQL injection and Cross-Site Scripting.

Figure 33 below give a close comparison of the two scanners performance in the categories mentioned above:

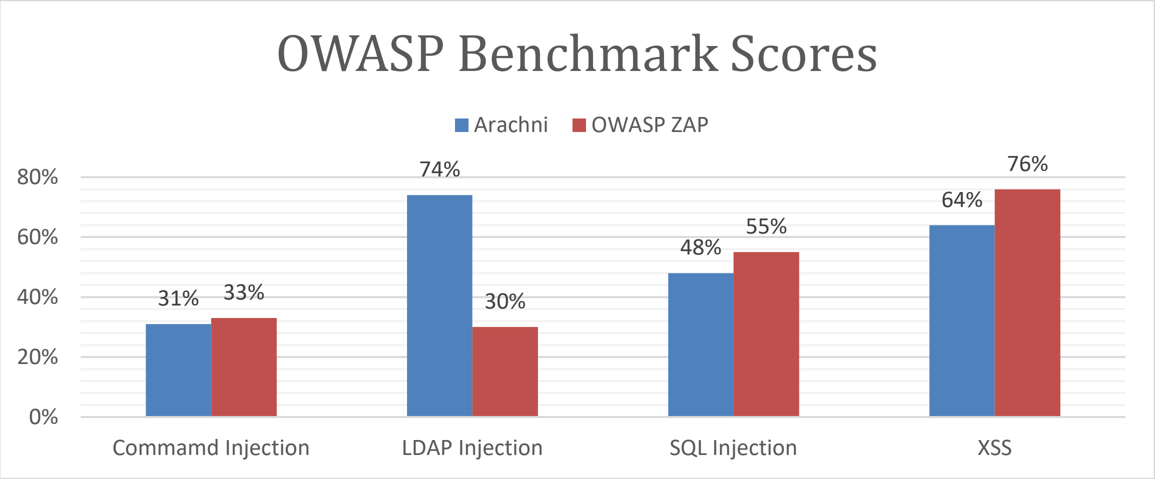


Figure 33: Side by side Comparison of OWASP Benchmark Scores for Arachni and ZAP in each category

As it can be seen in the above chart, scanners performed differently in each type of vulnerability. It has been deduced that Arachni had the highest score in LDAP injection of 74%. OWASP ZAP, on the other hand, outperformed Arachni in Command Injection, SQL Injection and XSS categories with the score of 33%, 55%, and 76% respectively. Although each scanner outperformed the other some categories, it is worth considering the percentage difference in the scores for a better evaluation of the performance of the scanners in each of the categories. To that end, it can be seen that ZAP performance was 2%,7%, and 12% higher than Arachni in its winning categories while Arachni scored 44% higher than ZAP in its winning category. This shows that although there is a need for both scanners to uplift their performance in their losing categories, it is evident that more work is needed in raising ZAP performance in its underperforming category as compared to Arachni. Table 7 below shows a summary of the differences in performance of the scanners in each category

CATEGORY	Scanners Performance Difference	
	ARACHNI	**OWASP ZAP**
Command Injection	-	2%
LDAP Injection	44%	-
SQL Injection	-	7%
XSS	-	12%
Total	44%	21%

Table 7: Arachni and ZAP perforce differences

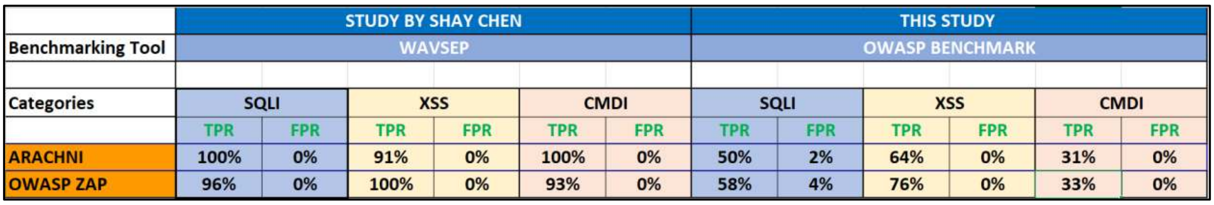
The above table shows that 44% work needs to be done in ZAP as compared to 21% in Arachni meaning that 22% more work is needed for ZAP to perform at the same level as Arachni in its losing category.

4.3 Comparison with WAVSEP benchmark Results

As mentioned in earlier chapters, the evaluation of Arachni and ZAP have been done before. However, WAVSEP benchmark has been used as the benchmark in these studies. In contrast, this study has evaluated these scanners based on OWASP benchmark. To highlight the importance of using a variety of Benchmarks to get an overall conclusion in the evaluation of the effectiveness of web application vulnerability scanners, we have compared our OWASP Benchmark results of Arachni and ZAP to a previous study that have evaluated these canners based on WAVSEP benchmark. We have therefore chosen the latest study by Shay Chen for this purpose. Our choice of Shay Chen's study was based on the accuracy of his results, and his reputation as a widely respected Information Security Researcher and author of WAVSEP benchmark. Additionally, his benchmarking results have never been contrasted with results based on other benchmarks.

Although our study has examined four critical categories, only three categories including SQLI, XSS, and CMDI will be considered for comparison purpose. This is because LDAP category has not been examined in Chen's study.

Table 8 below gives an overview of Chen's results and our results.



	STUDY BY SHAY CHEN						THIS STUDY					
Benchmarking Tool	WAVSEP						OWASP BENCHMARK					
Categories	SQLI		XSS		CMDI		SQLI		XSS		CMDI	
	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR
ARACHNI	100%	0%	91%	0%	100%	0%	50%	2%	64%	0%	31%	0%
OWASP ZAP	96%	0%	100%	0%	93%	0%	58%	4%	76%	0%	33%	0%

Table 8: Comparison Summary of Our Results to Previous study results by Shay Chen

A close examination of the results in Table 8 above demonstrates that there is some similarity in the performance pattern of the scanners in some categories such as XSS. However, there is a significant variation in detection rate and dissimilarities of scanners performance in some other categories such as SQLI. This variation is verifiable by examining Chen's results of XSS category which shows that ZAP had a 100% accuracy score and Arachni 91% whereas our results indicate that ZAP scored 76% and Arachni 64% in the same category. In SQLI on the other hand, our results indicate that ZAP has performed better than Arachni 58% and 50% respectively whereas Chen's results show the opposite (ZAP 96% and Arachni 100%). This difference in results, however, can be explained by the fact that our results were obtained from the latest version of ZAP (2.7) while Chen's study examined the previous version of ZAP (2.6). Moreover, our results have demonstrated that there has been much improvement in the performance of the current version of ZAP as compared to its predecessor in some categories. Nevertheless, there is still much difference in the score numbers in our results and those of Chen which is 100% for Arachni, 96 % for ZAP and 58% for ZAP and 50% for Arachni in our results.

What is interesting, however, is that the differences in the scanners performance scores in our results and Chen's results are both averaging to 3.5%. In other categories, ZAP outperformed Arachni by 9 % and 12% in XSS in Chen's results and our results respectively. Additionally, in SQLI there is 8% and 4 % performance difference in our results and Chen's results respectively. The graphical representation of these comparison results for each category is shown in the subsections below.

4.3.1 SQL Injection Comparison

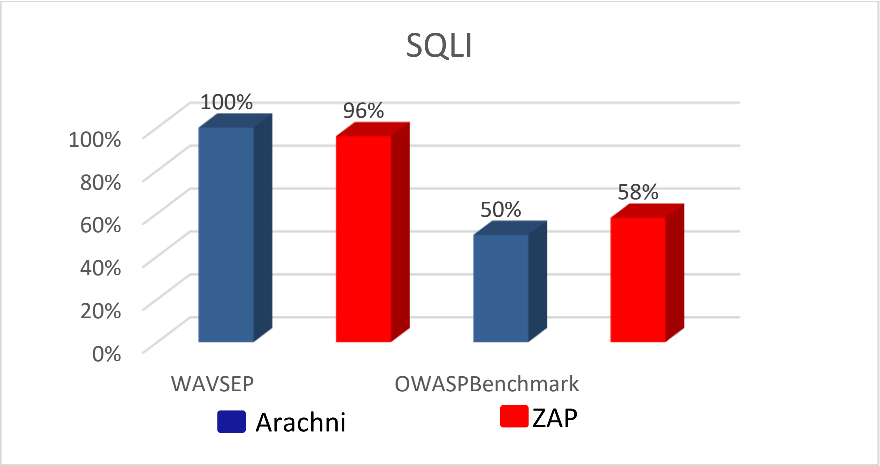


Figure 34: SQLI comparison results

As it can be seen in figure 30 above, there is a contrast between OWASP benchmark and WAVSEP benchmark results. Arachni outperformed ZAP by 4% in WAVSEP benchmark results whereas OWASP benchmark results indicate that ZAP outperformed Arachni by 8% in this category. Although it can be seen that Arachni has outperformed ZAP in the existing WAVSEP benchmark results with a score of 100% and 96% respectively, we consider OWASP benchmark results. This is because OWASP benchmark examined the latest version of ZAP whereas existing WAVSEP benchmark study examined an older version of ZAP. Furthermore, our discussion of OWASP benchmark results in chapter 4 has confirmed that there has been a significant improvement in the examined version of ZAP as compared to previous versions.

4.3.2 Cross Site Scripting (XSS) Comparison

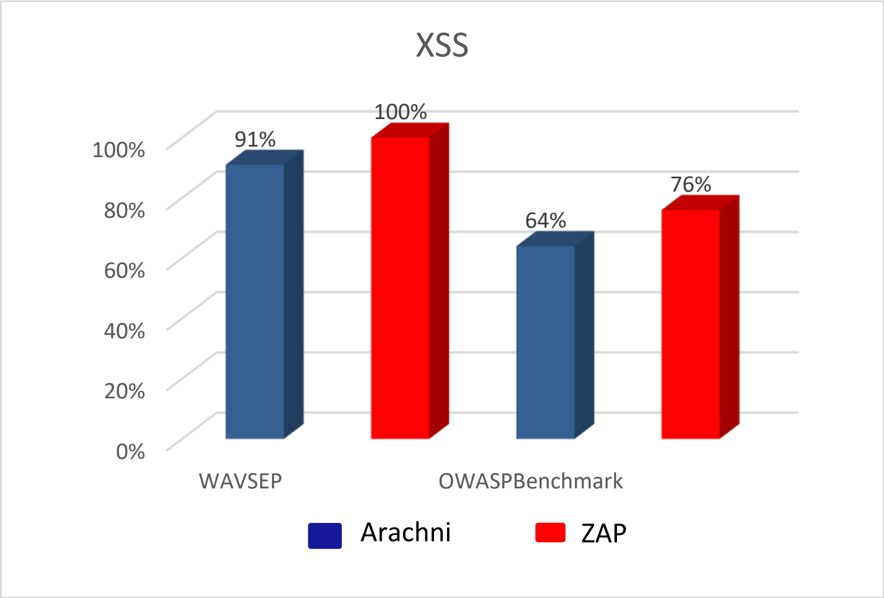


Figure 35: XSS Comparison Results

Figure 35 above clearly show that in this category ZAP performed better than Arachni both in Chen's results and our results. However, there is a difference in the detection rates in both results of 91% and 64% for Arachni in Chen result and our results respectively and 100% and 76% for ZAP. Once again this might be influenced by the use of different benchmarks.

4.3.3 Command Injection (CMDI)

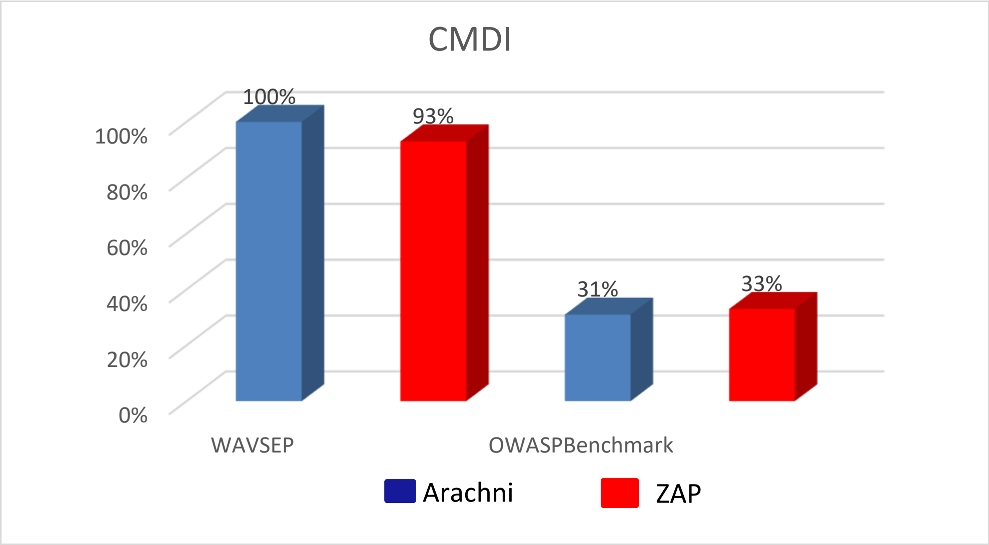


Figure 36: XSS Comparison Results

As it can be seen in this category, Arachni outperformed ZAP in WAVSEP benchmark results with 100% and 93% detection rates respectively, whereas the opposite occurred in OWASP benchmark results with ZAP scoring 33% and Arachni 31%. Although the scanners performance differences are not significant for both WAVSEP and OWASP benchmarks (with a difference of 7% and 2% respectively), WAVSEP benchmark detection rate for both scanners is three times higher than OWASP benchmark with an average of 96.5% and 32 % respectively.

CHAPTER 5 CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

In this thesis, the evaluation of the effectiveness of OWASP ZAP and Arachni based on OWASP benchmark was conducted. The variations in these scanners performance in different vulnerability categories were experimentally demonstrated. While previous studies have mainly paid attention to WAVSEP benchmark to evaluate scanners effectiveness, OWASP benchmark has never been used to evaluate Arachni and the latest version(V.2.7) of ZAP before. Thus, in this thesis, we have investigated the importance of using different benchmarks to evaluate the effectiveness of web application vulnerability scanners by comparing our OWASP benchmark results with existing WAVSEP benchmark results. This comparison is the first such study about these two benchmarks in literature. Our comparison results between these two benchmarks strongly support our claim that to obtain the best understanding of scanner effectiveness, multiple benchmarks should be used to evaluate scanners.

Besides concluding that no scanners suit all and multiple benchmarks should be used together in general, we also make recommendations on the following:

- Which scanner is better in a particular vulnerability category
- Which benchmark is stronger in particular vulnerability category
- Places to improve for vulnerability scanners and for benchmarks

1. Better Scanner for a Vulnerability Category

The results obtained in this study has revealed that scanners perform differently in different categories. Therefore, no scanner can be considered an all-rounder in scanning web vulnerabilities. Moreover, it was found that performances of scanners vary depending on the benchmark used for the evaluation. However, considering scanners performance in different categories, we have concluded that ZAP has performed better than Arachni in SQLI, XSS and CMDI categories.

Additionally, our results confirmed that there had been much improvement in this version of ZAP compared with its previous versions in the categories of SQLI, LDAP, and XSS as highlighted in Chapter 4. Arachni, on the other hand, performed much better in LDAP category with a score of 74%, which is about 2.5 times of the ZAP score of 30%. However, this conclusion is only based on OWASP benchmark results because the existing WAVSEP benchmark results did not include this category.

2. Stronger Benchmarks for a Vulnerability Category

We have mentioned that the performance evaluation results of each scanner vary depending on the benchmarks used. These variations are due to the number of test cases in each vulnerability category as well as the complexity and difficulty of test cases.

Specifically, our results of benchmarks comparison revealed that for both scanners and all the three vulnerability categories compared, the scores under WAVSEP benchmark are much higher than those under OWASP benchmark. Using the criterion that if benchmark A contains more cases that fail a scanner than benchmark B, we say benchmark A is stronger than benchmark B, we can conclude that OWASP benchmark is stronger than WAVSEP benchmark in all the three vulnerability categories evaluated in this thesis.

Although it is shown that OWASP benchmark is stronger than WAVSEP benchmark under the above criterion, there are still benefits in evaluating vulnerability scanners using both benchmarks simultaneously. These benefits include:

- Encouraging continuous improvement of vulnerability scanners effectiveness as a countermeasure to hacking activities that are becoming more sophisticated. As striving towards secure web application is a never-ending process that needs effective vulnerability scanners, benchmarking is one of the techniques that will encourage this by unveiling the scanner effectiveness in various ways.
- Additionally, Effective benchmarking will give web application testers a clear choice of what scanner can be used to find vulnerabilities in a particular category effectively.

3. **Places to improve for vulnerability scanners and benchmarks**

Based on the evaluation results of this thesis, we noticed the following places for scanners and benchmarks to improve.

For scanners, the places include improving crawling mechanism to guarantee the discovery of all URLs of the target applications without any omission and strengthening the scanners vulnerability databases to increase the coverage of vulnerabilities.

For benchmarks, it is necessary to improve the design of tests cases for evaluating the scanners with more complicated vulnerabilities in different categories.

5.2 Future work

Further studies can be considered from this work. Firstly, the effectiveness of web vulnerability scanners will be evaluated in all the possible vulnerability categories based on WAVSEP and OWASP benchmarks, while in this thesis, only four of the categories were examined, and only three were examined based on the two benchmarks. Secondly, the coverage of scanners vulnerability databases should be improved to increase the detection accuracy. Finally, Artificial Intelligence (especially Machine Learning) will be integrated into scanners to boost their capabilities to identify unknown vulnerabilities in web applications.

REFERENCES

[1] S. E. Idrissi, N. Berbiche, F. G. and, and M. Sbihi, "Performance Evaluation of Web Application Security Scanners for Prevention and Protection against Vulnerabilities.pdf," *International Journal of Applied Engineering Research,* vol. 12, pp. 11068-11076, 2017.

[2] Core_Security. (2018). *What is Penetration Testing?* Available: https://www.coresecurity.com/content/penetration-testing

[3] T. Laskos. (2017). *Arachni Apllication Security Scanner Framework*. Available: http://www.arachni-scanner.com/

[4] INFOSEC_Institute. (2016). *The History of Penetration Testing*. Available: http://resources.infosecinstitute.com/the-history-of-penetration-testing/#gref

[5] OWASP. (2016). *Fuzzing*. Available: https://www.owasp.org/index.php/Fuzzing

[6] Z. T. Watson_ C., "Automated-threat-handbook," 2016.

[7] A. C. Barus, D. I. P. Hutasoit, J. H. Siringoringo, and Y. A. Siahaan, "White box testing tool prototype development," in *2015 International Conference on Electrical Engineering and Informatics (ICEEI)*, 2015, pp. 417-422.

[8] S. Xu, L. Chen, C. Wang, and O. Rud, "A comparative study on black-box testing with open source applications," in *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 2016, pp. 527-532.

[9] Information Security Stack Exchange. (2017). *Effectiveness of Interactive Application Security Testing*. Available: https://security.stackexchange.com/questions/54865/effectiveness-of-interactive-application-security-testing

[10] P. E. Black, "Static Analyzers in Software Engineering.pdf," National Institute of Standards and Technology2009.

[11] Skoussa. (2018, January). *What do SAST, DAST, IAST and RASP mean to developers?* Available: https://www.softwaresecured.com/what-do-sast-dast-iast-and-rasp-mean-to-developers/

[12] Y. Wang and J. Yang, "Ethical hacking and network defense: Choose your best network vulnerability scanning tool," in *Proceedings - 31st IEEE International Conference on Advanced Information Networking and Applications Workshops, WAINA 2017*, 2017, pp. 110-113.

[13] OWASP. (2017). *OWASP Top Ten Project*. Available: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2017_Release_Candidate_1

[14] OWASP. (2016). *Cross Site Scripting*. Available: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

[15] PortSwigger_Ltd. (2018, 2018). *SQL injection*. Available: https://portswigger.net/kb/issues/00100200_sql-injection

[16] R. K., "A benchmark approach to analyse the security of web frameworks," Master, Computer Science, Radboud University Nijmegen, Nijmegen, Netherlands, 2014.

[17] OWASP. (2016). *SQL Injection*. Available: https://www.owasp.org/index.php/SQL_Injection

[18] Infosec_Institute. (2018). *File-Inclusion Attack*. Available: http://resources.infosecinstitute.com/file-inclusion-attacks/#gref

[19] M. El, E. McMahon, S. Samtani, M. Patton, and H. Chen, "Benchmarking vulnerability scanners: An experiment on SCADA devices and scientific instruments," in *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2017, pp. 83-88.

[20] PENTESTGEEK. (2018). *WHAT IS BURP SUITE*. Available: https://www.pentestgeek.com/what-is-burpsuite

[21] w3af.org. (2013). *Web Application Attack and Audit Framework*. Available: http://w3af.org/

[22] Wikipedia. (2017). *W3af*. Available: https://en.wikipedia.org/wiki/W3af

[23] N. Surribas. (2018). *Wapiti : The web-application vulnerability scanner*. Available: http://wapiti.sourceforge.net/

[24] F. Duchene, R. Groz, S. Rawat, and J. L. Richier, "XSS Vulnerability Detection Using Model Inference Assisted Evolutionary Fuzzing," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, 2012, pp. 815-817.

[25] 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)*, 2015, pp. 399-402.

[26] Micosoft. (2018). *Establishing an LDAP Session*. Available: https://msdn.microsoft.com/en-us/library/aa366102(v=vs.85).aspx

[27] N. Antunes and M. Vieira, "On the Metrics for Benchmarking Vulnerability Detection Tools," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 505-516.

[28] J. S. Akosa, "<Predictive Accuracy: A Misleading Performance Measure for Highly Imbalanced Data.pdf>," 2017.

[29] OWASP. (2017`). *OWASP Benchmark*. Available: https://www.owasp.org/index.php/Benchmark

[30] A. Baratloo, M. Hosseini, A. Negida, and G. El Ashal, "Part 1: Simple Definition and Calculation of Accuracy, Sensitivity and Specificity," *Emergency,* vol. 3, pp. 48-49, Spring
12//received
02//accepted 2015.

[31] N. I. Daud, K. A. A. Bakar, and M. S. M. Hasan, "A case study on web application vulnerability scanning tools," in *2014 Science and Information Conference*, 2014, pp. 595-600.

[32] S. Chen. (2017). *Price and Feature Comparison of Web Application Scanners*. Available: http://sectoolmarket.com/price-and-feature-comparison-of-web-application-scanners-unified-list.html

[33] A. Alzahrani, A. Alqazzaz, Y. Zhu, H. Fu, and N. Almashfi, "Web Application Security Tools Analysis," in *2017 ieee 3rd international conference on big data security on cloud (bigdatasecurity), ieee international conference on high performance and smart computing (hpsc), and ieee international conference on intelligent data and security (ids)*, 2017, pp. 237-242.

[34] D. Subramanian, H. T. Le, P. K. K. Loh, and A. B. Premkumar, "Quantitative Evaluation of Related Web-Based Vulnerabilities," in *2010 Fourth International Conference on Secure Software Integration and Reliability Improvement Companion*, 2010, pp. 118-125.

[35] Darknet. (2017). *wavsep-web-application-vulnerability-scanner-evaluation-project*. Available: https://www.darknet.org.uk/2011/09/wavsep-web-application-vulnerability-scanner-evaluation-project/

[36] OWASP. (2018). *OWASP Zed Attack Proxy Project*. Available: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

[37] Sarosys_LLC. (2017). *Arachni Web Application Security Scanner Framework*. Available: http://www.arachni-scanner.com/Sarosys LLC

[38] P. J. Fleming and J. J. Wallace, "How not to lie with statistics: the correct way to summarize benchmark results," *Communications of the ACM,* vol. 29, pp. 218-221, 1986.

[39] w3af.org. (2018). *Web Application Payloads*. Available: http://docs.w3af.org/en/latest/advanced-exploitation.html