applied sciences

MDPI

Article

On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications

Francesc Mateo Tudela [1], Juan-Ramón Bermejo Higuera [1,*], Javier Bermejo Higuera [1], Juan-Antonio Sicilia Montalvo [1,*] and Michael I. Argyros [2]

[1] Escuela Superior de Ingeniería y Tecnología, Universidad Internacional de La Rioja, Avda. de La Paz 137, 26006 Logroño, La Rioja, Spain; fmateo@gmail.com (F.M.T.); javier.bermejo@unir.net (J.B.H.)
[2] Department of Computing and Technology, Cameron University, Lawton, OK 73505, USA; Michael.Argyros@cameron.edu
* Correspondence: juanramon.bermejo@unir.net (J.-R.B.H.); juanantonio.sicilia@unir.net (J.-A.S.M.)

Received: 1 December 2020; Accepted: 18 December 2020; Published: 20 December 2020

Featured Application: This document provides a complete comparative study of how different types of security analysis tools, (static, interactive and dynamic) can combine to obtain the best performance results in terms of true and false positive ratios taking into account different degrees of criticality.

Abstract: The design of the techniques and algorithms used by the static, dynamic and interactive security testing tools differ. Therefore, each tool detects to a greater or lesser extent each type of vulnerability for which they are designed for. In addition, their different designs mean that they have different percentages of false positives. In order to take advantage of the possible synergies that different analysis tools types may have, this paper combines several static, dynamic and interactive analysis security testing tools—static white box security analysis (SAST), dynamic black box security analysis (DAST) and interactive white box security analysis (IAST), respectively. The aim is to investigate how to improve the effectiveness of security vulnerability detection while reducing the number of false positives. Specifically, two static, two dynamic and two interactive security analysis tools will be combined to study their behavior using a specific benchmark for OWASP Top Ten security vulnerabilities and taking into account various scenarios of different criticality in terms of the applications analyzed. Finally, this study analyzes and discuss the values of the selected metrics applied to the results for each n-tools combination.

Keywords: web application; security vulnerability; analysis security testing; static analysis security testing; dynamic analysis security testing; interactive analysis security testing; assessment methodology; false positive; false negative; tools combination

1. Introduction

In recent years, the use of web applications has increased in many types of organizations, such as public and private, government, critical infrastructures, etc. These applications have to be continuously developed in the shortest time possible to face the competitors. Therefore, developers make programing security vulnerabilities or use third-party modules or components that are vulnerable. Sometimes they have limited budgets. These cases often make them forget an essential component within the

development life cycle—security. In addition, companies find that developers do not have enough security knowledge, which increases the risk of producing insecure developments.

Taking into account the different types of analysis security testing (AST) [1] and the great amount of security vulnerabilities that web applications have in their code and configurations, it seems necessary that the security analyst making a security analysis should select the best static (SAST), dynamic (DAST) and interactive (IAST) analysis security testing tools in combination. The results of several studies [2–4] confirm that the web applications analyzed did not pass the OWASP Top Ten project [5]. SQL injection (SQLI) and cross site scripting (XSS) vulnerabilities continue to be the most frequent and dangerous vulnerabilities.

Several tools of the same type can be combined to achieve a better performance in terms of true and false positives [6–9]. Several works have also shown that better ratios of true and false positives can be obtained by combining different types of techniques to leverage the synergies that different type of tools can have [10–12]. These works show how a tools combination can reduce false positives (detected vulnerabilities that do not really exist) and false negatives (real vulnerabilities not found). Analyzed works concluded that each security vulnerability included in an AST tool report, including manual reviews, is required to be verified. False positives are not really a danger and can be fixed by the security analyst. However, a false negative is more difficult to find if the tool has not previously detected it, causing a real danger. These techniques include the use of static white box security analysis (SAST), dynamic black box security analysis (DAST) or interactive white box security analysis (IAST) tools. Manual analysis requires highly specialized staff and time. To carry out an analysis of web application security, using any method, it is necessary to cover the entire attack surface accessing all parts and application layers and using tools to automate security analysis as much as possible. There are some questions to investigate:

- How is each type of AST tool's average effectiveness, considering each type of vulnerability without combination?
- How is each type of AST tool's combinations average effectiveness, considering each type of vulnerability and the number of tools in a combination?
- How is each type of AST tool's average effectiveness at detecting OWASP Top Ten security category vulnerabilities without combination?
- How is the n-tools combination's average effectiveness at detecting OWASP Top Ten security vulnerabilities computing different metrics?
- Which are the best combinations for analyzing the security of web applications with different levels of criticality?

It is necessary to establish in the organizations a Software Development Life Cycle (SSDLC), as defined in the work of Vicente et al. [13], in order to standardize the use of SAST, DAST and IAST tools with the objective of deploying in production web applications as secure as possible.

This work aims to be the first of its kind—to study the best way to combine the three types of security analysis tools for web applications. Therefore, the first goal of this study is to investigate the behavior of the combination of two static tools (Fortify SCA by Microfocus, Newbury, United Kingdom, and FindSecurityBugs, OWASP tool created by Philippe Arteau, licensed under LGPL), two dynamic tools (OWASP ZAP open source tool with Apache 2 licenseand Arachni open source tool with public source License v1.0 created by Tasos Laskos) and two interactive tools (Contrast Community Edition by Contrasst Security, Los Altos, EE.UU. and CxIAST by Chekcmarx, Raman Gan, Israael) using a new specific methodology and a test bench for Java language with test cases for the main security vulnerabilities. We investigate specifically the security performance of two and three AST tools combinations, selecting adequate metrics. The second main goal of this study is to select the best combinations of tools taking into account different security criticality scenarios in the analyzed applications. To investigate the best combinations of AST tools according to each of the three levels that

are criticality established, different metrics are selected considering the true and false positive ratios and the time available to perform a subsequent audit to eliminate the possible false positives obtained.

The tools are executed against the OWASP Benchmark project [14] based on the OWASP Top Ten project [5] to obtain the results of n-tools combination effectiveness. Well-known metrics are selected for the execution results to obtain a strict rank of combination tools. Finally, the paper gives some practical recommendations about how to improve their effectiveness using the tools in combination.

Therefore, the contributions of this paper can be summarized as follows:

- A concrete methodology using the OWASP Benchmark project to demonstrate its feasibility evaluating n-tools combinations by the detection of web vulnerabilities using appropriate criteria for benchmark instantiation.
- An analysis of the results obtained by n-tools in combination using a defined comparative method to classify them according to different degree levels of web applications importance using carefully selected metrics.
- An analysis of results of leading commercial tools in combination for allowing the practitioners to choose the most appropriate tools to perform a security analysis of a web application.

The outline of this paper is as follows: Section 2 reviews background in web technologies security focusing on vulnerabilities, SAST, DAST and IAST tools, security web application benchmarks and related work. In Section 3 the OWASP Benchmark project to evaluate security analysis tools (AST) tools is presented. Section 4 describes the steps of the comparative methodology proposal designed by enumerating the steps followed to rank the SAST tools in combination using the selected benchmark. Finally, Section 4 contains the conclusions and Section 5 sketches the future work.

2. Background and Related Work

This section presents the background on web technologies security, benchmarking initiatives, security analysis tools, as well as a review and analysis of different security analysis tools combination results in previous comparatives.

2.1. Web Applications Security

The OWASP Top Ten project brings together the most important vulnerability categories. There are several works that confirm web applications tested did not pass the OWASP Top Ten project [2–4]. Web applications in organizations and companies connected through the Internet and Intranets imply that they are used to develop any type of business, but at the same time they have become a valuable target of a great variety of attacks by exploiting the design, implementation or operation vulnerabilities, included in the OWASP Top Ten project, to obtain some type of economic advantage, privileged information, denial, extortion, etc.

Today there are a great variety of web programing languages such as .NET framework languages (C# or Visual Basic), swift for iOS platforms, or PHP. Java is the most used language, according to several analyses [15,16]. NodeJs, Python and C ++ are among the most frequently chosen today. Modern web applications use asynchronous JavaScript language and XML (AJAX), HTML5, flash technologies [17,18] and Javascript libraries such as Angular, Vue, React, Jquery, Bootstrap, etc. Also. Vaadin is a platform for building, collaborative web applications for Java backends and HTML5 web user interfaces.

Developers should have training in secure code development to prevent security vulnerabilities in web application source code [1]. Another form of prevention is to use secure languages that do type and memory length checks at compile time. C#, Rust and Java are some of these languages [1]. Designers and developers have to use security benchmarks for all configurations of navigators, application and database servers. Complementary and specific measures of online protection are necessary as the deployment of a Web Application Firewall [19–21].

2.2. Analysis Security Testing

There are different types of testing techniques that a security auditor or analyst can select to perform a security analysis of a web application, static white box security analysis (SAST), dynamic black box security analysis (DAST) or interactive white box security analysis (IAST) techniques [1]. The OWASP Security Testing Guide Methodology v4.1 [22] suggests that to perform a complex web application security analysis it is necessary to automate as possible using static, dynamic and interactive analysis testing tools, including manuals checking to find more true positives, and to reduce the number of false positives.

2.2.1. Static Analysis Security Testing

A good technique to avoid security vulnerabilities in source code is prevention [1,23]. Vulnerabilities can be prevented if developers are trained in secure web application development to avoid making "mistakes" that could lead to security vulnerabilities [24]. Obviously, prevention will avoid some of the vulnerabilities, but programing mistakes will always be made in despite of applying preventive secure best practice. Therefore, other subsequent security analysis techniques are necessary once the source code is developed.

SAST tools performs a white box security analysis. It analyzes both source code and executable, as appropriate. SAST tools start with a problem because of the act of determining if a program reaches its final state, or not [25]. Despite this problem, security code analysis can reduce the review code effort [26]. SAST tools are considered the most important security activity within a SSDLC [13].

Security analysts need to improve on recognizing all types of vulnerabilities in the source code for a particular programing language [27]. In the study of Díaz and Bermejo [28], tools such as Fortify SCA, Coverity, Checkmarx or Klocwork are good examples of tools that provide a trace information for eliminating false positives. SAST tools interfaces can be more or less "friendly" in terms of the error trace facilities to audit a security vulnerability.

SAST tools analyze the entire application covering all attack surface. They can also revise the configuration files of the web application. For this reason, static analysis requires a final manual audit of the results to discard the false positives and find the false negatives (much more complicated). However, several works confirm that different SAST tools have distinct algorithm designs as Abstract Interpretation [29–31], Taint Analysis [32], Theorem Provers [33], SAT Solvers [34] or Model Checking [35,36]. Therefore, combining SAST tools can find different types of vulnerabilities and therefore obtain a better combination result [6,7].

2.2.2. Dynamic Analysis Security Testing

DAST tools are black box tools that allow the analyses of a running application attacking all external source inputs of a web application [37]. In a first phase they try to discover (crawling) the attack surface of the web application, that is, all the possible source inputs of the application. The crawling phase must be manual using the tool as an intercept proxy, while also performing an automatic crawling that includes some information of the web application such as programing languages, application server, database server, authentication or session methods. After a crawling phase, the tools perform a recursive attack with malicious payload to all source inputs of web application discovered. Next each http response is syntactically analyzed to check if a security vulnerability exits. Finally, the vulnerability report must be manually revised to discard false positives and discover possible false negatives. Unlike the white box tools, here the source code of the application is not known. Tests are launched against the interface of the running web application. DAST tools usually discover less true positives and also has less false positives than SAST tools [10,38].

DAST tools allow the detection of vulnerabilities in the deployment phase of software development. The behavior of an attacker is simulated to obtaining results for analysis. In addition, these tools that can be executed independently of the language used by the application. DAST tools evolve over time,

incorporating new authentication methods (JWT), attacks vectors (XML, JSON, etc.) and techniques to automate the detection of vulnerabilities [1], among which they stand out fuzzing techniques, based on tests on the application trying to make it fail, such as modifying the entries on forms.

2.2.3. Interactive Analysis Security Testing

Interactive analysis, the ability to monitor and analyze code as it executes, has become a fundamental tool in computer security research. Interactive analysis is attractive because it allows us to analyze the actual executions, and thus can perform precise security analysis based upon runtime information. IAST tools are white box tools and they are an evolution of the SAST tools [39]. They allow code analysis, but unlike the SAST tools, they do it in real time and in an interactive way similar to the DAST tools. The main difference is that the tool runs directly on the server and has to be integrated into the application. Therefore, it is recommended to have a testing environment to perform all the tests and detect the greatest number of vulnerabilities. They are based on the execution of an agent on the server side. This agent is responsible for monitoring the behavior of the application being integrated into all layers of it, providing a broader control of the application flow and data flow, creating possible attack scenarios. The main characteristics of IAST tools are [40]:

- It runs on the server as an agent, obtaining the results of the behavior generated by the end user on the published application. As they are agents that run on the server side, they must be compatible in the language that the application is designed. They can produce a relative runtime overhead in the application server.
- They can allow the sanitization of the entries, making the information that comes from the client side clean, eliminating possible injections or remote executions.
- The data correlation between SAST and IAST tools is usually more accurate as they are white box tools.
- They generate fewer false positives and they are not be able to detect client-side vulnerabilities.

One of the techniques used by IAST tools is taint analysis. The purpose of dynamic taint analysis is to track information flow between sources and sinks. Any program value whose computation depends on data derived from a taint source is considered tainted. Any other value is considered untainted. A taint policy determines exactly how taint flows as a program executes, what sorts of operations introduce new taint, and what checks are performed on tainted values [41–47]. Another technique used by IAST tools is symbolic execution. It allows the analyses of the behavior of a program on many different inputs at one time, by building a logical formula that represents a program execution. Thus, reasoning about the behavior of the program can be reduced to the domain of logic. One of the advantages of forward symbolic execution is that it can be used to reason about more than one input at once [48–51].

2.3. Related Work

In this section, we review the main and recent studies about the combination of different types of web applications security analysis tools with the main objectives of discovering more vulnerabilities and reducing the number of false positives. Several works combine static analysis tools with machine learning techniques for automatic detection of security vulnerabilities in web applications reducing the number of false positives [52,53]. Other approximations are based in attacks and anomalies detection using machine learning techniques [54].

2.3.1. SAST Tools Comparisons Studies

In the work of Diaz and Bermejo [28], nine SAST tools for C language comparison is accomplished to rank them according a set of selected metrics using SAMATE tests suites for C language. This comparison includes several leaders' commercial tools. It is very useful to the practitioners to select the best tool to analyze the source code security.

The work of Nunes et al. [9] present an approach to design benchmarks for evaluating SATS tools having into account distinct levels of criticality. The results of selected SAST tools show that the metrics could be improved to balance the rates of the true positives (TP) and false positives (FP). However, this approach could improve their representative with respect to security vulnerabilities coverage for other types, besides SQLI and XSS.

The work of Algaith et al. [6] is based on a previously published data set that resulted from the use of five different SAST tools to find SQL injections (SQLI) and cross-site scripting (XSS) vulnerabilities in 132 WordPress Content Management System (CMS) plugins. This work could be improved using a benchmark with more vulnerability categories and including leader commercial tools.

Finally, a paper [8] can be examined that presents a benchmarking approach [55] to study the performance of seven static tools (five commercial tools) with a new methodology proposal. The benchmark is representative and it is designed for the vulnerability categories included in the known standard OWASP Top Ten project for SAST tools evaluations.

2.3.2. DAST Tools Comparisons Studies

A study of current DAST tools [56] provides the background needed to evaluate and identify the potential value of future research in this area. It evaluates eight well-known DAST against a common set of sample applications. The study investigates what vulnerabilities are tested by the scanners and their effectiveness of. The study shows that DAST tools are adept at detecting straightforward historical vulnerabilities.

In the work [57], a comprehensive evaluation is performed on a set of open source DAST tools. The results of this comparative evaluation highlighted variations in the effectiveness of security vulnerability detection and indicated that there are correlations between different performance properties of these tools (e.g., scanning speed, crawler coverage, and number of detected vulnerabilities). There was a considerable variance on both types and numbers of detected web vulnerabilities among the evaluated tools.

Another study [58] evaluates five DAST tools against seven vulnerable web applications. The evaluation is based on different measures such as the vulnerabilities severity level and types, numbers of false positive and the accuracy of each DAST tool. The evaluation is conducted based on an extracted list of vulnerabilities from OWASP Top Ten project. The accuracy of each DAST tool was measured based on the identification of true and false positives. The results show that Acunetix and NetSparker have the best accuracy with the lowest rate of false positives.

The study of Amankwah el al. [59] proposes an automated framework to evaluate DAST tools vulnerability severity using an open-source tool called Zed Attach Proxy (ZAP) to detect vulnerabilities listed by National Vulnerability Database (NVD) in a Damn Vulnerable Web Application (DVWA). The findings show that the most prominent vulnerabilities, such as SQL injection and cross-site scripting found in modern Web applications are of medium severity.

2.3.3. SAST-DAST Comparisons Studies

The study of Antunes and Vieira [10], compares SAST vs DAST tools against web services benchmarks and show that the SAST tools usually obtain better true positive ratios and worse false positive ratios than DAST tools. Additionally, these works confirm that SAST and DAST can find different types of vulnerabilities.

2.3.4. SAST Tools Combinations Studies

The work of Xypolytos et al. [60] proposes a framework for combining and ranking tool findings based on tool performance statistics. The initial result shows potential capabilities of the framework to improve tool effectiveness and usefulness. The framework weights the performance of SAST Tools per defect type and cross-validates the findings between different SAST tools reports. An initial validation shows the potential benefits of the proposed framework.

The work of Tao Ye et al. [61], performed a comparison study of commercial (Fortify SCA and Checkmarx) and open source (Splint) SAST tools focusing on the Buffer Overflow vulnerability. In their work they describe the comparison of these tools separately, analyzing 63 open source projects including 100 bugs of this type. In their study, they include the combination of the tools in pairs (Fortify + Checkmarx, Checkmarx + Splint and Fortify + Splint).

In another paper [7], the problem of combining diverse SAST tools is investigated to improve the overall detection of vulnerabilities in web applications, considering four development scenarios with different criticality. It tested five SAST tools against two benchmarks, one with real WordPress plugins and another with synthetic test cases. This work could be improved using test benchmarks with more vulnerability categories than SQLI and XSS and including leaders' commercial tools.

2.3.5. SAST-DAST and SAST-IAST Hybrid Prototypes

The goal of the proposed approach in [62] is to improve penetration testing of web applications by focusing on two areas where the current techniques are limited: identifying the input vectors of a web application and detecting the outcome of an attempted attack. This approach leverages two recently developed analysis techniques. The first is a static analysis technique for identifying potential input vectors, and the second is a dynamic analysis to automate response analysis. The empirical evaluation included nine web applications, and the results show that the solution test the targeted web applications more thoroughly and discover more vulnerabilities than other two tools (Wapiti and Sqlmap), with an acceptable analysis time.

There are several recent works that combine SAST tools with IAST tools to runtime monitor attacks with the information of static analysis. The work of Mongiovi et al. [63] presented a novel data flow analysis approach for detecting data leaks in Java apps, which combines static and interactive analysis in order to perform reliable monitoring while introducing a low overhead in target apps. It provides an implementation as the JADAL tool, which is publicly available. The analysis shows that JADAL can correctly manage use cases that cannot be handled by static analysis, e.g., in presence of polymorphism. Additionally, it has a low overhead since it requires monitoring only a minimal set of points.

Another study presents a new approach to protect Java Enterprise Edition web applications against injection attacks as XSS [64]. The paper first describes a novel approach to taint analysis for Java EE. It explains how to combine this method with static analysis, based on the Joana IFC framework. The resulting hybrid analysis will boost scalability and precision, while guaranteeing protection against XSS.

Another paper [65] proposes a method where firstly vulnerabilities are detected through static code analysis. Then, code to verify these detected vulnerabilities is inserted into the target program's source code. Next, by actually executing the program with the verification code, it is analyzed whether the detected vulnerabilities can be used in a real attack. So, the proposed method comprises of vulnerability detection stage through static code analysis, instrumentation stage to link static code analysis and dynamic analysis, dynamic analysis stage of injecting an error and monitoring the impact on a program.

3. Method Proposal to Analyze AST n-Tools Combinations

In this section we present a new methodology repeatable to combine, compare and rank the SAST, DAST and IAST tools combinations (see Figure 1).

- Select the OWASP Benchmark project.
- Select the SAST, DAST and IAST tools. In this concrete instantiation of the methodology, we choice six commercial and open source tools according to the analysis of the related works in Section 2.3 and official lists of SAST, DAST and IAST tools.

- Run the selected tools against the OWASP Benchmark project with the default configuration for each tool.
- Select appropriate metrics to analyze results according three different levels of web applications criticality.
- Metrics calculation.
- Analysis, discussion and ranking of the results.

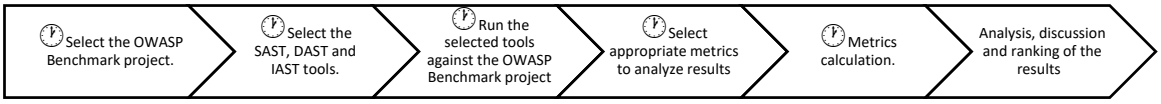

Figure 1. Method proposal to analyze analysis security testing (AST) n-tools combinations.

The following figure presents the proposed method in graphic form:

3.1. Benchmark Selection

An adequate test bench must be credible, portable, representative, require minimum changes and be easy to implement, and the tools execution must be under the same conditions [10]. We have investigated several security benchmarks for web applications as Wavsep used in the comparisons of [66,67]; Securebench Micro Project used in the works of [68,69]; Software Assurance Metrics And Tool Evaluation (SAMATE) project of National Institute of Standards and Technology (NIST) used in several works [9,28,70–73]; OWASP benchmark project [14]; Delta-bench by Pashchenko et al. [74] and OWASP Top Ten Benchmark [55] adapted for OWASP Top Ten 2013 and 2017 vulnerability categories projects and designed for static analysis only used in the work of Bermejo et al. [8].

Taking into account statistics of security vulnerabilities reported by several studies [2–4], the most adequate test bench for using SAST, DAST and IAST tools is OWASP benchmark project [14]. This benchmark is an open source web application in Java language deployed in Apache Tomcat. It meets the properties required for a benchmark [10] and it covers dangerous security vulnerabilities of web applications according to OWASP Top Ten 2013 and OWASP Top Ten 2017 projects. It contains exploitable test cases for detecting true and false positives, each mapped to specific CWEs, which can be analyzed by any type of application security testing (AST) tool, including SAST, DAST (like OWASP ZAP), and IAST tools. We have randomly selected 320 test cases from OWASP benchmark project and distributed them equally according to the number and type of vulnerabilities represented in the test bed. Of these cases half are false positives and half are true positives. It is easily portable as a Java project and it does not require changes with any tool. Table 1 shows the test cases distribution for each type of security vulnerability.

Table 1. OWASP Benchmark Project Test Cases.

CWE	Vulnerability Types	Number of Test Cases
78	Command Injection	40
643	Xpath Injection	20
79	Cross Site Scripting (XSS)	40
90	LDAP Injection	20
22	Path Traversal	40
89	SQL Injection	40
614	Secure Cookie flag	20
501	Trust Boundary Violation	20
330	Weak Randomness	40
327	Weak Cryptographic	20
328	Weak Hashing	20
	Total	320

3.2. SAST, DAST and IAST Tools Selection

- SAST and IAST tools are selected according with Java 2 Platform, Enterprise Edition (J2EE), the most used technology in web applications development, the programing language used by J2EE is Java, one of the labeled as more secure [75]. The next step is the selection of two (2) SAST tools for source code analysis that can detect vulnerabilities in web applications developed using the J2EE specification; two (2) IAST tools for detecting vulnerabilities in runtime and finally two (2) DAST tools:
- With the premises of the above comparatives and analyzing the availability of commercial and open source tools are selected three commercial and three open source meaning tools. Selected tools:
    - Fortify SCA (SAST Commercial tool by Microfocus, Newbury, United Kingdom) supports 18 distinct languages, the most known OS platforms and offers SaaS (Software as a service) and it detects more than 479 vulnerabilities.
    - FindSecurityBugs. (SAST OWASP open source tool, created by Philippe Arteau and licensed under LGPL). It has plugins are available for SonarQube, Eclipse, IntelliJ, Android Studio and NetBeans. Command line integration is available with Ant and Maven.
    - OWASP ZAP (DAST OWASP open source tool with Apache 2 license). Popular tool developed by the active OWASP Foundation community. It is a widely used tool in the field of penetration testing. It supports a great number of plugins, features and input vectors.
    - Arachni (DAST tool with public source License v1.0 created by Tasos Laskos). Open source vulnerability scanner highly valued by the community, and which allows the assessment of vulnerabilities such as SQLi, XSS, Command Injection and other included in OWASP Top Ten project.
    - Contrast Community Edition (IAST free version of the commercial tool for 30 days by Contrasst Security, Los Altos, EE.UU.). It allows the analyses of the application in Java language in an interactive way, making an agent in charge of reporting the vulnerabilities to the server. It provides the complete functionality of our paid platform solutions, Contrast Assess, and Contrast Protect. Contrast Community Edition's main limitations from the paid platform are language support (Java, .NET Core framework) and only one (1) application can be onboarded.
    - CxIAST (IAST commercial tool, by Chekcmarx, Raman Gan, Israael). It is an IAST engine for Java language. It inspects custom code, libraries, frameworks, configuration files, and runtime data flow. It supports Java, .Net framework and NodeJs languages. It can integrate with CxSAST (SAST tool) to correlate and improve results regarding true and false positives.

3.3. Metrics Selection

The selected metrics are widely accepted in others works [8–10,28,73,76] and in the work of Antunes and Vieira [77] that analyzes distinct metrics for different levels of the criticality of web applications. The metrics used in this methodology are:

- Precision (1). Proportion of the total TP detections:

$$TP \ / \ (TP + FP) \tag{1}$$

  where TP (true positives) is the number of true vulnerabilities detected in the code and FP (false positives) is the number of vulnerabilities detected that, really, do not exist.
- True positive rate/Recall (TPR) (2). Ratio of detected vulnerabilities to the number that really exists in the code:

$$TP \ / \ (TP + FN) \tag{2}$$

where *TP* (true positives) is the number of true vulnerabilities detected in the code and *FN* (false negatives) is the total number of existing vulnerabilities not detected in the code.

- False positive rate (FPR) (3). Ratio of false alarms for vulnerabilities that do not really exists in the code:

$$TN \ / \ (TN + FP) \tag{3}$$

where *TN* (true negatives) is the number of not detected vulnerabilities that do not exist in the code and *FP* (false positives) is the total number of detected vulnerabilities that do not exist in the code.

- F-measure (4) is harmonic mean of precision and recall:

$$\frac{(2 \ x \ precision \ x \ recall)}{(precision + recall)} \tag{4}$$

- F_β-Score (5) is a particular F-measure metric for giving more weight to recall or precision. For example, a value for β of 0.5 gives more importance to precision metric, however a value or 1.5 gives more relevance to recall precision:

$$\left(1 + \ \beta^2\right) x \ \frac{precision \ x \ recall}{((\beta^2 \ x \ precision) + recall)} \tag{5}$$

- Markedness (6) has as a goal to penalize the ratio of true positives considering the ratio of false positives and the ratio of false negatives.

$$\frac{TP * TN - FP * FN}{\sqrt{(TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)}} \tag{6}$$

- Informedness (7) computes all positive and negative cases found are reported, and for each true positive the result is increased in a ratio of 1/P while if not detected it is decreased in a ratio of 1/N, where P is the number of positives (test cases with vulnerability) and N is the number on Negatives (test cases with no vulnerability).

$$\frac{TP}{TP + FP} + \frac{TN}{FN + TN} - 1 = \frac{TP}{TP + FP} - \frac{FN}{FN + TN} \tag{7}$$

3.4. Metrics Calculation

In this section, the selected tools run against the OWASP Benchmark project test cases. We obtain the true positive and false positive results for each type of vulnerability. Next, the metrics selected in Section 3.4 are applied to obtain the most appropriate good interpretation of the results and draw the best conclusions.

To calculate all metrics, we have developed a C program to automatize the computation of metrics of each tool. Once executed each tool against OWASP Benchmark project the results are manually revised and included in a file that the C program reads to obtain the selected metrics.

3.4.1. 1-Tool Metrics Calculation

For each tool, Recall (Rec), FPR, Precision (Prec), F-measure (F-Mes), $F_{0.5}$Score ($F_{0.5}$S), $F_{1.5}$Score ($F_{1.5}$S), Markedness (Mark) and Informedness (Inf) metrics are computed. In Table 2 false positives (FP), false negatives (FN), true negatives (TN) and true positives (TP) are computed. Additionally, in Table 3, all selected metrics are computed for each tool. The abbreviations used for some of the tools are: FindSecurityBugs (FSB), Contrast Community Edition (CCE).

Table 2. Metrics applied to individual tools.

	FP	FN	TN	TP
CCE	0	0	160	160
FSB	107	3	53	157
CxIAST	0	15	160	145
Fortify	92	21	68	139
ZAP	8	118	152	42
Arachni	0	124	160	36

Table 3. Metrics applied to individual tools.

	Rec	FPR	Prec	F-Mes	$F_{0.5}S$	$F_{1.5}S$	Mark	Inf
CCE	1	0	1	1	1	1	1	1
FSB	0.99	0.67	0.59	0.74	0.65	0.82	0.56	0.32
CxIAST	0.91	0	1	0.95	0.98	0.93	0.91	0.91
Fortify	0.87	0.58	0.6	0.71	0.64	0.76	0.37	0.29
ZAP	0.29	0.01	0.98	0.45	0.67	0.37	0.56	0.29
Arachni	0.23	0	1	0.37	0.59	0.3	0.56	0.23

A first classification order according to the TPR score from left to right are showed in Table 3 for 1-tool in isolation.

Table 4 shows the execution times of each tool against OWASP Benchmark project. DAST tools (ZAP and Arachni) usually spend more time because they attack the target application externally by performing a penetration test.

Table 4. Execution times of each tool against OWASP Benchmark project.

Tool	Elapsed Time
FSB	22 min
Fortify	19 min and 86 s
ZAP	36 h
Arachni	13 h and 10 min
CCE	2 min and 43 s
CxIAST	4 min and 10 s

3.4.2. 2-Tools and 3-Tools Metrics Calculation

The rules showed in Table 5 have been considered for computing TP, TN, FP and FN metrics in n-tools combinations.

Table 5. Logic applied to the combination of two tools.

	Tool A		Tool N	N-Tools
Positives cases (P)	TP	or	TP	TP
	TP	or	FN	TP
	FN	or	TP	TP
	FN	and	FN	FN
Negative cases (N)	FP	or	FP	FP
	FP	or	TN	FP
	TN	or	FP	FP
	TN	and	TN	TN

We use the 1-out-of-N (1ooN) strategy for combining the results of the AST tools. The process proposed to calculate the combined results for two or more tools is based on a set of automated steps. 1ooN SAST for true positives detection: Any "TP alarm" from any of the n-tools in a test case for TP

detection will lead to a TP alarm in a 1ooN system. If no tool in a combination detects a TP in a test case, it is a FN result for the 1ooN system.

Additionally, we use the 1-out-of-N (1ooN) strategy for combining the results of the AST tools. The process proposed to calculate the combined results for two or more tools is based on a set of automated steps. 1ooN SAST for true positives detection: Any "FP alarm" from any of the n-tools in a test case for FP detection will lead to an FP alarm in a 1ooN system. If no tool in a combination detects a FP in a test case, it is a TN result for the 1ooN system.

Table 6 shows the metrics calculations for 2-tools combinations. In any 2-tools combination can be included tools of the same type.

Table 6. Application of the metrics to 2-tools combinations.

	Rec	FPR	Prec	F-Mes	FPR	$F_{0.5}S$	$F_{1.5}S$	Mark	Inf
Arachni + CCE	1	0	1	1	0	1	1	1	1
CCE + CxIAST	1	0	1	1	0	1	1	1	1
ZAP + CCE	1	0.05	0.95	0.98	0.05	0.96	0.98	0.95	0.95
Fortify + CCE	1	0.58	0.63	0.78	0.58	0.68	0.85	0.63	0.43
FSB + CCE	1	0.67	0.6	0.75	0.67	0.65	0.83	0.6	0.33
FSB + CxIAST	1	0.67	0.6	0.75	0.67	0.65	0.83	0.6	0.33
FSB + Fortify	1	0.78	0.56	0.72	0.78	0.62	0.81	0.56	0.23
Forti + CxIAST	0.99	0.58	0.63	0.77	0.58	0.68	0.84	0.6	0.41
FSB + ZAP	0.98	0.67	0.59	0.74	0.67	0.65	0.82	0.54	0.31
FSB + Arachni	0.98	0.67	0.59	0.74	0.67	0.65	0.82	0.54	0.31
Fortify + Arachni	0.94	0.58	0.62	0.75	0.58	0.66	0.81	0.49	0.36
Fortify + ZAP	0.93	0.58	0.62	0.74	0.58	0.66	0.8	0.47	0.35
Ar + CxIAST	0.92	0	1	0.96	0	0.98	0.94	0.92	0.92
ZAP + CxIAST	0.91	0.05	0.95	0.93	0.05	0.94	0.92	0.86	0.86
ZAP + Arachni	0.3	0.05	0.86	0.44	0.05	0.63	0.38	0.43	0.25

Table 7 shows the metrics calculations for 3-tools combinations. In any 3-tools combination tools of the same type can be included.

Table 7. Application of the metrics to 3-tools combinations.

	Rec	FPR	Prec	F-Mes	$F_{0.5}S$	$F_{1.5}S$	Mark	Inf
Ar + CCE + CxIAST	1	0	1	1	1	1	1	1
ZAP + Ararchi + CCE	1	0.05	0.95	0.98	0.99	0.98	0.95	0.95
ZAP + CCE + CxIAST	1	0.05	0.95	0.98	0.99	0.98	0.95	0.95
Fortify + ZAP + CCE	1	0.58	0.63	0.77	0.9	0.85	0.63	0.42
Fortify + Ararchi + CCE	1	0.58	0.63	0.78	0.9	0.85	0.63	0.43
Fortify + CCE + CxIAST	1	0.58	0.63	0.78	0.9	0.85	0.63	0.43
FSB + ZAP + CCE	1	0.67	0.6	0.75	0.88	0.83	0.6	0.33
FSB + ZAP + CxIAST	1	0.67	0.6	0.75	0.88	0.83	0.6	0.33
FSB + Ararchi + CCE	1	0.67	0.6	0.75	0.88	0.83	0.6	0.33
FSB + Ararchi + CxIAST	1	0.67	0.6	0.75	0.88	0.83	0.6	0.33
FSB + CCE + CxIAST	1	0.67	0.6	0.75	0.88	0.83	0.6	0.33
FSB + Fortify + ZAP	1	0.78	0.56	0.72	0.87	0.81	0.56	0.23
FSB + Fortify + Ararchi	1	0.78	0.56	0.72	0.87	0.81	0.56	0.23
FSB + Fortify + CCE	1	0.78	0.56	0.72	0.87	0.81	0.56	0.23
FSB + Fortify + CxIAST	1	0.78	0.56	0.72	0.87	0.81	0.56	0.23
Fortify + ZAP + CxIAST	0.99	0.58	0.63	0.77	0.89	0.84	0.6	0.41
Fortify + Arachni+ CxIAST	0.99	0.58	0.63	0.77	0.89	0.85	0.62	0.42
FSB + ZAP + Ararchi	0.98	0.67	0.59	0.74	0.87	0.82	0.54	0.31
Fortify + ZAP + Ararchi	0.94	0.58	0.62	0.74	0.85	0.81	0.49	0.36
ZAP +Arachni + CxIAST	0.92	0.05	0.95	0.93	0.92	0.93	0.87	0.87

3.5. Analysis and Discussion

Next, in this section, the main research questions formulated in the Introduction Section 1 are going to be analyzed.

3.5.1. What s Each Type of AST Tool's Average Effectiveness Considering Each Type of Vulnerability without Combination?

Figure 2 shows the average effectiveness considering each type of AST tool and each type of vulnerability. SAST tools obtain very good ratios of Recall/TPR between 0.80 and 1 scores and worse FPR results which are too high in several vulnerability types as CMDi, LDAPi, SQLi, Trust Boundary Violation or Xpathi. DAST tools have a very good ratio of TPR for all vulnerability types, however they have a wide improving margin with respect to Recall/TPR where the best score is 0.70 for XSS and 0.60 for SQLi vulnerabilities but the scores are very low for the rest of vulnerability types. IAST tools obtain excellent TPR and FPR scores for all vulnerability types. Therefore, combining two IAST tools or combining IAST tools with DAST or SAST tools can be a very good choice. Combining IAST tools with SAST tools the TPR ratio can be improved. However, these combinations can obtain a worse FPR combined ratio. Combining IAST tools with DAST tools can improve the TPR ratio.

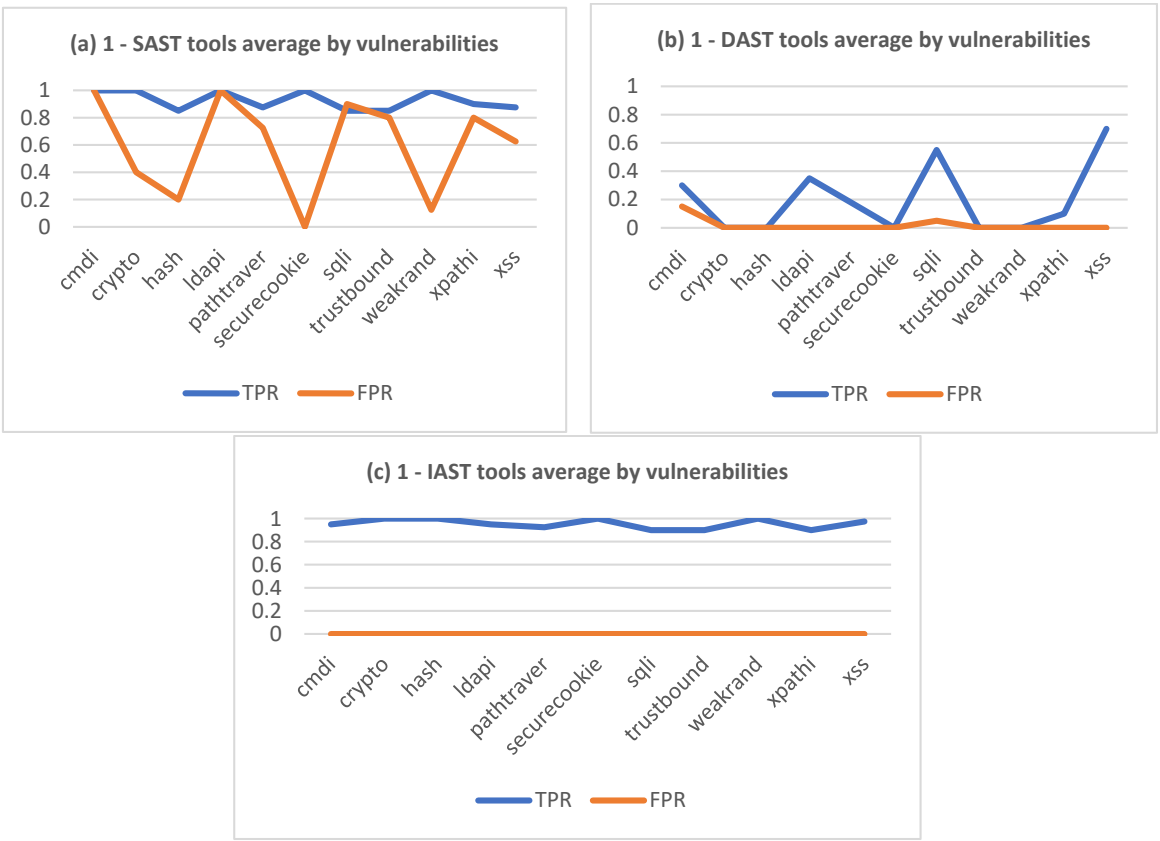

Figure 2. Effectiveness of each type of AST tool considering each type of vulnerability without combination.

3.5.2. What Is That Each Type of AST Tool's Combinations Average Effectiveness Considering Each Type of Vulnerability and the Number of Tools in a Combination?

The graphics of Figure 3 show the average effectiveness of the tools in combination. The TPR ratio increases as the number of tools in the combination increases, reaching practically the value 1 in all vulnerabilities. The FPR ratio increases mainly in several vulnerability types as CMDi, LDAPi, SQLi, Trust Boundary Violation or Xpathi.

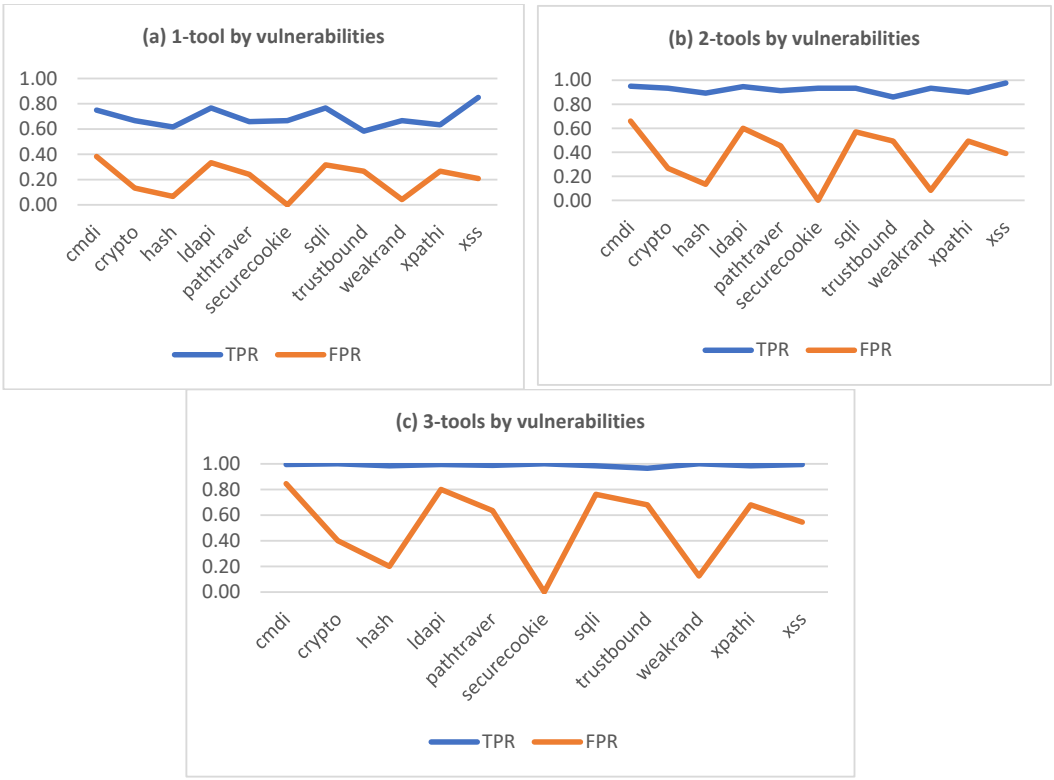

Figure 3. Average effectiveness considering n-tools combinations for each type of vulnerability.

3.5.3. What Is Each Type of AST Tool's Average Effectiveness Detecting OWASP Top Ten Security Category Vulnerabilities without Combination?

In the Figure 4 the results for each type of AST tool effectiveness detecting the OWASP Top Ten security category vulnerabilities without combination is shown. IAST tools obtain the best results in all metrics near the best score (1) and also for FPR where they obtain the best score (0). SAST tools obtain good results in general but they have a great improving margin with respect to FPR. DAST tools obtain good results with respect to FPR but they have a great improving margin with respect to Recall metric. The combination of Recall and TPR supposes a very good precision metric (>0.90). Depends on the specific vulnerability findings of each type of tool in a DAST-SAST tools combination, a concrete combination will be able to obtain better or worse results metrics. Combinations of IAST tools with SAST tools will obtain more false positives due to high FPR ratio of SAST tools.

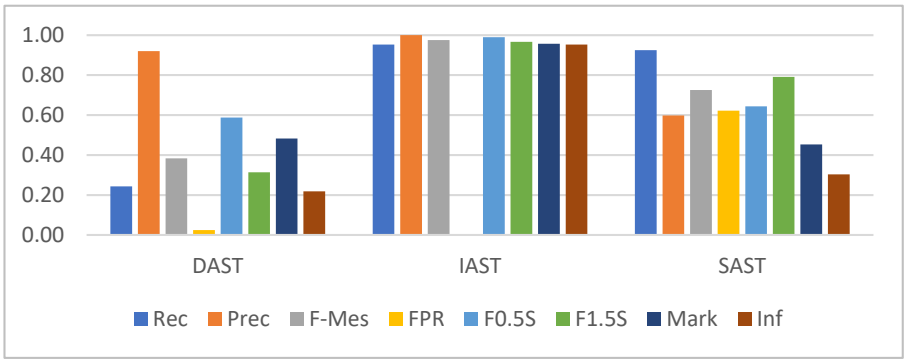

Figure 4. Tools metrics comparison without combination.

Combinations of IAST tools with DAST tools can obtain better metrics results due to DAST tools have a low ratio of FPR and they can find some distinct vulnerabilities than IAST tools.

3.5.4. What Is the n-Tools Combinations Average Effectiveness Detecting OWASP Top Ten Security Vulnerabilities Computing Different Metrics?

In Figures 5 and 6 the results for each type of AST n-tools combinations effectiveness detecting OWASP Top Ten security category vulnerabilities without combination is shown. Figure 5 shows the 2-tools combinations results for each combination and Figure 3 shows the 3-tools combinations results for each combination. In each combination can be included tools of the same type.

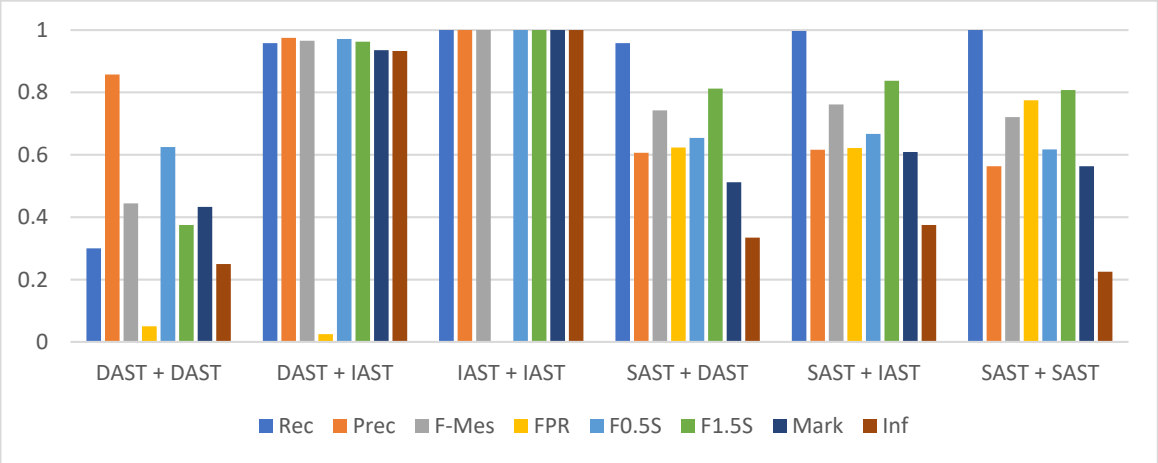

Figure 5. Metric results comparison of 2-tools combinations.

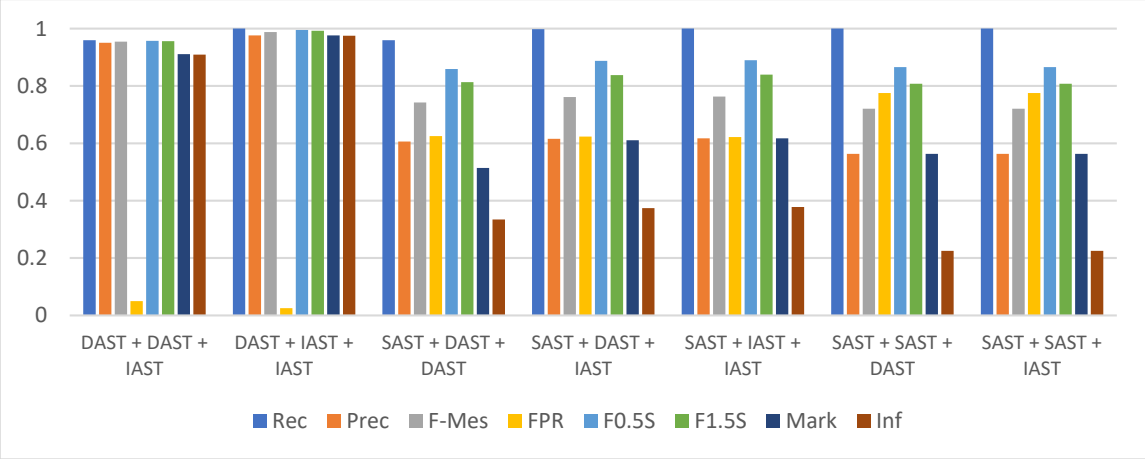

Figure 6. Metric results comparison of 3-tools combinations.

The results of 2-tools combinations show that the combination of two IAST tools obtain the maximum results for all metrics (1) and (0) for FPR metric. DAST+IAST tools combinations is the second-best combination with very good results for all metrics (>0.90) and also very good result for FPR metric. The IAST+SAT tools combinations is the third-best combination with excellent results for Recall metric but with high score for FPR, which make the other metric be worse. SAST+DAST combinations obtain better precision than SAST+SAST combinations, however SAST+SAST obtain better results with respect to Recall than SAST+DAST combinations. Depending on the objective of an analysis, the auditor will choose a combination type. This choice is going to be analyzed in

the next study's questions—considering distinct types of web applications criticality. In all cases, the combination of two tools improves the metrics results obtained by each tool in isolation.

Figure 6 shows that the conclusions about metrics results obtained by 3-tools combinations are similar to the ones 2-tools combinations but it is important to note that the combination of three tools improves the metrics results obtained by 2-tools combinations and the ones obtained by each tool in isolation. The IAST+IAST+DAST combinations obtain excellent results for all metrics followed by IAST+DAST+DAST combinations. Recall/TPR is excellent for all combinations, between 0.95 and 1 scores. For the rest of the combinations, the fact of increasing the FPR ratio increasing the FPR ratio implies a decrease in the values of the other metrics. The higher the FPR ratio, the worse the values of metrics precision and $F_{0.5}$-Score (SAST+SAST+DAST and SAST+SAST+IAST combinations). Similar metrics values are obtained by SAST+DAST+DAST, SAST+DAST+IAST, SAST+IAST+IAST.

3.5.5. Which Are the Best Combinations for Analyzing the Security of Web Applications with Different Levels of Criticality?

Each organization should adapt the security criticality levels of their applications according to the importance they consider appropriate. The criticality levels have been established according to the data stored by the applications, based on the importance of the exposure of sensitive data today, data governed by data protection regulations, but extendable to any type of application. Three distinct levels of web applications criticality, high, medium and low are proposed:

- High criticality—a high criticality environment would be those environments with very sensitive data that, if violated, can cause very serious damage to the company.
- Medium criticality—a medium critical environment would be those environments with especially sensitive data.
- Low criticality—a low criticality environment would be those environments that do not carry any risk for the disclosure of information published in it.

For high criticality web applications, the metrics that provide a greater number of true positive results have been taken into account, that is, FP and TP are detected. It has been considered that being a critical application, any alert detected by the tools should be reviewed. It is also important to ensure that there are no undetected vulnerabilities, i.e., FNs. The FN ratio should be low. TPR must be considered for this type of application, as the closer the value is to 1, the lower the FN. On the other hand, it does not matter that there is a high number of FP, since as it is a critical application, it is considered necessary to audit the vulnerability results to discard false positives.

We have considered for applications with medium criticality that those metrics provide a higher number of positive detections including a medium false positive ratio, as well as being as accurate as possible. That is why if you have to take into account both metrics, the best is to use F-Measure, which provides the harmonic between both. Later, because of its criticality, it is advisable to give priority to TPR and finally to Precision. This relationship will help you to choose the best tool for this purpose.

Finally, in the low criticality applications, the aim is to choose those metrics that provide the highest number of TP, having a very low number of FP, even if this means that there is some vulnerability that is not detected. Therefore, it is advisable to use the Markedness metric, which takes into account the number of TP and FP detections that have been correctly detected. It is considered in these cases that there is no much time to audit the report of vulnerabilities to discard the possible false positives.

In Table 8 are shown the selected metrics considering the distinct types of web application criticality.

Table 8. Selection of metrics by web application criticality.

Metric—Web Application Criticallity	High	Medium	Low
Recall	1	2	
Precision (Prec)		3	1
F-Measure (F-Mes)		1	
$F_{0.5}$Score ($F_{0.5}$S)			2
$F_{1.5}$Score ($F_{1.5}$S)	2		
Markedness (Mark)			3
Informedness (Inf)	3		

According to classification metrics included in Table 2, a ranking of n-tools combinations is going to be established for each degree of web applications criticality. This ranking is established considering three metrics for each level of criticality and the order established for each metric and for each level of criticality in Table 7.

Table 9 shows the classification for 2-tools combinations.

Table 9. Ranking of 2-tools combinations according to distinct level of web application criticality.

HIGH	Rec	$F_{1.5}$S	Info	MEDIUM	F-Mes	Rec	Prec	LOW	Prec	$F_{0.5}$S	Mark
Arachni + CCE	1.00	1.00	1.00	Arachni + CCE	1.00	1.00	1.00	Arachni + CCE	1.00	1.00	1.00
CCE + CxIAST	1.00	1.00	1.00	CCE + CxIAST	1.00	1.00	1.00	CCE + CxIAST	1.00	1.00	1.00
ZAP + CCE	1.00	0.98	0.95	ZAP + CCE	0.98	1.00	0.95	Arachni + CxIAST	1.00	0.98	0.92
Fortify + CCE	1.00	0.85	0.43	Arachni + CxIAST	0.96	0.92	1.00	ZAP + CCE	0.95	0.96	0.95
FSB + CCE	1.00	0.83	0.33	ZAP + CxIAST	0.93	0.91	0.95	ZAP + CxIAST	0.95	0.94	0.86
FSB + CxIAST	1.00	0.83	0.33	Fortify + CCE	0.78	1.00	0.63	ZAP + Arachni	0.86	0.63	0.43
FSB + Fortify	1.00	0.81	0.23	Fortify + CxIAST	0.77	0.99	0.63	Fortify + CCE	0.63	0.68	0.63
Fortify + CxIAST	0.99	0.84	0.41	FSB + CCE	0.75	1.00	0.60	Fortify + CxIAST	0.63	0.68	0.60
FSB + ZAP	0.98	0.82	0.31	FSB + CxIAST	0.75	1.00	0.60	Fortify + Arachni	0.62	0.66	0.49
FSB + Arachni	0.98	0.82	0.31	Fortify + Arachni	0.75	0.94	0.62	Fortify + ZAP	0.62	0.66	0.47
Fortify + Arachni	0.94	0.81	0.36	Fortify + ZAP	0.74	0.93	0.62	FSB + CCE	0.60	0.65	0.60
Fortify + ZAP	0.93	0.80	0.35	FSB + ZAP	0.74	0.98	0.59	FSB + CxIAST	0.60	0.65	0.60
Arachni + CxIAST	0.92	0.94	0.92	FSB + Arachni	0.74	0.98	0.59	FSB + ZAP	0.59	0.65	0.54
ZAP + CxIAST	0.91	0.92	0.86	FSB + Fortify	0.72	1.00	0.56	FSB + Arachni	0.59	0.65	0.54
ZAP + Arachni	0.30	0.38	0.25	ZAP + Arachni	0.44	0.30	0.86	FSB + Fortify	0.56	0.62	0.56

The ranking of Table 9 confirms that the benchmark contains vulnerability types that AST tools can detect and permits to establish an adequate order with respect to the tool effectiveness in terms of the selected metrics. IAST tools combinations obtain the best results for high, medium and low critical web applications. Another conclusion is that IAST and DAST tools combinations have very good results for medium and low critical applications. IAST and SAST tools combinations have very good results for high, applications and also good results for medium and low critical web applications. ZAP+Arachni obtain the worse results for high and medium web applications critical levels.

Figure 7 includes three graphics to show the rankings of 2-tools combinations for each criticality level taking into account the first metric used for each classification. High (a): Recall; Medium (b): F-measure and Low (c): $F_{0.5}$ Score. ZAP+ Arachni combination obtains a better result for low level due to DAST tools have a low false positives ratio and the $F_{0.5}$ Score metrics favors on a good FPR metric. However, FSB + Fortify combination obtains a worse result for low level due to SAST tools have a high ratio of false positives. Additionally, FSB + Fortify combination obtains a worse result for medium level due to SAST tools have a high ratio of false positives.

The rankings of Tables 9 and 10 confirm that the benchmark contains vulnerability types that AST tools can detect but it permits to establish an adequate order with respect to the tool effectiveness in terms of the selected metrics. Table 10 shows that IAST tools combinations obtain the best results. Another conclusion is that there is IAST and DAST tools combinations have very good results. Combinations with SAST tools generally obtain good results if spite of having a higher ratio of false positives than IAST and DAST tools they have a very good ratio of true positives. SAST tools have to be considered to include them in a security analysis of a web application because find distinct

vulnerabilities and the false positives can be eliminated in the necessary subsequent audit of the vulnerability reports. The Combinations integrated by SAST+DAST+IAST tools as Fortify + Arachni + CCE or Fortify + ZAP + CCE obtain a very good result in the HIGH, MEDIUM and LOW classifications. In the three 3 classifications the two first 2-tools combinations are the same due to the presence of one or two IAST tools in the combination, but from the third position the three criticality classifications are quite different. In the three 3 classifications the three first 3-tools combinations are the same due to the presence of one or two IAST tools in the combination, but from the fourth position the three criticality classifications are quite different. For high level of criticality, the results reached for the 3-tools combinations are between 0.92 and 1 for recall metric.

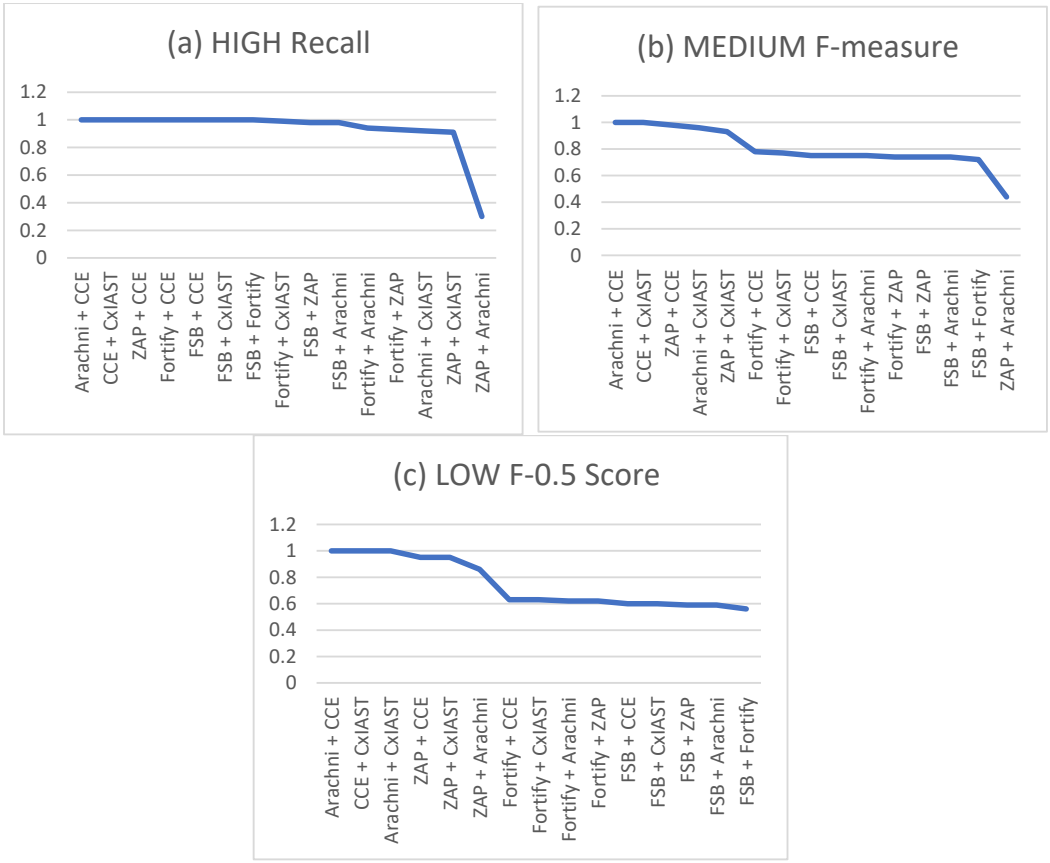

Figure 7. Rankings for (**a**) high, (**b**) medium and (**c**) low criticality levels for 2-tools combinations.

Table 10. Ranking of 3-tools combinations according to distinct level of web application criticality.

HIGH	Rec	$F_{1.5}S$	Info	MEDIUM	F-Mes	Rec	Prec	LOW	Prec	$F_{0.5}S$	Mark
Arachni + CCE + CxIAST	1.00	1.00	1.00	Arachni + CCE + CxIAST	1.00	1.00	1.00	Arachni + CCE + CxIAST	1.00	1.00	1.00
ZAP + Arachni + CCE	1.00	0.98	0.95	ZAP + Arachni + CCE	0.98	1.00	0.95	ZAP + Arachni + CCE	0.95	0.99	0.95
ZAP + CCE + CxIAST	1.00	0.98	0.95	ZAP + CCE + CxIAST	0.98	1.00	0.95	ZAP + CCE + CxIAST	0.95	0.99	0.95
Fortify + Arachni + CCE	1.00	0.85	0.43	ZAP + Arachni + CxIAST	0.93	0.92	0.95	ZAP + Arachni + CxIAST	0.95	0.92	0.87
Fortify + CCE + CxIAST	1.00	0.85	0.43	Fortify + Arachni + CCE	0.78	1.00	0.63	Fortify + Arachni + CCE	0.63	0.90	0.63
Fortify + ZAP + CCE	1.00	0.85	0.42	Fortify + CCE + CxIAST	0.78	1.00	0.63	Fortify + CCE + CxIAST	0.63	0.90	0.63
FSB + ZAP + CCE	1.00	0.83	0.33	Fortify + ZAP + CCE	0.77	1.00	0.63	Fortify + Arachni + CxIAST	0.63	0.89	0.62
FSB + ZAP + CxIAST	1.00	0.83	0.33	Fortify + Arachni + CxIAST	0.77	0.99	0.63	Fortify + ZAP + CCE	0.63	0.90	0.63
FSB + Arachni + CCE	1.00	0.83	0.33	Fortify + ZAP + CxIAST	0.77	0.99	0.63	Fortify + ZAP + CxIAST	0.63	0.89	0.60
FSB + Arachni + CxIAST	1.00	0.83	0.33	FSB + ZAP + CCE	0.75	1.00	0.60	Fortify + ZAP + Arachni	0.62	0.85	0.49
FSB + CCE + CxIAST	1.00	0.83	0.33	FSB + ZAP + CxIAST	0.75	1.00	0.60	FSB + ZAP + CCE	0.60	0.88	0.60
FSB + Fortify + ZAP	1.00	0.81	0.23	FSB + Arachni + CCE	0.75	1.00	0.60	FSB + ZAP + CxIAST	0.60	0.88	0.60
FSB + Fortify + Arachni	1.00	0.81	0.23	FSB + Arachni + CxIAST	0.75	1.00	0.60	FSB + Arachni + CCE	0.60	0.88	0.60
FSB + Fortify + CCE	1.00	0.81	0.23	FSB + CCE + CxIAST	0.75	1.00	0.60	FSB + Arachni + CxIAST	0.60	0.88	0.60
FSB + Fortify + CxIAST	1.00	0.81	0.23	Fortify + ZAP + Arachni	0.74	0.94	0.62	FSB + CCE + CxIAST	0.60	0.88	0.60
Fortify + Arachni + CxIAST	0.99	0.85	0.42	FSB + ZAP + Arachni	0.74	0.98	0.59	FSB + ZAP + Arachni	0.59	0.87	0.54
Fortify + ZAP + CxIAST	0.99	0.84	0.41	FSB + Fortify + ZAP	0.72	1.00	0.56	FSB + Fortify + ZAP	0.56	0.87	0.56
FSB + ZAP + Arachni	0.98	0.82	0.31	FSB + Fortify + Arachni	0.72	1.00	0.56	FSB + Fortify + Arachni	0.56	0.87	0.56
Fortify + ZAP + Arachni	0.94	0.81	0.36	FSB + Fortify + CCE	0.72	1.00	0.56	FSB + Fortify + CCE	0.56	0.87	0.56
ZAP + Arachni + CxIAST	0.92	0.93	0.87	FSB + Fortify + CxIAST	0.72	1.00	0.56	FSB + Fortify + CxIAST	0.56	0.87	0.56

Figure 8 includes three graphics to show the rankings of 3-tools combinations for each criticality level having into account the first metric used for each classification. High (a): Recall; Medium (b): F-measure and Low (c): $F_{0.5}$ Score. For high level the combinations that include 2 DAST tools have a wort results for recall metric. For high and medium levels, combinations between DAST and IAST tools obtain the best results.

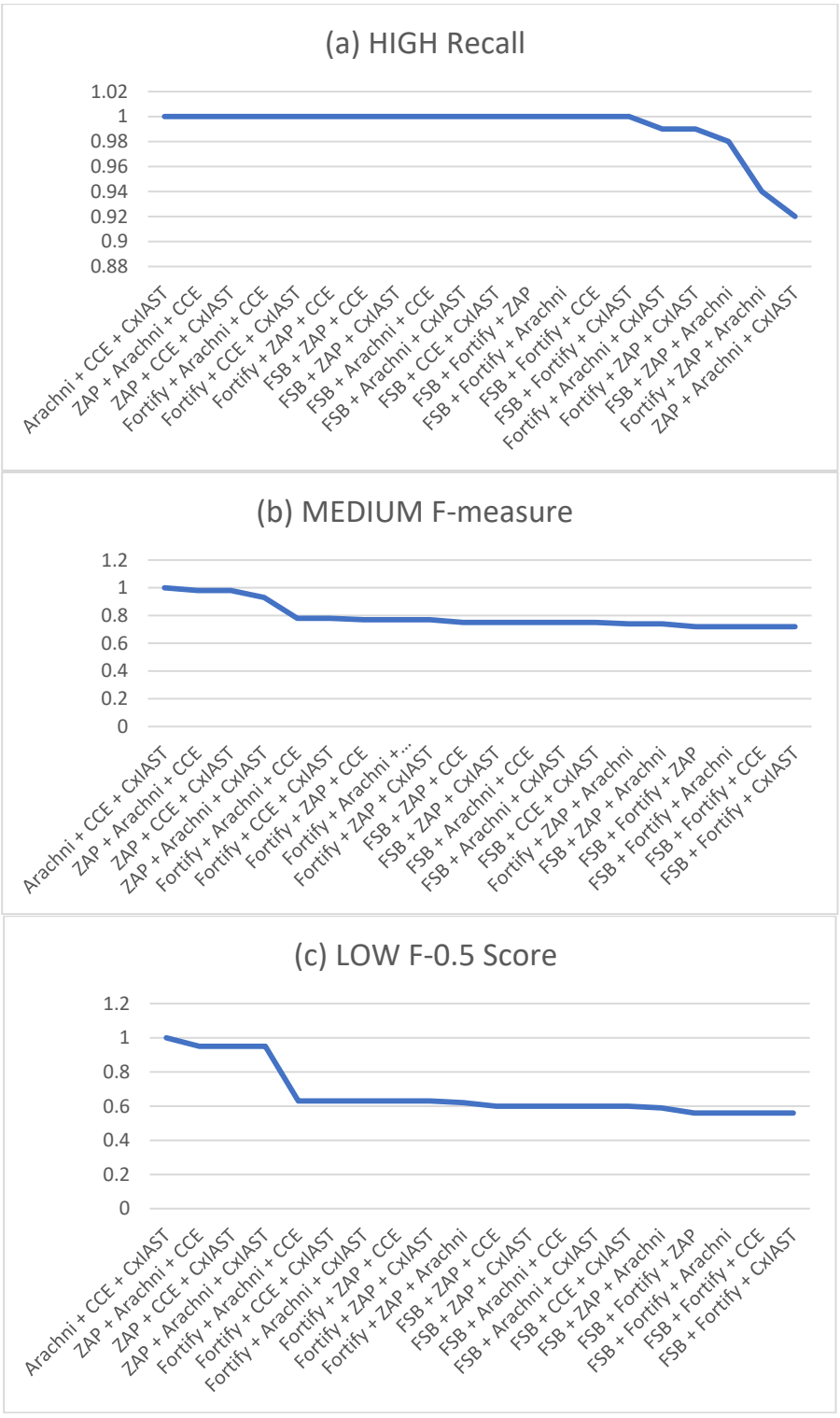

Figure 8. Rankings for (**a**) high, (**b**) medium, and (**c**) low criticality levels for 3-tools combinations.

4. Conclusions

Vulnerability analysis is still for many organizations, the great unknown. In others, it is a natural way of working, providing the necessary security required to keep the product free of vulnerabilities. Using different types of tools to detect distinct security vulnerabilities helps both developers and organizations to release applications securely, reducing the time and resources that must later be provided to fix initial errors. It is during the secure software development life cycle of an application where vulnerability detection tools help to naturally integrate security into the web application. It is necessary to use SAST, DAST and IAST tools in the development and test phases to develop a secure web application within a secure software development life cycle auditing the vulnerability reports to eliminate false positives and correlating the results obtained for each tool in a concrete combination of different types of tools.

In recent years, different combinations of tools have been seen to improve security, but until the writing of this study, it has not been possible to combine the three types of tools that exist on the market. It has been proven how, in a remarkable way, such a combination helps to obtain a very good ratios of true and false positives in many of the cases. IAST tools have obtain very good results metrics, the use of an IAST tool in a combination can improve the web applications security of an organization. Three tools included in this study are commercial, it is very interesting that the auditors and practitioners have the possibility of knowing how the security performance and the behavior of a commercial tool is as part of a n-tools combination.

To reach the final results we have used a methodology applied against a selected representative benchmark for OWASP Top Ten vulnerabilities to obtain results from different types of tools. Next, the results have been applied to a set of carefully selected metrics to perform the comparison of tools combinations to find out which of them are part of the best option when choosing these tools considering various levels of criticality in web applications to be analysed in an organization.

IAST tools combinations obtain the best results. Another conclusion is that IAST and DAST tools combinations have very good results. Combinations with SAST tools obtain generally good results if spite of having a higher ratio of false positives than IAST and DAST tools. However, SAST tools have a very good ratio of true positives. Therefore, SAST tools have to be considered to include them in a security analysis of a web application because find distinct vulnerabilities than DAST tools and the false positives can be eliminated in the necessary subsequent audit of the vulnerability reports. The combinations integrated by SAST+DAST+IAST tools as Fortify + Arachni + CCE or Fortify + ZAP + CCE obtain a very good result in the high, medium and low classifications.

The correlation of results between tools of different type is still an aspect that is not very widespread. It is necessary to develop a methodology or a custom-made software that allows in an automatic or semiautomatic way for the evaluation and correlation of the results obtained by several tools of different types.

It is very important to develop representative test benchmarks for the set of vulnerabilities included in the OWASP Top Ten categories that allow proper assessment and comparison of combinations of tools including SAST, DAST and IAST tools. It is also necessary to evolve them to include more test cases designed for more types of vulnerabilities included in the OWASP Top Ten categories. The rankings established in Tables 8 and 9 confirm that the benchmark contains vulnerability types that AST tools can detect but it permits to establish an adequate order with respect to the tool effectiveness in terms of the selected metrics.

5. Future Work

As future work we are going to work on the elaboration of a representative benchmark for web applications including a wide set of security vulnerabilities that permits an AST tools comparison. The main objective is that the evaluation of each combination of different tools can be the most effective possible using the methodology proposed in this work, allowing one to distinguish the best combinations of tools considering several levels of criticality in the web applications of an organization.

Author Contributions: Conceptualization, F.M.T., J.B.H. and J.-R.B.H.; methodology, J.B.H., F.M.T.; validation, J.-R.B.H., M.I.A. and J.-A.S.M.; investigation, J.B.H., F.M.T., J.-R.B.H. and J.-A.S.M.; resources, J.B.H., F.M.T., J.-R.B.H. and J.-A.S.M.; writing—original draft preparation, J.B.H., and F.M.T.; writing—review and editing, J.B.H., F.M.T., J.-R.B.H. and J.-A.S.M.; visualization, J.B.H., F.M.T., J.-R.B.H., M.I.A. and J.-A.S.M.; supervision, J.B.H., F.M.T., J.-R.B.H. and J.-A.S.M.; project administration, J.-R.B.H., M.I.A. and J.-A.S.M.; software M.I.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

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

References

1. Felderer, M.; Büchler, M.; Johns, M.; Brucker, A.D.; Breu, R.; Pretschner, A. Security Testing: A Survey. In *Advances in Computers*; Elsevier: Cambridge, MA, USA, 2016. [CrossRef]
2. Homaei, H.; Shahriari, H.R. Seven Years of Software Vulnerabilities: The Ebb and Flow. *IEEE Secur. Priv. Mag.* **2017**, *15*, 58–65. [CrossRef]
3. Barabanov, A.; Markov, A.; Tsirlov, V. Statistics of software vulnerability detection in certification testing. In *International Conference Information Technologies in Business and Industry 2018*; IOP Publishing: Tomsk, Russia, 2017. [CrossRef]
4. Sołtysik-Piorunkiewicz, A.; Krysiak, M. The Cyber Threats Analysis for Web Applications Security in Industry 4.0. In *Towards Industry 4.0—Current Challenges in Information Systems*; Studies in Computational Intelligence; Springer: Cham, Switzerland, 2020. [CrossRef]
5. OWASP Foundation. OWASP Top Ten 2017. Available online: https://www.owasp.org/index.php/Top_10_2017-Top_10 (accessed on 1 December 2020).
6. Algaith, A.; Nunes, P.; Fonseca, J.; Gashi, I.; Viera, M. Finding SQL injection and cross site scripting vulnerabilities with diverse static analysis tools. In Proceedings of the 14th European Dependable Computing Conference, IEEE Computer Society, Iasi, Romania, 10–14 September 2018. [CrossRef]
7. Nunes, P.; Medeiros, I.; Fonseca, J.C.; Neves, N.; Correia, M.; Vieira, M. An empirical study on combining diverse static analysis tools for web security vulnerabilities based on development scenarios. *Computing* **2018**, *101*, 161–185. [CrossRef]
8. Bermejo, J.R.; Bermejo, J.; Sicilia, J.A.; Cubo, J.; Nombela, J.J. Benchmarking Approach to Compare Web Applications Static Analysis Tools Detecting OWASP Top Ten Security Vulnerabilities. *Comput. Mater. Contin.* **2020**, *64*, 1555–1577. [CrossRef]
9. Nunes, P.; Medeiros, I.; Fonseca, J.C.; Neves, N.; Correia, M.; Vieira, M. Benchmarking Static Analysis Tools for Web Security. *IEEE Trans. Reliab.* **2018**, *67*, 1159–1175. [CrossRef]
10. Antunes, N.; Vieira, M. Assessing and Comparing Vulnerability Detection Tools for Web Services: Benchmarking Approach and Examples. *IEEE Trans. Serv. Comput.* **2015**, *8*, 269–283. [CrossRef]
11. Monga, M.; Paleari, R.; Passerini, E. A hybrid analysis framework for detecting web application vulnerabilities. In Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems, Vancouver, BC, Canada, 19 May 2009; pp. 25–32. [CrossRef]
12. Higuera, J.B.; Aramburu, C.A.; Higuera, J.-R.B.; Sicilia, M.-A.; Montalvo, J.A.S. Systematic Approach to Malware Analysis (SAMA). *Appl. Sci.* **2020**, *10*, 1360. [CrossRef]
13. Mohino, J.D.V.; Higuera, J.B.; Higuera, J.-R.B.; Montalvo, J.A.S.; Higuera, B.; Mohino, D.V.; Montalvo, J.A.S. The Application of a New Secure Software Development Life Cycle (S-SDLC) with Agile Methodologies. *Electronics* **2019**, *8*, 1218. [CrossRef]
14. OWASP Foundation. OWASP Benchmark Project. Available online: https://www.owasp.org/index.php/Benchmark (accessed on 1 December 2020).
15. Nanz, S.; Furia, C.A. A comparative study of programming languages in rosetta code. In Proceedings of the 37th International Conference on Software Engineering 2015, Florence, Italy, 16–24 May 2015; Volume 1, p. 778. [CrossRef]
16. Aruoba, S.B.; Fernández-Villaverde, J. A comparison of programming languages in macroeconomics. *J. Econ. Dyn. Control.* **2015**, *58*, 265–273. [CrossRef]
17. Beasley, R.E. Ajax Programming. In *Essential ASP.NET Web Forms Development*; Apress: Berkeley, CA, USA, 2020. [CrossRef]

18. Moeller, J.P. *Security for Web Developers: Using Javascript, HTML and CSS*; O'Reilly Media: Sebastopol, Russia, 2016.
19. Razzaq, A.; Hur, A.; Shahbaz, S.; Masood, M.; Ahmad, H.F.; Abdul, R. Critical analysis on web application firewall solutions. In Proceedings of the 2013 IEEE Eleventh International Symposium on Autonomous Decentralized Systems (ISADS), Mexico City, Mexico, 6–8 March 2013; pp. 1–6. [CrossRef]
20. Holm, H.; Ekstedt, M. Estimates on the effectiveness of web application firewalls against targeted attacks. *Inf. Manag. Comput. Secur.* **2013**, *21*, 250–265. [CrossRef]
21. Tekerek, A.; Bay, O. Design and implementation of an artificial intelligence-based web application firewall model. *Neural Netw. World* **2019**, *29*, 189–206. [CrossRef]
22. OWASP Foundation. OWASP Testing Guide, 2020. Available online: https://owasp.org/www-project-web-security-testing-guide/ (accessed on 1 December 2020).
23. Huth, M.; Nielsen, F. *Static Analysis for Proactive Security. Computing and Software Science. Lecture Notes in Computer Science*; Springer: Cham, Switzerland, 2019. [CrossRef]
24. Al-Amin, S.; Ajmeri, N.; Du, H.; Berglund, E.Z.; Singh, M.P. Toward effective adoption of secure software development practices. *Simul. Model. Pr. Theory* **2018**, *85*, 33–46. [CrossRef]
25. Sipser, M. *Introduction to the Theory of Computation*, 2nd ed.; Thomson Course Technology: Boston, MA, USA, 2006.
26. Singh, D.; Sekar, V.R.; Stolee, K.T.; Johnson, B. Evaluating How Static Analysis Tools Can Reduce Code Review Effort. In Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, Raleigh, NC, USA, 11–14 October 2017; pp. 101–105. [CrossRef]
27. Yang, J.; Tan, L.; Peyton, J.; Duer, K.A. Towards better utilizing static application security testing. In Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, Montreal, QC, Canada, 25–31 May 2019; IEEE Computer Society: Montreal, QC, Canada, 2019. [CrossRef]
28. Díaz, G.; Bermejo, J.R. Static analysis of source code security: Assessment of tools against SAMATE tests. *Inf. Softw. Technol.* **2013**, *55*, 1462–1476. [CrossRef]
29. Fromherz, A.; Ouadjaout, A.; Miné, A. Static Value Analysis of Python Programs by Abstract Interpretation. In *NASA Formal Methods. NFM 2018. Lecture Notes in Computer Science*; Dutle, A., Muñoz, C., Narkawicz, A., Eds.; Springer: Cham, Switzerland, 2018; Volume 10811. [CrossRef]
30. Urban, C.; Ueltschi, S.; Müller, P. Abstract interpretation of CTL properties. In *SAS '18. LNCS*; Springer: Berlin/Heidelberg, Germany, 2018; Volume 11002, pp. 402–422. [CrossRef]
31. Oortwijn, W.; Gurov, D.; Huisman, M. An Abstraction Technique for Verifying Shared-Memory Concurrency. *Appl. Sci.* **2020**, *10*, 3928. [CrossRef]
32. Ferrara, P.; Olivieri, L.; Spoto, F. BackFlow: Backward Context-Sensitive Flow Reconstruction of Taint Analysis Results. In *Verification, Model Checking, and Abstract Interpretation. VMCAI 2020. Lecture Notes in Computer Science*; Beyer, D., Zufferey, D., Eds.; Springer: Berlin/Heidelberg, Germany, 2020; Volume 11990. [CrossRef]
33. Khan, W.; Kamran, M.; Ahmad, A.; Khan, F.A.; Derhab, A. Formal Analysis of Language-Based Android Security Using Theorem Proving Approach. *IEEE Access* **2019**, *7*, 16550–16560. [CrossRef]
34. Biere, A.; Kröning, D. SAT-Based Model Checking. In *Handbook of Model Checking*; Clarke, E., Henzinger, T., Veith, H., Bloem, R., Eds.; Springer: Cham, Switzerland, 2018. [CrossRef]
35. Beyer, D.; Gulwani, S.; Schmidt, D.A. Combining Model Checking and Data-Flow Analysis. In *Handbook of Model Checking*; Clarke, E., Henzinger, T., Veith, H., Bloem, R., Eds.; Springer: Cham, Switzerland, 2018. [CrossRef]
36. Nielson, F.; Nielson, H.R.; Zhang, F. Multi-valued Logic for Static Analysis and Model Checking. In *Models, Mindsets, Meta: The What, the How, and the Why Not? Lecture Notes in Computer Science*; Margaria, T., Graf, S., Larsen, K., Eds.; Springer: Cham, Switzerland, 2019; Volume 11200. [CrossRef]
37. Mirjalili, M.; Nowroozi, A.; Alidoosti, M. A survey on web penetration test. *Adv. Comput. Sci.* **2014**, *3*, 107–121.
38. Vega, E.A.A.A.; Orozco, L.S.; Villalba, L.J.G. Benchmarking of Pentesting Tools. *Int. J. Comput. Electr. Autom. Control Inf. Eng.* **2017**, *11*, 602–605.
39. Pan, Y. Interactive Application Security Testing. In Proceedings of the 2019 International Conference on Smart Grid and Electrical Automation (ICSGEA), Xiangtan, China, 10 August 2019; pp. 558–561. [CrossRef]
40. Bermejo, J.R. Assessment Methodology of Web Applications Automatic Security Analysis Tools for Adaptation in the Development Life Cycle. Ph.D. Thesis, UNED, Madrid, Spain, 2014. Available online: http://e-spacio.uned.es/fez/view/tesisuned:IngInd-Jrbermejo (accessed on 1 December 2020).

41. Ren, Y.; Dong, W.; Lin, J.; Miao, X. A Dynamic Taint Analysis Framework Based on Entity Equipment. *IEEE Access* **2019**, *7*, 186308–186318. [CrossRef]
42. Zhao, J.; Qi, J.; Zhou, L.; Cui, B. Dynamic Taint Tracking of Web Application Based on Static Code Analysis. In Proceedings of the 2016 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), Fukuoka, Japan, 6–8 July 2016; pp. 96–101. [CrossRef]
43. Karim, R.; Tip, F.; Sochurkova, A.; Sen, K. Platform-Independent Dynamic Taint Analysis for JavaScript. *IEEE Trans. Softw. Eng.* **2018**. [CrossRef]
44. Kreindl, J.; Bonetta, D.; Mossenbock, H. Towards Efficient, Multi-Language Dynamic Taint Analysis. In Proceedings of the 16th ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes, MPLR 2019, Athens, Greece, 21–22 October 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 85–94. [CrossRef]
45. Cho, S.; Kim, G.; Cho, S.-J.; Choi, J.; Park, M.; Han, S. Runtime Input Validation for Java Web Applications Using Static Bytecode Instrumentation. In Proceedings of the International Conference on Research in Adaptive and Convergent Systems, RACS'16, Odense, Denmark, 11–14 October 2016; Association for Computing Machinery: New York, NY, USA, 2016.
46. Wang, R.; Xu, G.; Zeng, X.; Li, X.; Feng, Z. TT-XSS: A novel taint tracking based dynamic detection framework for DOM cross-site scripting. *J. Parallel Distrib. Comput.* **2018**, *118*, 100–106. [CrossRef]
47. Bichhawat, A.; Rajani, V.; Garg, D.; Hammer, C. Information Flow Control in WebKit's JavaScript Bytecode. In *Principles of Security and Trust, POST 2014*; Lecture Notes in Computer Science; Abadi, M., Kremer, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2015; Volume 8414. [CrossRef]
48. Joseph, P.N.; Jackson, D. Derailer: Interactive security analysis for web applications. In Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE '14), Vasteras, Sweden, 15–19 September 2014; ACM: New York, NY, USA, 2014; pp. 587–598.
49. Duraibi, S.; Alashjaee, A.M.; Song, J. A Survey of Symbolic Execution Tools. *Int. J. Comput. Sci. and Secur. (IJCSS)* **2019**, *13*, 244.
50. Baldoni, R.; Coppa, E.; D'Elia, D.C.; Demetrescu, C.; Finocchi, I. A survey of symbolic execution techniques. *ACM Comput. Surv.* **2018**, *51*, 50. [CrossRef]
51. Balasubramanian, D.; Zhang, Z.; McDermet, D.; Karsai, G. Dynamic symbolic execution for the analysis of web server applications in Java. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC '19, Limassol, Cyprus, 8–12 April 2019; pp. 2178–2185. [CrossRef]
52. Pistoia, M.; Tripp, O.; Lubensky, D. Combining Static Code Analysis and Machine Learning for Automatic Detection of Security Vulnerabilities in Mobile Apps. In *Application Development and Design: Concepts, Methodologies, Tools, and Applications*; IGI Global: Hershey, PA, USA, 2018. [CrossRef]
53. Pereira, J.D.; Campos, J.R.; Vieira, M. An Exploratory Study on Machine Learning to Combine Security Vulnerability Alerts from Static Analysis Tools. In Proceedings of the 2019 9th Latin-American Symposium on Dependable Computing (LADC), IEEE, Natal, Brazil, 19–21 November 2019. [CrossRef]
54. Riera, T.S.; Higuera, J.-R.B.; Higuera, J.B.; Martínez-Herráiz, J.-J.; Montalvo, J.A.S. Prevention and Fighting against Web Attacks through Anomaly Detection Technology. A Systematic Review. *Sustainability* **2020**, *12*, 4945. [CrossRef]
55. Bermejo, J.R. OWASP Top Ten-Benchmark. Available online: https://github.com/jrbermh/OWASP-Top-Ten-Benchmark (accessed on 1 December 2020).
56. Bau, J.; Bursztein, E.; Gupta, D.; Mitchell, J.C. State of the Art: Automated Black-Box Web Application Vulnerability Testing. In Proceedings of the IEEE Symposium on Security and Privacy, Berkeley/Oakland, CA, USA, 16–19 May 2010.
57. Alsaleh, M.; Alomar, N.; Alshreef, M.; Alarifi, A.; Al-Salman, A. Performance-based comparative assessment of open source web vulnerability scanners. *Secur. Commun. Netw.* **2017**, *2017*, 6158107. [CrossRef]
58. Qasaimeh, M.; Shamlawi, A.; Khairallah, T. Black box evaluation of web applications scanners: Standards mapping aproach. *J. Theor. Appl. Inf. Technol.* **2018**, *96*, 4584–4596.
59. Amankwah, R.; Chen, J.; Kudjo, P.K.; Agyemang, B.K.; Amponsah, A.A. An automated framework for evaluating open-source web scanner vulnerability severity. *Serv. Oriented Comput. Appl.* **2020**, *14*, 297–307. [CrossRef]

60. Xypolytos, A.; Xu, H.; Vieira, B.; Ali-Eldin, A.M.T. A Framework for Combining and Ranking Static Analysis Tool Findings Based on Tool Performance Statistics. In Proceedings of the 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Prague, Czech Republic, 25–29 July 2017; pp. 595–596. [CrossRef]
61. Ye, T.; Zhang, L.; Wang, L.; Li, X. An Empirical Study on Detecting and Fixing Buffer Overflow Bugs. In Proceedings of the 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST), Chicago, IL, USA, 11–15 April 2016; pp. 91–101. [CrossRef]
62. Halfond, W.G.J.; Choudhary, S.R.; Orso, A. Improving penetration testing through static and dynamic analysis. *Softw. Test. Verif. Reliab.* **2011**, *21*, 195–214. [CrossRef]
63. Mongiovi, M.; Giannone, G.; Fornaia, A.; Pappalardo, G.; Tramontana, E. Combining static and dynamic data flow analysis: A hybrid approach for detecting data leaks in Java applications. In Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, 13–17 April 2015; ACM: New York, NY, USA, 2015; pp. 1573–1579. [CrossRef]
64. Loch, F.D.; Johns, M.; Hecker, M.; Mohr, M.; Snelting, G. Hybrid taint analysis for java EE. In Proceedings of the 35th Annual ACM Symposium on Applied Computing, Brno, Czech Republic, 30 March–3 April 2020; ACM: New York, NY, USA, 2020. [CrossRef]
65. Kim, S.; Kim, R.Y.C.; Park, Y.B. Software Vulnerability Detection Methodology Combined with Static and Dynamic Analysis. *Wirel. Pers. Commun.* **2015**, *89*, 777–793. [CrossRef]
66. Alavi, S.; Bessler, N.; Massoth, M. A comparative evaluation of automated vulnerability scans versus manual penetration tests on false-negative errors. In Proceedings of the Third International Conference on Cyber-Technologies and Cyber-Systems, IARIA, Athens, Greece, 18–22 November 2018.
67. Idrissi, S.E.; Berbiche, N.; Sbihi, M. Performance evaluation of web application security scanners for prevention and protection against vulnerabilities. *Int. J. Appl. Eng. Res.* **2017**, *12*, 11068–11076.
68. Livshits, B.V.; Lam, M.S. Finding security vulnerabilities in java applications with static analysis. In Proceedings of the 14th Conference on USENIX Security Symposium USENIX Association, Berkeley, CA, USA, 31 July–5 August 2005.
69. Martin, B.; Livshits, B.; Lam, M.S. Finding application errors and security flaws using PQL: A program query language. In Proceedings of the 20th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, San Diego, CA, USA, October 2005. [CrossRef]
70. Krishnan, R.; Nadworny, M.; Bharill, N. Static analysis tools for security checking in code at Motorola. *ACM SIGAda Ada Lett.* **2008**, *28*, 76–82. [CrossRef]
71. Cifuentes, C.; Scholz, B. Parfait–designing a scalable bug checker. In Proceedings of the 2008 Workshop on Static Analysis, SAW '08, Tucson, AZ, USA, 12 June 2008; ACM: New York, NY, USA, 2008. [CrossRef]
72. Shrestha, J. Static Program Analysis. Ph.D. Thesis, Uppsala University, Uppsala, Sweden, 2013.
73. Goseva-Popstojanova, K.; Perhinschi, A. On the capability of static code analysis to detect security vulnerabilities. *Inf. Softw. Technol.* **2015**, *68*, 18–33. [CrossRef]
74. Pashchenko, I.; Dashevskyi, S.; Massacci, F. Delta-bench: Differential benchmark for static analysis security testing tools. In Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, IEEE Computer Society, Toronto, ON, Canada, 9–10 November 2017. [CrossRef]
75. Long, F.; Mohindra, D.; Seacord, R.C.; Sutherland, D.F.; Svoboda, D. *Java™ Coding Guidelines: 75 Recommendations for Reliable and Secure Programs*; Pearson Education: Boston, MA, USA, 2014.
76. Heckman, S.; Williams, L. A systematic literature review of actionable alert identification techniques for automated static code analysis. *Inf. Softw. Technol.* **2011**, *53*, 363–387. [CrossRef]
77. Antunes, N.; Vieira, M. On the metrics for benchmarking vulnerability detection tools. In Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Rio de Janeiro, Brazil, 22–25 June 2015. [CrossRef]