# Benchmarking Approach to Compare Web Applications Static Analysis Tools Detecting OWASP Top Ten Security Vulnerabilities

**Article** *in* Computers, Materials and Continua · June 2020

DOI: 10.32604/cmc.2020.010885



**5 authors**, including:

Juan-Ramón Bermejo Higuera
Universidad Internacional de La Rioja
**13** PUBLICATIONS **99** CITATIONS
SEE PROFILE

Javier Bermejo
Universidad Internacional de La Rioja
**16** PUBLICATIONS **59** CITATIONS
SEE PROFILE

Juan Antonio Sicilia Montalvo
Universidad Internacional de La Rioja
**49** PUBLICATIONS **229** CITATIONS
SEE PROFILE

*Computers, Materials & Continua* CMC, vol.64, no.3, pp.1555-1577, 2020

# Benchmarking Approach to Compare Web Applications Static Analysis Tools Detecting OWASP Top Ten Security Vulnerabilities

**Juan R. Bermejo Higuera[1, *], Javier Bermejo Higuera[1], Juan A. Sicilia Montalvo[1], Javier Cubo Villalba[1] and Juan José Nombela Pérez[1]**

**Abstract:** To detect security vulnerabilities in a web application, the security analyst must choose the best performance Security Analysis Static Tool (SAST) in terms of discovering the greatest number of security vulnerabilities as possible. To compare static analysis tools for web applications, an adapted benchmark to the vulnerability categories included in the known standard Open Web Application Security Project (OWASP) Top Ten project is required. The information of the security effectiveness of a commercial static analysis tool is not usually a publicly accessible research and the state of the art on static security tool analyzers shows that the different design and implementation of those tools has different effectiveness rates in terms of security performance. Given the significant cost of commercial tools, this paper studies the performance of seven static tools using a new methodology proposal and a new benchmark designed for vulnerability categories included in the known standard OWASP Top Ten project. Thus, the practitioners will have more precise information to select the best tool using a benchmark adapted to the last versions of OWASP Top Ten project. The results of this work have been obtaining using widely acceptable metrics to classify them according to three different degree of web application criticality.

**Keywords:** Web application, benchmark, security vulnerability, Security Analysis Static Tools, assessment methodology, false positive, false negative, precision, f-measure.

## 1 Introduction

Today companies and organizations use web applications to manage their data from any place, Intranets, Internet, etc. It can be assured that web applications have the gift of ubiquity, which implies at the same time they are more "attackable" and have to be security hardened.

The results of several studies [Homaei and Shahriari (2017); Barabanov, Markov and Tsirlov (2018); Sołtysik-Piorunkiewicz and Krysiak (2020)] confirm that the web applications analyzed did not pass the OWASP Top Ten project [OWASP Foundation (2017)]. The security vulnerabilities, that web applications have in their code should force organizations to make a security analysis using the best SAST tools. SQL injection (SQLI) and Cross Site

[1] Escuela Superior de Ingeniería y Tecnología, Universidad Internacional de La Rioja, La Rioja, 26006, Spain.

[*] Corresponding Author: Juan R. Bermejo Higuera. Email: juanramon.bermejo@unir.net.

Received: 03 April 2020; Accepted: 28 April 2020.

Scripting (XSS) vulnerabilities continue to be the most frequent vulnerabilities.

The task of manually scanning a web application with a high number of lines of code for security vulnerabilities can become an arduous and time-consuming task. It is necessary to study the state of art in automatic techniques such as source and binary code SAST tools. Some works confirm that one of the great advantages of this type of tool, as opposed to the purely manual analysis, is that they analyze all the code and the configurations of a web application achieving rates higher than 70% of security vulnerabilities [Díaz and Bermejo (2013); Antunes and Vieira (2015); Shrestha (2013); Nunes, Medeiros, Fonseca et al. (2018)]. However, the work of Nguyen et al. [Nguyen, Maleehuan, Aoki et al. (2019)] demonstrates that static tools have false positives (vulnerabilities detected that really does not exist) and false negatives (real vulnerabilities not found). Therefore, a final audit of a SAST tool report is required to confirm each security vulnerability. Other works have investigated another method that combine static and dynamic working together to take advantage of possible synergies to eliminate false positive and detect more true positives [Monga, Paleari and Passerini (2009); Nunes, Medeiros, Fonseca et al. (2019); Bermejo, Abad, Bermejo et al. (2020)]. However, a false negative is harder to find if the tool has not previously detected it causing a real danger. False positives are not really a danger and they could be fixed by the security analyst.

There are some questions to investigate:

- How is the SAST tools detection ratio with OWASP Top Ten security vulnerabilities using the approach benchmarking?
- How is the SAST tools false alarm ratio with OWASP Top Ten security vulnerabilities using the approach benchmarking?
- How is true positives/false positives balance of tools against different critical degree of the web applications?
- How is the best tool for analyzing the security of web applications with different levels of criticality?
- Is the OWASP Top Ten Benchmark adequate to comparing SAST tools?
- How are the results of this SAST tools comparative using OWASP Top Ten Benchmark compared with the results of comparatives using other benchmarks?

It is necessary to fix all security vulnerabilities discovered in the source code. The security points addressed in the work of Homaei et al. [Homaei and Shahriari (2017)] are related to the study of the most frequent and dangerous vulnerabilities that exist in the code. SAST tools need to be integrated into the Software Development Life Cycle (SSDLC) as the model described in the work of Vicente et al. [Vicente, Bermejo, Bermejo et al. (2019)] to detect security vulnerabilities early. Therefore, we consider that users and professionals of web applications should know which commercial and open source SAST tools have better effectiveness in terms of real detection rates (true positives), unreal detections (false positives) and vulnerabilities not found (false negatives).

Next, it is described the main contributions of our work. The first goal of this study is to design a benchmarking approach that includes a specifics test bank with test cases for OWASP Top Ten security vulnerabilities [Bermejo (2020)].

The second main goal of this study is to compare the effectiveness of seven SAST tools selected for J2EE specification commercial and open source projects (Spotbugs, FindSecurityBugs, Klocwork-insight, Fortify SCA, Checkmarx, Coverity and Xanitizer). The tools are executed against the new benchmark adapted to OWASP Top Ten project to obtain the results of tools effectiveness. We apply well-known metrics to the execution results to obtain a strict rank of tools. Finally, the paper gives some practical recommendations on how to improve their effectiveness using the tools.

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

- A general approach to design a benchmark for the evaluation of SAST capable of detecting software vulnerabilities, considering the OWASP Top Ten project vulnerability categories.
- A concrete instantiation of the OWASP Top Ten Benchmark approach to demonstrate its feasibility, evaluating seven SAST tools by the detection of web vulnerabilities using appropriate criteria for benchmark instantiation.
- An analysis of the results obtained by SAST tools using a defined comparative method to classify them according to different degree levels of web applications importance.
- An analysis of results of leading commercial SAST allows the practitioners to choose the most appropriate tool to perform a security analysis.

The outline of this paper is as follows: Section 2 reviews background in web technologies security focusing in vulnerabilities, SAST tools, security benchmarks and related work. In Section 3 the OWASP Top Ten benchmarking approach to evaluate SAST tools is presented. Section 4 describes the steps of the comparative methodology proposal designed by enumerating the steps followed to rank the SAST tools using the benchmark approach. Finally, Section 5 collects the conclusions and Section 6 sketches the future work.

## 2 Background and related work

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

### *2.1 Web technologies security*

The growing development of web applications in the environment of organizations and companies connected through the Internet means that they have become a precious target of attack by those who wish to make a profit exploiting the vulnerabilities that these applications may have. Organizations, to this constant threat, must be aware that investment in web application security must be planned from the beginning, generally referred to start in the initial stages of application development.

The most used languages today are some of the .NET technology such as C # or Visual Basic, there is also an increase in the popularity of Android and iOS, PHP and C/C ++. Java is still the most used language, as is also confirmed by different studies [Nanz and Furia (2015); Arouba and Fernández-Villaverde (2015)]. In other publications the vulnerabilities of the new generation of Web applications are discussed: Web 2.0 focusing on the types of attacks on applications that use Asynchronous Javascript and Xml (AJAX), HTML5 or

flash technologies [Cannings, Dwivedi and Lackey (2008); Scambray, Liu and Sima (2010); Sema (2012); Moeller (2016)].

We use the term "secure" to refer to languages automatically perform runtime checks to prevent the programs violate the limits of the allocated memory. The work by Long et al. [Long, Mohindra, Seacord et al. (2014)] confirm that Java is essentially a secure language.

In order to build adequate benchmarks to test and select the best performance SAST tools, we need to investigate which security vulnerabilities are more frequent and dangerous in the web applications. Some studies confirm that the vulnerabilities more frequent and dangerous are the included in OWASP Top Ten project and that the web applications analyzed did not pass the OWASP Top Ten project [Homaei and Shahriari (2017); Barabanov, Markov and Tsirlov (2018); Sołtysik-Piorunkiewicz and Krysiak (2020)].

### *2.2 Benchmarks*

There have been some attempts for benchmarking static analysis tools for web applications:

- Wavsep project is designed for a reduced set of vulnerabilities as XSS, Path Traversal, Remote File Inclusion (RFI), Open Redirect and SQLI, though with a lot of test cases for the vulnerabilities that it covers. It is included in the dynamic security tools' comparisons [Alavi, Bessler and Massoth (2018); Idrissi, Berbiche and Sbihi (2017)].
- Securebench Micro project focuses in Java vulnerabilities not for J2EE. Diverse comparisons can be found using Securebench Micro [Livshits and Lam (2005); Martin, Livshits and Lam (2005)].
- Software Assurance Metrics and Tool Evaluation (SAMATE) project of National Institute of Standards and Technology (NIST) includes the test suite Juliet composed of 13782 test cases that covers widely vulnerabilities. This benchmark has been used in several works [Krishnan, Nadworny and Bharill (2008); Cifuentes and Scholz (2012); Shrestha (2009); Díaz and Bermejo (2013); Goseva-Popstojanova and Perhinschi (2015); Nunes, Medeiros, Fonseca et al. (2018)].
- OWASP benchmark project [OWASP Foundation (2020)]. The Benchmark is used in several works [Burato, Ferrara and Spoto (2017), Deshlahre and Tiwari (2020)]. It contains a set of test cases are fully runnable and exploitable. It includes test cases for XSS, SQLI, Xml Path Injection (XPATHI), command injection, path traversal and other sensitive data exposure vulnerabilities included in OWASP Top Ten 2013.
- Delta-bench by Pashchenko et al. [Pashchenko, Dashevskyi and Massacci (2017)] is a benchmarking approach based in 108 security vulnerabilities as SQLI, XSS, LFI, Session fixation, Exec code or Information disclosure of Apache Tomcat. It was evaluated using six open source SAST tools and Fortify SCA commercial tool.

### *2.3 SAST tools*

The study of Huth et al. [Huth and Nielson (2019)] confirms that the most desirable to avoid vulnerabilities in a web application code is prevention. Developers should have received training in web security programming to avoid making "mistakes" related to

programming vulnerabilities. Despite the very good training of security programmers, there will always be vulnerabilities in the code and, once the first version of the application or parts of it have been developed, it will be much more complicated to review the source code or perform a dynamic security analysis. Manual static analysis requires highly specialized staff and time. The work of Sttutard et al. [Stuttard and Pinto (2008)] concludes that the manual dynamic analysis in an attempt to perform an "ethical hacking" of the implemented application requires highly specialized personnel, it is very difficult to cover the entire attack surface of the application takes a long time to complete. To carry out an analysis of web application security, made by any method, you must cover the entire surface of attack covering all parts and application layers and using tools to automate security analysis as much as possible, combining various types of tools and achieve the better results [Felderer, Büchler, Johns et al. (2016)].

One type of security analysis is white box analysis performed by SAST tools, which analyzes both source code and executable, as appropriate. SAST tools start with a problem because the act of determining if a program reaches its final state, or not. For further reading on the computational theory is recommended Sipser's "*Introduction to the Theory of Computation*", second Edition [Sipser (2005)]. Despite this problem, security code analysis using these tools are considered the most important security activity within a SSDLC [Vicente, Bermejo, Bermejo et al. (2019); Duclervil and Liou (2019)].

A final audit of a SAST tool report is required to eliminate the false positives and find the false negatives (much more complicated). Security analysts need to adequate the training to reconnaissance the security vulnerabilities in the code for a particular programming language [Yang, Tan, Peyton et al. (2019)]. SAST tools interfaces can be more or less "friendly" in terms of the error trace facilities to audit a security vulnerability. In the study of Díaz et al. [Díaz and Bermejo (2013)] tools such as Fortify SCA, Coverity, Checkmarx or Klocwork are good examples of tools that provide a very good information for eliminating false positives.

One of the most important advantages of SAST tools is that they analyze the entire application covering all source inputs. The studies of Antunes et al. [Antunes and Vieira (2010, 2015)], compare SAST tools vs DAST tools against web services benchmarks and the SAST tools generally obtain better true positive ratios and worse false positive ratios. This gives an idea of the importance of static analysis including a manual audit of the results. However, the works of Antunes et al. [Antunes and Vieira (2010, 2015)] confirm that SAST and DAST can find distinct types of vulnerabilities and consequently is a good idea using the two types of analysis correlating their results improving true and false positive ratios. Other works combine SAST tools with IAST tools to runtime monitor attacks with the information of static analysis [Mongiovi, Giannone, Fornaia et al. (2015); Loch, Johns, Hecker et al. (2020)]. The work of Pistoia et al. [Pistoia, Tripp and Lubensky (2018)] combines static analysis with machine learning techniques for automatic detection of vulnerabilities in mobile applications. Another distinct approximation is the design and implement of a Multidimensional and Hierarchical Web Anomaly Detection System [Guan, Li and Jiang (2019)].

### *2.4 Related work*

The comparison of Ware et al. [Ware and Fox (2008)] concludes that, of the total of nine involved tools, only two are valid for J2EE web applications: Spotbugs and Fortify SCA.

The master thesis of Shrestha [Shrestha (2013)] studies the state of art of open source static analysis tools. This study is focused on the research of C/C++ and Java based static analyzers, which are open source tools. In particular, Shrestha work studies Spotbugs, analyzes the results and compares it with Parasoft Jtest commercial tool. Spotbugs detects a 20% of vulnerabilities and Jtest detects only a 0.05% of vulnerabilities in SAMATE Juliet test suites. Spotbugs has a high number of false positives. One conclusion was that commercial SAST tools should be having into account.

A study of three open source tools is accomplished also with SAMATE Java and C/C++ test cases [Goseva-Popstojanova and Perhinschi (2015)]. It concluded that three tools missed 11% of Java vulnerabilities. Only one detected some vulnerabilities or combination of two tools; 41% of C/C++ and 21% of Java vulnerabilities were detected by three tools. The study is considered unrepresentative because it only computes three open source tools and more open source and commercial tools need to be included in the study to make it more representative. Besides, the study is performed against web vulnerability categories without having into account OWASP Top Ten vulnerability categories.

NIST Static Analysis Tool Exposition's (SATE). From 2008, there have been six SATE different projects with diverse objectives. SATE V project proposes a methodology to assess tool effectiveness. Others can use this methodology to determine which tools fit their requirements [NIST (2018)].

With respect to benchmarking approaches, the work of Antunes et al. [Antunes and Vieira (2015)] is a benchmarking approach to analyze the effectiveness of security analyzers tools in Web services environments. This approach defined two concrete benchmarks for SQL Injection vulnerability detection tools. The two benchmarks are used to evaluate and compare several widely used tools, four penetration testers, three SAST tools, and one anomaly detector. However, it could improve their representative with respect to security vulnerabilities coverage for other security vulnerabilities besides SQLI.

Of particular interest is another work of Nunes et al. [Nunes, Medeiros, Fonseca et al. (2018)] that addressed the problem of choosing adequate SAST tools for vulnerability detection in web applications. They proposed an approach to design benchmarks for evaluating such SATS tools considering different levels of criticality. Each scenario uses different metrics to rank the tools. To evaluate the approach, they created a benchmark for WordPress plugins and tested it with five free SASTs searching for XSS and SQLI vulnerabilities in 134 WordPress plugins with real vulnerabilities, developed in PHP. The experimental results showed that the best tool changes from one scenario to another and depends on the class of vulnerabilities being detected. The results show that the metrics could be improved to balance the weight of the true positives (TP) and false positives (FP) in the computation of the metrics. However, it could improve their representative with respect to security vulnerabilities coverage for others besides SQLI and XSS.

Some studies about how SAST tools combining is very interesting [Algaith, Nunes, Fonseca et al. (2018); Nunes, Medeiros, Fonseca et al. (2019)]. These work combines

diverse SAST tools to improve the performance detection of security vulnerabilities in web applications, having into account four development scenarios with different criticality levels and constraints obtaining precise combining results of the tools according the proposed metrics. As previously analyzed studies, it could be improved designing the used benchmark with additional security vulnerabilities than SQLI and XSS.

The main conclusions of the related work examined are that the existing comparatives do not include an adequate number of leading commercial tools and the benchmarks used are not complete and representatives with respect to OWASP Top Ten project.

## 3 Benchmarking approach

An adequate test bench must comply with the properties and the considerations made in the work [Antunes and Vieira (2015)]. Another premise is that the tools execution must be under the same conditions. The new benchmark approach meets these properties, it is credible, portable, representative, require minimum changes and easy to implement and run [Bermejo (2020)].

The benchmark will be used to compare and rank seven SAST tools with minimum changes in their configurations, five SAST commercial tools and two open source tools. In a first phase true positive and false positive metrics will be obtained and in a second phase other metrics will be calculated with the objective of rank the tools having into account the different levels of criticality that the web applications can have.

According to Martin et al. [Martin and Barnum (2008)]: "*a benchmark could also motivate its use as a referential standard by community players such as OWASP, the SANS (SysAdmin, Audit, Networking, and Securityand many others*."

The OWASP Top Ten benchmarking approach [Bermejo (2020)] covers the most dangerous security vulnerabilities of web applications according to OWASP Top Ten 2013 and OWASP Top Ten 2017 projects and having into account statistics of security vulnerabilities reported by several studies [Barabanov, Markov and Tsirlov (2018); Sołtysik-Piorunkiewicz and Krysiak (2020)]. Tab. 1 shows the security vulnerabilities included in the benchmarking approach selecting the more adequate test cases for the main vulnerability categories of OWASP Top Ten 2013 and 2017 projects from SAMATE Juliet benchmark. The result is a set of vulnerability categories (for example injection) each one with a set of vulnerability types (for example SQL injection, LDAP injection, etc.) with a number of test cases to test SAST tools behavior and performance. Each test case is designed with a concrete vulnerability included in OWASP Top Ten 2013 and 2017 projects. The benchmark is easily portable as a java project. Also, it does not require changes with any tool and finally the benchmark is representative according to OWASP Top Ten 2013 and 2017 projects.

**Table 1:** Benchmarking approach for OWASP Top Ten [Bermejo 2020]

| CWE | Vulnerability categories and types by category | TP test cases | FP test cases |
|---|---|---|---|
| **INJECTION** | | | |
| 89 | SQL_Injection | 19 | 58 |
| 90 | LDAP_Injection | 11 | 17 |
| 566 | Access_Through_SQL Primary | 10 | 30 |

| | | | |
|---|---|---|---|
| 78 | Command_Injection | 10 | 16 |
| 113 | HTTP_Response_Splitting | 32 | 88 |
| 643 | Unsafe_Treatment_XPath Input | 2 | 4 |
| **BROKEN AUTENTICATION AND SESSIONS** | | | |
| 256 | Plaintext_Storage_of Password | 2 | 4 |
| 257 | Storing_Password Rec._Format | 2 | 4 |
| 259 | Hard_Coded_Password | 2 | 3 |
| 293 | Using_Referer_Field_for Auth. | 2 | 4 |
| 315 | Plaintext_Storage_in_a Cookie | 2 | 8 |
| 321 | Hard_Coded_Cryptographic Key | 2 | 4 |
| 523 | Unprotected_Cred_Transport | 2 | 4 |
| 547 | Hardcoded_Security Constants | 2 | 4 |
| 549 | Missing_Password_Masking | 2 | 3 |
| 603 | Client_Side_Authentication | 2 | 4 |
| 613 | Insufficient_Session Exp. | 2 | 2 |
| 614 | Sensitive Cookie Without Secure | 2 | 4 |
| **SENSITIVE DATA EXPOSURE** | | | |
| 209 | Information_Leak_Error | 2 | 5 |
| 319 | Plaintext_Tx_Sensitive_Info | 2 | 8 |
| 489 | Leftover_Debug_Code | 2 | 4 |
| 497 | Information_Leak_SystemData | 2 | 2 |
| 598 | Information_Leak QueryString | 2 | 4 |
| 615 | Info_Leak_By_Comment | 2 | 4 |
| **BROKEN ACCESS CONTROL** | | | |
| 23 | Relative_Path_Traversal | 11 | 18 |
| 36 | Absolute_Path_Traversal | 9 | 14 |
| 378 | Creation_of_File_with Insec_Per | 2 | 4 |
| 367 | TOC_TOU | 2 | 2 |
| 567 | Unsynchronized_Shared_Data | 1 | 1 |
| **SECURITY MISCONFIGURATION** | | | |
| 328 | Reversible_One_Way_Hash | 2 | 4 |
| 330 | Insufficiently_Random Values | 2 | 2 |
| 336 | Same_Seed_in_PRNG | 2 | 3 |
| 759 | Unsalted_One_Way_Hash | 2 | 3 |
| **CROSS SITE SCRIPTING** | | | |
| 80 | XSS | 11 | 15 |
| 81 | XSS_Error_Message | 13 | 23 |
| 83 | XSS_Attribute | 8 | 13 |
| **USING COMPONENTS WITH KNOWN VULNERABILITIES** | | | |
| 327 | Use_Broken_Crypto | 2 | 4 |
| 338 | Weak_PRNG | 2 | 4 |
| 760 | Predictable_Salt_One_Way Hash | 2 | 4 |
| **CSRF** | | | |
| 352 | Cross_Site_Request_Forgery | 7 | 20 |
| **OPEN REDIRECT** | | | |
| 601 | Open_Redirect_Servlet | 11 | 17 |

| | | |
|---|---|---|
| Nº TEST CASES | 209 | 439 |

For each test case we have selected variants in flow complexity and input source to the application. Each test case has a function called bad() (see Fig. 1) which has an input source that is not validated (badsource) and a line in the code also not validated where the vulnerability materializes (badsink). It also indicates the variation of code complexity, flow variant and distinct ways of source inputs. Examples of inputs to the application are console_readline, environment, fromDB, fromFile, GetCookieServlet, GetParameterServlet, GetQueryStringServlet, connect_tcp, etc. The final composition of OWASP Top Ten Benchmark comprises 209 bad functions in test cases for calculating the true positive rate and 439 good functions test cases for calculating false positive rate. The different number of bad and good functions is due to a test case that has one bad function, but it can have several versions of good functions (from 1-4 depending of each test case) with good source input, (goodsource), or good sink (goodsink).

```
/* CWE: 80 Cross Site Scripting (XSS) * BadSource: getCookiesServlet Read data from the first cookie * GoodSource: A hardcoded string * BadSink: Servlet querystring parameter not sanitized * Flow Variant: 01 Baseline */
package testcases CWE80_XSS;
import testcasesupport.*;
import javax.servlet.http.*;
import javax.servlet.http.*;
import java.util.logging.Logger;
public class CWE80_XSS__Servlet_getCookiesServlet_01 extends AbstractTestCaseServlet
{   public void bad(HttpServletRequest request, HttpServletResponse response) throws Throwable // uses badsource and badsink
   { String data;
     Logger log_bad = Logger.getLogger("local-logger");
     Cookie cookieSources[] = request.getCookies(); // Source → read parameter from cookie
     if (cookieSources != null)  {
       data = cookieSources[0].getValue();
     } else { data = null;}
     if (data != null){  /* POTENTIAL FLAW: data not validated */
        response.getWriter().println("<br>bad() - Parameter name has value " + data);} //Sink
 }
```

**Figure 1:** XSS test case example with bad() function [Bermejo (2020)]

## 4 Methodology proposal to compare SAST tools

In this section we present a new methodology repeatable to compare and rank the SAST tools.

- Select the OWASP Top Ten Benchmark designed.
- Select the SAST tools. In this concrete instantiation of the methodology we choice seven commercial and open source SAST tools according to the analysis of the related

works in Section 2.4 and official lists of SAST tools.

- Run the selected SAST tools against the OWASP Top Ten Benchmark designed with the default configuration for each tool.
- Select appropriate metrics to analyze results.
- Rank the SAST tools according the result metrics.
- Analysis, discussion and ranking of the results.

### *4.1 SAST selection*

The next step is the selection of seven (7) commercial and open source static analysis tools for source or executable code that can detect vulnerabilities in web applications developed using the J2EE specification. SAST tools are selected according with J2EE, the most used technology in web developing, the programming language used by J2EE, Java, is one of the labeled as more secure [Long, Mohindra, Seacord et al. (2014)].

With the premises of the above comparatives and analyzing the availability of commercial and open source tools are selected seven (7) meaning tools.

Selected tools:

- Fortify SCA. It supports 18 distinct languages, the most known OS platforms and offers SaaS (Software as a service) and it detects more than 479 vulnerabilities.
- Checkmarx CxSAST. It supports Java, JSP, C#, and ASP, VB.NET, VB6, C++, PHP, APEX, Javascript and VBscript languages.
- Klocwork Insight. It supports Java-J2EE, C#, C/C++ languages and Windows, UNIX, Mac, Android platforms and eclipse plugin. Also, it detects a wide set of vulnerabilities.
- Xanitizer. It supports only Java language, but it gives the possibility to the developers of sanitizing the inputs variables in the code.
- Coverity supports a great quantity of languages, as C/C++, Java, C#, Javascript, HTML5, Typescript and others.
- FindSecurityBugs. (Open source). Plugins are available for SonarQube, Eclipse, IntelliJ, Android Studio and NetBeans. Command line integration is available with Ant and Maven.
- SoptBugs 2.0. (Open source). It supports only J2EE language and it can integrate in Eclipse. It supports a reduced set of vulnerabilities.

### *4.2 Metrics to analyze the results*

A selection of adequate metrics must be applied to the results of the performed test to better understand the obtained measurements. The used metrics are widely accepted in others works [Heckman and Williams (2011); Antunes and Vieira (2010, 2015); Díaz and Bermejo (2013); Goseva-Popstojanova and Perhinschi (2015); Nunes, Medeiros, Fonseca et al. (2018)].

The summary of metrics used is:

- %TP, number and percentage of true positives TP (correct detections).
- %FP, number and percentage of false positives FP (detecting no error).

- Number of vulnerability categories for which the tool is designed.
- Precision (1). Proportion of the total TP detections:

$$TP/(TP + FP) \quad (1)$$

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.

- Recall (2). Ratio of detected vulnerabilities to the number that really exists in the code. Recall is also referred to as the True Positive Rate:

$$TP/(TP + FN) \quad (2)$$

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.

- Harmonic mean (3):

$$\frac{n}{\left(\frac{1}{X_1}+\frac{1}{X_2}\ldots+\frac{1}{X_n}\right)} \quad (3)$$

where n: number of variables and $x_{n:}$ value of variable n.

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

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

- $F_\beta$-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:

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

### *4.3 Ranking of SAST tools*

In this section, the selected tools run against the OWASP Top Ten Benchmark approach test cases. In each test execution, we obtain the true positive and false positive results for each type of vulnerability. Next, the metrics selected in Section 4.2 are applied to obtain the most appropriate measures to promote good interpretation of the results and to draw the best conclusions.

In Tab. 2 the number of detected vulnerabilities (true positives) is accounted. The total of analyzed test cases was 209.

The number of test cases in each type of vulnerability is variable inside a vulnerability category. To normalize the result of detections in each vulnerability category (p.e. injection) we calculate the percentage of detections for each type of vulnerability (SQLI, XPATHI) included in a concrete category. Following the arithmetic mean of all types of vulnerability, the detection percentage is calculated for each vulnerability category. Finally, last file of Tab. 3 shows also the arithmetic mean of detection percentage for all categories of vulnerabilities for each tool.

**Table 2:** Vulnerabilities detection. True positive ratio

| | Xanitizer | Coverity | Checkmarx | Klocwork | Fortify | SpotBugs | FsecBugs |
|---|---|---|---|---|---|---|---|
| Injection | 66% | 40,41% | 76,6% | 35% | 83,3% | 5,4% | 63,5% |
| Broken auth | 25% | 0% | 8,3% | 8,3% | 25% | 0% | 20,8% |
| Sensitive data | 0% | 0% | 8,3% | 16,6% | 50% | 0% | 0% |
| Broken A.C | 16,14% | 49,72% | 60% | 71,9% | 40% | 3,6% | 57,7% |
| Broken conf | 75% | 50% | 25% | 25% | 50% | 0% | 50% |
| XSS | 90,9% | 51,5% | 29,5% | 61,5% | 47,6% | 3% | 5,5% |
| Comp. Vuln. | 66,3% | 33,3% | 33,3% | 33,3% | 66,6% | 0% | 66,6% |
| CSRF | 0% | 0% | 100% | 71,4% | 100% | 0% | 14,2% |
| Open redirect | 81,8% | 81,8% | 63,6% | 81,8% | 54,5% | 27,2% | 100% |
| **TP percent** | **47%** | **34%** | **44,9%** | **45%** | **57,4%** | **4,3%** | **42%** |

Tab. 3 shows a summary of results in the total of 439 false positive test cases. There are more test cases than in Tab. 2 because the bad function of each test case has several good functions versions for each bad function.

With the purpose of normalizing the result of false positives results in each vulnerability category (for example injection) we calculate the percentage of false positive alarms for each type of vulnerability (SQLI, XPATHI, etc.) included in a concrete category. Following the arithmetic mean of all types of vulnerability false positive percentage is calculated for each vulnerability category. Finally, last file of Tab. 3 shows also the arithmetic mean of false positive percentage for all categories of vulnerabilities for each tool.

**Table 3:** Vulnerabilities detection. False positive ratio

| | Xanitizer | Coverity | Checkmarx | Klocwork | Fortify | SpotBugs | FsecBugs |
|---|---|---|---|---|---|---|---|
| Injection | 56,9% | 34,43% | 58,7% | 30% | 75,7% | 4,15% | 40% |
| Broken Auth | 33,3% | 16,6% | 8,3% | 4,2% | 20,8% | 0% | 8,3% |
| Sensitive data | 16,6% | 8,3% | 8,3% | 16,6% | 33,3% | 0% | 0% |
| Broken A. C. | 7,66% | 29% | 47% | 65,8% | 32,5% | 1% | 42,8% |
| Broken conf | 25% | 12,5% | 25% | 25% | 50% | 0% | 25% |
| XSS | 57,2% | 30,7% | 26,1% | 44,7% | 32,2% | 0% | 4,4% |
| Comp. Vuln. | 16,6% | 25% | 33,3% | 33,3% | 50% | 0% | 16,6% |
| CSRF | 0% | 0% | 85% | 55% | 90% | 0% | 0% |
| Open redirect | 47,3% | 21% | 52,9% | 70,5% | 41,1% | 0% | 29,4% |
| **FP percent** | **28,4%** | **19,7%** | **38,2%** | **38%** | **47,2%** | **0,6%** | **19%** |

### *4.4 OWASP Top Ten Benchmark results analysis*

This section aims to obtain a classification of the tools according to the metrics applied to the results obtained from the execution of the tools against OWASP Top Ten Benchmark.

Subsequently we have calculated the selected metrics: precision, recall, false positives and F-measure. Recall metric is adequate for high critical applications where the objective is to discover the highest number of vulnerabilities. Precision metric penalizes true positive ratio having into account the false positive score. Usually a tool has a direct proportionality between its true and false positives results. A good tool should break this direct proportionality. F-measure is the best metric for selecting the tool that detects a high number of vulnerabilities while reporting a low number of false positives for a best effort (heightened-critical applications). The work of Antunes and Vieira [Antunes and Vieira (2015b)] analyzes distinct metrics for different level of the importance of web applications. The maximum score in all metrics is (1). Also, we calculate other two derived metrics from F-measure, $F_{1,5}$-Score and $F_{0,5}$-Score metrics. $F_{1.5}$ Score metric normalize precision and recall metrics giving more weight to recall metric. This metric is the most adequate metric for critical applications as it allows to reward the tools with better recall. This makes tools with high recall to obtain much better results. $F_{0,5}$ Score metric favors precision and is adequate metric for non-critical applications where the time of development can be quicker as it allows to reward the tools with better precision. This makes tools with high precision to obtain much better results.

The results assessment of the execution of tools against the benchmark is accomplished applying the following metrics of Section 4.2, false positive ratio (percent), Recall, Precision and three ranking classifications: F-measure (best effort), $F_{1,5}$-score and $F_{0,5}$-score.

Metrics calculation:

- TP mean a vulnerability category is calculated obtaining the percent of TP for each vulnerability type included in each vulnerability category and finally is calculated the mean for each vulnerability category.
- Recall is a TP mean/100.
- FP mean a vulnerability category is calculated obtaining the percent of FP for each vulnerability type included in each vulnerability category and finally is calculated the mean for each vulnerability category.

F-measure, $F_{1.5}$ Score and $F_{0.5}$ Score are calculated according to their equations described in the Section 4.2, Eqs. (4) and (5) respectively.

The classification order according to the F-measure score are showed in Tab. 4.

**Table 4:** Assessment results computing and ranking the selected metrics by F-measure

| Metric / Tool | TP Mean | FP Mean | Precision | Recall | $F_{0.5}$ Score | $F_{1.5}$ Score | F-measure |
|---|---|---|---|---|---|---|---|
| Fortify | 57,4 | 47,2 | 0,548 | 0,574 | 0,553 | 0,566 | **0,561** |
| Xanitizer | 47 | 28,4 | 0,623 | 0,470 | 0,584 | 0,508 | **0,535** |
| Findsecbugs | 42 | 19 | 0,689 | 0,420 | 0,610 | 0,477 | **0,521** |
| Klocwork | 45 | 38 | 0,542 | 0,450 | 0,520 | 0,474 | **0,491** |
| Checkmarx | 44,9 | 38,2 | 0,540 | 0,449 | 0,518 | 0,473 | **0,490** |
| Coverity | 34 | 19,7 | 0,633 | 0,340 | 0,539 | 0,396 | **0,446** |
| Spotbugs | 4,3 | 0,6 | 0,877 | 0,043 | 0,178 | 0,058 | **0,081** |

### *4.5 Analysis and discussion*

In this section, we analyze and discuss about the research questions formulated in Section 1 replying them according to the method proposed with the corresponding process, presenting the results.

- **How is the SAST tools detection ratio with OWASP Top Ten security vulnerabilities using the benchmarking approach?**

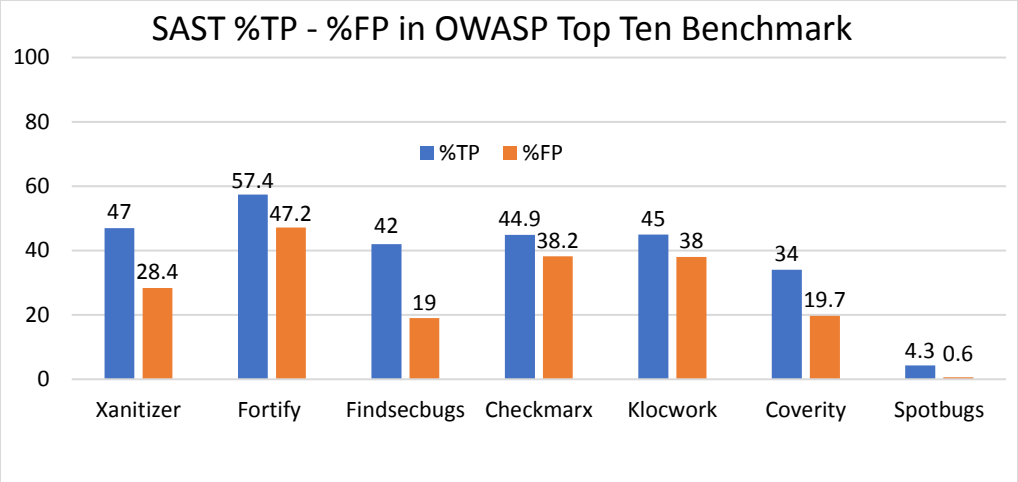


**Figure 2:** True and false positive percentajes obatined by the SAST tools

According to the Tabs. 3 and 4. (see Section 4.3), the results of the tools are different in terms of true and false positives detecting vulnerabilities suggest that using more than one SAST tool in combination can improving the TP ratio.

Types of vulnerabilities included in injection category are detected by FindSecurityBugs with a 63,5%, Xanitizer with a 66%, Checkmarx with a 76.6%, Fortify is the best tool with 83,3%. Klocwork obtains a 35%. Spotbugs with a 5,4% obtains the worst result.

Analyzing the results, Sensitive Data Revealed and Broken Authentication and Sessions are the vulnerability categories with worse TP ratio, lesser than 12%. Open Redirect is the vulnerability category with the best TP ratio (70%).

All tools obtain different results for the same vulnerability categories in many cases and six of the tools have similar TP ratio, between 34% and 58%, except for Spotbugs with a 4,3%. Fortify has the best TP ratio with 57,4% of detections, see Fig. 2.

- **How is the SAST tools false alarms ratio with OWASP Top Ten security vulnerabilities using the approach benchmarking?**

Analyzing the results, all tools obtain different results for the same vulnerability categories in many cases. Xanitizer, Klocwork, Checkmarx and Fortify obtain higher FP ratio from 28% to 47,2%. Coverity and FindSecurityBugs obtain a more moderate FP ratio (19%) and Spotbugs obtains a 0, 57% but it has a TP ratio of 4,3% of detections.

- **How is the balance of true and false positives of the SAST tools?**

According to Fig. 2, if it is considered the biggest difference between TP and FP ratios, FindSecurityBugs, Xanitizer and Coverity obtain the best balance between True and False positives, but it is necessary to analyze the precision metric that is calculated with TP and FP metrics.

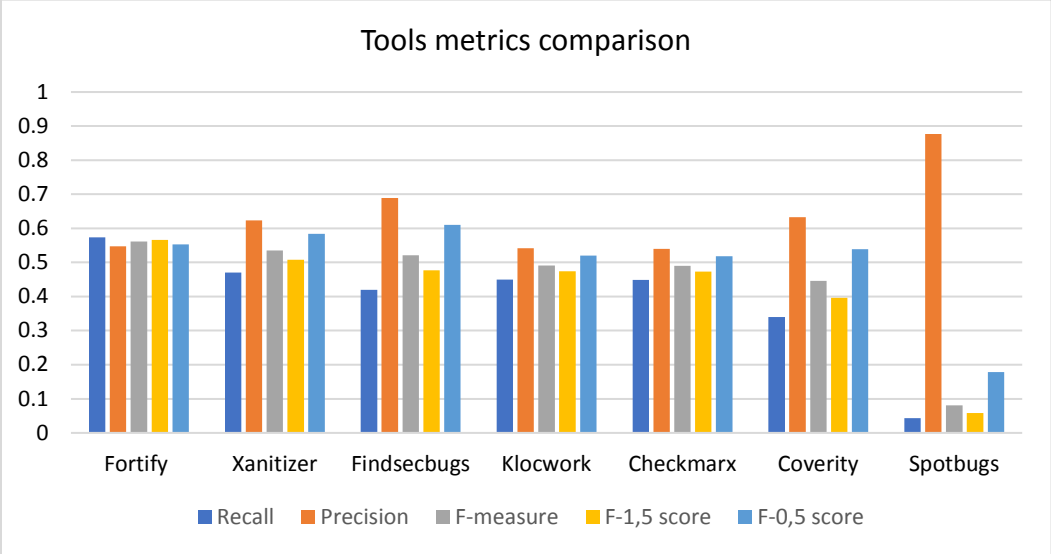


**Figure 3:** Metrics obtained by the SAST tools comparison

Fig. 3 shows a comparative graphic of the metrics results of all tools included in this analysis. Normally the TP ratio has a direct proportionality relationship with FP ratio. The best balance for this relationship in a concrete tool is having the higher TP ratio with a lesser ratio of FP ratio breaking the direct proportionality relationship. Precision metric normalizes TP and FP metrics penalizing the ratio of TP with the ratio of FP. The average ratio of precision for all analyzed tools is 0.636 what it suggests that the used SAST tools have a wide margin of improving. Spotbugs and FindSecurityBugs (open source tools) have the best results for precision metric (0,877 and 0,689).

In the case of Soptbugs it has few false positives but also it obtains very few true positives what it suggests that other metrics must have into account for characterize the Spotbugs performance. The rest of the tools obtains a precision value that goes from 0.548 to 0,689. Xanitizer, Coverity and FindSecurityBugs have similar precision value (0,623, 0,633 and 0,689).

The obtained values suggest that all examined tools, except for Spotbugs, must improve having a smaller number of false positives and a better precision.

- **How is the balance of true and false positives by vulnerability category?**

Fig. 4 shows SAST average of TP and FP ratios by vulnerability category. The vulnerability category with the best balance between TP and FP ratios are Open Redirect (TP percent=70,1%, FP percent=37,46%). Using Components with Known Vulnerabilities, XSS, and Broken Configurations have a balance between 23% of FP ratio and 43% TP ratio. However, CSRF, Injection and especially Sensitive Data Revealed and Broken Configuration have a worse result with respect to the balance between TP and FP ratios, in the case of Sensitive Data Revealed the FP ratio is higher than TP ratio.

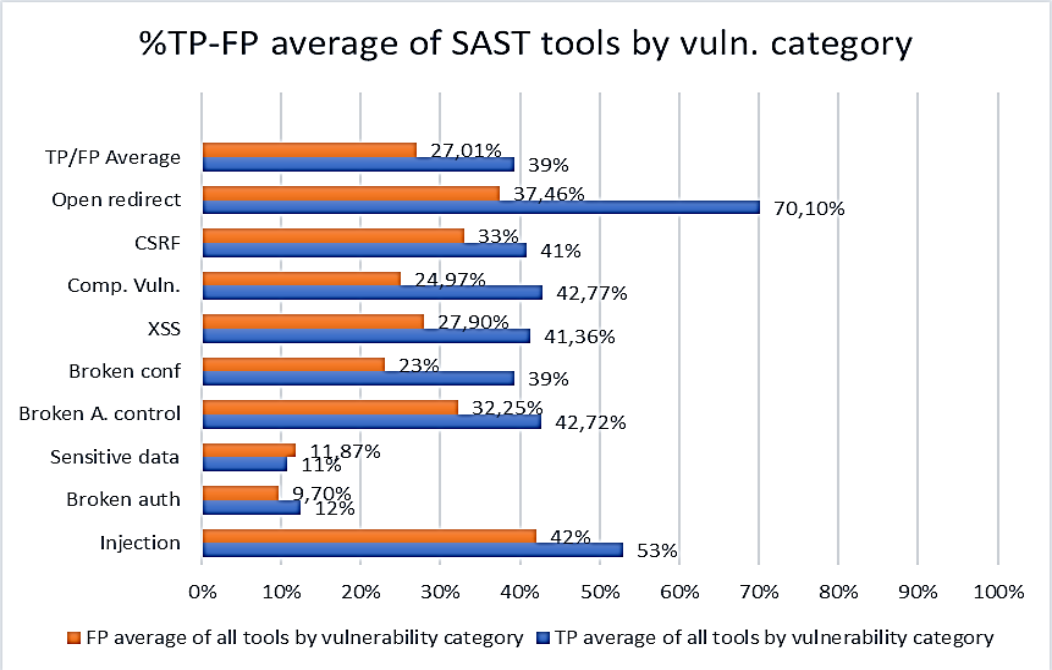


**Figure 4:** TP- FP average percent of SAST tools by vulnerability category

The TP and FP average ratios of all vulnerability categories obtained by all tools are 39% and 27% respectively indicating that in general their balance between TP an FP ratios is an important objective to improve by all tools in all vulnerability categories.

- **How is the best tool for analyzing the security of web applications with different levels of criticality?**

We have built three classifications of the performance of the tools attending to F-measure, $F_{0,5}$-score and Recall metrics. Tab. 5 shows the ranking of SAST tools by F-measure, Recall and $F_{0,5}$-score metrics.

Recall metric must be having into account for those web applications classified as business-critical, because it indicates the capacity of a SAST tool to discover the higher number of vulnerabilities. This metric is the most adequate metric for critical applications as it allows to find select tools with the best TP ratio. Fortify, Xanitizer and Klocwork are the best tools ranked for critical applications. As alternative to recall, it can be used $F_{1,5}$-score metric as it allows to reward the tools with better recall than precision metric.

**Table 5:** Ranking the tools by F-Measure, $F_{0,5}$-Score and Recall metrics

| Tool | F-measure | Tool | $F_{0.5}$ Score | Tool | Recall |
|---|---|---|---|---|---|
| Fortify | 0,561 | Findsecbugs | 0,610 | Fortify | 0,574 |
| Xanitizer | 0,535 | Xanitizer | 0,584 | Xanitizer | 0,470 |
| Findsecbugs | 0,521 | Fortify | 0,553 | Klocwork | 0,450 |
| Klocwork | 0,491 | Coverity | 0,539 | Checkmarx | 0,449 |
| Checkmarx | 0,490 | Klocwork | 0,520 | Findsecbugs | 0,420 |
| Coverity | 0,446 | Checkmarx | 0,518 | Coverity | 0,340 |
| Spotbugs | 0,081 | Spotbugs | 0,178 | Spotbugs | 0,043 |

However, we have selected F-measure metric (Tab. 5) for web applications considered heightened-critical. This scenario represents the development and assessment of not critical application or less important, which might have hard time to market constraints or be on a tight budget. F-measure represents the best effort and it means that the objective for less critical web applications is to discover a greater number of true positives having the smallest number of false positives. Fortify, Xanitizer and FindSecurityBugs are ranked as the best tools for heightened-critical applications.

For non-critical applications, this scenario represents the development and assessment of applications that do not have criticality concerns and are not very exposed to attacks. $F_{0,5}$-score metric favors precision and is adequate metric for non-critical applications where the time of development can be quicker as it allows to reward the tools with better precision. This makes tools with high precision to obtain much better results. Tab. 5 indicates that FindSecurityBugs, Xanitizer and Fortify have the best score.

Having into account the three classifications together, the best tool is Fortify because it appears in first position in two classifications and it appears in third position $F_{0,5}$-score classification. Xanitizer can be considered the second-best tool. It is classified in second position in the three classifications. FindSecurityBugs appears in first position in $F_{0,5}$-score classification and it appears in third position F-measure classification, it is the third best tool. Klocwork, Checkmarx and Coverity are in a second group of tools by this order of performance.

- **Is the OWASP Top Ten Benchmark adequate to comparing SAST tools?**

Evaluating commercial SAST tools for Web Applications is necessary. According to benchmark properties (Section 3), the cost of the benchmark implementation is justified with the results obtained with execution of the Web application SAST tools to be compared and ranked using a new repeatable methodology. The benchmark reports similar results when run more than once over the same tool and is easily portable to any operative system with 4 Gigabytes of RAM memory.

This benchmark represents the state of the art of security vulnerabilities for web applications based on the OWASP Top Ten project and is based on realistic code that includes different source entries and code complexity. In addition, it allows you to increase the number of categories and types of vulnerabilities to make them scalable. Finally, the benchmark is easy to implement even using default tool settings obtaining short run times with the SAST tools.

- **How are the results of this SAST tools comparative using OWASP Top Ten Benchmark compared with the results of comparatives using other benchmarks?**

The results and analysis of this work can be compared with other assessments included in related work. For example, one of them that it is based on OWAS Top Ten project. OWASP Benchmark project that has built its own benchmark for OWASP Top Ten vulnerability categories [OWASP Foundation (2020)]. It includes test cases written in Java for some types of security vulnerabilities included in OWASP Top Ten, but we think that it must be more representative including more types of vulnerabilities in each category. It uses the metric SCORE=(TP-FP) to classify the tools executed against the benchmark. It includes results for static and dynamic analysis tools. These results are showed for open source tools, for example, the TP results for FindSecurityBugs is 96,8% and the result in our comparative is 42% due to the inclusion of more type de vulnerabilities in our benchmark that FindSecurityBugs not detect well as types included in Sensitive data (0%), XSS (5%) and CSRF (14,2%). However, it can detect well all vulnerabilities included in OWASP Benchmark project.

For Spotbugs the results are similar, 5,12% in Benchmark project and 4,3% in our benchmark.

The names for commercial static tools are anonymous but their TP results are showed: SAST-01=32%, SAST-02=56.1%, SAST-03=46,3%, SAST-04=61,45, SAST-05=47,7% and SAST-06=85,02% and in our benchmark with commercial SAST tools that can be different goes from 34% to 57%. For commercial tools the results the TP results are similar except for SAST-06.

## 5 Conclusions

In this work an OWASP Top Ten vulnerabilities benchmarking approach is built for the assessment of the security performance of SAST tools including a complete of different vulnerability types of test cases in each vulnerability OWASP Top Ten category. The assessment uses a new and repeatable methodology for comparing and ranking the SAST tools.

The vulnerability detection percentages achieved by the analyzed tools indicate that the tools have a wide margin of improving. The tools obtain a recall value between 0,34 and 0,57 except for Spotbugs that obtain a worse result. The number of false positives, high in general must be reduced in a subsequent audit of the results. A comprehensive audit process of the vulnerability results accomplished by an experienced user or team, with security skills in the language used in the target code and specific security vulnerabilities for each language, is always necessary.

The TP and FP average ratios of all vulnerability categories obtained by all tools are 39% and 27% respectively, indicating that in general their balance between TP and FP ratios is an important objective to improve by all tools in all vulnerability categories. When any of the compared tools do not detect a real security vulnerability in a test case, they do not give false positive warnings in the corrected version of the test case.

The assessment obtains a strict ranking of seven SAST tools according to adequate and wide accepted metrics applied to the results of tools execution against benchmarking approach. In addition, it ranks the tools having into account three distinct metrics for different degrees of importance for web applications. Five leaders commercial SAST tools have been included in the assessment and ranked showing their results executed against the new benchmarking approach.

In general, the vulnerability detections in the categories of vulnerabilities related to disclosure of information in the code and broken authentication and sessions are a point of improving for all tools. The changing nature and evolution of web application technologies implies also changes in the vulnerability categories over time. It requires a future study to adapt the tools to discover the most frequent and important vulnerability categories periodically. Therefore, OWASP Top Ten must be modified frequently.

The analysis confirms that the results of the tools are different in terms of true and false positives detecting vulnerabilities. It suggests that using more than one SAST tool in combination can improving the TP and FP ratios.

It is also important to consider the possibility of using a SAST for binary code very useful for the cases that a company has not availability of web applications source code. Using of these types of tools will permit to analyze third party software. Also, is essential to develop new benchmarks for all categories of vulnerabilities and for mores languages to accomplish new comparisons that aid companies and developers choose the best SAST tool.

## 6. Future work

We are currently working on studying how SAST tools and other types of tools such as DAST, IAST, Real Analysis Self Protection (RASP) and HYBRID tools can be integrated and combined into the different SSDLC phases of web applications, according to their characteristics to complement between them and correlate their reports and get an optimized whole result. We have the main objective of designing a methodology for security analysis in the SSDLC of web applications as a function of the types of tools that might be available to accomplish a new web application secure development.

**Acknowledgement:** The authors extend their appreciation to the Software Engineering

and Security research group (SES) of Universidad Internacional de La Rioja.

**Funding Statement:** This work was supported in part by the Software Engineering and Security research group (SES) of Universidad Internacional de La Rioja.

**Conflicts of Interest:** The authors declare that they have no conflicts of interest to report regarding the present study.

## References

**Alavi, S.; Bessler, N.; Massoth, M**. (2018): A comparative evaluation of automated vulnerability scans versus manual penetration tests on false-negative errors. *Third International Conference on Cyber-Technologies and Cyber-Systems*. Iaria, Athens, Greece.

**Algaith, A.; Nunes, P.; Fonseca, J.; Gashi, I.; Viera, M.** (2018): Finding SQL injection and cross site scripting vulnerabilities with diverse static analysis tools. *14th European Dependable Computing Conference*, IEEE Computer Society, Iasi, Romania.

**Antunes, N.; Vieira, M.** (2010): Benchmarking vulnerability detection tools for web services. *Proceeding of the IEEE International Conference on Web Service*, Miami, Florida.

**Antunes, N.; Vieira, M.** (2015): Assessing and comparing vulnerability detection tools for web services: benchmarking approach and examples. *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 269-283.

**Antunes, N.; Vieira, M.** (2015b): On the metrics for benchmarking vulnerability detection tools. *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Rio de Janeiro, Brazil.

**Arouba, S. B.; Fernández-Villaverde, J.** (2015): A comparison of programming languages in macroeconomics. *Journal* of *Economic Dynamics* and *Control*, vol. 58, no. C, pp. 265-273.

**Barabanov, A.; Markov, A.; Tsirlov, V.** (2018): Statistics of software vulnerability detection in certification testing. *International Conference Information Technologies in Business and Industry*, IOP Publishing, Tomsk, Russia.

**Bermejo, J. R.** (2020): *OWASP Top Ten-benchmark.*

**Bermejo, J.; Abad, C.; Bermejo, J. R.; Sicilia, M. A.; Sicilia, J. A.** (2020): Systematic approach to malware analysis. *Applied Sciences*, vol. 10, no. 4, pp. 1360.

**Burato, E.; Ferrara, P.; Spoto, F.** (2017). Security analysis of the OWASP benchmark with Julia. *Proceedings of the 1st Italian Conference on Security*. DBLP Computer Science Bibliography, Venice, Italy.

**Cannings, R.; Dwivedi, H.; Lackey, Z.** (2008): *Hacking Exposed Web Applications: Web 2.0 Security Secrets and Solutions.* McGraw Hill, USA.

**Cifuentes, C.; Scholz, B.** (2008): Parfait-designing a scalable bug checker. *SAW '08: Proceedings of the Workshop on Static Analysis*. ACM, New York, USA.

**Deshlahre, R., Tiwari, N.** (2020): A review on benchmarking: comparing the static analysis tools (SAST) in web security. *Social Networking and Computational Intelligence. Lecture Notes in Networks and Systems*. Springer, Singapore.

**Díaz, G.; Bermejo, J. R.** (2013): Static analysis of source code security: assessment of tools against SAMATE tests. *Information and Software Technology*, vol. 55, no. 8, pp. 1462-1476.

**Duclervil, S. R.; Liou, J. C.** (2019): The study of the effectiveness of the secure software development life-cycle models in IT project management. *16th International Conference on Information Technology-New Generations. Advances in Intelligent Systems and Computing*. Springer, Las Vegas, USA.

**Felderer, M.; Büchler, M.; Johns, M.; Brucker, A. D.; Breu, R. et al.** (2016): Security testing: a survey. *Advances in Computers*. Elsevier, Cambridge, USA.

**Goseva-Popstojanova, K.; Perhinschi, A.** (2015): On the capability of static code analysis to detect security vulnerabilities. *Information and Software Technology*, vol. 68, no. 1, pp. 18-33.

**Guan, J.; Li, J.; Jiang, Z.** (2019): The design and implementation of a multidimensional and hierarchical web anomaly detection system. *Intelligent Automation and Soft Computing*, vol. 25, no. 1, pp. 131-141.

**Heckman, S.; Williams, L.** (2011): A systematic literature review of actionable alert identification techniques for automated static code analysis. *Information and Software Technology*, vol. 53, no. 4, pp. 363-387.

**Homaei, H.; Shahriari, H. R.** (2017): Seven years of software vulnerabilities: the ebb and flow. *IEEE Security & Privacy*, vol. 15, no. 1, pp. 58-65.

http://suif.stanford.edu/~livshits/securibench/.
https://github.com/jrbermh/OWASP-Top-Ten-Benchmark.
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-326.pdf.
https://www.owasp.org/index.php/Benchmark.
https://www.owasp.org/index.php/Top_10_2017-Top_10.

**Huth, M.; Nielsen, F.** (2019): Static analysis for proactive security. Computing and Software Science. *Lecture Notes in Computer Science. Springer*, Cham, Switzerland.

**Idrissi, S. E.; Berbiche, N.; Sbihi, M.** (2017): Performance evaluation of web application security scanners for prevention and protection against vulnerabilities. *International Journal of Applied Engineering Research*, vol. 12, no. 21, pp. 11068-11076.

**Krishnan, M.; Nadworny, N.; Bharill, N.** (2008): Static analysis tools for security checking in code at motorola. *ACM SIGAda Ada Letters*, vol. 28, no. 1, pp. 76-82.

**Livshits, B. V.; Lam, M. S.** (2005): Finding security vulnerabilities in java applications with static analysis. *Proceedings of the 14th Conference on USENIX Security Symposium*. USENIX Association, Berkeley, USA.

**Loch, F. D.; Johns, M.; Hecker, M.; Mohr, M.; Snelting, G.** (2020): Hybrid taint analysis for java EE. *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. ACM, New York, USA.

**Long, F.; Mohindra, D.; Seacord, R. C.; Sutherland, D. F.; Svoboda, D.** (2014): *Java™ Coding Guidelines: 75 Recommendations for Reliable and Secure Programs.* Pearson Education, Boston, USA.

**Martin, B.; Livshits, B.; Lam, M. S.** (2005): Finding application errors and security

flaws using PQL: a program query language. *20th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Application*s. ACM, San Diego, California, USA.

**Martin, R. A.; Barnum, S.** (2008): Creating the secure software testing target list. *Proceedings of the 4th Annual Workshop on Cyber Security and Information Intelligence Research: Developing Strategies to Meet the Cyber Security and Information Intelligence Challenges Ahead*. ACM, New York, USA.

**Moeller, J. P.** (2016): *Security for Web Developers: Using Javascript, HTML and CSS*. O'Reilly Media, Sebastopol, Russia.

**Monga, M.; Paleari, R.; Passerini, E.** (2009): A hybrid analysis framework for detecting web application vulnerabilities. *Proceedings of the 5th International Workshop on Software Engineering for Secure Systems*. IEEE Computer Society, Washington, USA.

**Mongiovi, M.; Giannone, G.; Fornaia, A.; Pappalardo, G.; Tramontana, E.** (2015): Combining static and dynamic data flow analysis: a hybrid approach for detecting data leaks in Java applications. *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, New York, USA.

**Nanz, S.; Furia, C. A.** (2015): A comparative study of programming languages in rosetta code. *Proceedings of the 37th International Conference on Software Engineering*, vol. 1, pp. 778-778.

**Nguyen, T. T.; Maleehuan, P.; Aoki, T; Tomita, T.; Yamada, I.** (2019): Reducing false positives of static analysis for sei cert c coding standard. *Proceedings of the Joint 7th International Workshop on Conducting Empirical Studies in Industry and 6th International Workshop on Software Engineering Research and Industrial Practice*. *IEEE Computer Society*, Montreal, Canada.

**NIST.** (2018): *NIST Special publication 500-326, SATE V report: ten years of static analysis tool expositions*.

**Nunes, P.; Medeiros, I.; Fonseca, J. C.; Neves, N.; Correia, M. et al.** (2018): Benchmarking static analysis tools for web security. *IEEE Transactions on Reliability*, vol. 67, no. 3, pp. 1159-1175.

**Nunes, P.; Medeiros, I.; Fonseca, J. C.; Neves, N.; Correia, M. et al.** (2019): An empirical study on combining diverse static analysis tools for web security vulnerabilities based on development scenarios. *Computing*, vol. 101, no. 2, pp. 161-185.

**OWASP Foundation.** (2017): *OWASP top ten 2017*.

**OWASP Foundation.** (2020): *OWASP benchmark project*.

**Pashchenko, I.; Dashevskyi, S.; Massacci, F.** (2017): Delta-bench: differential benchmark for static analysis security testing tools. *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE Computer Society, Toronto, Canada.

**Pistoia, M.; Tripp, O.; Lubensky, D.** (2018): Combining static code analysis and machine learning for automatic detection of security vulnerabilities in mobile apps. *Application Development and Design: Concepts, Methodologies, Tools, and Applications*. IGI Global, USA.

**Scambray, J.; Liu, V.; Sima, C.** (2010): *Hacking Exposed Web Applications 3*. McGraw-Hill, USA.

**Securebench Micro Project.** (2009): *Introduction to Stanford SecuriBench*.

**Sema, M.** (2012): *Hacking Web Apps Detecting and Preventing Web Application Security Problems*. Elsevier, Netherlands.

**Shrestha, J.** (2013): *Static Program Analysis (Ph. D. Thesis)*. Uppsala University, Sweden.

**Sipser, M.** (2005): *Introduction to the Theory of Computation*. Second Edition. Thomson Course Technology, Boston, USA**.**

**Sołtysik-Piorunkiewicz, A.; Krysiak, M.** (2020): The cyber threats analysis for web applications security in industry 4.0. Towards Industry 4.0-Current Challenges in Information Systems. *Studies in Computational Intelligence*. *Springer*, Cham, Switzerland.

**Stuttard, D.; Pinto, M.** (2008): *The web application hacker's handbook: finding and exploiting security flaws*. John Wiley & Sons, Indianapolis, USA.

**Vicente, J.; Bermejo, J.; Bermejo, J. R.; Sicilia, J. A.** (2019): The application of a new secure software development life cycle (S-SDLC) with agile methodologies. *Electronics*, vol. 8, no. 11, pp. 1218.

**Ware, M.; Fox, C.** (2008): Securing java code: heuristics and an evaluation of static analysis tools. *Proceedings of the workshop on Static analysis*. ACM, New York, USA.

**Yang, J.; Tan, L.; Peyton, J.; Duer, K. A.** (2019): Towards better utilizing static application security testing. *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*. IEEE Computer Society, Montreal, Canada.