Computer Modeling in Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2021.010700

ARTICLE

Hybrid Security Assessment Methodology for Web Applications

Roddy A. Correa[1], **Juan Ramón Bermejo Higuera**[2], **Javier Bermejo Higuera**[2], **Juan Antonio Sicilia Montalvo**[2], **Manuel Sánchez Rubio**[2] **and Á. Alberto Magreñán**[3,*]

[1]Universidad Técnica Particular de Loja, San Cayetano Alto, Loja, Ecuador

[2]Escuela Superior de Ingeniería y Tecnología, Universidad Internacional de La Rioja, Logroño, 26006, Spain

[3]Universidad de la Rioja, Logroño, 26006, Spain

[*]Corresponding Author: Á. Alberto Magreñán. Email: angel-alberto.magrenan@unirioja.es

Received: 21 March 2020 Accepted: 23 June 2020

ABSTRACT

This study presents a methodology to evaluate and prevent security vulnerabilities issues for web applications. The analysis process is based on the use of techniques and tools that allow to perform security assessments of white box and black box, to carry out the security validation of a web application in an agile and precise way. The objective of the methodology is to take advantage of the synergies of semi-automatic static and dynamic security analysis tools and manual checks. Each one of the phases contemplated in the methodology is supported by security analysis tools of different degrees of coverage, so that the results generated in one phase are used as feed for the following phases in order to get an optimized global security analysis result. The methodology can be used as part of other more general methodologies that do not cover how to use static and dynamic analysis tools in the implementation and testing phases of a Secure Software Development Life Cycle (SSDLC). A practical application of the methodology to analyze the security of a real web application demonstrates its effectiveness by obtaining a better optimized vulnerability detection result against the true and false positive metrics. Dynamic analysis with manual checking is used to audit the results, 24.6 per cent of security vulnerabilities reported by the static analysis has been checked and it allows to study which vulnerabilities can be directly exploited externally. This phase is very important because it permits that each reported vulnerability can be checked by a dynamic second tool to confirm whether a vulnerability is true or false positive and it allows to study which vulnerabilities can be directly exploited externally. Dynamic analysis finds six (6) additional critical vulnerabilities. Access control analysis finds other five (5) important vulnerabilities such as Insufficient Protected Passwords or Weak Password Policy and Excessive Authentication Attacks, two vulnerabilities that permit brute force attacks.

KEYWORDS

Web applications; security vulnerability; weakness; security analysis; white box; black box; interactive application security testing; static application security testing; dynamic application security testing

1 Introduction

Software security is a very important feature to be considered in the software development processes resulting from the evolution of technology, information and services [1].

Frequently, the lack of security is conceived by the lack of security measures taken by software engineers [2], it means that much of the software that goes into production is vulnerable to one or more types of attacks. Against this, organizations such as OWASP [3] and SANS [4] are constantly working to identify, analyze and determine ways to mitigate software vulnerabilities, contributing to the methods, procedures, classifications and different ways of the community to solve problems due to software vulnerabilities. Identifying vulnerabilities and ensuring security functionality through security testing is a widely applied measure to evaluate and improve software security [5].

Software security in companies or organizations that develop information systems has to be constituted as an intangible asset and an important feature for which development companies are recognized, offering customers a set of standards and good practices to apply in their projects in order to guarantee and provide the necessary reliability on the data and sensitive information they handle. To establish a management system for information security, consider the three pillars of McGraw [6]: Risk management, knowledge management, security and good security practices in any implementation of a Lifecycle secure software development (SSDLC) [7]. The implementation of the three pillars will imply a return on the initial investment made in the acquisition of security assets.

The popularity of web applications, as a solution to interconnect users with data and services [8] has made them, by their exposure, the main objective of cyber attackers to identify and take advantage of weak points to violate their privacy [9,10]. Therefore, it has become an obligation to evaluate the security of the software that unfolds in each of the phases of the software life cycle (SDLC).

There are several methods and ways of evaluating software security, among which the use of Applications Security Testing (AST) stands out, they are one of the resources most widely used by developers to guarantee the security of applications [11]. The AST tools divide their vulnerability detection methods into static vulnerability detection through tools of the SAST type (Static Application Security Testing) and dynamic vulnerability detection through tools of the DAST type (Dynamic Application Security Testing) [12]. SAST is most commonly used for vulnerability detection through source code review [13], without the need to run the software [14]; while DAST performs a verification on the attack surface of the software at runtime, where all possible data entries are verified [15]. There are many benefits to this type of analysis, however, the tools create many false positives [12], so a strategy is required to make the best of each type of analysis.

The software evaluation process represents an important human and technical effort, due to the degree of depth involved in carrying out each type of security analysis [13,16,17]. Generally, white box and black box security tests require the support of semi-automatic tools combined with audits of the results that have to be performed by humans, to rule out the false positives generated by the tools, which is one of the greatest weaknesses of security analysis tools and to detect additional vulnerabilities. Studying how to combine diverse types of security analysis using different tools is important because it allows you to automate much of the security analysis work so that a great benefit will be obtained in terms of the total time spent on a security analysis of a web application.

Software security vulnerabilities are one of the most critical problems and one of the greatest concerns related to computer security. The method analyzes its strengths and weaknesses and takes advantage of the synergies that both types of analysis have to optimize the result of the security evaluation of a web application.

For the development of the methodology, we investigated the characteristics of static and dynamic security analysis, the tools available to carry them out and their use in isolation. The objective is to get an appropriate knowledge that can be used to combine them in several phases, feed the results between them and correlate the partial results obtained in each phase to maximize the overall result of the security evaluation of a web application.

To evaluate the methodology and its applicability to any web application, we used the CMS Made Simple case study [18]. This web application was first launched in 2004 and focuses on creating and managing content for static web pages. This CMS is maintained by CMS Made Simple Foundation, it is an open source project, written in PHP, which stands out for having won some awards like two second places and a first place as the "best general open source content management system" by Pact Publishing [19] and in 2017 it won the annual CMS Critic award for Best Open Source Content Management [20].

Finally, the results classified by the metrics included in the methodology will be evaluated and published, in such a way that the experimental security analysis presents a first scope of the effectiveness of the methodology.

This paper makes the followings contributions:

1. It presents a new, light, repeatable and flexible methodology that allows evaluating the security of any type of web application, being independent of the type of development and implementation technology.
2. It uses two different security analysis perspectives that allow having an internal software evaluation surface using SAST tools and an external one with DAST tools. In addition, manual verification criteria are defined, which allow identifying and minimizing the false positives produced by the tools, and identifying false negatives that, due to the scope of the applications, often cannot be detected in a timely manner.
3. Demonstrates that the proposed methodology is simple to apply and effective in evaluating the security of web applications. Furthermore, its ease of understanding allows the methodological process to be used in any traditional or agile development context or practices of integration and continuous delivery.
4. It shows how the methodology can be used by others more general methodologies that do not explain how to combine static and dynamic security analysis tools in web applications.
5. It can be integrated into both traditional and agile SDLC.

The rest of the paper is organized as follows. Section 2 is a background of security testing technologies, Section 3 describes the new testing methodology approach, Section 4 is a practical application of the methodology, and Section 5 reviews the related work about web applications security testing tools, conclusions and future work.

2 Background

2.1 Web Applications

Web applications are software products that are accessed from a web browser and operate under a client-server architecture that communicates through the HTTP communication protocol (see Fig. 1).

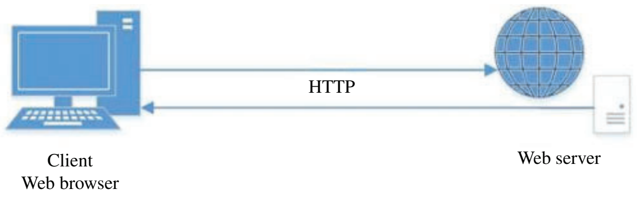


Figure 1: Basic architecture of a web application

According to Ginige et al. [21] in their article entitled "Web Engineering: An Introduction," web applications have grown rapidly in scope and use, so they have positively affected the lives of human beings, such as industries, businesses and others, since they have to be migrated to the Web and the Internet facilitating processes and accessibility capabilities to their services through the development and use of web applications.

Although web applications have facilitated many tasks, their exposure and access to all types of end users has created new threats [22], which can be classified as design, implementation and operational vulnerabilities [16]. Other authors provide specific vulnerabilities and attacks taxonomies for specific environments [23]. In addition, there are those who claim that most web applications have security vulnerabilities [24,25] and are prone to mainly SQL injection attacks and cross site scripts attacks. The possibility of avoiding such attacks should be considered and it is necessary to use techniques and tools that allow the detection of software vulnerabilities [26], as well as guaranteeing the use of good practices such as the inclusion of SSDLC processes that allow covering and protecting software in each of the phases of the life cycle.

Web applications are subject to continuous changes in usage trends, implementation technologies and new security vulnerabilities, as shown by the results of the 2017 survey conducted by Stack Overflow [27], where it is appreciated that 72.6 percent of a total of 36,000 identify themselves as web developers and title their learning trends with contributions that allow them to improve their skills within the focus of web application developers (see Fig. 2).

2.2 Security Analysis Types and Tools for Vulnerability Detection

Given the growing trend in the use of new technologies for the development of web applications and the need to mitigate security weaknesses, research papers such as [24,26,28], mention that one must act "before;" that is, before suffering an attack due to security vulnerabilities, so it is essential to perform all security tests and validations before going into production. For this, several types of analysis must be considered [29], supported by semi-automatic tools that facilitate the analysis of the entire web application.

There is not a perfect security analysis technique, normally, it suffers from generating alarms for security vulnerabilities that do not really exist (false positives) and from not detecting some security vulnerabilities (false negatives) [30,31]. Due to the inaccuracy of detection techniques, we say that the tools are "semi-automatic" since they will require a manual audit of the results to confirm the veracity of the detected vulnerabilities. Therefore, there is a need to identify the scope of the analysis, the technology and the impact of the requirement that is being developed, in order to correctly select the tools that support the security analysis based on its ability to detect security vulnerabilities.

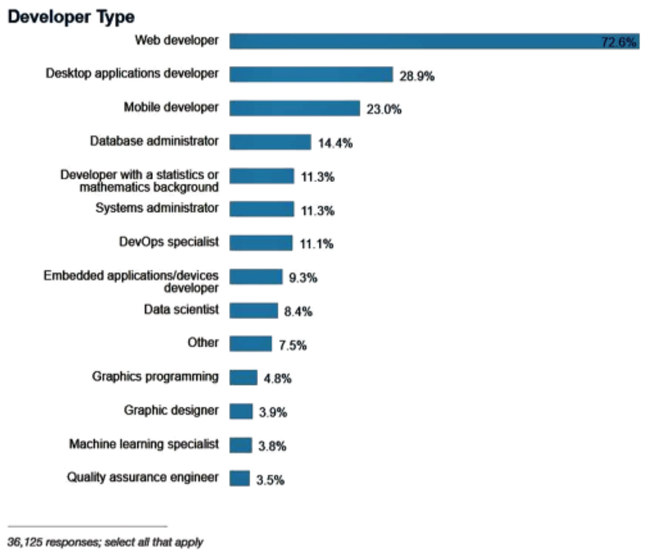


Figure 2: Interest trend of developers [27]

2.2.1 *Static Analysis and SAST Tools*

In the context of SSDLC, Static Application Security Testing (SAST) is a white box type vulnerability analysis technique, which bases its process on evaluating the source code to detect possible security vulnerabilities.

Its main advantage is the ability to find errors and coding problems in web applications that produce security vulnerabilities without having to run the application [32]; given its importance and capacity, it is considered as the main security technique in the Secure Software Development Life Cycle of McGraw [19].

Manual code security analysis is arduous and complex if the number of lines of code derived from the process of software development is high, which would mean a longer evaluation time [33]. Therefore, this type of analysis should be based on the use of semi-automatic tools that help to carry out a thorough security analysis of the source code in a more reduced period of time. The workflow of the static analysis involves four steps, it starts with the establishment of objectives to execute the tools against the source code and vulnerabilities found are audited and corrected. Note that Steps 2 (run tools to find vulnerabilities), 3 (review code to confirm the vulnerabilities found by the tool) and 4 (make fixes of confirmed vulnerabilities), go into an iterative process as appropriate to identify and correct all security vulnerabilities identified in the analysis (see Fig. 3).

SAST (Static Application Security Testing) tools build a source code model to perform a static process to evaluate and identify security vulnerabilities in early stages of development, in order to generate products software with fewer weaknesses from the beginning of code implementation [34–37].

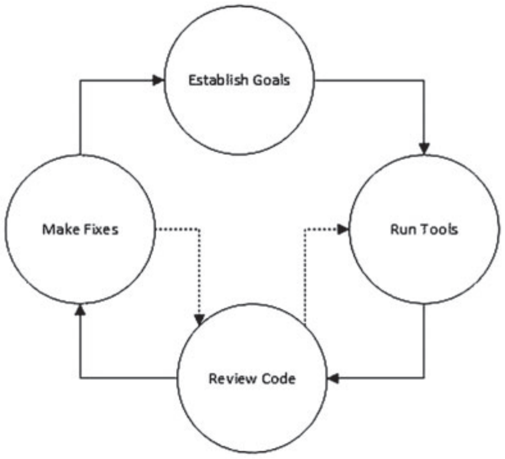


Figure 3: Static analysis process [33]

The method described in [16] highlights, as the main advantage, the code review and evaluation process carried out by these tools, to identify security vulnerabilities in the source code that can lead to attacks in production environments. On the other hand, he also highlights that currently SAST analysis algorithms suffer from false positives and negatives, so their use requires a subsequent audit. However, despite its lack of precision and its almost mandatory need for an audit of results, SAST continues to maintain its popularity in the identification of security vulnerabilities [38].

SAST tools still have a wide margin for improvement, both their internal design and the set of vulnerabilities they can detect determine their degree of effectiveness. Each of the tools have a different degree of coverage and, in general terms, the commercial tools prevail in the market and have a higher degree of effectiveness than free tools, as shown by the studies [34,39,40] and the classification of Gartner in Fig. 4, where tools such as HP Fortify SCA [36], Veracode SAST [37], Checkmarx [35] or IBM Security App Scan [41] dominate the market based on their results.

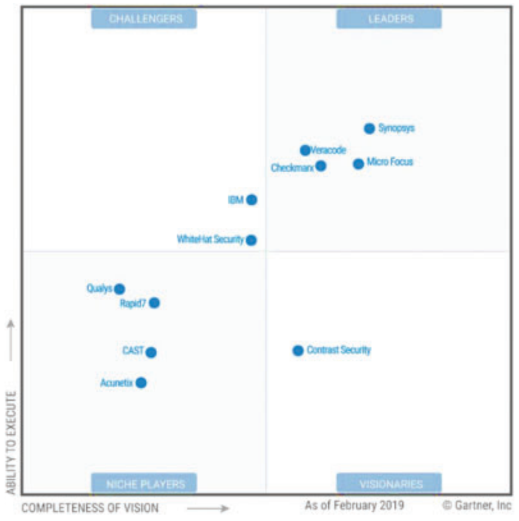


Figure 4: SAST Gartner magic quadrant [42]

2.2.2 Dynamic Analysis

Dynamic analysis is a security vulnerability detection technique that includes black box and white box testing. This type of analysis makes a check of the running software [43], so that it can detect abnormal behaviors due to security vulnerabilities and provide useful information to the programmers about their software applications. This type of analysis allows us to identify possible forms of security attacks, configuring multiple attack vectors that can also include tests of possible configurations in testing and pre-production environments, in order to identify and mitigate possible ways to exploit software security vulnerabilities.

A dynamic black box analysis makes an evaluation directly with the Frontend of the application using fuzzing test in the entry inputs of an application, in order to determine and identify vulnerabilities and weaknesses in the architecture of web applications [15]. A dynamic white box analysis implements the executable or source code by rewriting it before compiling it. A dynamic white box tool used acts as an agent to obtain precise information about assignment of values to the input variables, logic flow, data flow and configuration to decide if a payload could be an attack attempt or not [44].

There are basically two categories of dynamic analysis tools:

- Dynamic Application Security Testing (DAST).
- Interactive Application Security Testing (IAST) and real-time protection (RASP), are white box tools and can complement the work done by the black box type identifying security vulnerabilities.

DAST (Dynamic Application Security Testing) tools try to discover the components, forms of a web application in a first phase. Once the web site is discovered, the tool sends many malicious requests to the input fields of the application forms and analyze the responses to see if there are security vulnerabilities [45], In addition, they can perform fuzzing on a specific entry field. Finally, the vulnerability report is obtained.

The advantages of DAST are:

- It can detect vulnerabilities in the final version of the software product.
- It simulates the behavior of a malicious user performing different types of attacks.
- It is independent of a programming language.

The disadvantages of DAST are:

- It is based on trial and error techniques, so they can not cover all attack surface.
- It cannot detect problems related to software logic.
- It is complex to cover all possible attack vectors.

DAST occupy a prominent place among the activities to be carried out within the life cycle of secure software development in the penetration testing phase, see Fig. 5 [19].

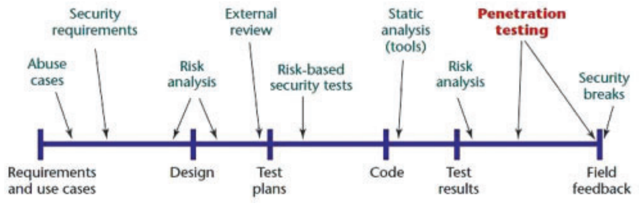


Figure 5: Comparative of security tools types [19]

IAST (Interactive Application Security Testing), RASP (Runtime Application Self-Protection). IAST-RASP are part of a second generation of security analysis tools. In their process of evolution some research papers [46,47] compared some IAST tools to the first-generation of SAST, DAST tools with the objective of improving the percentages of false positives and false negatives that they present.

IAST are dynamic white box analysis tools, unlike SAST, which are static tools. The advantage of IAST, as explained by Rohr [48], is that type of tools analyzes the code at run time in such a way that they can identify and understand its context. On the other hand, a RASP tool [49] is an agent that works by identifying and blocking security vulnerabilities at runtime. Although IAST and RASP are similar in the operation, it is important to specify the difference for the execution environment in which they are used, IAST tools are used in test environment, whereas a RASP tool operates in an environment of production as a software type firewall.

According to the studies of Sureda et al. [50] and Contrast Security [51] IAST and RASP tools are achieving very good results. Contrast study compares IAST against SAST and DAST tools. IAST achieves better results, generating 0 percent of false positives detected (see Fig. 6). More comparisons studies are required for analyzing the effectiveness of these tools.

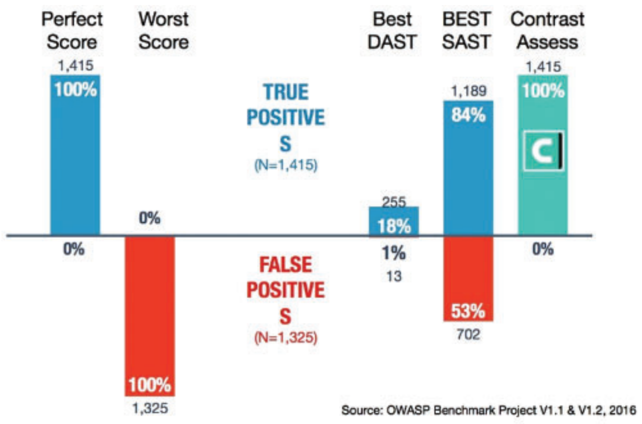


Figure 6: SSDLC security techniques [19]

2.2.3 *Hybrid Analysis*

Hybrid analysis is a type of security analysis that emerges from the combination of static, black and white box dynamic analysis, so we seek to take advantage of the synergies that may exist between the types of tools and extend the attack surface of the application to obtain a global increase in the effectiveness of the security analysis of a web application. An example of this is the combination of tests with SAST and DAST tools, which integrate their capabilities to achieve greater accuracy in the detection of security vulnerabilities and false positives. At present, it is possible to find some combinations of tools that perform static, dynamic or interactive analysis, such as the integration of DAST-SAST [52,53], SAST-RAST [54], DAST-RAST or SAST-DAST-RAST [54].

2.3 Security Analysis Methodologies

Currently, there are security methodologies that already establish ways to measure the security of web applications based on a series of phases and security analysis steps that allow a complete perspective of security in web applications.

Shakeel [55], in his contribution on methodologies and success standards for security, highlights the following methodologies:

- Penetration Testing Execution Standard (PTES)
- OWASP Testing Guide
- NIST 800-115 2008
- Open Source Testing Methodology Manual (OSSTMM)

2.3.1 Penetration Testing Execution Standard (PTES)

PTES is a standard designed to offer security services based on its penetration testing analysis [56]. It establishes the creation of a frame of reference with an integral approach in software intrusion testing as an objective [57]. The reference framework bases its process on seven phases:

1. Interactions prior to commitment
2. Information gathering
3. Threat modeling
4. Vulnerability analysis
5. Exploitation
6. Post-exploitation
7. Reports

Its advantage, according to Shanley et al. [58], their article of penetration testing methodologies, lies in the exclusivity of the PTES not to reinvent processes, but to incorporate good own and external practices, such as security assessments proposed by OWASP.

2.3.2 OWASP Testing Guide

The OWASP Test Guide is a referential framework that includes a set of control points, based on the collection of good security practices used in web applications [59]. Its objective [60,61], is to help people to understand what, why, when, where and how to evaluate web applications, for which, the guide proposes an organized tour and systematizes all variants of attack vectors to the web applications. The methodology includes two parts: The first part consists of a frame of reference with five phases composed of techniques and tasks to be used in each of them, these five phases are:

1. Before starting the development
2. During the design and definition
3. During the development
4. During the implementation
5. Maintenance and operations

The second part contains a set of techniques that serve to identify the different security problems that a web application may suffer. The techniques specified in version four of the guide includes:

1. Information collection (OTG-INFO)
2. Configuration and deployment management tests (OTG-CONFIG)
3. Identity management tests (OTG-IDENT)
4. Authentication tests (OTG-AUTHN)
5. Authorization tests (OTG-AUTHZ)
6. Session management tests (OTG-SESS)
7. Input validation tests (OTG-INPVAL)
8. Error handling tests (OTG-ERR)
9. Weak cryptography tests (OTG-CRYPST)
10. Business logic tests (OTG-BUSLOGIC)
11. Customer side tests (OTG-CLIENT)

This methodology establishes the parts of a web application that must be tested successively using different types of tools such as SAST, DAST, etc. However, it does not delve into how to use the tools together and take advantage of the synergies that may exist between them.

2.3.3 NIST 800-115 2008

This methodology is a standard developed by the National Institute of Standards and Technology of the United States [62]. For Souppaya et al. [63], this methodology provides guidelines on the planning and execution of security tests, information evaluation, analysis of results and mitigation strategies. This methodology proposes three phases of execution:

1. Planning
2. Execution
3. Post-Execution

2.3.4 Open Source Testing Methodology Manual (OSSTMM)

OSSTMM is a general security testing methodology mainly focused on audit areas [64] and created by the Institute of Security of Open Methodologies (ISECOM). Authors such as Shanley et al. [58], Alvarado et al. [64], agree that OSSTMM is one of the most complete professional standards and it is used when assessing the security of Internet Systems. Its evaluation process integrates software security and physical security of the organization, for which its execution is based on the analysis of the following aspects:

1. Security of the information
2. Process safety
3. Security of Internet technologies
4. Security of communications
5. Wireless security
6. Physical security

Each of the mentioned aspects contains a list of controls and techniques to meet the needs of the organization. It covers all security areas including web applications.

3 Methodological Design

The main objective of the proposed methodology is to carry out the security analysis activities of the SSDLC through a procedure that attempts to automate the method using semi-automatic tools identified in Section 2.3 of both types: Static analysis and dynamic analysis. This methodology may be part of other more general methodologies that contemplate the use of static and dynamic analysis tools, such as, the OWASP Testing guide v4 methodology.

The combination of this type of tools SAST and DAST, was born with the intention of taking advantage of the characteristics of each one to achieve a greater degree of coverage with respect to the security analysis. Studies conducted by Ball [43] and Intel Software [65], have shown that the use of these approaches is complementary, since no type of analysis is able to fully evaluate the security of a software product on its own.

The proposed approach contemplates a three-phase process at the macro level of definition, execution and results, where the core of the methodology is in the execution phase, which, in itself, its operation is fed by the results of the static analysis and dynamic. They are verified and validated through a correlation of results that allows to manually establish scenarios that, from an attacker's perspective, allow us to discard false positives and identify possible false negatives. Finally, we emphasize on evaluating in a particular way the access control of the applications to increase the degree of vulnerability coverage that allows obtaining the greatest possible effectiveness during the analysis. The proposed methodology contemplates three phases shown in Fig. 7. Each of the phases includes a sub-process based on evaluation activities, so that, in general terms, the methodology is structured as follows:

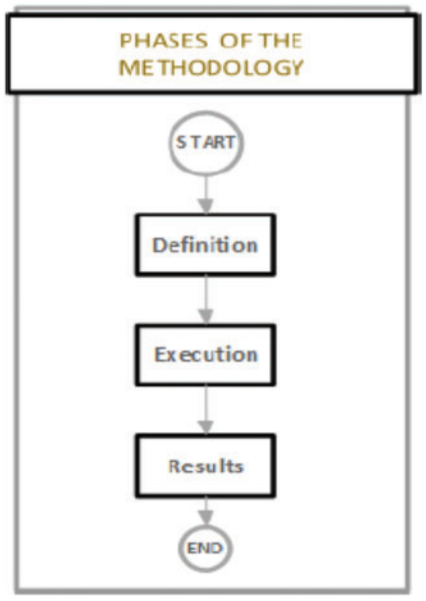


Figure 7: Phases of the methodology

Phase 1: Definition

1. Evaluation selection objective
2. Identification of evaluation objective characteristics

3. Tools selection
4. Metrics selection

Phase 2: Execution

1. Semi-automatic static analysis
2. Checking static analysis results with dynamic analysis
3. Semi-automatic dynamic analysis
4. Access control analysis

Phase 3: Classification of results

1. Analysis of results using selected metrics

The phases are detailed in the following sections.

3.1 Phase 1: Definition

This phase is the beginning of the security assessment, in which we define all the basic aspects of the evaluation process. The start is important because of the definition of the aspects exposed in the process (see Fig. 8) of this Phase, the effectiveness of the security analysis is defined and guaranteed.

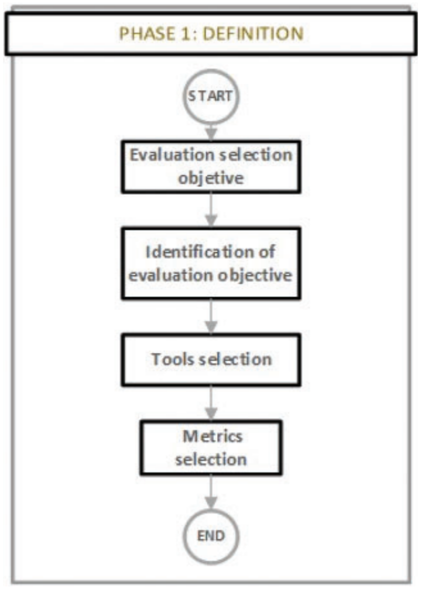

PHASE 1: DEFINITION
START
Evaluation selection objetive
Identification of evaluation objective
Tools selection
Metrics selection
END

Figure 8: Definition phase

The process involved in this phase implies:

1.1 The establishment of evaluation objectives: It defines the web application to be evaluated, the objectives, the scope of the analysis, the evaluation criteria and the vulnerabilities that will be covered by the analysis. This step is important, as the tool selection process is highly dependent on this.

1.2 Identification of characteristics of the objective: All the technical details of the web application (programming language, execution environment and implementation) are identified and the context and operation of the web application are validated.
1.3 Selection of analysis tools: Based on the previous point 1.2, the SAST and DAST tools to be used are identified, and it is verified that the selected tools are compatible with the evaluation environment.
1.4 Metrics definition: The metrics that will be used to classify the results are established. For the proposed methodology the metrics will be:
    - TP: True positive: A detection by any type of vulnerability analysis that is a real security vulnerability.
    - FP: False positive: A detection by any type of vulnerability analysis that is not a real security vulnerability.
    - TP percentage of total vulnerabilities.
    - FP Percentage of total vulnerabilities.

Once the definition phase is complete, the evaluation team has the technological and functional context of the web application, the tools and the metrics that will be necessary to continue with the execution phase of the security evaluation.

3.2 Phase 2: Execution

The execution phase is where the security assessment process of the web applications is accomplished, it follows the process shown in Fig. 9.

The security analysis execution procedure uses the information obtained in the definition phase to execute the corresponding analyzes, for which the following process is followed.

2.1 Semi-automatic static analysis (white-box): In this step, the security analysis of the web application to be evaluated is executed, for which it is necessary to execute the static analysis through the SAST tool selected in Step 1.3. Subsequently, the security auditor must manually audit and classify the results that the tools generate and classify them as true positives and false positives.
2.2 Checking results with dynamic and analysis: To carry out this activity, the results classified in Step 2.1 are required as input. Thus, the dynamic analysis will be carried out through the DAST tool selected in Step 1.3, with manual verification. This will allow having the first classification of security analysis results.
2.3 Semi-automatic dynamic analysis (black-box): This step is independent of Steps 2.1 and 2.2, and can be executed in parallel with the previous steps. This step will serve to identify the vulnerabilities that the static analysis does not cover. From this step, new results will be obtained that will serve to extend the degree of coverage of the analyzes carried out.
2.4 Dynamic access control analysis: This security analysis will be used to carry out security checks regarding the authentication and authorization processes of the web application.
2.5 Results: This step is fed by Steps 2.1, 2.2, 2.3, and 2.4. In this step, the degree of coverage of the security analysis executed will be analyzed, correlated and identified based on the vulnerabilities and metrics identified. It is important to mention that this step is key, since it allows to locate vulnerabilities and that the auditor, or the development team

can later improve the quality [66] and security of the software through the results of the analysis.

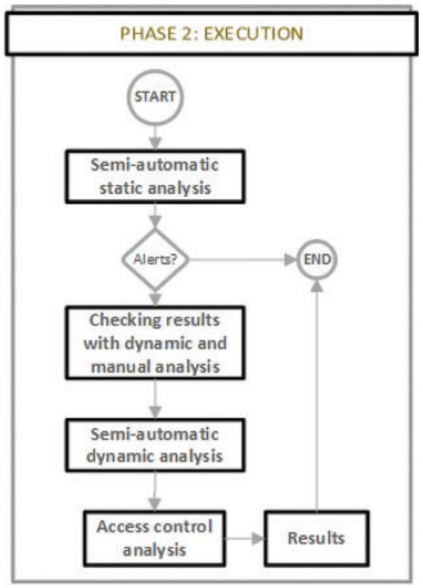

PHASE 2: EXECUTION
START
Semi-automatic static analysis
Alerts?
END
Checking results with dynamic and manual analysis
Semi-automatic dynamic analysis
Access control analysis
Results

Figure 9: Execution phase

In the security assessment process, the evaluator may face some doubts due to the scope of each one of the types of analysis executed, that is why some considerations have been established to be considered during this phase:

- The established process can be iterated as many times as the evaluator believes necessary to ensure the security of the web application.
- In Step 1.3 of tool selection, the evaluator can select the evaluation tool that best suits the context of functional and technological execution of the web application, and can also choose how to use the tool based on the best performance of the tool. The identification of the TP and FP metrics. See the related job section for tool comparisons. See related work section for tool comparisons.
- The execution process is flexible; that is, it allows us to use or integrate another or different types of analysis to the satisfaction of the evaluator.
- When selecting tools, one should investigate the degree of coverage of the vulnerabilities for which they are designed and the differences between the types of analysis and the tools to cover the greatest possible attack surface.
- A vulnerability database checking process such as NVD (National Vulnerability Database) must be performed to discover CVEs (Common Vulnerabilities Exposures) in relation to third-party software. In addition, a static code security analysis process must be performed to determine its degree of exposure.
- The results phase must contemplate minimally the detailed structure in its corresponding phase.

3.3 Phase 3: Classification of Results

In this final phase, we apply the selected metrics to the results of execution phase to classify the security vulnerabilities detected. Complementary statistical graphs and detailed information of the identified vulnerabilities will be shown to clarify the results of the security analysis based on this methodology. In execution phase, the information is organized according to the following data:

- Static analysis of source code (white-box)
  - —Audit Results
  - —Results classification based on defined metrics
- Confirm of the static analysis results with dynamic analysis
  - —Confirmed security vulnerabilities
  - —Not confirmed security vulnerabilities
  - —Results classification based on defined metrics
- Dynamic analysis (black box) and Dynamic access control analysis
  - —Audit Results
  - —Results classification based on defined metrics

Results of security vulnerabilities classified by metrics

4 Practical Application of the Methodology

This section summarizes the methodology application process developed in order to measure the degree of accuracy in the identification of security vulnerabilities. The application process is detailed in the following sections according to each of the phases proposed in the methodology.

4.1 Phase 1: Identification

In this phase, according to the proposed methodology, we define all the basic aspects of the evaluation process:

- Scope of evaluation: Evaluate security vulnerabilities resulting from tools that are classified as high/critical.
- Evaluation objective: CMS Made Simple v2.2.1
- Characteristics of the application Type: CMS or Website content manager.
- Programming language: PHP
- Application Server: Apache
- Database Management Server: MySQL
- Analysis tools selected SAST: SCA Fortify. DAST: OWASP ZAP
- The metrics to use in this methodology are:
  - —Vulnerability enumeration and classification Common Weakness Enumeration: CWE.
  - —TP: True positive: A detection by any type of vulnerability analysis that is a real security vulnerability.
  - —FP: False positive: A detection by any type of vulnerability analysis that is not a real security vulnerability.
  - —TP percentage.
  - —Percentage.

In this phase, for the experimental process of applying the methodology, a test environment was prepared using the service manager for XAMPP that contains a package of services and applications that is used to run PHP web applications, then, the selected application was installed in a local environment as part of objective evaluation.

Furthermore, for this process, it is important to explain that the tools were selected based on the information mapped for the case study, considering the scope of the study and the technological context of the web application to be evaluated. It is important to emphasize that the methodology is capable of adapting to any technological context, therefore, based on the evaluation objective identified in Phase 1, the tools that best suit the auditor's needs can be selected.

4.2 *Phase 2: Execution*

4.2.1 Semi-Automatic Static Analysis

Initially, through the SAST tool, the evaluation of the source code of the evaluation objective was configured. The analysis in this step generated the results shown in Fig. 10.

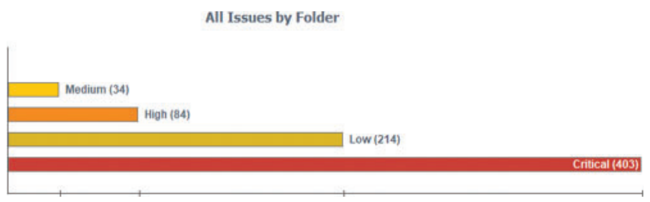


Figure 10: Summary of SAST analysis results

In summary, the results included 403 vulnerability issues of critical type, 84 of high type, 34 of medium type and 214 of low type were obtained, of which, according to the scope of the study, critical vulnerability issues were evaluated, the same as detailed in Tab. 1.

After obtaining the results, the audit was carried out for each of the vulnerabilities, obtaining the first classification of security vulnerabilities. Fig. 11 shows an example using the Fortify SCA interface to audit a security vulnerability (XSS) by analyzing graphically the code from the input source to the sink instruction whose execution would involve a real attack. In line number 44, the input parameter "user" is validated before it is sent to the browser. This is a false positive.

Fig. 12 shows a sample of auditory of a true positive security vulnerability (XSS) reported by SAST tool. In this case, the input parameter "group" is sent with POST method and it is assigned to variable "group" in line 95 without previous validation.

4.2.2 Checking Results with Dynamic and Manual Analysis

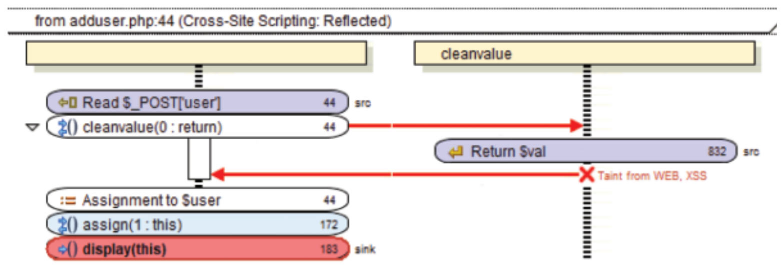
In this step, each of the identified and classified vulnerabilities of the results generated by the SAST tool was checked in two steps:

1. For each security vulnerability identify audited by SAST is performed a dynamic analysis penetration testing technique using the selected DAST tool.

2. Once the active scan has finished a manual verification of each vulnerability is performed to classify it as a true positive or a false positive.

Table 1: SAST critical results by vulnerabilities categories

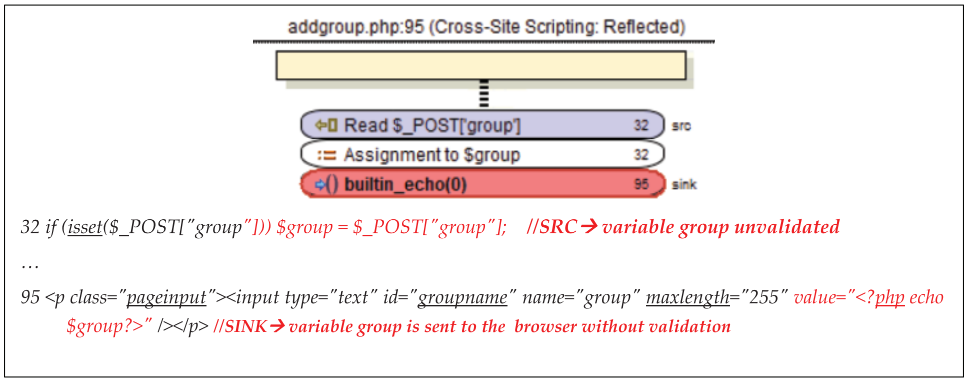
CWE	Detections
CWE-79 XSS persistent	44
CWE-79 XSS reflected	244
CWE-94 dangerous file inclusion	20
CWE-676 dangerous function	2
CWE-95 code injection	5
CWE-91 JSON injection	2
CWE-321 hardcoded encryption key	1
CWE-601 open redirect	16
CWE-259 hardcoded password	2
CWE-22 path manipulation	48
CWE-359 privacy violation	17
CWE-89 SQL injection	1
CWE-497 system information leak	1
Total	**403**



```
44 $user = isset($_POST["user"]) ? cleanValue($_POST["user"]) : '';
...
172 $smarty->assign('user', $user);
...
183 $smarty->display('adduser.tpl');
```

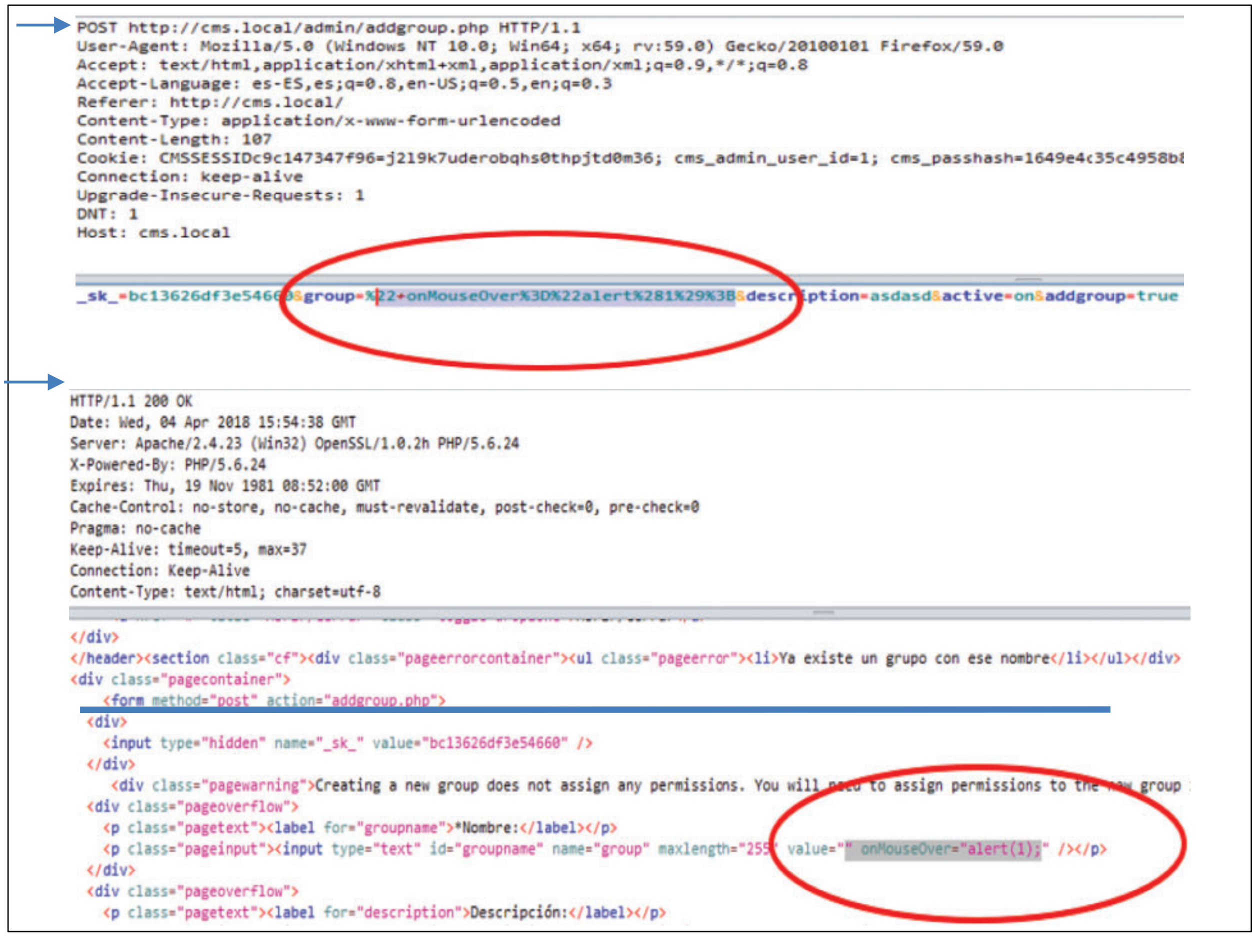
Figure 11: Vulnerability audit (false positive) with fortify SCA

According to the aspects mentioned in relation to the execution phase, there were incidents of security vulnerabilities identified by SAST that did not enter within the range of action or attack surface of DAST due to the different depth and characteristics of each type of analysis, so, in those cases, the result of SAST was considered. The security vulnerability audited in Fig. 12 after static analysis is also verified with the DAST tool (Fig. 13) assigning the malicious javascript payload "*on MouseOver = "alert(1);*" to the group parameter. The POST request is submitted, and the response shows that the javascript payload is executed.


addgroup.php:95 (Cross-Site Scripting: Reflected)
Read $_POST['group'] 32 src
Assignment to $group 32
builtin_echo(0) 95 sink

```
32 if (isset($_POST["group"])) $group = $_POST["group"];   //SRC→ variable group unvalidated
…
95 <p class="pageinput"><input type="text" id="groupname" name="group" maxlength="255" value="<?php echo
      $group?>" /></p> //SINK→ variable group is sent to the  browser without validation
```

Figure 12: Vulnerability audit (true positive) with SAST tool and manual checking



```
POST http://cms.local/admin/addgroup.php HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:59.0) Gecko/20100101 Firefox/59.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Referer: http://cms.local/
Content-Type: application/x-www-form-urlencoded
Content-Length: 107
Cookie: CMSSESSIDc9c147347f96=j219k7uderobqhs0thpjtd0m36; cms_admin_user_id=1; cms_passhash=1649e4c35c4958b8
Connection: keep-alive
Upgrade-Insecure-Requests: 1
DNT: 1
Host: cms.local

_sk_=bc13626df3e54660&group=%22+onMouseOver%3D%22alert%281%29%3B&description=asdasd&active=on&addgroup=true
```

```
HTTP/1.1 200 OK
Date: Wed, 04 Apr 2018 15:54:38 GMT
Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2h PHP/5.6.24
X-Powered-By: PHP/5.6.24
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Keep-Alive: timeout=5, max=37
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
```

```
</div>
</header><section class="cf"><div class="pageerrorcontainer"><ul class="pageerror"><li>Ya existe un grupo con ese nombre</li></ul></div>
<div class="pagecontainer">
    <form method="post" action="addgroup.php">
  <div>
    <input type="hidden" name="_sk_" value="bc13626df3e54660" />
  </div>
     <div class="pagewarning">Creating a new group does not assign any permissions. You will need to assign permissions to the new group
  <div class="pageoverflow">
    <p class="pagetext"><label for="groupname">*Nombre:</label></p>
    <p class="pageinput"><input type="text" id="groupname" name="group" maxlength="255" value="" onMouseOver="alert(1);" /></p>
  </div>
  <div class="pageoverflow">
    <p class="pagetext"><label for="description">Descripción:</label></p>
```

Figure 13: Vulnerability audit (false positive) with DAST tool and manual checking

4.2.3 Semi-Automatic Dynamic Analysis

The analysis performed using the DAST tool, which is required in a first instance, human support performs a manual crawling of the web application to identify the entire attack surface of the web application. The tool selected for analysis (OWASP ZAP) was configured as an interception proxy, in order to visualize and record all manual crawling traffic.

Once the attack surface was identified, an automatic active scan was performed using the DAST tool. This scan injects recursively distinct payloads or malicious strings in order to exploit and discover vulnerabilities in the input fields of the web application. This step is very time-consuming. The tool in the active analysis process detected five types of high-level vulnerabilities, six medium-type and 13 low-level.

Finally, the vulnerabilities result of the active scan were audited manually with the help of DAST tool on the web site. The results of the manual audit of High security vulnerabilities are shown in Tab. 2.

Table 2: DAST security vulnerabilities audited

CWE	file	DAST results
CWE-352 anti CSRF token scan	login.php	TP[1]
CWE-79 XSS	addgroup.php	TP
CWE-79 XSS	adduser.php	TP
CWE-79 XSS	editgroup.php	TP
CWE-79 XSS	edituser.php	TP
CWE-79 XSS	myaccount.php	TP
CWE-79 XSS	moduleinterface.php	TP
CWE-79 XSS	siteprefs.php	TP
CWE-89 SQLi	ajax-content.php	TP
CWE-89 SQLi	moduleinterface.php	TP
CWE-89 SQLi	edituser.php	TP
CWE-89 SQLi	moduleinterface.php	TP
CWE-540 source code disclosure	adduser.php	FP
CWE-540 source code disclosure	listbookmarks.php	FP
CWE-540 source code disclosure	listgroups.php	FP

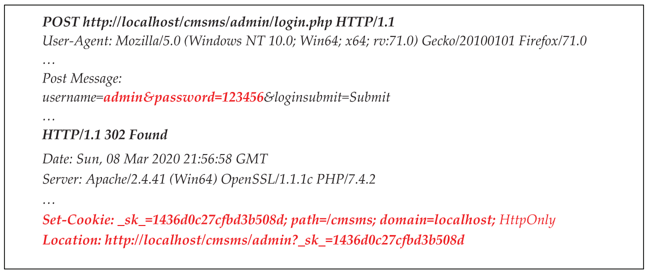
[1]TP = true positive.

4.2.4 Dynamic Security Analysis of Access Control

In this phase, two access control checks are carried out:

1. Dynamic analysis using the selected tool of type DAST. The Owasp ZAP tool has an access control plugin to check accesses to all web application's URLs.
2. Brute force attack using a list of payloads extracted from the OWASP SecLists Project [67], which contains a collection of lists containing users, passwords, URLs, etc., that have been compiled from multiple security assessments. List types include usernames, passwords, URLs, sensitive grep data strings, fuzzy payloads, and many more. The resulting analysis confirmed that there is a weak access control validation policy and from its results we resume security conclusions obtained from the access control security analysis. Fig. 13 shows a brute force attack using ZAP DAST tool against the username and password

parameters of the login.php page. This brute force attack submits many requests recursively to the web application by assigning different values to the password parameter using all values included in the password dictionary of the OWASP SecLists Project. The response in the Fig. 14, shows how the user is redirected to the admin panel after a successful login. All web applications should implement a robust password policy.



```
POST http://localhost/cmsms/admin/login.php HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:71.0) Gecko/20100101 Firefox/71.0
...
Post Message:
username=admin&password=123456&loginsubmit=Submit
...
HTTP/1.1 302 Found
Date: Sun, 08 Mar 2020 21:56:58 GMT
Server: Apache/2.4.41 (Win64) OpenSSL/1.1.1c PHP/7.4.2
...
Set-Cookie: _sk_=1436d0c27cfbd3b508d; path=/cmsms; domain=localhost; HttpOnly
Location: http://localhost/cmsms/admin?_sk_=1436d0c27cfbd3b508d
```

Figure 14: Brute force attack with ZAP tool against login.php page

Tab. 3 shows the results of security vulnerabilities related to the access control mechanism security analysis.

Table 3: DAST security vulnerabilities audited

CWE	Results
CWE-20 improper input validation	TP[1]
CWE-522 insufficiently protected credentials	TP
CWE-307 improper restriction of excessive authentication attempts	TP
CWE-521 weak password policy	TP
CWE-564 SQLi hibernate XSS	TP

[1]TP = true positive.

Access control security analysis conclusions:

1. CWE-20. Brute force attack detected that there is no validation regarding discrimination between uppercase and lowercase letters in the user field.
2. CWE-522. The credentials are transported in plain text and are not encrypted in any of the layers of the web application.
3. CWE-307. Access control does not have a security policy of blocking requests that limit brute force attacks.
4. CWE-521. There is no password creation policy, any password can be configured.
5. CWE-564. The web application does not validate or sanitize the entries, resulting in the access control breaking.

4.3 Phase 3: Classification of Results

Next, the results obtained mainly in the execution phase of the methodology classified by each of the steps proposed are detailed below.

4.3.1 Semi-Automatic Static Analysis

The percentage of true positives obtained is 89.33, compared to a 10.66 percent of false positives (see Tabs. 4, 5). This a very good percentage of true positives comparing to other SAST comparison results obtained by some studies [34,39,40,68].

Table 4: SAST: TP and FP result metrics

CWE	Detections	TP[1]	FP[2]
CWE-79 XSS-P	44	44	0
CWE-79 XSS-R	244	238	6
CWE-94	20	9	11
CWE-676	2	2	0
CWE-95	5	5	0
CWE-91	2	2	0
CWE-321	1	0	1
CWE-601	16	16	0
CWE-259	2	2	2
CWE-22	48	25	23
CWE-359	17	17	0
CWE-89	1	1	0
CWE-497	1	1	0
Total	403	360	43
TP/FP percentages		**89.3**	**10.6**

[1]TP = true positive; [2]FP = false positive.

The vulnerability with the highest number of detections in this step was persistent and reflected type Cross Site Scripting (CWE-79), these vulnerabilities are the ones that most affect the analyzed web application.

4.3.2 SAST Results Checking with Dynamic Analysis

Based on the considerations of the methodology, in this analysis there were vulnerabilities detections that could not be validated through dynamic analysis, so Tab. 7 details that a total of 73.9 percent, could not be validated because the depth of the attack. The dynamic analysis did not allow to validate and in this case the verdict of step one was taken for each detection. Therefore, in the case of unproven vulnerabilities, it was identified that these vulnerabilities have no direct relationship with the attack surface of the application, and they cannot be tested with DAST or manually externally. This step is interesting because it allows to identify and associate the source inputs variables in the code with the input fields in the application web forms accessible from the web interface.

In Tab. 5, the column N/A refers to the number of vulnerabilities of each category that cannot be accessed externally by a DAST tool and manual checks. Notice that the checked vulnerabilities by DAST were 105 (26,1 per cent) of the total 403 vulnerability detections in the static security analysis of the source code. Tab. 7 also shows the results of the SAST detections manually checked by DAST. The fact of auditing a vulnerability as TP by three (SAST + DAST + MANUAL) distinct analysis types lets confirm the vulnerability as TP or FP.

Table 5: Vulnerabilities detected-audited by SAST and audited by DAST

CWE	SAST detections	SAST TP	SAST FP	DAST TP	DAST FP	N/A by DAST
CWE-79 XSS persistent	44	44	0	–	–	44
CWE-79 XSS reflected	244	238	6	90	6	142
CWE-94 dangerous file inclusion	20	9	11	9	0	2
CWE-676 dangerous function	2	2	0	–	–	2
CWE-95 code injection	5	5	0	–	–	5
CWE-91 JSON injection	2	2	0	–	–	2
CWE-321 hardcoded encryption key	1	0	1	–	–	0
CWE-601 open redirect	16	16	0	–	–	16
CWE-259 hardcoded password	2	2	2	–	–	2
CWE-22 path manipulation	48	25	23	–	–	25
CWE-359 privacy violation	17	17	0	–	–	17
CWE-89 SQL injection	1	1	0	–	–	1
CWE-497 system information leak	1	1	0	–	–	1
Total	403	360	43	99	6	298
TP FP N/A percentages[1]		**89.3**	**10.6**	**24.6**	**1.5**	**73.9**

[1]TP = true positive, FP = false positive, N/A = not applied.

Table 6: Audit for DAST common vulnerabilities with previous steps

CWE	File	DAST manual audit
CWE-79 Refleted XSS	addgroup.php	TP[1]
CWE-79 Refleted XSS	adduser.php	TP
CWE-79 Refleted XSS	editgroup.php	TP
CWE-79 Refleted XSS	edituser.php	TP
CWE-79 Refleted XSS	myaccount.php	TP
CWE-79 Refleted XSS	siteprefs.php	TP

[1]TP = true positive.

4.3.3 Semi-Automatic Dynamic Analysis

The results of the DAST audit (see Tab. 4) also with manual checking are presented below, based on their classification as common (see Tab. 8) and non-common (see Tab. 9) vulnerabilities with results of previous steps.

Tab. 6 shows the detections by dynamic security analysis detected by static security analysis, (6) of (244) reflected XSS vulnerabilities have been commonly detected, in a first step by security static analysis and in a second step by security dynamic analysis. This result implies that, generally, both types of analysis find some distinct categories of vulnerabilities due to their different designs.

Table 7: Audit for DAST non common vulnerabilities with previous steps

CWE	File	DAST manual audit
CWE-352 anti CSRF	login.php	TP[1]
CWE-79 R-XSS	moduleinterface.php	TP
CWE-89 SQLI	ajax-content.php	TP
CWE-89 SQLI	moduleinterface.php	TP
CWE-89 SQLI	edituser.php	TP
CWE-89 SQLI	moduleinterface.php	TP
CWE-540	adduser.php	FP[2]
CWE-540	listbookmarks.php	FP
CWE-540	listgroups.php	FP

[1]TP = true positive; [2]FP = false positive.

Table 8: Access control vulnerabilities identified

CWE	TP[1]	FP[2]
CWE-20 improper input validation	1	0
CWE-522 insufficiently protected credentials	1	0
CWE-307 excessive authentication attempts	1	0
CWE-521 weak password policy	1	0
CWE-564 SQLi hibernate	1	0
Total	**5**	**0**

[1]TP = true positive; [2]FP = false positive.

Table 9: Results of true and false positive vulnerabilities. DAST-NC = vulnerabilities not found (not common) with static analysis

Analysis	Detections	TP	FP	%TP	%FP
SAST	403	360	43	89.33	10.66
DAST-NC(DAST)	9(15)	6(12)	3	80	20
A. CONTROL	5	5	0	100	0
Total	**417**	**371**	**46**	**88.9**	**11.03**

Tab. 6 shows the detections by dynamic security analysis detected by static security analysis, (6) of (244) reflected XSS vulnerabilities have been commonly detected, in a first step by security static analysis and in a second step by security dynamic analysis. This result implies that, generally, both types of analysis find some distinct categories of vulnerabilities due to their different designs.

Tab. 7 shows the detections by dynamic security analysis not detected by static security analysis. Nine (9) vulnerabilities were detected and the following auditory results confirm six (6) new vulnerabilities to the results obtained by SAST analysis.

One (1) CSRF, four (4) SQLI and one (1) Reflected XSS. The results confirm that independents SAST analysis find more vulnerabilities that isolate the DAST analysis: The SAST audited analysis confirm (360) vulnerabilities and the DAST audited analysis confirms (12) vulnerabilities (6 common with SAST and 6 non-common with SAST).

4.3.4 Dynamic Security Analysis of Access Control

Tab. 8 shows the results obtained from the security analysis of the access control mechanism. Checking access control mechanism is a critical requisite and the results confirm significant vulnerabilities. The application has many improper input validation vulnerabilities such as Reflected XSS that permit steal the session ID stored in a cookie and SQLi Hibernate that can allow an attacker to modify the statement's meaning or execute arbitrary SQL commands. Credentials are transported in plain text because the application does not implement the HTTP protocol and there is no a robust password policy that permit a good defense against brute force attacks.

All discovered access control vulnerabilities confirm that it is possible to gain access to the application and subsequently gain additional privileged access to other resources of the application such as files or database objects. The access control analysis has found few vulnerabilities, but they are all really critical.

5 Results Analysis of the Methodology Practical Application and Discussion

Based in the results obtained, we formulate four research question to validate the methodology practical application:

RQ1. Does the methodology analyze static and dynamic security tools types to extract the best way for combining them in several phases to obtain a better result?

—The methodology takes advantage of using the main skills of SAST and DAST tools to combine them with the aim of reducing the number of false positives and finding a greater number of true positives. This objective is reached by checking SAST results using a DAST tool that includes manual checks and correlating the results of the isolation DAST analysis with the previous results obtained by the SAST analysis checked by a DAST tool that includes manual checks. Besides the methodology performs access control checks using several plugins of DAST tool and manual checks to discover additional vulnerabilities.

RQ2. Is the methodology new and repeatable for security assessment of web applications that can be generalized for any web technology and web languages?

—The revision of the state of the art demonstrates that there are no other methods to use different types of tools on the market to take advantage of the main skills of the SAST and DAST tools in combination. The state of the art reveals that there are specific hybrid SAST-DAST tools, but there is no general and repeatable methodology that allows a process such as the one presented in this study to be followed to analyze the security of a web application that takes advantage of the synergies of the analyzes performed. In this study, in addition, this methodology presents the necessary mechanism how to adapt to any process of software development, technology or web language. It is also not closed, which allows one or more analysis activities to be included to improve the effectiveness obtained.

Our methodology is general and open to incorporate any SAST tool to be adapted to any web language application server and also it is open to incorporate any DAST tool for combining with the SAST tool. Related work shows diverse performance comparatives of tools [40,68,69] to make the best election a SAST or DAST tools.

RQ3. Does the methodology demonstrate that it is effective to evaluate the security of real-world web applications?

—We have selected a real web application, CMS Simple used by many users on the Internet, to demonstrate the degree of performance of this new methodology. The results of the practical application of the methodology In Tab. 9, details the number of vulnerabilities classified as true positives and false positives.

Based on the results obtained from the implementation of the evaluation methodology, the degree of coverage of the security analysis methodology for web applications is presented below. Considering the number of true positives and false positives, it was found that the degree of effectiveness is 88,9 percent based on the identification of vulnerabilities classified as true positives and false positives (see Tab. 9).

The static analysis (white box) finds many security vulnerabilities with few false positives (10,69 per cent) identified and confirmed by the posterior auditory of each reported vulnerability.

Dynamic analysis with manual checking is used to audit the results, 24,6 per cent of security vulnerabilities reported by the static analysis has been checked and it allows to study which vulnerabilities can be directly exploited externally. This phase is very important because it permits that each reported vulnerability can be checked by a second tool (DAST) to confirm whether a vulnerability is true or false positive.

Dynamic analysis (black box) finds six (6) additional critical vulnerabilities as SQL injections (4), CSRF (1) in login.php page, XSS (1).

Access control analysis finds other five (5) important vulnerabilities, such as Insufficient Protected Passwords or Weak Password Policy and Excessive Authentication Attacks, two vulnerabilities that permit brute force attacks. The access control vulnerabilities found are very critical because they allow brute force attacks and XSS or SQLI to steal session state or application credentials.

XSS is the most frequent vulnerability, 288 different XSS vulnerabilities are found and classified as critical by SAST and DAST tools. However, in general, all vulnerabilities found as a result of the practical application of the methodology are included in the OWASP Top Ten 2017 project as the most critical vulnerabilities.

The application of the methodology confirms that their true positive number (371) is better than the true positive number obtained by only SAST analysis (360) due to the methodology combines the DAST analysis (6) and access control analysis (5). Also, the methodology performs auditory steps to reduce the number of false positives as much as possible in the phases:

- 4.2.1 (SAST analysis) It confirms 43 FP
- 4.2.2 (Audit SAST results with DAST and manual checks) It confirms 90 SAST TP/9 SAST FP
- 4.2.3 (DAST analysis) It confirms 6 FP
- 4.2.4 (Access Control analysis) It confirms that all vulnerabilities discovered (5) are TP

This method can be extended to any other web application you consider to select a SAST tool capable of analyze the source code language used by the web application and a DAST tool selected according to diverse performance comparatives as showed in [40,68,69].

RQ4. Can the methodology be used by others more general methodologies that do not explain how to combine static and dynamic security analysis tools in web applications?

—General methodologies to analyze the security of web applications as the OWASP testing guide version 4 explains the types of tools to use to analyze each part of a web application, client, server side, database server, application code, configurations, input validation, authentication, sessions, authorization, logging, etc., but it does not explain how to take advantage of the use of different types of tools in combination. Our methodology can be integrated as a part of OWASP testing guide to analyze each part of a web application that can be analyzed with SAST, DAST and manual checks according to the OWASP testing guide. Thus, these two methodologies can benefit each other to achieve a better result in their joint.

RQ5. How this methodology differs from other methodologies that evaluate the security of web applications?

—This methodology in comparison to other proposals how [70–74], differs in the first place in that the necessary criteria that allow taking advantage of the coverage of static and dynamic analyzes as a whole are not defined, but rather, they use each One of these approaches separately, also do not establish a clear way of how to discern the results produced by tools, which generate many false positives. They also do not use the criteria for checking results manually. The method designed in [73] establishes the penetration testing mechanism (PenTesting). The work implemented in [74] designs an injection tool for SQLi and XSS, however its degree of coverage is very limited. The method designed in [72] uses analysis tools based on the Kali Linux Operating System. The work [71] defines graphical analysis models of results obtained from security analysis tools, however it does not clearly define how to use the tools from different degrees of coverage and the work shown in [70] establishes static and dynamic analysis techniques in a student context, where the tools are expected to support developer training, so that based on their results they can become aware of the importance of writing clean and safe code. Finally, our methodology collects and bases its mechanism on the use of static and dynamic analysis to expand the analysis coverage in a clear and simple way, in addition to taking advantage of manual checks, which define scenarios that attackers could use to violate web applications allowing the assurance to be effective through the identification of the false positives that the tools generate, reaching the point of identifying possible vulnerabilities that do not cover each one of the analyzes. In addition, depending on the objective intended in [70], the simplicity and ease of using our methodology could serve as a basis for training and raising awareness among developers when writing code that guarantees the security of web applications.

RQ6. Can the methodology be integrated into an agile SDLC such as SCRUM or traditional SDLC such as WATERFALL or the similar?

—The flexibility of the presented methodology shows that it can be applied to any method of Software development. In SDLC, it can be shown that the methodology can support the processes and activities carried out in each of the phases to ensure the security of the software being built. For example, in SDLC it should be used in the development, implementation and testing phases so that its results validate the context of the security of the web application before going to a production stage. In the same way with WATERFALL, the process of evaluation and correlation of results allows that the results obtained in each phase of software development can be used as an security criterion in each change of stage so that the levels of security are guaranteed during the process. However, despite its easy adaptation to any context, we emphasize, in the processes of agile development and continuous deliveries. The simplicity that our methodology demonstrates makes it a validation and assurance mechanism for web applications in an agile development environment, since the incremental process and the small launches that are made at the end of each Sprint in SCRUM, allow them to be analyzed and validate at a more specific level of detail than with other software development methods, in addition, the acceptance criteria of each User Story can be used to determine the levels of security through the definition of criteria or stories of the attacker how It is mentioned [75,76] that they can be combined with the manual validations presented by the methodology and thus further improve the effectiveness of each of security evaluations.

6 Related Work

In this section we include related works about techniques to combine static and dynamic analysis security tools and studies that analyze and compare Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST). All the methodologies analyzed first perform a SAST analysis to generate a set of test cases to test each vulnerability found by SAST with a DAST tool for TP verification. These methods try to automatize the task of eliminating the false positives detecting by previous SAST analysis.

Our methodology also uses DAST tools to confirm SAST detections, but our method performs manual audits of independent SAST and DAST analysis and finally correlates all analysis results to present a complete report. We demonstrate that manual audit of vulnerabilities with the help of tools is also necessary to verify all vulnerabilities detected by SAST and DAST tools. Moreover, our repeatable method permits select any SAST or DAST and IAST tool to adapt to any web technology and vulnerability categories.

6.1 Studies About Combining Static and Dynamic Security Analysis

Kim et al. [12] propose a method to combine static and dynamic detection to reduce false positives created from static vulnerability detection. It is limited to check the false positives discovered by static using dynamic analysis in a second phase.

Csallner and Smaragdakis describe several implementations of this type of tools. Despite its simplicity they can find bugs that would require complex static analysis efforts. Check 'n' Crash [77] uses JCrasher as a post-processing step to the powerful static analysis tool ESC/Java. JCrasher [78] is a simple, mostly dynamic analysis tool that generates JUnit test cases. As a result, Check 'n' Crash is more precise than ESC/Java alone and generates better targeted test cases than JCrasher in isolation. DSD-Crasher [79] adds a reverse engineering step to Check 'n' Crash to rediscover the program's intended behavior. This enables DSD-Crasher to suppress false positives with respect to the program's informal specification.

Babic et al. [80] presents a new technique to exploit static analysis to guide the automated dynamic generation of security test for binary programs, prioritizing the paths to be explored. The technique is a three-stage process, which alternates dynamic and static analysis. Preliminary experiments on a suite of benchmarks extracted from real applications show that static analysis allows exploration to reach vulnerabilities that it would not otherwise reach, and the generated test inputs prove that the static warnings indicate true positives.

PHP vulnerability hunter [53] is a white box PHP web application fuzzer that scans for several different classes of vulnerabilities *via* static and dynamic analysis. By instrumenting the application code, PHP Vulnerability Hunter is able to achieve greater code coverage and uncover more vulnerabilities.

Dogan et al. [81] perform a systematic literature review about web application testing. They discuss emerging trends in web application security testing and the implications for researchers and professionals in this area. The results of the study can help researchers gain an overview of existing web application testing approaches, failure models, tools, metrics and empirical evidence, and subsequently identify areas in the field that require more attention from the research community.

Prokhorenko et al. [82], surveys the web application protection techniques, aiming to systematize the existing approaches and review static and dynamic white box and black box security analysis techniques describing their advantages. The advantage of implementing black-box approaches is that such approaches can be deployed for any web application (even if the source code is unavailable). However, a deeper understanding of how the application should behave may not be possible with black-box approaches, potentially leading to a lower attack detection rate. Regardless of the details of the analysis used, due to the requirement to have access to the application sources, the use of white-box approaches is conceptually limited to either open source software or applications developed in-house.

Kiss et al. [83], designed a combined tool named FLINDER SCA for vulnerability detection, implemented on top of FRAMA-C, a platform for collaborative verification of C programs, and Search Lab for FLINDER testing tool. FLINDER-SCA includes three steps. First, abstract interpretation and taint analysis are used to detect potential vulnerabilities, then program division is applied to reduce the initial program, and finally a testing step tries to confirm detected security alarms by fuzzing on the reduced program. They describe the proposed approach and the tool, illustrate its application for the recent OpenSSL HeartBeat Heartbleed vulnerability, and discuss the benefits and industrial application perspectives of the proposed verification approach.

Nunes et al. [11] study the problem of combining diverse SAST to improve the overall detection of vulnerabilities in web applications, considering four development scenarios with different criticality goals and constraints. The findings about XSS and SQLI vulnerabilities revealed that combining the outputs of several SAST does not always improve the vulnerability detection performance in a single SAST. By using this procedure, a developer is able to choose which is the best combination of SAST that fits better in the project requirements. This work can be improved by testing SAST using a benchmark with more vulnerabilities.

Díaz [70] explains how to evaluate the security of student projects focused on the construction of web applications, for which, he proposes the use of free or open source security analysis tools to carry out security tests using analysis techniques static and dynamic. In addition, it uses the results as an awareness mechanism for training developers and emphasizes the use of good practices when writing clean and safe source code.

Lakshmi et al. [71] emphasize the difficulty of development teams in conducting security tests in continuous development environments where launch cycles are becoming shorter and shorter. For this, in their study they present a comparison of tools, techniques and outstanding models and support for the security analysis of web applications for heterogeneous technological contexts. In addition, this paper highlights the current trend of research for some of the web application testing techniques.

Babincev et al. [72] in their study describe how the Kali Linux operating system works through its controls and capabilities. It supports the security assessment processes of web applications, highlighting a group of tools compatible with the operating system. In addition, it makes a specific explanation of how to use each of these tools, however, it does not emphasize any specific type of analysis.

Skoruppa [73] in his doctoral thesis describes how to take advantage of static analysis methods to discover security vulnerabilities in web applications based on JavaScript and PHP. It also defines a graphical framework that allows developers to analyze the code in PHP based on defined graphical criteria to model common vulnerabilities based on the results obtained by the static analysis.

Le et al. [74] propose a protective, extensible and hybrid platform called GuruWS, to automatically detect vulnerabilities in malicious web applications and web shells. For this, they use the original PHP THAPS vulnerability scanner as a base, however, its degree of coverage is very limited and it focuses specifically on scanning web applications written in PHP, which does not allow their tool to be used in other technological contexts.

6.2 Security Analysis Tools Comparisons

These studies will permit to select the most suitable SAST, DAST or IAST tools to best adapt to any web technology and vulnerability categories. Wagner et al. [84] analyzed three open source static analysis tools: FindBugs, PMD and QJ Pro that were executed against six projects, one of them was a development project which were in the final testing phase. The tools obtained very different results.

According to Elizabeth Fong [85,86], in two interesting articles about DAST tools should be able to identify a subset of acceptable security vulnerabilities of web applications and generate a report for each detected vulnerability, indicating an action or set of actions that suggest the vulnerability and having an acceptable false positive rate, which of course can also have these tools.

The study of Ware et al. [87] "Securing Java Code: Heuristics and an Evaluation of Static Analysis Tools" concludes that, of the total of nine tools involved, only two are valid tools for J2EE web applications: Find bugs and HP Fortify SCA. One interesting finding obtained is that, from the total of 115 different vulnerabilities (not J2EE), only 50 were identified by summing all the detections of the 9 tools. The best result is achieved by HP-Fortify SCA that identifies 27 vulnerabilities. Lastly, the authors mention the need for a subsequent audit result.

The work of Diaz et al. [40], presents a methodology that allows strictly ranking the nine static tools analyzed, in terms of vulnerabilities coverage and effectiveness detecting the highest number of vulnerabilities having few false positives. The best results for F-measure metric are obtained by Prevent, K8-insight and SCA. The differences between all tools are relevant, detecting different kinds of vulnerabilities. The authors recommend combining different tools in the security evaluation process.

The OWASP Benchmark project [69], according to its web site "is designed to evaluate the speed, coverage, and accuracy of automated vulnerability detection tools. Without the ability to measure these tools, it is difficult to understand their strengths and vulnerabilities, and compare them to each other". The Benchmark contains thousands of test cases are fully executable and exploitable (see Tab. 10).

Table 10: Owasp Benchmark results

CWE	Detections	TP [1]	FP [2]
FindSecBugs v1.4.6	96.84	57.74	39.10
FindBugs v3.0.1	5.12	5.19	-0.07
OWASP ZAP v-2016	19.95	0.12	19.84
PMD v5.2.3	0.00	0.00	0.00
SonarQube Java Plugin v3.14	50.36	17.02	33.34
Commercial SAST-01	28.96	12.22	26.74
Commercial SAST-02	56.13	25.53	30.60
Commercial SAST-03	46.33	21.34	24.89
Commercial SAST-04	61.45	28.81	32.64
Commercial SAST-05	47.74	29.03	18.71
Commercial SAST-06	85.02	52.09	32.93

[1]TP = true positive; [2]FP = false positive.

It includes security test cases for XSS, SQLI, XPATH, command injection, path traversal and other sensitive data exposure vulnerabilities included in OWASP top ten 2017. It has computed results for six commercial tools (SAST, DAST and IAST), but is not publicly available due to licensing reasons. Analyzing the project, it is difficult to obtain conclusions about SAST commercial tools because their results are anonymous. The project provides the average results of all commercial tools including tools categories, SAST tools (HP-Fortify, Checkmarx, Coverity, IBM-Appscan, Parasoft Jtest), DAST (Arachni, ZAP, Acunetix, Burp, HP-Webinspect, Rapid7) and IAST (Contrast). The results of 5 SAST free tools, PMD, FindBugs, FindSecurityBugs, SonarQube and OWASP ZAP (dynamic black box tool) are available against version 1.2 beta of the Benchmark. SonarQube and FindSecurityBugs obtain the best results with a similar balance between true and false positives. FindSecurityBugs finds more vulnerabilities but has higher rate of false positives. Actually, SonarQube besides its Java plugin integrates PMD, FindBugs and FindSecurityBugs plugins for increasing the detection ratio of security vulnerabilities. Java and PMD plugins of SonarQube are focused in quality bugs not in security vulnerabilities, therefore SonarQube relies in FindBugs and FindSecurityBugs to find security vulnerabilities.

Bau et al. [68] analyze Black-box web application vulnerability scanners (DAST), the results show the promise and effectiveness of automated tools, as a group, and also some limitations. In particular, "stored" forms of Cross Site Scripting and SQL Injection vulnerabilities are not currently found by many tools and they can stand improvement on CSRF, Malware or session vulnerabilities. Because their goal is to assess the potential for future research, not to evaluate specific vendors, they do not report comparative data or make any recommendations about purchasing specific tools.

7 Conclusions and Future Work

In this study, the development and execution of an evaluation methodology based on the combination of static, dynamic analysis and access control techniques was carried out, which left the following conclusions. This methodology is repeatable and flexible allowing it to be adapted to any language, or web technology by selecting the SAST tools adapted to specific languages and selecting the DAST tools adapted to specific web technologies, such as input vectors, authentication methods, etc.

The application of this methodology to analyze the security of a web application shows that the static analysis finds many security vulnerabilities with few false positives (10,69 per cent) identified and confirmed by the posterior auditory of each reported vulnerability. Dynamic analysis with manual checking is used to audit the results, 24,6 per cent of security vulnerabilities reported by the static analysis has been checked and it allows to study which vulnerabilities can be directly exploited externally. This phase is very important because it permits that each reported vulnerability can be checked by a second tool (DAST) to confirm whether a vulnerability is true or false positive. Dynamic analysis finds six (6) additional critical vulnerabilities as SQL injections (4), CSRF (1) in login.php page, XSS (1). Access control analysis finds other five (5) important vulnerabilities, such as Insufficient Protected Passwords or Weak Password Policy and Excessive Authentication Attacks, two vulnerabilities that permit brute force attacks. The access control vulnerabilities found are very critical because they allow brute force attacks and XSS or SQLI to steal session state or application credentials. The results obtained from the security tests carried out revealed that the analyzed web application presents cross Site Scripting (XSS) vulnerabilities in most cases, due to a weak process of verification of data entries by the application.

The joint use of the SAST and DAST tools to carry out each step of the proposed methodology allowed the security testing process to be more productive and effective. The dynamic analysis tool is used first to manually verify the results of static analysis to effectively discard false positives, following a dynamic analysis is performed and finally, the results of the static and dynamic analysis are correlated obtaining an improvement in the effectiveness of the results in terms of true and false positives.

The methodology used in this study only evaluates the application as such, it is important that security implementations are involved in all layers of the same or that the deployment of a web application integrates all possible levels of security.

This methodology can be integrated as a part of OWASP testing guide to analyze each part of a web application that can be analyzed with SAST, DAST and manual checks according to the OWASP testing guide. Thus, these two methodologies can benefit each other to achieve a better result in their joint.

In the security analysis process of web applications, there are a lot of security techniques, however, new ways of violating applications are still being discovered today, so it would be important to establish a formal method to teach developers that involves tests, standards, norms, good practices and not only language teaching and ways to program, in order to minimize the security risk presented by the human factor. In addition, emphasis should be placed on the use of the secure software development life cycle for all software projects since the SSDLC already involves security testing in each of the development phases and discard methods that propose the execution of security tests in the final phases of software development. The flexibility of the presented methodology shows that it can be applied to any method of Software development. In SDLC, it can be shown that the methodology can support the processes and activities carried out in each of the phases to ensure the security of the software being built. For example, in SDLC it should be used in the development, implementation and testing phases so that its results validate the context of the security of the web application before going to a production stage.

The developed methodology generated a high and precise degree of effectiveness, which makes it an alternative to be used in any software development approach, especially in agile processes and continuous delivery, where deliveries are expected to be partial and in a short time. In this way, the proposed methodology would allow members of the development team to carry out the necessary security validations in each of the planned launches. Furthermore, it can be integrated into other methodologies that suggest using the same types of analysis, but do not specify how to do it. This methodology details how to integrate diverse types of analysis to optimize results. The practical application shows that the integrated analysis SAST, DAST and manual obtain better results.

The methodology, like any other, can be improved, its improvements include the implementation of analysis tools such as IAST, which according to some research presents much more accurate results when detecting vulnerabilities, among its results is the minimization of false positives or the scope of coverage of improved vulnerabilities among other essential features that support the security verification process. IAST tools could be integrated into the semi-automatic dynamic analysis subphase of the execution phase (see Section 4.2.3). The combined work of the SAST, DAST, IAST tool types aided by manual checks required to audit the tools' individual results will result in the highest detection of vulnerabilities with the lowest number of false positives. In addition, we will study the possibility of including in the method the combination of two SAST tools, two DAST tools and two different IAST tools in each case. Combining two tools of the same type improves the results due to their different designs in terms of the algorithms they use.

Author Contributions: All authors have equally contributed to this work. All authors read and approved the final manuscript.

Funding Statement: The author(s) received no specific funding for this study.

Conflicts of Interest: The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

1. Rosado, D. G., Blanco, C., Sánchez, L. E., Fernández-medina, E., Piattini, M. (2010). La Seguridad como una asignatura indispensable para un Ingeniero del software. *XVI Jornadas de Enseñanza Universitaria de La Informática*, pp. 205–212. Santiago de Compostela, Spain.
2. Veracode (2010). How vulnerabilities get into software. https://info.veracode.com/how-do-vulnerabilities-get-into-software-whitepaper-resource.html.
3. OWASP (2017). *OWASP*. https://www.owasp.org/index.php/.
4. SANS (2017). SANS information security training cyber certifications research. https://www.sans.org/.
5. Felderer, M., Büchler, M., Johns, M., Brucker, A. D., Breu, R. et al. (2016). Security testing: A survey. *Advances in Computers, 101,* 1–51.
6. McGraw, G. (2006). *Software security: Building security in.* USA: Addison-Wesley.
7. Vicente Mohino, 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, 8(11),* 1218. DOI 10.3390/electronics8111218.
8. Trustwave (2017). Trustwave globa security report. https://www2.trustwave.com/rs/815-RFM-693/images/2015_TrustwaveGlobalSecurityReport.pdf.
9. ESET (2016). ESET security report latinoamérica 2016. https://www.welivesecurity.com/wp-content/uploads/2016/04/eset-security-report-latam-2016.pdf.

10. Ghaffarian, S. M., Shahriari, H. R. (2017). Software vulnerability analysis and discovery using machine-learning and data-mining techniques. *ACM Computing Surveys, 50(4),* 1–36. DOI 10.1145/3092566.
11. Nunes, P., Medeiros, I., Fonseca, J., 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, 101(2),* 161–185. DOI 10.1007/s00607-018-0664-z.
12. Kim, S., Kim, R. Y. C., Park, Y. B. (2016). Software vulnerability detection methodology combined with static and dynamic analysis. *Wireless Personal Communications, 89(3),* 777–793. DOI 10.1007/s11277-015-3152-1.
13. Chess, B., McGraw, G. (2004). Static analysis for security. *IEEE Security and Privacy Magazine, 2(6),* 76–79. DOI 10.1109/MSP.2004.111.
14. Egele, M., Scholte, T., Kirda, E., Kruegel, C. (2012). A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys, 44(2),* 1–42. DOI 10.1145/2089125.2089126.
15. Pandikumar, T., Eshetu, T. (2016). Detecting web application vulnerability using dynamic analysis with penetration testing. *International Research Journal of Engineering and Technology, 3(10),* 430–433.
16. Bermejo Higuera, J. R. (2014). *Assessment methodology of web applications automatic security analysis tools for adaptation in the dev*. http://espacio.uned.es/fez/view/tesisuned:IngInd-Jrbermejo.
17. Russo, A., Sabelfeld, A. (2010). Dynamic *vs*. static flow-sensitive security analysis. *IEEE Computer Security Foundations Symposium*, pp. 186–199. Edinburgh, UK.
18. CMS Made Simple (2017). About CMS made simple. http://www.cmsmadesimple.org/about-link/.
19. CMS Made Simple (2017). CMS made simple. http://www.cmsmadesimple.org/.
20. CMS Critic Awards (2020). CMS critic awards, https://www.cmscritic.com/awards/#best-open-source-cms.
21. Ginige, A., Murugesan, S. (2001). Web engineering: An introduction. *IEEE Multimedia, 8(1),* 14–18. DOI 10.1109/93.923949.
22. Rudman, R. J. (2010). Incremental risks in Web 2.0 applications. *Electronic Library, 28(2),* 210–230. DOI 10.1108/02640471011033585.
23. Pan, Y., White, J., Schmidt, D., Elhabashy, C., Sturm, A. et al. (2017). Taxonomies for reasoning about cyber-physical attacks in IoT-based manufacturing systems. *International Journal of Interactive Multimedia and Artificial Intelligence, 4(3),* 45–54. DOI 10.9781/ijimai.2017.437.
24. Nguyen-Tuong, A., Guarnieri, S., Greene, D., Shirley, J., Evans, D. (2005). Automatically hardening web applications using precise tainting. In: *Security and Privacy in the Age of Ubiquitous Computing*. Boston: Springer.
25. Scholte, T., Balzarotti, D., Kirda, E. (2012). Have things changed now? An empirical study on input validation vulnerabilities in web applications. *Computers and Security, 31(3),* 344–356. DOI 10.1016/j.cose.2011.12.013.
26. Hernández Saucedo, A. L., Mejia Miranda, J. (2015). Guía de ataques, vulnerabilidades, técnicas y herramientas para aplicaciones web. *ReCIBE, 4(1),* 1–17.
27. Stack Overflow (2017). Developer survey 2017 . https://insights.stackoverflow.com/survey/2017.
28. Ricca, F., Tonella, P. (2001). Analysis and testing of web applications. *Proceedings of the 23rd International Conference on Software Engineering*, pp. 25–34. Toronto, Canada.
29. Garg, V. (2015). Approaches, tools and techniques for security testing. https://www.3pillarglobal.com/insights/approaches-tools-techniques-for-security-testing.
30. Turing, M. (1937). On computable numbers, with an application to the entscheidungs problem. *Proceedings of the London Mathematical Society, 42(1),* 230–265. DOI 10.1112/plms/s2-42.1.230.
31. Sipser, M. (2012). Introduction to the theory of computation. DOI 10.1007/s13398-014-0173-7.2.
32. Livshits, V. B., Lam, M. S. (2005). Finding security vulnerabilities in java applications with static analysis. https://www.usenix.org/legacy/event/sec05/tech/full_papers/livshits/livshits.pdf.
33. Chess, B., West, J. (2013). *Secure programming with static analysis,* USA: Addison-Wesley.
34. Emanuelsson, P., Nilsson, U. (2008). A comparative study of industrial static analysis tools. *Electronic Notes in Theoretical Computer Science, 217,* 5–21. DOI 10.1016/j.entcs.2008.06.039.

35. Checkmarx (2017). Static application security testing. https://www.checkmarx.com/glossary/static-application-security-testing-sast/.
36. Hewlett Packard Enterprise (2017). Static analysis, static application security testing, SAST | hewlett packard enterprise. https://saas.hpe.com/en-us/software/sca.
37. Veracode (2017). Static analysis (SAST) veracode. https://www.veracode.com/products/binary-static-analysis-sast.
38. Koussa, S. (2016). What do SAST, DAST, IAST, and RASP mean to Developers? https://softwaresecured.com/what-do-sast-dast-iast-and-rasp-mean-to-developers.
39. Damodaran, A., Di Troia, F., Visaggio, C. A., Austin, T. H., Stamp, M. (2017). A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques, 13(1),* 1–12. DOI 10.1007/s11416-015-0261-z.
40. Díaz, G., Bermejo, J. R. (2013). Static analysis of source code security: Assessment of tools against SAMATE tests. *Information and Software Technology, 55(8),* 1462–1476. DOI 10.1016/j.infsof.2013.02.005.
41. IBM (2017). IBM security AppScan family. http://www-03.ibm.com/software/products/es/appscan.
42. Gartner (2018). Magic quadrant for application security testing. https://software.microfocus.com/es-es/assets/enterprise-security-products/magic-quadrant-for-application-security-testing.
43. Ball, T. (1999). *The concept of dynamic analysis,* Berlin: Springer.
44. Kim, S. (2016). *Software vulnerability detection methodology combined with static and dynamic analysis.* New York: Springer.
45. Arkin, B., Stender, S., McGraw, G. (2005). Software penetration testing. *IEEE Security and Privacy Magazine, 3(1),* 84–87. DOI 10.1109/MSP.2005.23.
46. Al Hamami, Z. (2017). IAST, RASP, and runtime instrumentation-DZone security. https://dzone.com/articles/iast-rasp-and-runtime-instrumentation.
47. Williams, J., Dabirsiaghi, A. (2012). Interactive vulnerability analysis enhancement results. http://www.dtic.mil/docs/citations/ADA568544.
48. Rohr, M. (2017). IAST: A new approach for agile security testing. https://blog.secodis.com/2015/11/26/the-emerge-of-iast/.
49. Cisar, P., Cisar, S. M. (2016). The framework of runtime application self-protection technology. *IEEE 17th International Symposium on Computational Intelligence and Informatics*, pp. 81–86. Budapest, Hungary.
50. Sureda, T., Bermejo, J. R. (2017). Comparison of the effectiveness of WAF and RASP tools against attacks. http://reunir.unir.net/123456789/4742.
51. Contrast Security (2015). Contrast scores high marks running OWASP benchmark. https://www.contrastsecurity.com/owasp-benchmark.
52. AppSecure (2018). Fusion lite insight. http://www.iappsecure.com/products.html.
53. PHP Vulnerability Hunter (2018). PHP vulnerability hunter-CodePlex archive. https://archive.codeplex.com/?p=phpvulnhunter.
54. Livshits, B. (2006). Improving software security with precise static and runtime analysis. https://suif.stanford.edu/livshits/papers/pdf/thesis.pdf.
55. Shakeel (2016). Penetration testing methodologies and standards. http://resources.infosecinstitute.com/penetration-testing-methodologies-and-standards/.
56. PTES (2009). Penetration testing execution standard. http://www.pentest-standard.org/index.php/Main_Page.
57. Dinis, B., Serrão, C. (2015). Using PTES and open-source tools as a way to conduct external footprinting security assessments for intelligence gathering. *Journal of Internet Technology and Secured Transactions, 3(4),* 271–279. DOI 10.20533/jitst.2046.3723.2014.0035.
58. Shanley, A., Johnstone, M. N. (2015). Selection of penetration testing methodologies: A comparison and evaluation. *Australian Information Security Management Conference*, Perth, Western Australia. DOI 10.4225/75/57b69c4ed938d.

59. Guasch, A. (2014). OWASP testing guide v4. http://www.securitybydefault.com/2014/09/publicada-la-owasp-testing-guide-v4.html.
60. López, A. (2014). OWASP testing guide v4.0. Guía de seguridad en aplicaciones web. https://www.certsi.es/blog/owasp-4.
61. OWASP (2020). OWASP testing guide v4.0. https://www.owasp.org/.
62. US Department of Commerce, N (2013). *National Institute of Standards and Technology*. https://www.nist.gov/.
63. Souppaya, M. P., Scarfone, K. A. (2008). Technical guide to information security testing and assessment. https://www.nist.gov/publications/technical-guide-information-security-testing-and-assessment.
64. Valdez Alvarado, A. (2013). OSSTMM 3. In: *Revista de información, tecnología y sociedad,* pp. 29–30. La Paz, Bolivia: Revistas Bolivianas.
65. Intel Software (2018). Dynamic analysis vs. static analysis. https://software.intel.com/en-us/inspector-user-guide-windows-dynamic-analysis-vs-static-analysis.
66. Yang, J., Tan, L., Peyton, J., Duer, K. A. (2019). Towards better utilizing static application security testing. *IEEE/ACM 41st International Conference on Software Engineering*, pp. 51–60. Montreal, Canada.
67. OWASP (2014). OWASP SecLists project-OWASP. https://www.owasp.org/index.php/OWASP_SecLists_Project.
68. Bau, J., Bursztein, E., Gupta, D., Mitchell, J. (2010). State of the art: Automated black-boxweb application vulnerability testing. *IEEE Symposium on Security and Privacy*, Berkeley/Oakland, USA. https://ieeexplore.ieee.org/document/5504795.
69. OWASP (2018). OWASP benchmark project. https://owasp.org/www-project-benchmark/.
70. Diaz, J. (2018). *Security analysis methodology for student web applications: A case study of the mills college computer science department alumni website (Master's Thesis)*. Oakland: Department of Math and Computer Science, Mills College.
71. Lakshmi, D. R., Mallika, S. S. (2017). A review on web application testing and its current research directions. *International Journal of Electrical and Computer Engineering, 7(4),* 2132.
72. Babincev, I. M., Vuletić, D. V. (2015). Analiza bezbednosti web aplikacija operativnim sistemom Kali Linux. *Vojnotehnički glasnik/Military Technical Courier, 64(2),* 513–531.
73. Skoruppa, M. H. A. (2017). *Automated Security Analysis of Web Application Technologies (Ph.D. Thesis).* University of Saarland, Germany.
74. Le, V. G., Nguyen, H. T., Pham, D. P., Phung, V. O., Nguyen, N. H. (2019). GuruWS: A hybrid platform for detecting malicious web shells and web application vulnerabilities. In: *Transactions on computational collective intelligence XXXII*. Berlin: Springer.
75. Denipotti, R. (2017). Secure Agile SDLC BSides. https://es.slideshare.net/RaphaelDenipotti/secure-agile-sdlc-bsides-14-2017-raphael-denipotti.
76. Cloud Software Finland (2014). Handbook of the secure agile software development life cycle. http://www.n4s.fi/2014magazine/article2/assets/guidebook_handbook.pdf.
77. Csallner, C., Smaragdakis, Y. (2005). Check 'n' crash: Combining static checking and testing. *International Conference on Software Engineering*, pp. 422–431. St. Louis, Missouri, USA.
78. Csallner, C., Smaragdakis, Y. (2004). JCrasher: An automatic robustness tester for Java. *Software—Practice & Experience, 34(11),* 1025–1050. DOI 10.1002/spe.602.
79. Csallner, C., Smaragdakis, Y. (2006). DSD-crasher: A hybrid analysis tool for bug finding. *ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 245–254. Portland, USA.
80. Babic, D., Martignoni, L., McCamant, S., Song, D. (2011). Statically-directed dynamic automated test generation. *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, pp. 12–22. Toronto, Canada.
81. Doğan, S., Betin-Can, A., Garousi, V. (2014). Web application testing: A systematic literature review. *Journal of Systems and Software, 91,* 174–201. DOI 10.1016/j.jss.2014.01.010.
82. Prokhorenko, V., Choo, K. K. R., Ashman, H. (2016). Web application protection techniques: A taxonomy. *Journal of Network and Computer Applications, 60,* 95–112. DOI 10.1016/j.jnca.2015.11.017.

83. Kiss, B., Kosmatov, N., Pariente, D., Puccetti, A. (2015). Combining static and dynamic analyses for vulnerability detection: Illustration on heartbleed. In: *Hardware and software: Verification and testing*. Cham: Springer.
84. Wagner, S., Jürjens, J., Koller, C., Trischberger, P. (2005). Comparing bug finding tools with reviews and tests. In: *Testing of communicating systems*. Berlin: Springer.
85. Fong, E., Okun, V. (2007). Web application scanners: Definitions and functions. *Proceedings of 40th Annual Hawaii International Conference on System Sciences*, Hawaii, USA.
86. Fong, E., Gaucher, R., Okun, V., Black, P. E., Dalci, E. (2008). Building a test suite for web application scanners. *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, Hawaii, USA.
87. Ware, M. S., Fox, C. J. (2008). Securing Java code: Heuristics and an evaluation of static analysis tools. *SAW'08 Proceedings of the 2008 Workshop on Static Analysis*, pp. 12–21. Tucson, Arizona, USA.