WP2

DIGIT B1 - EP Pilot Project 645

Deliverable 9: List of Requirements for Code Reviews

Specific contract n°226 under Framework Contract n° DI/07172 – ABCIII

April 2016

DIGIT Fossa WP2 – Governance and Quality of Software Code – Auditing of Free and Open Source Software.

Deliverable 9: List of requirements for code reviews

Author:

Disclaimer

Contents

CONTENTS **3**

LIST OF TABLES **5**

LIST OF FIGURES **6**

ACRONYMS AND ABBREVIATIONS **7**

1 INTRODUCTION **8**

1.1. OBJECTIVE OF THIS DOCUMENT AND INTENDED AUDIENCE 8
1.2. SCOPE 8
1.3. DOCUMENT STRUCTURE 9
1.4. KEY SUCCESS FACTORS 9
1.5. DELIVERABLES 10

2 METHODOLOGICAL APPROACH TO BUILDING THE ANALYSIS **11**

2.1. IDENTIFICATION OF THE REQUIREMENTS FOR THE CODE REVIEW METHODOLOGY (CODE REVIEW PROCESS) 11
2.2. IDENTIFICATION OF THE REQUIREMENTS FOR THE PRE-SELECTION OF THE CODE REVIEW TOOLS 12
2.3. IDENTIFICATION OF THE REQUIREMENTS FOR THE SELECTION OF THE CODE REVIEW TOOLS 12
2.4. PRE-SELECTION OF CODE REVIEW TOOLS 12
2.5. EVALUATION OF CODE REVIEW TOOLS 13

3 REQUIREMENTS FOR CODE REVIEW METHODOLOGY **14**

3.1. REQUIREMENTS FOR THE PLANNING PHASE 14
3.2. REQUIREMENTS FOR THE CODE REVIEW PHASE 17
3.2.1. INFORMATION GATHERING 17
3.2.2. AUTHENTICATION MANAGEMENT 18
3.2.3. AUTHORIZATION (ACCESS CONTROL) 21
3.2.4. DATA/INPUT VALIDATION OF DATA FROM ALL UNTRUSTED SOURCES & ENCODING 22
3.2.5. CRYPTOGRAPHY 24
3.2.6. USER AND SESSION MANAGEMENT 25
3.2.7. ERROR AND EXCEPTION HANDLING/INFORMATION LEAKAGE 26
3.2.8. AUDITING AND LOGGING 26
3.2.9. BUSINESS LOGIC TESTING 27

4 REQUIREMENTS FOR CODE REVIEW TOOLS **29**

4.1. PRE-SELECTION REQUIREMENTS 29

4.2. Selection Requirements for Code Review Tools 31
4.2.1. Requirements for the Code Review Planning Phase 31
4.2.2. Requirements for the Code Review Phase 32
4.2.2.1. Information Gathering 32
4.2.2.2. Authentication Management 33
4.2.2.3. Data/Input Validation of Data from all Untrusted Sources & Encoding 33
4.2.2.4. Cryptography 35
4.2.2.5. Error and Exception Handling/Information Leakage 36
4.3. Requirements for the Results of Code Review Phase 37
4.3.1. Requirements for Mandatory Features 37
4.3.2. Requirements for Optional Features 37

5 PRE-SELECTION AND EVALUATION OF CODE REVIEW TOOLS 39

5.1. Pre-selection of Tools 39
5.1.1. Tools to Evaluate 39
5.1.2. Evaluation and Pre-selection of Tools 40
5.1.3. Results of the Pre-selection 42
5.2. Evaluation of Code Review Tools 43
5.2.1. FindBugs 43
5.2.2. PMD 44
5.2.3. RIPS 44
5.2.4. SonarQube 45
5.2.5. VCG 46
5.2.6. Yasca 47
5.3. Conclusions 47

6 BIBLIOGRAPHICAL REFERENCES 49

7 ANNEXES 50

7.1. Description of the OWASP Verification Levels 50

List of Tables

Table 3-1 - Requirement template 14

Table 4-1 - Requirements for Pre-selection of the Tools for Code Review 29

Table 5-1 Tools to be preselected 39

Table 5-2 Tools excluded from the ranking 41

Table 5-3 other tools excluded from the ranking 41

Table 5-4 Selection requirements 42

List of Figures

Figure 1. WP2 Tasks.....9

Figure 2. Methodological approach to building the analysis 11

Acronyms and Abbreviations

API	Application Programming Interface
CERN	Conseil Européen pour la Recherche Nucléaire (renamed as Organisation Européen pour la Recherche Nucléaire in 1954 but acronym was retained)
DG	Directorate General
EP	European Parliament
ESAPI	Enterprise Security Application Programming Interface
EUI	European Institutions
FOSSA	Free and Open Source Software Auditing
FOSS	Free and Open Source Software
NIST	National Institute of Standards and Technology
OS	Operating System
OSS	Open Source Software
OWASP	Open Web Application Security Project
SAST	Static Application Security Testing
SDLC	Software Development Life Cycle
SEO	Search Engine Optimization
WP	Work Package

1 Introduction

1.1. Objective of this Document and Intended Audience

This document represents the deliverable 9 and contains the results of TASK-06: Requirements for the Code Reviews.

The objective of this document is to propose a list of requirements for the code review process, to be followed inside the European Institutions, as well as a list of requirement for the selection of the tools that will support this process.

This document targets the DIGIT and ITEC departments interested and/or responsible for the code reviews, related practices and tools.

1.2. Scope

To entirely understand the scope of the document, it is necessary to understand the aim of the Work Package (WP) 2. The WP2 has four tasks:

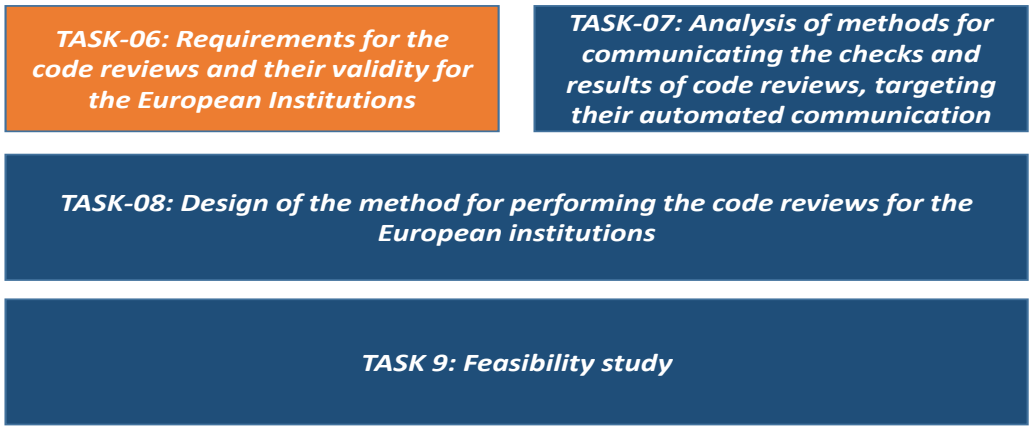
- Task 6: Requirements for the code reviews and their validity for the European Institutions that aim to define the list of requirements for proper code reviews, and to prepare an analysis of how they fit into the working methods of the European Commission and the European Parliament.
- Task 7: Analysis of the methods for communicating the checks and results of the code reviews, targeting their automated communication.

  Tasks 6 and 7 will provide the requirements that the methodology defined in task 8 needs to fulfil. For this reason, deliverables 9 and 10 (output of tasks 6 and 7) are complementary.
- Task 8: Design of the code review process to be used in the European Institutions, taking into account the requirements defined in tasks 6 and 7.
- Task 9: Feasibility study of the method defined to perform code review to be used inside the European Institutions.

This deliverable is deliverable 9, the result of task 6, and therefore covers the requirements for the code review methodology and the code review.

A graphical representation of WP2 tasks and their relationship is depicted in Figure 1.

Figure 1. WP2 Tasks



1.3. Document Structure

This document consists of the following sections:

- Section 1: **Introduction**, which describes the objectives of this deliverable, intended audience and Scope.
- Section 2: **Methodological Approach to Building the Analysis**, which describes the steps that we followed to conduct the evaluation of the requirements for the methodology and for the tools used to perform the code review, in line with the scope.
- Section 3: **Requirements for Code Review Methodology** to be valid for the European Institutions.
- Section 4: **Requirements for Code Review Tools**, which identifies the requirements for the selection of tools, for them to be adequate within the code review.
- Section 5: **Pre-selection and Evaluation of Code Review Tools,** according to the requirements detailed in Section 4.
- Section 6: **Bibliographical** references.
- Section 7: **Annexes**.

1.4. Key Success Factors

All steps described in Section 2 - Methodological Approach to Building the Analysis, will ensure the fulfilment of the key success factors related to this deliverable:

- The specific requirements that must be accomplished by the code reviews performed within the European Institutions are proposed.

1.5. Deliverables

1 *Deliverable 3: Analysis of Software Development Methodologies Used in European Institutions*
2 *Deliverable 4: Analysis of Software Development Methodologies Used in FOSS communities*
3 *Deliverable 7: Comparative Study*
4 *Deliverable 10: List of Methods for Communicating the Checks and Results of Code Reviews.*
5 *Deliverable 11: Design of the Method for Performing the Code Reviews for the European Institutions.*
6 *Deliverable 12: Feasibility Study and Method for Doing Code Reviews of Free and Open Source Projects in European Institutions, Targeting Automatic Communication of Checks and Results.*

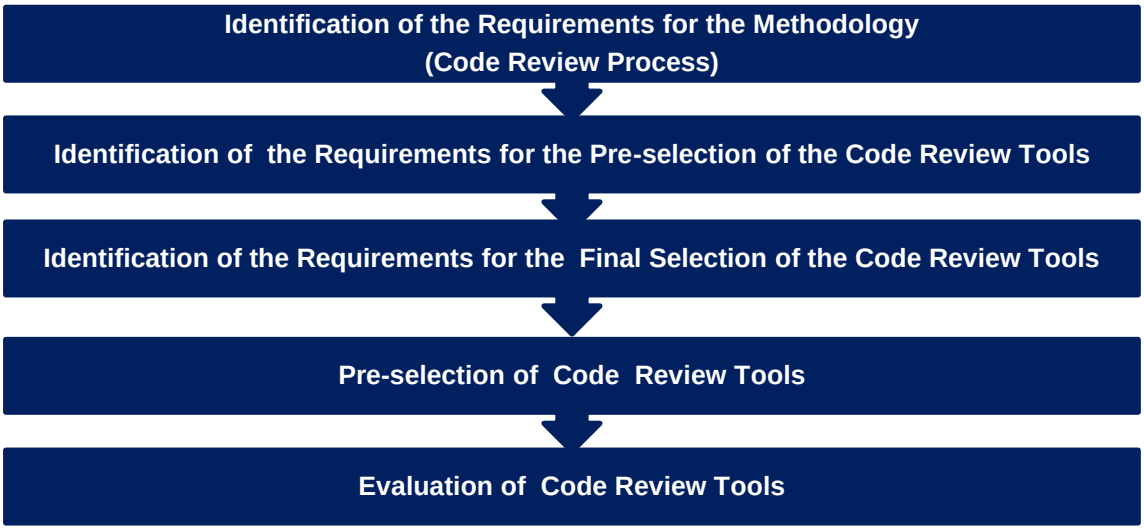
2 Methodological Approach to Building the Analysis

The goal of this task is to identify a set of requirements for the code review methodology and the code review tools that need to be followed inside the European Institutions. The evaluation of the code review tools that will support the code review process is initiated in this task and the results will be included in this deliverable.

The methodology that will be defined in TASK-08: Design of the Method for Performing the Code Reviews for the European Institutions, will fullfill all of the requirements defined in Section 3.

The approach used to execute this task has been divided in five main steps:

Figure 2. Methodological approach to building the analysis



In the following sections we explain each step in more detail.

2.1. Identification of the Requirements for the Code Review Methodology (Code Review Process)

For this step, we conducted the following activities:

- A research through forums, documents and articles from secure code-related organisations, such as OWASP, The MITRE Corporation, NIST and CERN Computer Security.
- An analysis of the inputs of WP1 to see if they may be transformed in requirements for the code review process.
- An analysis of the requirements from other clients (both private and public).

To conduct the analysis of the requirements, we developed and Excel sheet with all the information gathered, and analysed it to confirm its applicability for the European Institutions and the context of the projects that are included in the scope.

The applicable requirements were completed with a detailed description of the requirement, the measure, and the measure result.

2.2. Identification of the Requirements for the Pre-selection of the Code Review Tools

Part of the requirements identified in the previous step need to be supported by an automatic tool. Therefore, the selection of the automatic tools is critical for the code review method.

To evaluate the code review tools, and having as background the knowledge acquired during WP1, we defined a short list of requirements that the tools must fulfil, and how to measure the fulfilment of the requirements in a quantitative way.

These requirements are defined in Section 4.1. Only the tools that fulfil all the requirements are pre-selected.

2.3. Identification of the Requirements for the Selection of the Code Review Tools

The pre-selected tools need to fulfil these requirements in order to be chosen for the final evaluation and selection. The list of requirements is created based on:

- The best practices and controls that should be covered by the code review process aiming to a highly automated process;
- The specific issues learnt during the WP1;
- The results of different audits to different kind of applications, ensuring that code flaws are found if this tool is selected.

The applicable selection requirements are completed with a description, measure and measure result.

2.4. Pre-selection of Code Review Tools

In this step, we developed a list of potential code review tools, including:

- The tools that everis already know and have previously worked with.
- Additional code review tools developed by NIST, OWASP and CERN Computer Security

The pre-selection of the tools to be further evaluated was done by comparing the pre-selection requirements with each tool of the complete list of code review tools and by measuring each tool. The tools were ranked by the sum of the measurement of the selection requirements. We selected six tools for the final evaluation.

2.5. Evaluation of Code Review Tools

As a final step, we installed and tested the six selected tools, evaluating each tool against the requirements defined in Step 2.3.

The final objective is to present the results of the evaluation, and, together with DIGIT, agree on the tools that we will use for conducting the code review.

3 Requirements for Code Review Methodology

In this section we will detail the requirements of the methodology to conduct the code review, divided in two phases:

1) **Planning Phase**
2) **Code Review Phase**

The requirements are detailed in a template like the following:

Table 3-1 - Requirement template

REQ-01:
Description:
Measure: **Measure result:**

Where:

- **Requirement:** Name of the requirement.
- **Description:** A brief description about what the requirement entails.
- **Measure:** How the requirement is measured, in terms of the compliance with it
- **Measure result**: Compliance/Noncompliance

In the following sub-sections, we identify the requirements that the code review methodology must meet.

3.1. Requirements for the Planning Phase

The requirements for this phase are defined in this section, and include aspects such as the scope of the source code review, limitations, test plan, etc.

REQ-01:Each Code Review will Have a Sponsorship Inside DIGIT (Contact person)
Description: The contact person (sponsor) is responsible for all the logistic aspects to ensure the necessary information for the code review is ready and on time.
Measure: A person is appointed as Sponsor and he/she accepts the role. **Measure result**: Compliance/Noncompliance

REQ-02:The Code Review Methodology Needs to be Adaptable to the Defined Verification Levels
Description: Following OWASP documentation, three different levels of verification are defined: • Level 1 - Opportunistic • Level 2 - Standard • Level 3 – Advanced The depth of the analysis is defined in each level by a set of security verification requirements that must be addressed. The code review methodology has to be valid for the three levels.
Measure: The methodology clearly specifies how to cover level 1, level 2 or level 3 depending on the necessities of the code. **Measure result:** Compliance/Noncompliance
REQ-03:The Code Review Covers All Contemplated Languages
Description: The code review methodology must cover all the languages that have been included in the methodological approach. Currently, only JAVA and PHP are covered, but it is possible to add other languages in the future if it is deemed appropriate.
Measure: The tools selected will cover both JAVA and PHP. At the date of publication of the current version of this deliverable, the proposed controls fully cover all considered JAVA and PHP vulnerabilities. **Measure result:** Compliance/Noncompliance

REQ-04:The Methodology Covers All Applicable Internal Security Guidelines in Use in the European Institutions
Description: The methodology is consistent with any applicable Internal Security Guidelines of the European Institutions. Additionally, the methodology can be further developed in order to cover any new security guidelines or requirements that may be added to the European Institutions in the future.
Measure: The methodology takes into account the best practices detected in the following deliverables: 1. *Deliverable 3: Analysis of Software Development Methodologies Used in European Institutions* related to *TASK-01: Analysis of Software Development Methodologies Used in the European Institutions.* 2. *Deliverable 4 - Analysis of Software Development Methodologies Used in FOSS communities* related to *TASK-02: Analysis of Software Development Methodologies Used in the Open Source Software (FOSS) communities.* 3. *Deliverable 7: Comparative Study* related to *TASK-05: Comparison of the Results of task 1 and task 2 and Drafting of a Comparative Study* and can be evolved in case new best practices are adopted. **Measure result:** Compliance/Noncompliance
REQ-05:The Code Review Requires the Full Code to be Available
Description: To perform the code review, the source code needs to be available; therefore the project owner must ensure that it is provided on a timely manner.
Measure: The project owner must provide the source code to be reviewed prior to the code review. **Measure result**: Compliance/Noncompliance
REQ-06:The Methodology Covers Manual and Automated Code Review
Description: To cover the potential flaws within the code as much as possible, the methodology covers three cases: Managed (automatic tool), Defined (manual test to verify and expand the results of the tool) and Optimised (manual tests to evaluate specific scenarios). Each consecutive mode is incremental and complementary of the previous one. The architecture of the application is also covered, by means of analysing those controls related to it, that are included in each of the eight categories defined in Annex 1: Control Checklist of Deliverable 11: Design of the Method for Performing the Code Reviews for the European Institutions
Measure: The methodology must cover both manual and automated review. **Measure result**: Compliance/Noncompliance

3.2. Requirements for the Code Review Phase

The requirements for the Code Review phase are defined in this section, and include all the different aspects to be taken into account during the code review process. The requirements are divided in 9 categories, as follows:

3.2.1.Information Gathering

REQ-07:The Code Review Methodology Evaluates the Application Structure to Identify Applicable Optional Requirements
Description: Before commencing the security testing, understanding the structure of the application is paramount. Without a thorough understanding of the layout of the application, it is unlikely that it will be tested thoroughly.
Measure: The methodology must provide steps to investigate and understand the usage of the code identifying its attack surface. **Measure result:** Compliance/Noncompliance
REQ-08:The Code Review Methodology Analyses the Application Entry Points
Description: Entry points can be defined as the interfaces that allow external entities to contact with the application, sending information, requiring data, etc. The methodology will cover the process of reviewing existing entry points, to determine if they are known and defined, and check their security configuration.
Measure: There are controls defined to identify entry points, validate them and evaluate their security level. **Measure result:** Compliance/Noncompliance

REQ-09:The Code Review Methodology Analyses Web Services Identification and Fingerprinting
Description: Any services usually give out by default information about their versions, libraries used and other configuration characteristics. As this information can be used to create more detailed targeted attacks (as knowing the version can facilitate the discovery of known vulnerabilities), it is important to include controls in the methodology to check for this kind of information leakage. (It only applies on code reviews of applications that have online services, APIs, REST services or similar).
Measure: The methodology includes controls to check for default version of configuration information on the services published on the application, API or REST service. **Measure result:** Compliance/Noncompliance
REQ-10:The Code Review Methodology Examines Web Content Comments and Metadata Searching for Information Leakage
Description: Comments and metadata included into the code of a web application can potentially reveal internal information that should not be available to potential attackers (such as paths or configuration parameters). Comments and metadata review must be done in order to determine vulnerabilities related with information leakage.
Measure: The controls defined in the methodology must cover the analysis of comments and metadata within the code to avoid information leakage. **Measure result:** Compliance/Noncompliance

3.2.2.Authentication Management

REQ-11: The Code Review Methodology Verifies the Usage of a Proper Authentication Process
Description: The authentication process used to gain access to the application must be adequate and contemplate different security measures depending on the severity of the application and the data it manages.
Measure: The methodology must establish controls to determine if the authentication process is adequate to the severity level of the application and the data it will manage. **Measure result:** Compliance/Noncompliance

REQ-12: The Code Review Methodology Verifies the Protection of Credentials, both in Transit and At Rest
Description: The user credentials must be protected from third parties in order to guarantee their privacy. This means that credentials must be protected during the login transaction, communication, storage and exchange with other services by applying appropriate security measures, such as content encryption (TLS, AES, etc.). Hashing should be used only to protect the user passwords in order to ensure that they are not recoverable if stolen.
Measure: The methodology must carefully review each aspect of the authentication mechanisms to ensure that user's credentials are protected at all times, while they are at rest (e.g. on disk), and while they are in transit (e.g. during login). Review every available mechanism for changing a user's credentials to ensure that only an authorized user can change them. **Measure result:** Compliance/Noncompliance
REQ-13:The Code Review Methodology Evaluates the Password Policy In Use
Description: The use of a password policy is required to ensure that the user uses a secure, strong and hard-to-guess password to access the application. This includes features such as password length, complexity (use of symbols and numbers), and history.
Measure: The methodology must evaluate if a password policy is in use, and verify the security features being used. **Measure result:** Compliance/Noncompliance
REQ-14:The Code Review Methodology Analyses if Credentials Are Protected When Stored
Description: Credentials stored by the application must be protected in order to avoid unauthorized access to them. This is usually accomplished via encryption. User passwords, however, should never be stored; instead, their hash should be encrypted and stored.
Measure: The methodology must provide controls to verify the proper and secure storage of user credentials and passwords where applicable. **Measure result:** Compliance/Noncompliance

REQ-15: The Code Review Methodology Analyses Implicit Trust Between Components of the Application
Description: Avoid implicit trust between components of the application whenever possible. Each component should authenticate itself to any other component it is interacting with unless there is a strong reason not to do so (such as performance or lack of a usable mechanism). If trust relationships are required, strong procedural and architecture mechanisms should be in place ensuring that such trust cannot be abused as the site architecture evolves over time.
Measure: The code review methodology has to take care of the trust relationships between the components of the application. Controls to watch out and minimize the use of implicit trust between those components must be defined. **Measure result:** Compliance/Noncompliance
REQ-16:The Code Review Methodology Validates Existing Account Protection Measures
Description: The user account needs to be protected against unauthorized use. This includes features such as a limited number of login attempts, restricting connections from uncommon locations (require two-factor validation) and protection measures for the password recovery process. If a two-factor authentication is offered, the possible combinations that it allows will have to be reviewed too in order to ensure that they are appropriate.
Measure: The controls of the methodology must confirm the existence of proper mechanisms to manage and protect the authentication processes. **Measure result:** Compliance/Noncompliance

3.2.3.Authorization (Access Control)

REQ-17:The Code Review Methodology Evaluates the Access Control System
Description: Authorization is the process where requests to access a particular resource are granted or denied. Authorization includes the execution rules that determine what functionality and data the user (or Principal) may access, ensuring the proper allocation of access rights after authentication is successful. Various access control design methodologies are available, among others: • Role Based Access Control (RBAC) • Mandatory Access Control (MAC) • Discretionary Access Control (DAC) To choose the most appropriate one, the code review must identify threats and vulnerabilities specific to the application.
Measure: The code review methodology must provide controls to evaluate the proper access control. **Measure result:** Compliance/Noncompliance
REQ-18:The Code Review Methodology Analyses the Privilege Schema
Description: The principle of least privilege (POLP) requires that every module (such as a process, a user or a program, depending on the subject) must be able to access only the information and resources that are necessary for its legitimate purpose. Therefore, it is common to define a privilege schema that defines the required privileges per user profile, and which should be reviewed to ensure it is properly structured.
Measure: The methodology must have controls to analyse the privilege schema (if available). **Measure result:** Compliance/Noncompliance
REQ-19:The Methodology Evaluates Any Implemented Privilege Revision Controls
Description: If there are roles and/or different privileges assigned per user type (i.e. administrators, reviewers, users, visitors, etc.) then it is necessary to have in place controls to verify the privileges assigned to each one. The methodology will have to review these controls, their periodicity, their effectiveness and the way in which it notifies of any non-conformity identified.
Measure: The methodology must have controls to analyse the privileges assigned to users. **Measure result:** Compliance/Noncompliance

REQ-20:The Methodology Evaluates Trust Levels
Description: The trust levels represent the access rights that the application will grant to external entities. The trust levels are cross-referenced with the entry points and assets. This allows a team to define the access rights or privileges required at each entry point, and those required interacting with each asset.
Measure: The code review methodology provides controls to verify the existence of trust levels **Measure result:** Compliance/Noncompliance

3.2.4. Data/Input Validation of Data from all Untrusted Sources & Encoding

REQ-21: The Code Review Methodology Verifies All Input Entries in the Code
Description: The most common application security weakness is the failure to properly validate input from the user or environment. This weakness leads to almost all of the major vulnerabilities in applications, such as Interpreter Injection, locale/Unicode attacks, file system attacks and buffer overflows. Data from the user should never be trusted since the user has every possibility to tamper with the data. The code review must verify that the code validates all inputs filtering by data type, format, minimum/maximum lengths, etc. Also the code review process must analyse any SQL sentences used and carry out buffer overflow and Cross Site Scripting (XSS) validations in order to ensure that there are no vulnerabilities related to these concepts
Measure: The methodology must provide proper controls to ensure that the application is robust against all forms of input data, whether obtained from the user, infrastructure, external entities or database systems including Data canonicalization and proper Character Encoding. **Measure result:** Compliance/Noncompliance

REQ-22:The Code Review Methodology Analyses the XML Validation Against Schema
Description: Failure to enable validation when parsing XML gives an attacker the opportunity to supply malicious input. Most successful attacks begin with a violation of the programmer's assumptions. By accepting an XML document without validating it against a DTD or XML schema, the programmer leaves a door open for attackers to provide unexpected, unreasonable, or malicious input. It is not possible for an XML parser to validate all aspects of a document's content; a parser cannot understand the complete semantics of the data. However, a parser can do a complete and thorough job of checking the document's structure and therefore guarantee to the code that processes the document that the content is well-formed.
Measure: The methodology ensures that the code validates all XML input data against the applicable schema. Alternatively, if no schema is defined, any specific XML input validation controls will be reviewed. **Measure result:** Compliance/Noncompliance
REQ-23: The Code Review Methodology Verifies the Use of Whitelists/Blacklists for Incoming Data
Description: In order to protect the application entry points, it is possible to use whitelists or blacklists that allow the restriction of incoming information accepting or discarding it before it is even processed. This allows protecting the application against known threats (blacklists) or unknown/unexpected data (whitelists).
Measure: The methodology provides controls to check if one or both of these kinds of lists are used, and the process to implement them. **Measure result:** Compliance/Noncompliance

3.2.5.Cryptography

REQ-24:The Methodology Verifies the Cryptographic Libraries Used
Description: In order to implement cryptographic features in an application, it is common to make use of already existing libraries that implement them as they are already optimized and usually quite robust. The methodology will review any cryptographic libraries used and check their version and implementation on the application or code reviewed. Additionally, any non-standard implementation made of cryptographic libraries will also be noted and further evaluated.
Measure: The methodology provides controls to verify if proper cryptographic libraries are used **Measure result:** Compliance/Noncompliance
REQ-25:The Code Review Methodology Evaluates the Encryption Algorithms/Cyphers Used
Description: Aside from the cryptographic library used, another point to consider is the algorithms and cyphers that are selected to be used from it. It is important to carefully consider which cyphers and algorithms are supported in order to avoid the use of weak or obsolete ones (usually kept available in common cryptographic libraries for legacy support reasons).
Measure: The code review methodology has controls to evaluate if proper encryption algorithms and cyphers are used. **Measure result:** Compliance/Noncompliance
REQ-26:: **The Code Review Methodology Reviews the Security Parameters In Use**
Description: Another point that must always be reviewed is the security parameters applied to the cryptographic options selected. These include key length, generation entropy, salted hashes, etc.
Measure: The methodology must attend the following issues related to encryption keys: • Minimum encryption key length : The key length is at least 128 bits • Strong entropy is used for secure key generation. • Usage of salted hashes: Hash functions are used with salts against dictionary attack, versus a list of password hashes and against pre-computed rainbow table attacks **Measure result:** Compliance/Noncompliance

3.2.6.User and Session Management

REQ-27:The Code Review Methodology Analyses the Session Creation Process
Description: The process of validating the user login and creating a unique session for him/her is a critical factor to evaluate as it impacts directly on the security and privacy of the application. Therefore both the login process and session creation process must be reviewed. On the other hand, the user's session object may also hold authorization data and must also be evaluated.
Measure: The methodology has controls to validate the login and session creation process, including any session object created in these processes. **Measure result:** Compliance/Noncompliance
REQ-28: The Methodology Analyses the Session ID Management and its Protection
Description: The session ID must be robust, cryptographically secure, privately generated and not predictable. In addition sessions must be created and invalidated in a timely manner. A user should be assigned a new unique session once authenticated to mitigate session fixation attacks.
Measure: The code review methodology makes available controls to ensure the appropriate ID management and protection. **Measure result:** Compliance/Noncompliance
REQ-29: The Code Review Methodology Verifies the Proper Session Lifecycle
Description: Sessions created by the application must expire after reaching well defined timeouts: once created, the session must have limits in place to protect the user. The most common ones are inactivity timeout and a hard-limit timeout that is imposed regardless of the user activity.
Measure: The methodology provides controls to analyse the session lifecycle **Measure result:** Compliance/Noncompliance
REQ-30: The **Methodology Analyses the Implementation of the Logoff Process**
Description: As sessions eventually expire (after reaching a timeout or due to the user exiting the application), it is necessary to implement a secure and robust logout process that controls all the closed operations required.
Measure: The code review methodology has controls to ensure the usage of this mechanism. **Measure result:** Compliance/Noncompliance

3.2.7.Error and Exception Handling/Information Leakage

REQ-31:The Code Review Methodology Analyses the Error Handling Processes
Description: By default, applications tend to provide additional information regarding the causes of an error. While this information can be very useful to diagnose the reason behind the error, it can also be used by attackers to gain inside information about the workings of the application, and even use it to exploit it. On the other hand, these errors can sometimes also leak internal information about the server, application, or the data contained within.
Measure: The methodology must review the error handling processes to ensure that they do not disclose any sensitive or internal information. **Measure result:** Compliance/Noncompliance

3.2.8.Auditing and Logging

REQ-32:The Methodology Will Evaluate Log Configuration and Management
Description: The logs generated by the application must be protected and access to them restricted, as in most cases they can contain internal configurations, sensitive information or program error data. Furthermore, the log storage and exchange between the application and the logging system must also be protected to avoid information leaks.
Measure: The code review methodology provides controls to analyse the log configuration and the logging process (storage, exchange, compression, encryption…). **Measure result:** Compliance/Noncompliance

REQ-33:The Methodology Will Review the Events that are Generated by the Application
Description: The events generated by the application should be reviewed in order to ensure that they do not include sensitive or personal information; for example a user password or unique access token. Some events that could be generated and logged include: • input validation failures • authentication successes and failures • authorization failures • session management failures
Measure: The code review methodology provides controls to verify that the proper events are registered into the log system and that no personal/sensitive information is included on them. **Measure result:** Compliance/Noncompliance

3.2.9.Business Logic Testing

REQ-34:The Methodology Evaluates Specific Framework Requirements
Description: Usually, applications or code reviewed will make use of existing frameworks to implement advanced features (such as web interfaces, database management, etc.). These frameworks must be reviewed, making sure that up-to-date versions are used, and specific checks are done to evaluate their special functions or characteristics (will vary from one framework to another).
Measure: The code review methodology evaluates the requirements related to the specific framework of the code **Measure result:** Compliance/Noncompliance
REQ-35:The Code Review Methodology Evaluates the Security Configuration Parameters
Description: Security misconfiguration arises when the security of the application is compromised due to improperly configured parameters, or incorrect usage of default ones. Good security requires a secure configuration defined and deployed for the application, web server, database server, and platform. These setting are enabled and configured in static configuration files, commonly in XML format, but may also be expressed as annotations within the code.
Measure: The methodology provides controls to analyse the security configuration and to ensure the nonexistence of security misconfiguration **Measure result:** Compliance/Noncompliance

REQ-36:The Methodology Analyses the Usage of JavaScript
Description: JavaScript provides powerful features that are executed on the clients' side, avoiding server overhead and speeding up operations that would otherwise require a lot of bandwidth or intermediate resources. However, due to this fact, JavaScript is inherently dangerous if improperly coded, as it can cause security breaches on the clients' browser and computer, as well as having adverse effects on the application itself (for example if improper requests are sent). The most common issues include: • Code meets the JavaScript Style Guide • JavaScript code is placed into a separate JavaScript file • There is not JavaScript code in any server side code.
Measure: The methodology must evaluate the JavaScript scripts that are used by the application in order to make sure that they are robust and secure, and do not perform unsafe, unneeded or suspicious actions. **Measure result:** Compliance/Noncompliance

4 Requirements for Code Review Tools

In this section we will detail the requirements of the code review tools, divided in two phases:

1) **Pre-selection Requirements:** during this phase, a total of 9 aspects are defined, such as programming language flexibility, type of license, update frequency, etc., in order to narrow the list of tools that will be evaluated
2) **Selection Requirements for Code Review Tools:** during this phase, all the necessary requirements for selecting the most suitable tool are defined.

In the following sub-sections, we identify the requirements that the code review tools must meet.

4.1. Pre-selection Requirements

In order to optimise the tool analysis conducted in section 5.2, we need to conduct a pre-selection of the tools that will be evaluated against the requirements.

To accomplish this, we defined the minimum requirements that must be met by the tools in order to be selected for the final, more thorough evaluation.

Table 4-1 - Requirements for Pre-selection of the Tools for Code Review

No.	Requirement	Evaluation	Description
1	Security Analyser/Scanner	2 - Security scanner/analyser 1 - Code quality analyser with some/few secure code review capabilities 0 - Non secure code review capabilities. **A result of 1 excludes the tool**	If it does not check security the tool is excluded
2	Can review Java and/or PHP	2 - for Java 2 - for PHP 1 - Others **A result of 1 excludes tool**	The tool must be able to check at least one of these languages: JAVA and/or PHP Example: If the tool can review JAVA and others, but not PHP, it will get 3 points (2+1). If it covers Java, PHP and others, then it will get 5 (2+2+1).
3	Programming language flexibility	2 - Yes 0 - No	This requirement evaluates if the tool can cover other programming languages in the future (as stated in their websites or forums)

No.	Requirement	Evaluation	Description
4	Type of License	3 – Free Software/Open Source with commercial Support 1 - Proprietary	The FOSS license will be preferred over a commercial one
5	Update frequency	3 - Within six months 2 - Between six months and 1 year 0 - More than 1 year	It is very important to know how frequently rules/plugins/tools are updated.
6	Support available	3 – Yes, Commercial and non-commercial 2 – Yes, Commercial 1 – Yes, Non-commercial 0 - No support available	Support can come from forums, email contact or any other channels of communication
7	Deployment and maintenance	2 - for 1 h 0 - for more than 1h	The time it takes for the deployment, installation, configuration, and other processes to go live.
8	Reports	3 - The tool has its own reports 2 - The tool needs other tools to show the reports 0 - No reports	If the tool has the functionality of generating its own reports in a pdf, doc or similar format, or if the tools generates reports in a format such HTML, XML, CVS, etc. in which case another tool is required to generate the report, or no reporting capabilities.
9	Country	2 - European 0 - Other country	If the developer/s, or the company which they belong to, is/are from a country within the EU or not

4.2. Selection Requirements for Code Review Tools

In this section we develop a form for each of the requirements that we will evaluate in Section 5.2 Evaluation of Code Review tools.

4.2.1.Requirements for the Code Review Planning Phase

T-REQ-01:The Tool Covers the Appropriate Language/s.
Description: The code review tool must cover the appropriate language. It has to be functional for the language used to implement the source code.
Measure: The tool covers the language in which the source code that will be reviewed is implemented. **Measure result**: Compliance/Noncompliance/Not Applicable.
T-REQ-02: Availability of a User Manual for the Code Review Tool
Description: Existence of a User Manual for tool usage.
Measure: The tool has a User Manual Guide. **Measure result**: Compliance/Noncompliance/Not Applicable.
T-REQ-03:The Tool Should Be Able to Cover Other Programming Languages in the Future.
Description: The potential of code review tools to cover new languages through new features or plugins.
Measure: The tool has the capacity to evolve and cover new languages thanks to new features or plugins. **Measure result**: Compliance/Noncompliance/Not Applicable
T-REQ-04:The Tool Has Some Kind of Installation Guidelines
Description: The availability of an installation guide for the tool to make the installation process easier.
Measure: The tool has a step-by-step installation guide or something similar. **Measure result**: Compliance/Noncompliance/Not Applicable

T-REQ-05: The Tool is Easy to Install
Description: The complexity of the installation process, regardless of whether an installation guide exists or not
Measure: The process of installing the tool is easy to follow. **Measure result**: Compliance/Noncompliance/Not Applicable
T-REQ-06:The Tool can be Integrated into an IDE (Integrated Development Environment)
Description: If the tool can be used as a utility for an IDE (like eclipse or similar)
Measure: The tool can be used as a feature for an IDE **Measure result**: Compliance/Noncompliance/Not Applicable
T-REQ-07:The Current Version of the Tool was Released Within the Last Year
Description: The newer the version of the tool is, the more issues it should detect
Measure: The current version of the tool is not older than a year. **Measure result**: Compliance/Noncompliance/Not Applicable

4.2.2.Requirements for the Code Review Phase

4.2.2.1.Information Gathering

T-REQ-08: The Tool Can Review Webpage Comments and Metadata for Information Leakages
Description: Comments and metadata included into the code of a web application can potentially reveal internal information that should not be available to potential attackers (such as paths or configuration parameters).
Measure: The tool is able to identify comments and metadata in order to detect possible information leakages. **Measure result**: Compliance/Noncompliance/Not Applicable

4.2.2.2.Authentication Management

T-REQ-09: The Tool Checks the Strength of the Password Requested to the User
Description: The use of a password policy is required to ensure that the user uses a secure, strong and hard-to-guess password to access the application. This includes features such as password length, complexity (use of symbols and numbers), and history.
Measure: The tool can check the strength of the passwords. **Measure result**: Compliance/Noncompliance/Not Applicable
T-REQ-10:The Tool Verifies that There are No Plain Text Passwords within The Code
Description: Plain text passwords lead to unauthorised access
Measure: The tool can detect if passwords are in plain text **Measure result**: Compliance/Noncompliance/Not Applicable
T-REQ-11:The Tool Verifies that Passwords are Encrypted Prior to Be Stored
Description: Password must be encrypted before their storage to prevent other individuals from accessing them.
Measure: The tool can check if passwords are protected before they are stored. **Measure result**: Compliance/Noncompliance/Not Applicable

4.2.2.3. Data/Input Validation of Data from all Untrusted Sources & Encoding

T-REQ-12:The Tool Verifies that the Format of the Input is Validated
Description: Accurate input data validation must be performed to avoid application crashes that can lead to the exploitation of possible vulnerabilities. For example when we expect digits as an input.
Measure: The tool can check if the format of the inputs is validated **Measure result**: Compliance/Noncompliance/Not Applicable

T-REQ-13:The Tool Verifies that SQL Injection Validations are Implemented
Description: Any SQL sentences used must be analysed in order to ensure that there are no vulnerabilities related to SQL Injections
Measure: The tool can detect possible SQL Injection vulnerabilities. **Measure result**: Compliance/Noncompliance/Not Applicable
T-REQ-14:The Tool Verifies that Cross Site Scripting (XSS) Validations are Implemented
Description: Cross Site Scripting (XSS) validations have to be carried out in order to ensure that there are no vulnerabilities related to these concepts.
Measure: The tool can detect possible Cross Site Scripting (XSS) vulnerabilities. **Measure result**: Compliance/Noncompliance/Not Applicable
T-REQ-15: The Tool Verifies that Buffer Overflow Validations are Implemented
Description: Buffer overflow validations have to be carried out in order to ensure that there are no vulnerabilities related to these concepts.
Measure: The tool can detect possible Cross Site Scripting (XSS) vulnerabilities. **Measure result**: Compliance/Noncompliance/Not Applicable
T-REQ-16:The Tool Verifies the Implementation of a XML Validation Against Schema
Description: Failure to enable validation when parsing XML gives an attacker the opportunity to supply malicious input. Most successful attacks begin with a violation of the programmer's assumptions. By accepting an XML document without validating it against a DTD or XML schema, the programmer leaves a door open for attackers to provide unexpected, unreasonable, or malicious input. It is not possible for an XML parser to validate all aspects of a document's content; a parser cannot understand the complete semantics of the data. However, a parser can do a complete and thorough job of checking the document's structure and make sure it is well formed.
Measure: The tool ensures that the code validates all XML input data against the applicable schema. Alternatively, if no schema is defined, all specific XML input validation controls will be reviewed. **Measure result**: Compliance/Noncompliance/Not Applicable

T-REQ-17:The Tool Verifies the Use of Whitelists and/or Blacklist for Specific Incoming Data
Description: In order to protect the application entry points, it is possible to use whitelists or blacklists that allow the restriction of incoming information accepting or discarding it before it is even processed. This allows protecting the application against known threats (blacklists) or unknown/unexpected data (whitelists).
Measure: The tool can check if one or both of these kinds of lists are used, and the process to implement them. **Measure result**: Compliance/Noncompliance/Not Applicable

4.2.2.4.Cryptography

T-REQ-18: The Tool Identifies the Type of Cryptographic Libraries Used
Description: In order to implement cryptographic features to an application, it is common to make use of already existing libraries that implement them, since they are already optimised and usually quite robust. The methodology will review any cryptographic libraries used and check their version and implementation in the application or code reviewed. Additionally, any non-standard implementation made of cryptographic libraries will also be noted and further evaluated.
Measure: The tool can verify if proper cryptographic libraries are used **Measure result**: Compliance/Noncompliance/Not Applicable
T-REQ-19:The Tool Identifies the Usage of Salted Hashes
Description: Hash functions are used with salts against dictionary attacks, against a list of password hashes and against pre-computed rainbow table attacks.
Measure: The tool can verify the usage of salted hashes. **Measure result**: Compliance/Noncompliance/Not Applicable

4.2.2.5.Error and Exception Handling/Information Leakage

T-REQ-20: The Tool Identifies Exception Handling Processes
Description: By default, applications tend to provide additional information regarding the causes of an error. While this information can be very useful to diagnose the reason behind the error, it can also be used by attackers to gain inside information about the workings of the application, and even use it to exploit it. On the other hand, these errors can sometimes also leak internal information about the server, application, or the data contained within it.
Measure: The tool ensures that the error handling process is conducted properly. **Measure result**: Compliance/Noncompliance/Not Applicable

4.3. Requirements for the Results of Code Review Phase

4.3.1. Requirements for Mandatory Features

T-REQ-21: The Tool Offers Weaknesses Reporting
Description: The main objective of using a code review tool is to provide a report of possible weaknesses and flaws that can guide the tester through a further manual code review
Measure: The tool provides a report of the weaknesses detected **Measure result**: Compliance/Noncompliance/Not Applicable
T-REQ-22:The Tool Provides Weaknesses Locations within The Code
Description: The tool must help to find where the detected vulnerability is inside the code, providing a direct link, or at least the information about the line of code and the file that contains the issue.
Measure: The tool points to the location of the issue within the code for each vulnerability. **Measure result**: Compliance/Noncompliance/Not Applicable
T-REQ-23: The Tool Generates a Low Rate of False Positives
Description: Is important to know if the tool conducts an accurate detection of vulnerabilities.
Measure: The tool provides a low rate of false positives concerning vulnerabilities. **Measure result**: Compliance/Noncompliance/Not Applicable

4.3.2. Requirements for Optional Features

T-REQ-24: The Tool Provides Reports in XML Format
Description: If the results of the code review done with the tool can be useful for further reports.
Measure: The tool can generate reports in XML format **Measure result**: Compliance/Noncompliance/Not Applicable

T-REQ-25:The Tool Allows the Application of Filters to Weaknesses Reports
Description: The functionality of filtering the results of the code review can help to check specific types of vulnerabilities.
Measure: The tool permits to filter the results of the code review. **Measure result**: Compliance/Noncompliance/Not Applicable
T-REQ-26:The Tool Results Meet CWE Identification
Description: [1]The Common Weakness Enumeration (CWE) is a formal list of software weakness types created to: • Serve as a common language for describing software security weaknesses in architecture, design, or code. • Serve as a standard measuring stick for software security tools targeting these weaknesses. • Provide a common baseline standard for weakness identification, mitigation, and prevention efforts.
Measure: The tool uses the CWE identification within its results to categorise the weaknesses detected **Measure result**: Compliance/Noncompliance/Not Applicable

5 Pre-selection and Evaluation of Code Review Tools

5.1. Pre-selection of Tools

5.1.1.Tools to Evaluate

Table 5-1 lists the code review tools that we will evaluate and pre-select in Section 5.1.2

Table 5-1 Tools to be preselected

No.	Tool	Focus	Description
1.	Brakeman	Security Scanner Ruby on rails	It is an open source vulnerability scanner specifically designed for Ruby on Rails applications.
2.	Codesake Dawn	Security Scanner Ruby on rails	It is a security source code scanner for ruby powered code
3.	Find Bugs	Security Java Quality	It is a scanning tool that evaluates Java static code. One of the most recognised tools in the Java open source field. It executes security and code quality scans.
4.	Flawfinder	C/C++	It is a simple program that examines C/C++ source code and reports possible security weaknesses (“flaws”) sorted by risk level.
5.	FxCop	Security .Net	It is a scanning tool that evaluates .NET code. It is a free tool, but not open source.
6.	OWASP SWAAT Project		It is an open source web application source code analysis tool. SWAAT searches through source code and analyses against the database of potentially dangerous strings given in the .xml files. Thus it does NOT positively identify the existence of a vulnerability.
7.	PMD	Java	It is a static Java source code analyser. It uses rule-sets to define when a piece of source is erroneous. PMD includes a set of built-in rules and supports the ability to write custom rules.
8.	Rips-scanner (RIPS)	Security Scan PHP	It is a scanning tool to evaluate PHP static code. One of the most recognised tools in the PHP open source field. It executes security and code quality scans.
9.	SonarQube	Multiplatform Multilanguage Quality	It is an open source platform for continuous inspection of code quality. It executes security and code quality scans.

No.	Tool	Focus	Description
10.	VCG	Multilanguage	It scans C/C++, Java, C# and PL/SQL for security issues and for comments which may indicate defective code
11.	Yasca	Multiplatform security Scan	It is a scanning tool that evaluates multiplatform static code. It analyses the following languages: Java, PHP, .Net, Perl, C, C++, HTML; Java Script, Cobol, ColdFusion and VB.

5.1.2.Evaluation and Pre-selection of Tools

After evaluating the tools listed in Section 5.1.1 – Pre-selection of Tools, against the requirements of Table 5-1, we ranked and selected the tools. This activity was conducted in two steps:

Step 1. We evaluated Requirement 1 and Requirement 2 to further refine the list of selected tools.

Requirement 1: The tool must be a Static Application Security Testing (SAST). We assign the following values to the specific cases, for ranking purposes:

- **Case 1:** Security scanner/analyser = 2
- **Case 2:** Code quality analyser with some/few secure code review capabilities = 1
- **Case 3:** Without secure code review capabilities = 0. ***In this case, the tool is automatically excluded from the ranking***

Requirement 2: The tool must be capable of reviewing Java and/or PHP code:

- **Case 1:** Can review JAVA AND PHP AND Other Languages = 5
- **Case 2:** Can review JAVA AND PHP BUT NOT Other Languages = 4
- **Case 3:** Can review JAVA OR PHP AND Other Languages = 3
- **Case 4:** Can review Other Languages BUT NOT JAVA OR PHP= 1. ***In this case, the tool is automatically excluded from the ranking***

After applying Requirement 1–Case 3 and Requirement 2–Case 4, Table 5-2 shows the resulting list of tools that will not be evaluated in Step 2.

Table 5-2 Tools excluded from the ranking

No,	Tool	Reason for exclusion	Description
1.	Brakeman	Requirement 2 – Case 4	Only covers Ruby code
2.	Codesake Dawn	Requirement 2 – Case 4	Only covers Ruby code
3.	Flawfinder	Requirement 2 – Case 4	Only covers C/C++ code
4.	FxCop	Requirement 2 – Case 4	Only covers .NET code.
5.	OWASP SWAAT Project	Requirement 1 – Case 3	The tool is not exactly a security code analyser. Currently it searches the source code and the database for potentially dangerous strings in the .xml files. Thus it does NOT positively identify the existence of a vulnerability - this generally requires the application of contextual knowledge

In addition, Table 5-3 shows one tool that is excluded from the pre-selection for a different reason, even though it is evaluated in Step 2.

Table 5-3 other tools excluded from the ranking

No,	Tool	Reason for exclusion	Description
6.	Rips-scanner	DEVELOPMENT ABANDONED	According to its webpage (http://rips-scanner.sourceforge.net/) the development has been abandoned temporarily.

Step 2. We evaluated the remaining tools against the requirements of Table 5-1. The results are shown on Table 5-4

Table 5-4 Selection requirements

SELECTION REQUIREMENTS

No.	Requirement	Evaluation	Explanation	Range Explanation	BrakeMan	Codesake Dawn	Find Bugs	Flawfinder	PxCop	OWASP SWAAT Project	PMD	Rips-scanner	SonarQube	VCG	Yasca
1	Security Analyzer	2 - Security scanner/analyzer 1 - Code quality analyzer with some/few secure code review capabilities 0 - Non secure code review capabilities. Non compliance excludes tool	Other main requirement. If does not check security the tools is unworthy		2	2	2	2	2	0	2	2	1	2	2
2	Can review Java and/or PHP	2 for Java 2 for PHP 1 Others A result of 1 at this check excludes the tool	The main requirement. The tool shall be able to check at least one of these languages.	Example: If the tools can review Java an other out of PHP will get 3 points (2+1). If covers Java, PHP and others then will get 5	1	1	2	1	1	5	3	2	5	5	5
3	Language programming flexibility	2 if yes 0 if not	If the tool can cover other programming languages in the		0	0	0	0	0	2	0	0	0	0	0
4	Type of License	3 - FOSS / OSS 1 - COMMERCIAL	FOSS OSS COMMERCIAL		3	3	3	3	1	3	3	3	3	3	3
5	Update frequency	How frequently are rules/plugins/tools update is a very important point 3 - Within six months 2 - between six months and 1 year 0 - More than 1 year			3	3	2	2	2	0	3	0	3	0	3
6	Support available	3 - Yes, commercial and non-commercial 2 - Yes, commercial 1 - Yes, non-commercial 0 - No			3	1	1	0	2	1	0	1	2	1	2
7	Deploy and maintenance	2 - 1 h 0 - + de 1h			2	2	2	2	2	2	2	2	2	2	2
8	Reports	3 - The tool has its own reports 2 - The tool need other tool to show the reports 0 - No reports			3	3	2	3	3	3	2	2	2	2	2
9	Country	2-European 0- Other country (only relevant for FOSS and OSS Tools)			2	2	0	0	0	2	2	2	2	2	0
10	TOTAL SCORE				19	17	14	13	13	18	17	14	20	17	19

5.1.3.Results of the Pre-selection

The evaluation resulted in the following ranking of the tools:

No	Tool	Developed by	Country	Score
1.	SonarQube	SonarSource (http://www.sonarsource.com/) (FOSS Tool)	Switzerland	20
2.	Yasca	Michael Scovetta (https://www.linkedin.com/in/scovetta) (https://twitter.com/scovetta)	USA	19
3.	PMD	Andreas Dangel and Romain Pelisse	Germany	17
4.	VCG	n1ckdunn (https://twitter.com/n1ckdunn)	UK	17
5.	RIPS	Johannes Dahse (https://twitter.com/fluxreiners)	Germany	14

No	Tool	Developed by	Country	Score
6.	Find Bugs	University Of Maryland The active members of the FindBugs development team in early 2015 were as follows: • Bill Pugh (project lead and primary developer) • Andrey Loskutov (Eclipse plugin) • Keith Lea (web cloud))	USA	14

5.2. Evaluation of Code Review Tools

For this step we have conducted code review tests for each tool. A summary of the evaluation results for each tool is presented in the next sections

5.2.1.FindBugs

FindBugs is a code quality review tool that can also find some security vulnerabilities.

The flaws detected and labelled as “Security” or “Malicious Code Vulnerability” in the results are interesting for our study. Another useful issue is the BadUseOfReturnValue detector feature.

The configuration can be changed via filters (Preferences --> Filters) to hide those bugs or flaws that are not relevant for the code review.

The installation process is very simple except if you do not have JDK version 1.6 or later and Ant already installed. In that case you have to install these utilities and set them up prior to the installation of FindBugs itself. After that you just need to download, unzip and run the tool.

This analysis has been conducted in a 64-bit Windows 7 environment with JDK 1.8 and ant 1.9.6.

Once the tool is running, and before starting the review, it asks for the location of the bytecode files, the compiled files of the source code. The tool accepts files such as jar, war, etc., and also the path to the directory that gathers the compiled files.

Furthermore it is imperative to indicate the libraries that the code needs in order to be compiled, to cover the dependencies of the source code.

Finally it has an optional, but useful, requirement. You can provide the path to the source code itself, the non-compiled code. It helps to browse the code that supports the possible bug.

Once you have provided the bytecode files to review, the classes that cover the dependencies, etc., you run the analysis.

The results of the code review are quite hard to follow since its nomenclature for the vulnerabilities does not meet any known standard like CWE. In fact it has its own way to categorise the flaws detected.

Regarding security-related issues, besides the code quality, there is also those categorised under the Security tag and the Malicious Code Vulnerability tag. It seems that it does not clearly detect XSS vulnerabilities, but it does detect possible SQL vulnerabilities. It does not detect the strength of the password, but it detects hardcoded passwords within the code.

As we previously mentioned, if you provide the path to the source code from which the bytecode files come, the tool points to the specific line of code where the vulnerability has been found, which is useful to identify why, or why not, the code is unsecure.

5.2.2.PMD

PMD is a source code analyser oriented to reviewing the quality of the code. It finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, etc. It supports the following languages: JAVA, JavaScript, XML, and XSL, but does not focus on security issues. It needs JRE 1.6 OR higher to run

The tool offers the possibility of writing custom rulesets to scan the code and find the issues declared in those rulesets.

Despite that this tools is a great quality code review tool its performance regarding security issues is not enough to cover the needs in this study, clearly oriented to secure code reviews.

5.2.3.RIPS

Rips is a static source code analyser for vulnerabilities in PHP scripts. The tool is written in PHP. By tokenising and parsing all source code files, RIPS is able to transform PHP source code into a program model and to detect sensitive sinks (potentially vulnerable functions) that can be tainted by user input (influenced by a malicious user) during the program flow. Besides the structured output of found vulnerabilities, RIPS also offers an integrated code audit framework for further manual analyses.

Types of Security Vulnerabilities that are covered:

- Server-Side
  - Code Execution
  - Command Execution
  - File Disclosure
  - File Inclusion
  - File Manipulation
  - LDAP Injection
  - PHP Object Injection
  - Protocol Injection
  - Reflection Injection
  - SQL Injection
  - XPath Injection

    - Other
- Client-Side
    - Cross-Side Scripting
    - HTTP Response Splitting
    - Session Fixation

RIPS is managed via web. It allows scanning the source code (local directory) for the above-mentioned vulnerability types, or performing regular expression searches. Desirable vulnerabilities can be selected, as well as scan levels, among other options. In order to use the RIPS tool, we used XAMPP as a local webserver in a 64-bit Windows 7 environment.

The vulnerabilities detected are presented on web dashboard together with an extract of the affected code lines and a vulnerability summary.

For each vulnerability, an integrated code viewer is offered in order to highlight the affected code lines in the original source code.

Furthermore, there is a help option, the 'get help' button on the left corner of the vulnerability that provides additional information to understand the vulnerability. Exploits will be automatically generated, when possible.

The interface also offers a list of scanned PHP files, user-defined functions, and detected sources.

5.2.4.SonarQube

SonarQube is an Open Source platform for managing code quality. Its main strength stems from the fact that its functionality can be extended by using plugins, commercial or Open Source, to cover a wide range of programming languages, such as JAVA, PHP, C, C++, etc., as well as managing the detection of security flaws and vulnerabilities on top of the quality bugs within the code. We can find plugins associated to OWASP Top Ten, FindBugs, etc.

The installation of the tool can be tedious, due to the lack of documentation for this process. A guide of installation has been developed during this analysis for possible future users, with step-by-step details. The installation and tests were conducted in Ubuntu Mate VM running on VirtualBox.

By default the tool does not provide much functionality, out of the web app, beyond the loading of a previous code analysis. Thus it is necessary to install diverse plugins to complement its usage.

The tool provides a lot of utilities, like a wide tagging of vulnerabilities, depending on the rules enabled for the review, which allow, for example, filtering the results by vulnerability type.

With regard to the reporting of code review results, we have detected one under-commercial license which allows reporting the results in XML format. There is another plugin that allows generating the reports in HTML format but it does not seem to work correctly.

The results of the PHP code review do not seem to be good enough. The tool does not report as many vulnerabilities as expected, out of a set of more than 200,000 lines of PHP code that belong to one source code.

As for the JAVA code review, the results of the tool look much better in terms of security-related issues, even when using the FindBugs plugin. In the latter case the results about security vulnerability detection are more satisfactory, yet below our expectations.

It is worth noticing that as the development of plugins improves, the better the results of the code reviews will be, in terms of security. However, at this time, the tool is not efficient enough.

With regard to the performance, it is similar for both languages.

In conclusion, we can say that SonarQube is a good code quality-oriented review tool, but it will take time for it to reach the level expected for a secure code review tool. Nevertheless, it is worth not losing sight of this tool.

5.2.5.VCG

VCG, which stands for Visual Code Grepper, is a multilingual tool that contributes to the code review.
The installation process is the easiest of all the tools in this study. The installation was conducted in a 64-bit Windows 7 environment.

For this analysis we used four projects, two for the JAVA review and two for the PHP review. The results from both types of reviews are noteworthy, especially for PHP code reviews, because the tool detects more vulnerabilities than the other tools in the study, but it can also lead to a higher ratio of false positives. This is not a serious problem because it can help us know what and where we have to check within the code to identify the security problems.

Once the automated code review is finished, the tool provides within its GUI a report of the issues and vulnerabilities detected.

The tool allows reviewing the code associated with the flaw, with an external text editor and checking the line of code where the issue was found.

For each problem found, the tool provides a brief description of the consequences of the vulnerability. Also the tool allows filtering the results by problem type. The issues are classified by severity (Critical, High, Medium, Standard, Low, and Suspicious)

The reports can be stored in two formats: XML and CSV. This allows to further check the code reviews, even creating portable code review results that can be analysed from other computers.

The tool does not allow the installation of plugins, but these do not seem necessary. The tool is very useful by itself.

The scan conducted in our test was ‘Full Scan’. There are more types of scans, but the full one is a thorough scan that encompasses all the others.

5.2.6.Yasca

Yasca is a command-line code review tool developed by Michael Scovetta.

Its functionality can be increased by using their own plugins or third-party plugins like PHPLint, Findbugs, PMD, etc.

Among the languages that the tool can review are JAVA and PHP.

Both processes, installation and code review, have to be done by command line.

This analysis has been conducted in a 64-bit Windows 7 environment. To avoid further 'Memory Exhaustion' problems we have changed the value assigned to the *memory_limit* parameter to 1 GB, since by default it is set to 256Mb, but this is the minimum capacity required for the performance of the tool, leaving no memory for other tasks.

During the code review there is no information as to what the tool is doing, or the on-going stage of the process.

The tool seems to cover a lot of the requirements considered worthy for our study.

With regard to the reporting of code review results, the tool generates a file in HTML format, which is stored in a folder located in the Desktop of the machine running the code review. This file has all the information about the results of the code review.

The report shows all the issues and flaws detected within the code, as well as features, in the shape of buttons, showing the specific line of code where the vulnerability is, a detailed explanation of the vulnerability, and a functionality to filter the flaws.

On the other hand the classification of the issues in the report is not very intuitive, sometimes even not easy to follow. It might be better to use some kind of categorisation, making it easier to search for specific problems when you are checking the report.

5.3. Conclusions

After the analysis of the six tools, we can recommend the use of the following tools:

1. **For Java projects:** FindBugs
2. **For PHP projects**: RIPS
3. **For Java and PHP:** VCG
4. **For Java and PHP:** YASCA

All the tools within the scope of this study are more or less efficient. SonarQube has a lot of potential as well, since its plugins are constantly being improved. PMD does not seem to be very valuable for secure code reviews, however it is a great tool for quality code review.

The complete results of the analysis can be found by clicking on the icon:

Deliverable 9: List of requirements for code reviews

Requirements for code review tools

6 Bibliographical References

[1] MITRE, “Common Weakness Enumeration - About CWE,” The MITRE Corporation, 11 July 2014. [Online]. Available: https://cwe.mitre.org/about/). [Accessed April 2016].

[2] OWASP, “OWASP Application Security Verification Standard Project,” 07 March 2016. [Online]. Available: https://www.owasp.org/images/6/67/OWASPApplicationSecurityVerificationStandard3.0.pdf. [Accessed April 2016].

[3] CERN Computer Security, “Static Code Analysis Tools,” CERN Computer Security, [Online]. Available: https://security.web.cern.ch/security/recommendations/en/code_tools.shtml. [Accessed April 2016].

[4] OWASP, “Testing Guide Introduction,” OWASP, 26 January 2015. [Online]. Available: https://www.owasp.org/index.php/Testing_Guide_Introduction. [Accessed April 2016].

[5] OWASP, “File:OWASP AlphaRelease CodeReviewGuide2.0.pdf,” 29 February 2016. [Online]. Available: https://www.owasp.org/index.php/File:OWASP_AlphaRelease_CodeReviewGuide2.0.pdf. [Accessed April 2016].

[6] NIST, “Source Code Security Analyzers,” National Institute of Standards and Technology (NIST), [Online]. Available: https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html. [Accessed April 2016].

[7] NIST, “Source Code Security Analysis,” [Online]. Available: https://samate.nist.gov/docs/source_code_security_analysis_spec_SP500-268_v1.1.pdf. [Accessed April 2016].

[8] MITRE, “Using Automated Static Analysis Tools for Code Reviews,” The MITRE Corporation, 20 August 2013. [Online]. Available: https://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/using-automated-static-analysis-tools-for. [Accessed April 2016].

[9] MITRE, “The Importance of Manual Secure Code Review,” The MITRE Corporation, 16 January 2016. [Online]. Available: https://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/the-importance-of-manual-secure-code-review. [Accessed April 2016].

[10] S. Moiras, “Source Code Review Using Static Analysis Tools (CERN openlab Summer Student Report 2015),” July-August 2015. [Online]. Available: https://zenodo.org/record/31938/files/SummerStudentReport-StavrosMoiras.pdf. [Accessed April 2016].

7 Annexes

7.1. Description of the OWASP Verification Levels

A description of the OWASP Verification Levels, from the OWASP Application Security Verification Standard [2], is provided below:

Level 1: Opportunistic

An application achieves ASVS Level 1 (or Opportunistic) if it adequately defends against application security vulnerabilities that are easy to discover, and is included in the OWASP Top 10 and other similar checklists.

Level 1 is typically appropriate for applications where low confidence in the correct use of security controls is required, or to provide a quick analysis of a fleet of enterprise applications, or assisting in developing a prioritized list of security requirements as part of a multi-phase effort. Level 1 controls can be ensured either automatically by tools or simply manually without access to source code. We consider Level 1 the minimum required for all applications.

Threats to the application will most likely be from attackers who are using simple and low effort techniques to identify easy-to-find and easy-to-exploit vulnerabilities. This is in contrast to a determined attacker who will spend focused energy to specifically target the application. If data processed by your application has high value, you would rarely want to stop at a Level 1 review.

Level 2: Standard

An application achieves ASVS Level 2 (or Standard) if it adequately defends against most of the risks associated with software today.

Level 2 ensures that security controls are in place, effective, and used within the application. Level 2 is typically appropriate for applications that handle significant business-to-business transactions, including those that process healthcare information, implement business-critical or sensitive functions, or process other sensitive assets.

Threats to Level 2 applications will typically be skilled and motivated attackers focusing on specific targets using tools and techniques that are highly practiced and effective at discovering and exploiting weaknesses within applications.

Level 3: Advanced

ASVS Level 3 is the highest level of verification within the ASVS. This level is typically reserved for applications that require significant levels of security verification, such as those that may be found within areas of military, health and safety, critical infrastructure, etc. Organisations may require ASVS Level 3 for applications that perform critical functions, where failure could significantly impact the organization's operations, and even its survivability.

Deliverable 9: List of requirements for code reviews

> […]
>
> An application at ASVS Level 3 requires more in depth analysis, architecture, coding, and testing than all the other levels. A secure application is modularized in a meaningful way (to facilitate e.g. resiliency, scalability, and most of all, layers of security), and each module (separated by network connection and/or physical instance) takes care of its own security responsibilities (defence in depth), that need to be properly documented. Responsibilities include controls for ensuring confidentiality (e.g. encryption), integrity (e.g. transactions, input validation), availability (e.g. handling load gracefully), authentication (including between systems), non-repudiation, authorization, and auditing (logging).

