COMPUTERS & SECURITY 111 (2021) 102470

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

On the adoption of static analysis for software security assessment–A case study of an open-source e-government project

Anh Nguyen-Duc[a,*], Manh Viet Do[b], Quan Luong Hong[b], Kiem Nguyen Khac[c], Anh Nguyen Quang[d]

[a] University of South Eastern Norway, Norway
[b] MQ ICT SOLUTIONS, Viet Nam
[c] Hanoi University of Science and Technology, Viet Nam
[d] University of Transport and Communications, Viet Nam

A R T I C L E   I N F O

Article history:
Received 13 January 2021
Revised 18 June 2021
Accepted 9 September 2021
Available online 20 September 2021

Keyword:
Security testing
Software vulnerability
SAST
Case studies
Experiments
Combined SAST tools
empirical study
e-government

A B S T R A C T

Static Application Security Testing (SAST) is a popular quality assurance technique in software engineering. However, integrating SAST tools into industry-level product development for security assessment poses various technical and managerial challenges. In this work, we reported results from a case study of adopting SAST as a part of a human-driven security assessment process in an open-source e-government project. We described how SASTs are selected, evaluated, and combined into a novel approach and adopted by security experts for software security assessment. The approach was preliminarily evaluated using semi-structured interviews. Our results show that while some SAST tools out-perform others, it is possible to achieve better performance by combining more than one SAST tools. The combined approach has the potential to aid the security assessment process for open-source software.

1. Introduction

Software vulnerabilities continue to be the major security threats to software-intensive systems. Vulnerability is defined as a quality attribute of a software system that could be accidentally triggered or intentionally exploited and result in a security failure (Source code security analysis tool functional specification version 1.0 2007). One of the vulnerability discovery techniques is a static analysis of source code, which investigate the written source code and hence can be applied in the life cycle, does not require the system to be executable. Static application security testing tools (SATs) play an important role to ensure the product meets the quality requirements (Chess and McGraw, 2004). Tools for static analysis have rapidly evolved in the last decade, from a simple lexical analysis to a set of much more comprehensive and complex analysis techniques. Various SAST tools have been proposed to facilitate the automatic detection of vulnerabilities in continu-

* Corresponding author.
E-mail addresses: angu@usn.no (A. Nguyen-Duc), dovietmanh@mqsolutions.vn (M.V. Do), luong.hong.quan@mqsolutions.vn (Q. Luong Hong), kiem.nguyenkhac@hust.edu.vn (K. Nguyen Khac), anhnq@utc.edu.vn (A. Nguyen Quang).

https://doi.org/10.1016/j.cose.2021.102470

ous software development. Not all tools are the same, so developers need to decide how they can benefit from using the right set of tools for their projects.

While SAST has been commercialized becomes a common element in software development environment, they are also known to generate misleading (or irrelevant) warnings (Aloraini et al., 2019; Pashchenko, 2017). Given this limitation, the selection of an appropriate tool based on empirical grounds becomes extremely important from an industrial perspective. The combination of different SAST tools or the combination of these tools with other types of security testing techniques could help to increase its applicability in real-world projects. Yet, we only find a few comparative empirical studies (Okun et al., 2013; Aloraini et al., 2019; Pashchenko, 2017; Baca et al., 2013; Hofer, 2010; Okun et al., 2012). Most of these studies base on the test data and do not reflect the adoption of this type of tools in a real-world context.In the scope of our research, we are interested in practical aspects of SAST applications: (1) which type of security issues can actually be detected by SASTs, (2) what is the practical use of multiple tools to achieve the best performance. This knowledge will not only help practitioners to take informed decisions when selecting the tool(s) to use, but also tool providers to enhance them and improve the level of support provided to developers.

We investiaged the SASTs in a concrete case study about the developmentof an open-source solution for secured software repositories in an e-government system. The repository consists of open-source software to be used in all computers in the e-government system. Vulnerability assessment is an important element of the secured software repository, ensure that all software will go through a security analysis before going further to end-users. In this project, SAST tools are experimented with and adopted to assist vulnerability assessment of open source software.

This paper reports our experience with adopting SAST tools in an e-government project through a research-driven process. We proposed three Research Questions (RQs) to guide the development of this paper. Firstly, we would like to investigate the state-of-the-art SAST tools and the ability to combine them in detecting software vulnerabilities. Secondly, we explore an industrial experience that adopted a combination of SAST tools in supporting security assessment in an e-government project.

- RQ1. Which SAST tool has the best performance against the Juliet Test suite?
- RQ2. Is the performance of SAST increased when combining different tools?
- RQ3: How SAST tools can support security assessment activities in an open-source e-government project?

The contribution of this paper as follows:

- An overview of state-of-the-art SAST tools and their applications
- An experiment that evaluates the performance of these tools
- An in-depth case study about the adoption of SAST tools in e-government projects.

The paper is organized as follows. Section 2 presents backgrounds about software security, SAST, and security in e-government sectors. Section 3 describes our case study of SOREG – Secured Open source-software Repository for E-Government. Two main research components of the projects are presented in this paper, an experiment with different SAST tools in Section 4 and a qualitative evaluation of a combined SAST approach for supporting security assessment in Section 5. After that, Section 6 discusses the experience in this paper, and Section 7 concludes the paper.

2. Background

2.1. *Software security and vulnerabilities*

Software security is “the idea of engineering software so that it continues to function correctly under malicious attack” (McGraw and Potter, 2004). Software security is always associated with the products and data is protected, the skills and resources of adversaries, and the costs of potential assurance remedies (Felderer et al., 2016; McGraw and Potter, 2004). In this research, we looked for software vulnerability, which can be defined as defects or weaknesses in software design, implement or operation management and can be used to break through security policies (Dowd et al., 2007).

Vulnerability databases record structured instances of vulnerabilities with their potential consequences. It helps developers and testers be aware of and keep track of existing vulnerabilities in their developing systems (Ghaffarian and Shahriari, 2017). Many of the databases are the results of a global effort by communities to leverage the existing large number of diverse real-world vulnerabilities. Security experts and communities maintain different databases and taxonomies of vulnerabilities, for instance, CVE, CWE NPD, MFSA, OWASP, and Bugzilla. National Vulnerability Database (NVD)[1] is the database operated by the US National Institute of Standards and Technology that stores vulnerable information in the form of security checklists, security related software flaws, misconfigurations, product names, and impact metrics.

Common Vulnerabilities and Exposures (CVE)[2] is the dataset is a publically available dictionary for vulnerabilities to allow for a more consistent and concise use of security terminology. In addition to the CVE entry, the vulnerability will also be classified in a set of Common Weakness Enumeration (CWE)[3] categories. The CWE provides a common language to describe software security weaknesses and classifies them based on their reported weaknesses. For instance, a category cross-site scripting (XSS) describes vulnerabilities that occur when form input is taken from a user and not properly validated, hence, allowing for malicious code to be injected into a web browser and subsequently displayed to end-users. SQL-Injection is another common type of vulnerability, where user input is not correctly validated and directly inserted in a database query. A path manipulation type occurs when users

1 https://nvd.nist.gov/.
2 https://cve.mitre.org/.
3 https://cwe.mitre.org/.

Table 1 – CWE categories and examples (Oyetoyan et al., 2018).

Class Id	Weakness class	Example Weakness (CWE Entry)
W321	Authentication and Access Control	CWE-259: Use of Hard-coded Password
W322	Buffer Handling (C/C++ only)	CWE-121: Stack-based Buffer Overflow
W323	Code Quality	CWE-561: Dead Code
W324	Control Flow Management	CWE-483: Incorrect Block Delimitation
W325	Encryption and Randomness	CWE-328: Reversible One-Way Hash
W326	Error Handling	CWE-252: Unchecked Return Value
W327	File Handling	CWE-23: Relative Path Traversal
W328	Information Leaks	CWE-534: Information Exposure Through Debug Log Files
	Initialization and Shutdown	CWE-404: Improper Resource Shutdown or Release
W329	Injection	CWE-134: Uncontrolled Format String
W3210	Malicious Logic	CWE-506: Embedded Malicious Code
W3211	Number Handling	CWE-369: Divide by Zero
W3212	Pointer and Reference Handling	CWE-476: NULL Pointer Dereference

can view files or folders outside of those intended by the application. Buffer handling vulnerability allows users to exceed the buffer's bounds which can result in attacks ranging from writing instructions to gaining full system access or control. The overview of different CWE categories is given in Table 1. We will use CWE categories to address the type of vulnerabilities that SASTs can detected in this study.

2.2. *Static application security testing (SAST)s*

Security testing is known as a process intended to reveal flaws in the security mechanisms of an information system that protect data and maintain functionality as intended. There are two major types of security testing, i.e. static testing and dynamic testing. Static Application Security Testing (SAST) utilizes a static code analysis tool to analyze source code to identify potential vulnerabilities or software faults. Different from dynamic approaches, SAST examines source code without executing it, and yields result by checking the code structure, the sequences of statements, and how variable values are processed throughout the different function calls. Common techniques using in SAST tools include:

1 Syntactic analysis such as calling insecure API functions or using insecure configuration options. An example of this category would be an analysis of Java programs that call to java.util.random (which does not provide a cryptographically secure random generator).
2 Semantic analysis that requires an understanding of the program semantics such as the data flow or control flow of a program. This analysis starts by representing the source code by an abstract model (e.g., call graph, control-flow graph, or UML class/sequence diagram). An example of this class would be a check for direct data-flows from a user form input to a SQL statement (indicating a potential SQL Injection vulnerability).

As SAST tools work as a white box testing and does not actually run the source code, a reported vulnerability from the tool might not necessarily be an actual one. This can because of two reasons:

- The source code is secure (true negative)
- The source code has a vulnerability but it is not reported by the tools (false negative).
- For each correct finding from the tool, a human expert is necessary to decide:
- If the finding represents a vulnerability, i.e., a weakness that can be exploited by an attacker (true positive), and, thus, needs to be fixed.
- If the finding cannot be exploited by an attacker (false positive) and, thus, does not need to be fixed.

Research about SAST tools is not new. The Center for Assured Software (CAS) developed a benchmark test suite with “good code” and “faulted code” across different languages to evaluate the performance of static analysis tools (Okun et al., 2012). They assessed five commercial SAST tools and reported the highest recall score of 0.67 and the highest precision score of 0.45. Dıaz and Bermejo compared the performance of nine SAST tools, including commercial ones, and found an average recall value of 0.527 and average precision value of 0.7 (Okun et al., 2013). Charest investigated four different SAST tools in detecting a class of CWE using the Juliet test suite (Charest and Wu, 2016). The best observed performance in terms of recall was 0.46 for CWE89 with an average precision of 0.21. Baca et al. evaluated the use of a commercial SAST tool and found it is difficult to apply in an industrial setting (Baca et al., 2013). In his case, the process of correcting false positive findings leads to additional vulnerability in the existing secure source code. Hofer conduced few experiments and found that different SAST tools detected different kinds of weaknesses (Hofer, 2010). In this research, we will analyze the effectiveness of seven different SAST tools. Different from previous research, we also investigate the performance of combining these tools.

2.3. *Security concerns in e-Government*

e-Government is commonly conceptualized as governments' use of information technologies combined with organizational change to improve the structures and operations of government (Field et al., 2003). The implementation of e-government is expected to help governments deliver services and transform relations with citizens, businesses, and different units in public sectors (Grönlund, 2001). In contrast to developed countries where e-government is well-established, there are many challenges for e-government in developing countries, such as an inadequate digital infrastructure, a lack of skills and competencies for design, implementation, use, and management of e-government systems, and a lack of trust in the security and privacy of the systems

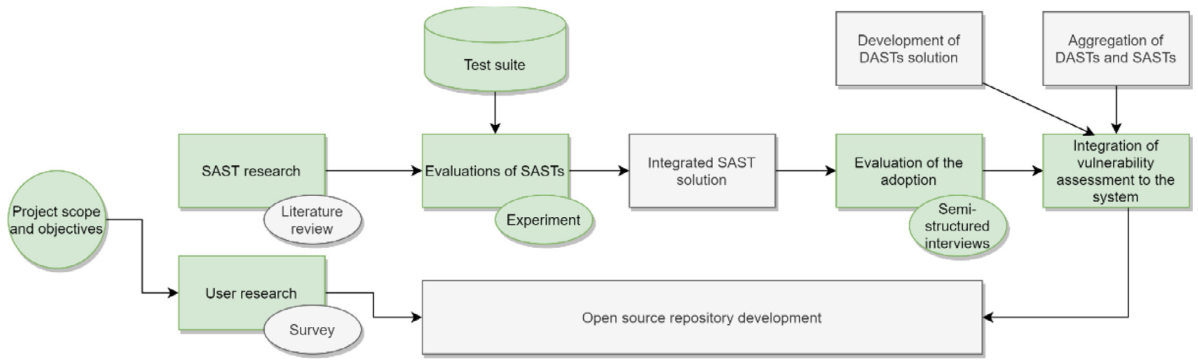
Test suite
Development of DASTs solution
Aggregation of DASTs and SASTs
SAST research
Literature review
Evaluations of SASTs
Experiment
Integrated SAST solution
Evaluation of the adoption
Semi-structured interviews
Integration of vulnerability assessment to the system
Project scope and objectives
User research
Survey
Open source repository development

Fig. 1 – An overview of the research (green boxes) and development (grey boxes) process.

(Twizeyimana and Andersson, 2019; Nkohkwo and Islam, 2013; Twizeyimana, 2017).

Security and privacy threats are always major concerns for e-government, such as cyberspace identity thefts and privacy violations (Twizeyimana and Andersson, 2019). If e-government systems are not well secured, security attacks may harm e-government systems at any time, leading to different types of financial, psychological and personal damages. Commonly reported security attacks include Denial of Service (DoS) attacks, unauthorised network access, theft of personal information, online financial fraud, website defacement, application-layer attacks such as cross-site scripting (XSS), and penetration attacking (Bélanger and Carter, 2008). Alshehri emphasized that privacy and security must be protected to increase the user's trust while using e-government services (Alshehri and Drew, 2010).

Security measures in an e-government system can be implemented at physical, technical, or management levels (Bélanger and Carter, 2008). As a software engineering research, we focused on the technical part of security measure. Firewalls, intrusion detection and prevention software, encryption, and secure networks are the most common security measures that have been implemented to secure e-government agencies against cyber threats (Carlos et al., 2012). SAST is a part of the vulnerability assessment system, belongs to the intrusion detection and prevention measures.

3. Research methodology

3.1. *Research context*

Vietnam is prioritizing e-government as a central pillar of the national digital transformation strategy to increase digital infrastructure, solutions, and capacity in government, industry, and society. During the last few decades, several initiatives have been implemented at regional and national levels to increase the digital capacity of the government, provide e-services to some extent, develop IT infrastructure and integrate national information systems and database. Aligned with the national strategy, a government-funded project, entitled "*Secured Open source-software Repository for E-Government (SOREG)*" has been conducted. It is note that the project is among several funded R&D projects towards the implementation of the whole e-government systems in a large-scale.

Aligned with the e-government policy, the development and security assurance of an open-source repository is necessary. The repository will serve for ca. 2.8 million government officers. Therefore, the security aspect is of high priority. The project was funded by the Ministry of Science and Technology of Vietnam from Nov-2018 to Oct-2020. The initiated budget is 200,000 Eur. The project team includes 23 key members who participate in project planning, implementation, and closure. The main objectives of SOREG are (1) proposal and development of a prototype of a community-driven open-source software repository, (2) development and validation of a security assessment approach using SAST and DAST tools. The security assessment module focuses on software vulnerability. The expectation is that the module can detect existing vulnerabilities from dependent components, such as libraries, frameworks, plug-ins, and other software modules. Issues with X-injections, e.g. SQL injection, LDAP injection should be detected at a practical rate. Other types of vulnerabilities as described in CWE should also be covered at an acceptable level.

The project was led by a domestic software company so-called MQ Solution. We participated in the project with both passive and participant observation. The first author of the paper participated in the project as a researcher, and is responsible for the plan and conducting an experiment with SAST tools. The research design was conducted and the experiment was carried on without affecting the original project plan. The second, third, and fourth authors of the paper directly performed the experiment following a predetermined design and also participated in developing different software modules in the projects. The first part of SOREG with literature review and market research has been partly reported in our previous work (Nguyen Duc and Chirumamilla, 2019).

3.2. *Research design*

Since the project involves both research and development activities, we will only focus on the research-relevant parts. Fig. 1 describes the research process leading to the selection and evaluation of SAST tools in supporting vulnerability assessment in SOREG. In the scope of this work, we focus on the research activity; hence, the open-source repository development is not mentioned (represented as a grey box). We also

Table 2 – A list of popular SAST tools by the end of 2018.

Tool name	Description
SonarQube	Scans source code for more than 20 languages for Bugs, Vulnerabilities, and Code Smells
Infer	Or “Facebook Infer”, static check for C, C++, Objective-C and Java, work for iOS and Android
IntelliJ IDEA	integrated development environment (IDE) written in Java, support multiple languages
VCG	an automated code security review tool that handles C/C++, Java, C#, VB and PL/SQL
Huntbug	A Java bytecode static analyzer tool based on Procyon Compiler tools aimed to supersede the FindBugs
PMD	A cross-language static code analyzer
Spotbug	A static analysis for Java code, a successor of FindBugs

exclude the research and development of Dynamic Application Security Testing (DAST) and the integration of DAST and SAST tools in this paper (the other two grey boxes in Fig. 1).

This paper reports a part of the case study with two parts, an experiment that investigates SAST tools with a test suite and a qualitative evaluation of the proposed SAST solution. When the project had started, we conducted an ad hoc literature review to understand the research area of software security testing and particularly SAST and DAST tools. After that, as a feasibility analysis, we selected a set of SAST tools and conducted an experiment to evaluate the possibility of combining SAST tools. A development activity follows with the architectural design of the integrated SAST solution and prototype development. After that, we conducted a preliminary evaluation of the prototype with expert interviews.

We described which SAST tools are selected (Section 4.1), the test suite to compare them (Section 4.2), the evaluation metric (Section 4.3), and the experiment result (Section 4.4). We briefly describe the outcome of the integrated SAST solution (Section 4.5) in this paper. The analysis of semi-structured interviews is shown in Section 5.

All software submitted to the repository must go through a vulnerability assessment, as shown in Fig. 2. The assessment module includes two parts (1) tools including both DAST and SAST tools, and (2) experts as moderators. The experts received reported results from tools and decide the vulnerability level of the inputted software. If the software passes the check, it will be published in the repository and available to all users. Otherwise, the software will be sent back to the submitters and not accepted for publishing.

3.3. *The selected SAST tools*

The literature review on SAST and experts' opinions are the two main inputs for selecting SAST tools. We gathered seven tools that attract both research and practitioners by the time the research project was conducted (end of October 2018). The list of the tools is summarized in Table 2.

SonarQube[4] is one of the most common open-source static code analysis tools for measuring quality aspects of source code, including vulnerability. SonarQube implements two fundamental approaches to check for issues in source code: (1) syntax trees and API basics and (2) semantic APIs. The code analyzer parses the given source code file and produces the syntax tree. The structure is used to clarify the problem as well as determine the strategy to use when analyzing a file. In addition to enforcing rules based on data provided by the syntax tree, SonarQube provides more information through a semantic representation of the source code. However, this model currently only works with Java source code. This semantic model provides information regarding each symbol manipulated. Using the API, Sonar has built-in several popular and proven tools available in the open-source community. These tools, through the implementation of standardized testing source code, consider possible errors and errors, each in their own opinion. The nature of checks ranges from small styles, for example detecting unwanted gaps, to more complex spaces that are more prone to potential errors, such as variables that cannot qualify checks result in null references.

Infer,[5] also referred to as "Facebook Infer", is a static code analysis tool developed by an engineering team at Facebook along with an open-source community. It cover multiple lanuages, such as Java, C, C++, and Objective-C, and is deployed at Facebook in the analysis of its Android and iOS apps. Its foundation is theories of formal verification of software (Churchill et al., al., Sergio). Infer uses a technique called bi-abduction to perform a compositional program analysis that interprets program procedures independently of their callers (https). Common quality checks include null pointer exceptions, resource leaks, annotation reachability, missing lock guards, and concurrency race conditions in Android and Java code. It checks for null pointer problems, memory leaks, coding conventions, and unavailable API's in C, C++ and Objective C. It is claimed that this enables Infer to scale to large codebases and to run quickly on code-changes in an incremental fashion, while still performing an inter-procedural analysis that reasons across procedure boundaries (Calcagno et al., 2015).

Intellij IDEA[6] is a static code analysis feature that provides an on-the-fly code check when using Intellij development environment. Various inconsistencies, probable bugs, redundancies, spec violations, etc. are highlighted in the editor right while you are typing. After IntelliJ IDEA has indexed a source code folder, it claims to produce a fast and intelligent experience by giving relevant suggestions in every context, i.e. intelligent quick-fixes and on-the-fly code analysis.

VCG[7] is an automated code security review tool that handles C/C++, Java, C#, VB and PL/SQL. In addition to performing some more complex analysis it has a portable and expandable configuration that allows users to add any bad functions (or other text). It provides a nice visualization (for the entire codebase and for individual files) showing relative proportions of code, whitespace, comments, "ToDo" style comments and bad code. It also searches intelligently to identify buffer overflows and signed/unsigned comparisons.

[4] www.sonarqube.org.

[5] https://fbinfer.com/.

[6] https://www.jetbrains.com.

[7] https://security.web.cern.ch/recommendations/en/codetools/vcg.shtml.

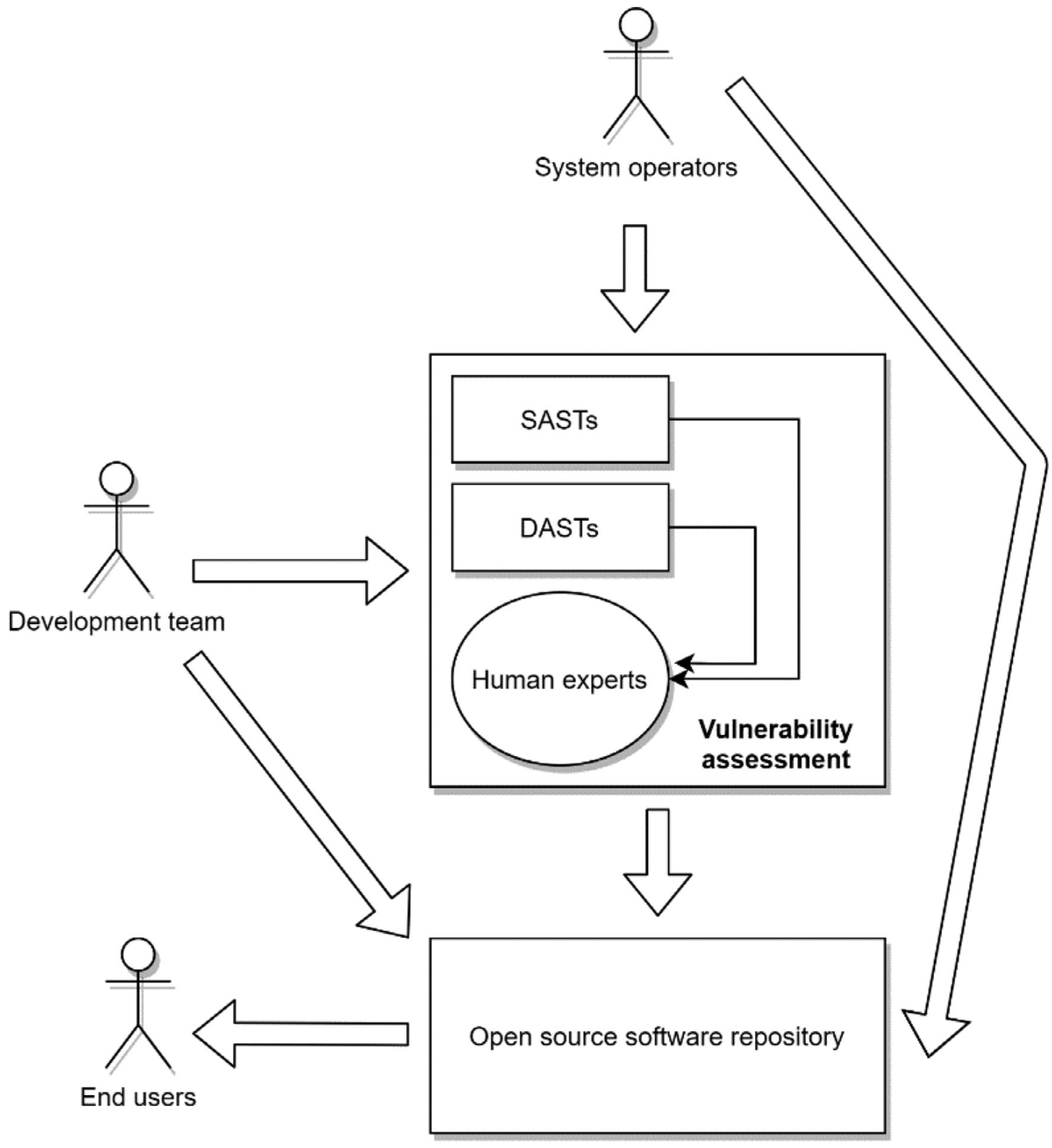

Fig. 2 – An overview of the open source software repository.

Huntbug[8] is a open source Java static analyzer tool based on Procyon Compiler Tools which aims at outperforming the famous tool FindBugs. The analysis applied by the tool is based on detecting bug patterns.

PMD[9] is a source code analyzer. It finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, and so forth. It supports Java, JavaScript, Salesforce.com Apex and Visualforce, PLSQL, Apache Velocity, XML, XSL. Similar to SonarQube or Huntbug, PMD operates using a set of rules.

SpotBugs[10] is a program that uses static analysis to look for bugs in Java code. SpotBugs checks for more than 400 bug patterns.

3.4. *The test suite*

Different benchmark test suites exist for testing security tools. Two popular examples are the Juliet test suite and OWASP Benchmark. We decided to use the Juliet test suite because it is not only limited to the top 10 vulnerabilities as of the OWASP benchmark dataset. In addition, the test suite is designed for all ranges of weaknesses and not only limited to web-based weaknesses. One of the goals for developing the Juliet test

8 https://github.com/amaembo/huntbugs.
9 https://pmd.github.io/.
10 https://spotbugs.github.io/.

suite was to enable open dataset for empirical research. The test suite has been popular among software and security engineering research (Oyetoyan et al.; Díaz and Bermejo, 2013; Velicheti et al., 2014; Oyetoyan et al., 2018). The latest version (ver 1.3) comprises 64,099 test cases in C/C++ and 28,881 test cases in Java.[11] It is fair to focus on these two programming languages due to the dominance of them in SOREG's source code.

3.5. *Evaluation metrics*

There exist a massive amount of research evaluating software quality models and tools and the performance metrics have been "de facto" standard in software engineering research. We reported the following metrics for each SAST (Oyetoyan et al., 2018; Okun et al., 2013):

- **True Positive (TP):** The number of findings where the tool correctly reports the flaw that is the target of the test case.
- **False Positive (FP):** The number of findings where the tool reports a flaw with a type that is the target of the test case, but the flaw is reported in non-flawed code.
- **False Negative (FN):** This is not a tool result. A false negative result is added for each test case for which there is no true positive.
- **Blank (Incidental flaws):** This represents tool's result where none of the types above apply. More specifically, either the tool's result is not in a test case file or the tool's result is not associated with the test case in which it is reported.
- **Recall** = TP / (TP + FN)
- **Precision** = TP / (TP + FP)
- **F1 Score** = 2 (Recall x Precision)/ (Recall + Precision)

It is possible to have both TP and FP in a test file. In this case, our SAST is not sophisticated enough to discriminate for instance when the data source is hardcoded and therefore does not need to be sanitized. We adopt the "strict" metrics defined by CAS (Velicheti et al., 2014) as they truly reflect a real-world situation.

4. Results

The answers to RQ1 to RQ3 are described in Sections 4.1, 4.2 and 4.3 correspondingly.

4.1. RQ1. *Which SAST tool has the best performance against the Juliet test suite?*

We report the evaluation results of the seven tools on Juliet Test Suite v1.3. as shown in Table 3. Looking at the number of outputs from each tool, Intellij is on top of the list with 37.694 reported issues. PMD is in the second place with 37.405 reported issues and after that Sonarqube finds 28.875 reported issues. To measure the accuracy of the tools, we calculated the F1 score as shown in Table 3. The top three most accurate tools in our experiment are Intellij, PMD, and Sonarqube accordingly. The successors of FindBugs, i.e. Huntbugs and Spotbugs detect a small number of issues, showing their limited capacity in a software security area. Infer, the SAST promoted by Facebook finds only 428 issues from our test suite.

False Positives are also of our concern since this is one of the main barriers to adopt SAST tools in industrial projects (Oyetoyan et al., 2018). This reflects the value of precisions. The rank of SAST is a bit different here, as Sonarqube has the best precision value (0,6), following by VCG (0,59) and Intellij (0,57).

Answer to RQ1: Sonarqube has the best precision score of 0.6. Intellij has the best F1 score of 0.69. For a single CWE class, the best achieved F1 score is from PMD for the error handling class.

We looked into details how each tool performs regarding CWE categories. Table 4 reports the F1 score of the seven tools across our 12 weakness categories.

- Authentication and Authorization include vulnerabilities relating to unauthorised access to a system. Intellij IDEA has the best F1 score of 0.53 and followed by Sonar Qube with 0.26. Overall the ability to detect issues with SAST tools in this category is quite limited.
- Code quality includes Issues not typically security related but could lead to performance and maintenance issues. Intellij IDEA has the best F1 score of 0.83 and followed by PMD with 0.79. Sonar Qube has an F1 score of 0.63, which is in third place. Other tools are not able to detect any issues in this category. This can be explained by the comprehensiveness of SAST tools, since tools such as SonarQube and Intellij cover not only vulnerabilities but also many other types of concerns, e.g. code smells, bugs, and hot spots
- Control Flow Management explores issues of sufficiently managing source code control flow during execution, creating conditions in which the control flow can be modified in unexpected ways. PMD has the best F1 score of 0.69 and followed by Sonarqube with an F1 score of 0.62.
- Encryption and Randomness refer to a weak or improper usage of encryption algorithms. Intellij IDEA has the best F1 score of 0.53 and followed by Sonar Qube with an F1 score of 0.26. Overall the ability to detect issues with SAST tools in this category is quite limited.
- Error Handling includes failure to handle errors properly that could lead to unexpected consequences. PMD has the best F1 score of 0.84 and followed by Intellij IDEA with an F1 score of 0.73.
- File Handling includes checks for proper file handling during read and write operations to a file on the hard-disk. Intellij IDEA has the best F1 score of 0.7 and followed by PMD with an F1 score of 0.63. Sonarqube works not so well with this category as its F1 score is only 0.35.
- Information Leaks contains vulnerabilities about exposing sensitive information to an actor that is not explicitly authorized to have access to that information. SonarQube has the best F1 score of 0.67 and followed by Intellij IDEA and PMD with similar scores.

[11] https://samate.nist.gov/SARD/testsuite.php.

Table 3 – Evaluation results of SAST tools against Juliet 3.1. Testsuite.

Tool	No. Detections	TP	FP	FN	Recall	Precision	F1-Score
Sonarqube	28,875	9381	6216	17,321	0.35	0.6	0.44
Infer	428	1564	1364	45,768	0.03	0.53	0.06
Intellij	37,694	52,276	40,026	8502	0.86	0.57	0.69
VCG	8143	8900	6164	38,053	0.19	0.59	0.29
PMD	37,405	12,094	10,389	8791	0.58	0.54	0.56
Huntbugs	2677	1873	2138	43,519	0.04	0.47	0.07
SpotBug	624	347	313	45,572	0.01	0.53	0.02

Table 4 – Evaluation results of SAST tools across different CWE categories.

CWE Class	Sonar Quebe	Infer	Intellij IDEA	VCG	PMD	Huntbugs	Spotbugs
Authentication and Access Control	0.26	0.17	0.53	0.19	0.25	0.07	0
Code quality	0.63	0	0.83	0	0.79	0	0
Control Flow Management	0.38	0	0.69	0.28	0.65	0	0
Encryption and Randomness	0.62	0	0.6	0.15	0.68	0	0
Error Handling	0.64	0	0.73	0	0.84	0	0
File Handling	0.35	0	0.7	0.23	0.63	0.07	0
Information Leaks	0.67	0	0.62	0.45	0.63	0	0
Initialization and Shutdown	0.16	0	0.73	0.39	0.72	0	0
X-Injection	0.51	0	0.72	0.36	0.68	0.15	0.04
Malicious Logic	0.79	0.02	0.8	0.08	0.6	0	0
Number Handling	0.29	0.12	0.56	0.17	0.29	0	0
Pointer and Reference Handling	0.23	0	0.71	0.21	0.75	0	0

- Initialization and Shutdown concern improper initializing and shutting down of resources. We see that IntelliJ IDEA has the best F1 score of 0.73 and followed by PMD with an F1 score of 0.72.
- X-Injection: a malicious code injected in the network which fetched all the information from the database to the attacker. This is probably one of the most important types of vulnerability. In this category, IntelliJ IDEA is the most accurate tool with an F1 score of 0.72 followed by PMD (0.68) and Sonar Qube (0.51)
- Malicious Logic concerns the Implementation of a program that performs an unauthorised or harmful action (e.g. worms, backdoors). This is also a very important type of vulnerability and probably among the most common ones. In this category, IntelliJ IDEA is the most accurate tool with an F1 score of 0.8 followed by Sonar Qube (0.79) and PMD (0.6)
- Number Handling include issues with incorrect calculations, number storage, and conversion weaknesses. Intellij IDEA has the best F1 score of 0.56 and followed by PMD and Sonar Qube with the same F1 score of 0.29.
- Pointer and Reference Handling issues, for example, the program obtains a value from an untrusted source, converts this value to a pointer, and dereferences the resulting pointer. PMD has the best F1 score of 0.75 and followed by PMD and Intellij IDEA with the same F1 score of 0.71.

4.2. *RQ2 is the performance of SAST increased when combining different tools?*

We also investigated if combining various SAST tools can produce better performance. We run the combination of two, three, and four SAST tools and compare them against our performance metrics. Table 5 presents our results in three categories (1) the combination that gives the best F1 score, (2) the combination that gives the best precision, and (3) the combination that produces the largest number of outputs. We reported again the result with IntelliJ IDEA as a benchmark for comparison. There are several worth-noticing assumption for this experiment:

- Firstly, we look for a tool or a set of tools that are practically useful. There is no interest in developing a new tool or extending a tool for better performance.
- Secondly, tools are treated as black boxes. Hence all the aggregation was done at the scanning result level. We wrote glue code that gathers all result files from different SASTs and combined these tools by aggregating the result files. The aggregation process is described in the workflow in Fig. 3. All the evaluation metrics, including TP, FP, TN, FN, Recall, Precision and F-score are recalculated basing on the aggregated files F_final (Fig. 3)

The result shows that it is possible to increase some performance metrics by combining different SAST tools. However, there are none of the combinations can produce the best value for all performance metrics. Interestingly, we see that Sonarqube appears in all best combinations, even though the SAST does not perform the best among single SAST tools. From Table 5, we can say that the 5-tools combination including SonarQube, Inteliji, VCG, Infer and SpotBug would probably give the best outcome in all of our metrics. However, this is beyond our experiment.

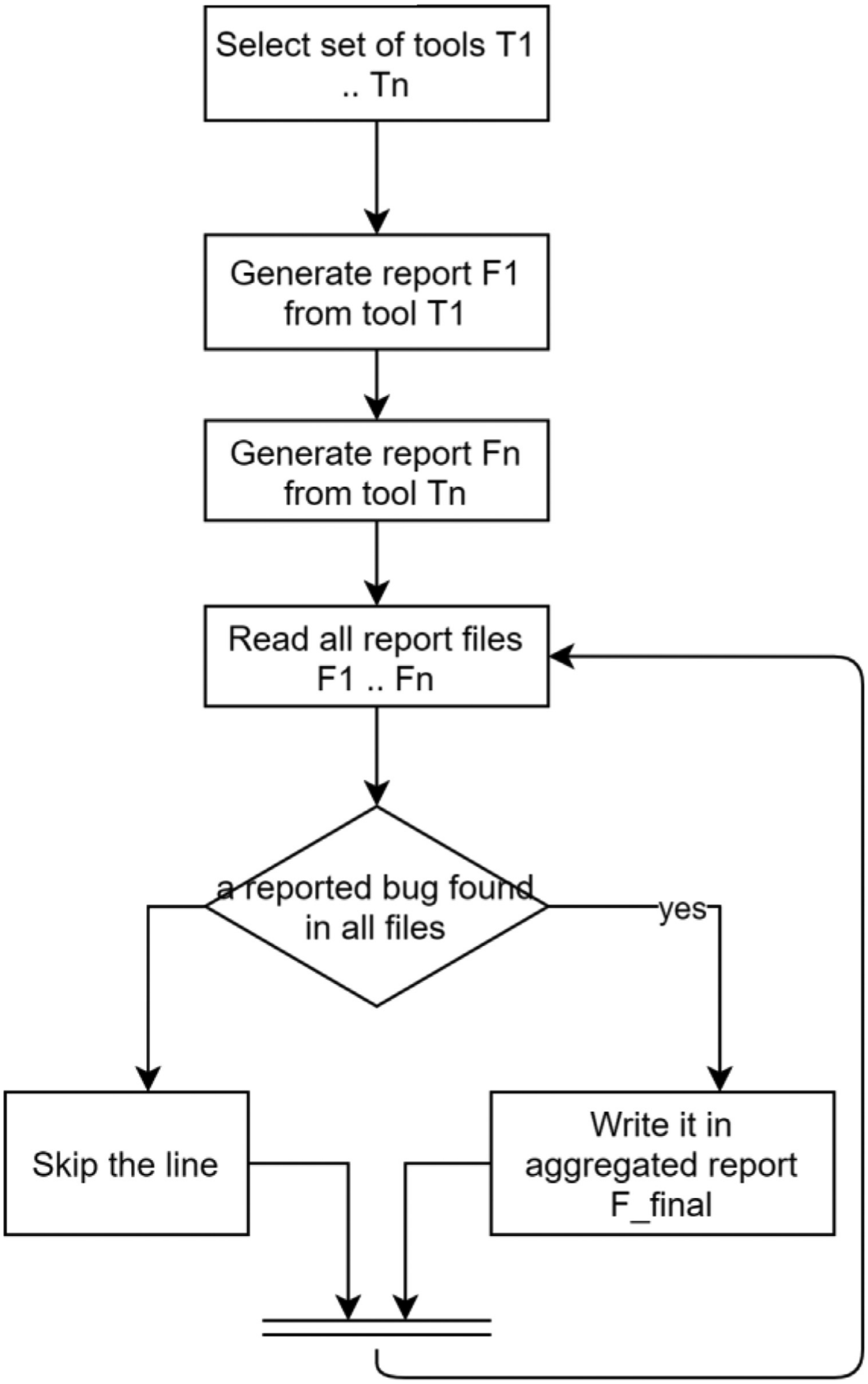
Select set of tools T1 .. Tn
Generate report F1 from tool T1
Generate report Fn from tool Tn
Read all report files F1 .. Fn
a reported bug found in all files
yes
Skip the line
Write it in aggregated report F_final

Fig. 3 – The work-flow of the tool aggregation process.

Table 5 – The effectiveness of combining various SAST tools.

Tool	TP	FP	FN	Recall	Precision	F-score	No of outputs	Evaluation criteria
Sonarqube + Inteliji	65,927	49,518	2199	0,97	0,57	0,72	43,997	Best F-Score
Sonarqube + SpotBug	9722	6544	17,296	0,36	0,62	0,45	28,900	Best Precision
Sonarqube + Inteliji + VCG + Infer	73,811	55,692	2095	0,97	0,57	0,72	44,101	Largest amount of outputs
Inteliji IDEA	52,276	40,026	8502	0,86	0,57	0,69	37,694	

Answer to RQ2: Combining SAST tools does give a better performance than of a single SAST, and this depends on the performance metrics

4.3. *The integrated SAST tools solution*

The experiment produces an input for the design and development of the SAST module in the security assessment part (as shown in Fig. 2) of the project. We present only briefly the architectural decisions that are taken and some architectural views of the solution.

From practical aspects, there are several requirements for SAST modules from the repository development and operation team. The team emphasized five criteria while working with SAST tools:

1 Result accuracy. SAST modules should scan within a limited timeframe and its output compared against a test application for which the results are known a-priori. The development team emphasizes the importance of False Positives (TPs). It is an active research on the effectiveness of SAST, however, many SAST tools have been used in practice.
2 Simplicity. Scanning and visualizing the result should be straightforward so that users without security background can also operate with or without manual documents. Scanning source code should not require the user to perform excessive operations to start running the tool.
3 Vulnerability coverage. We focused on the ability to detect different categories in the CWE database. Besides, detection of common vulnerabilities as identified by other industry standards such as OWASP Top 10 or SANS is desirable. Since the vulnerability taxonomy and ratings differ by each SAST vendor, it is necessary to receive from each SAST vendor their list and normalize them one against the other for a true vulnerability coverage comparison.
4 Supports multiple languages. Ensure that the SAST tool supports the popular programming languages such as Java and C++. It should also have a possibility to support emerging technologies as these may prove to be significant in the long run. For example, mobile or updated development languages (e.g. Android, Objective C, Ruby on Rails)
5 Customizability. The ability to adapt the scan results to different output format and integrate to different business logics. Each organization uses its' own framework for accessing databases and so the SAST tool must be customizable to the proprietary code. This capability also eliminates false positives that occur due to the custom code and the organization's business logic

Especially, when looking at the focus on FP, simplicity and the ability to customize, the development team had decided to select the combination of SonarQube and SpotBug as the most practical solution with SAST tools. The further development includes a spotbug plugin to a community-version SonarQube and a new SonarQube widget to customize the scanning result. The logical view and development view of the integrated solution is shown in Fig. 4.

The overall architecture of modules of SAST tools integrated into SonarQube is presented in Fig. 5. The blue boxes includes the aggregator that produces and process the aggregated result files F_finals, the configuration elements for AST Dev Plugin and the user interface using HTML template in widgets.

4.4. RQ3: *how SAST tools can support security assessment activities in an open-source e-government project?*

Semi-structured interviews were conducted with four stakeholders in the project. The profile of the interviewee is given in Table 6. The same interview guideline was used, including three main sections (1) Questions regarding the usability of the tool, (2) Questions regarding the usefulness of the tools for vulnerability assessment, and (3) Questions regarding the performance of the tools. The interviews ranged from 20 to 30 mins in total. We did note-taking during the interviews and summarized them into three main themes. It is noted that all interviews were done in Vietnamese, so the quotes are translated into English.

Perceptions about the usability of the SAST tools: there has been a consensus among interviewees about the usability of the tool. The user interface has been continuously improved and the final version has been tested with different stakeholders in the project. Before the tool's effectiveness can be evaluated, it is practically important that it can be accessed and operable by targeted users. Some feedback about the interfaces are:

- "*The user interface is lean and intuitive!*" (P04)
- "*I think the web interface looks good. It is easy to use and for me all the key features are visible. I do not need any manual documents to work with this tool*" (P02)

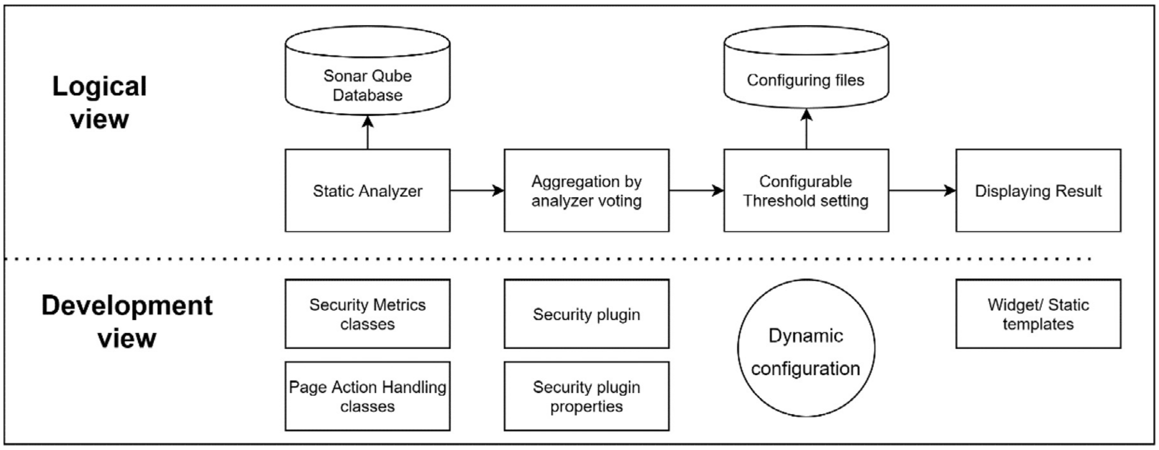

Fig. 4 – The logical view and development view of an integrated SAST solution.

Table 6 – Profiles of interviewees.

Interview No.	Title	Role in the project	Familiar with software security (from 1 to 5)
P01	Project manager	Lead the project from planning to completion. Coordinating vulnerability assessment modules with the repository	2
P02	Security expert	Representative of an expert user of a pilot organization	5
P03	System operator	Representative of a system operator in the pilot organization	4
P04	Project developer	A developer of the repository	3

Perceptions about the performance of the SAST tools: the performance of the tools originally refers to the technical capacity of the tools against real-world applications. Our interviewee highlighted the importance of showing the possibility of capture vulnerabilities across different categories of weakness:

- “*SonarQube has a wide range of test coverage. The adopted version inherited this from the community version and covers different types of vulnerabilities. It is important to be aware of different possible threats to our repository*” (P03)

We also see that performance is interpreted as a practical performance, that is the tools should be an indicator of security level and can be integrated into other ways of security assessment:

- “*The experiments show that the effectiveness of top 3 SAST tools does not differ much from each other. Then we care about how difficult it is and how much time it takes to develop and integrate the selected SAST tools to our repository. The proposed architecture looks great and integral into the overall system.*” (P04)
- “*Testing SAST tools is an important step that giving us confidence in adopting the right tool in the next step. Performance and coverage are important insights for expert teams to decide the security level of software apps*” (P01)

Perceptions about the usefulness of the SAST tools: in the nutshell, it seems that the automated tools will not be fully automatically operated in this project. The tools are perceived as useful as a complementary means to assess security. It should be combined with another type of security testing, i.e. DAST and other types of security assessment to give a triangulated result.

- “*SAST or even the combination of SAST tools and DASTs and other automated tools are not sufficient to ensure a safe software repository. I think the tools play important roles as inputs for expert teams who operate and manage security aspects of the repository.*” (P03)
- “*I think the tool has a good potential! I am looking forward to seeing how DASTs and SAST tools can be combined in this project*” (P02)

Answer to RQ3: SAST tools should be used towards a practical performance and in the combination with triangulated approaches for human-driven vulnerability assessment in real-world projects.

5. Discussion

5.1. *Discussing the research questions*

The experiment conducted in this research strengthens the findings from previous empirical studies on SAST tools. We updated the research of SAST tools with the state of the art

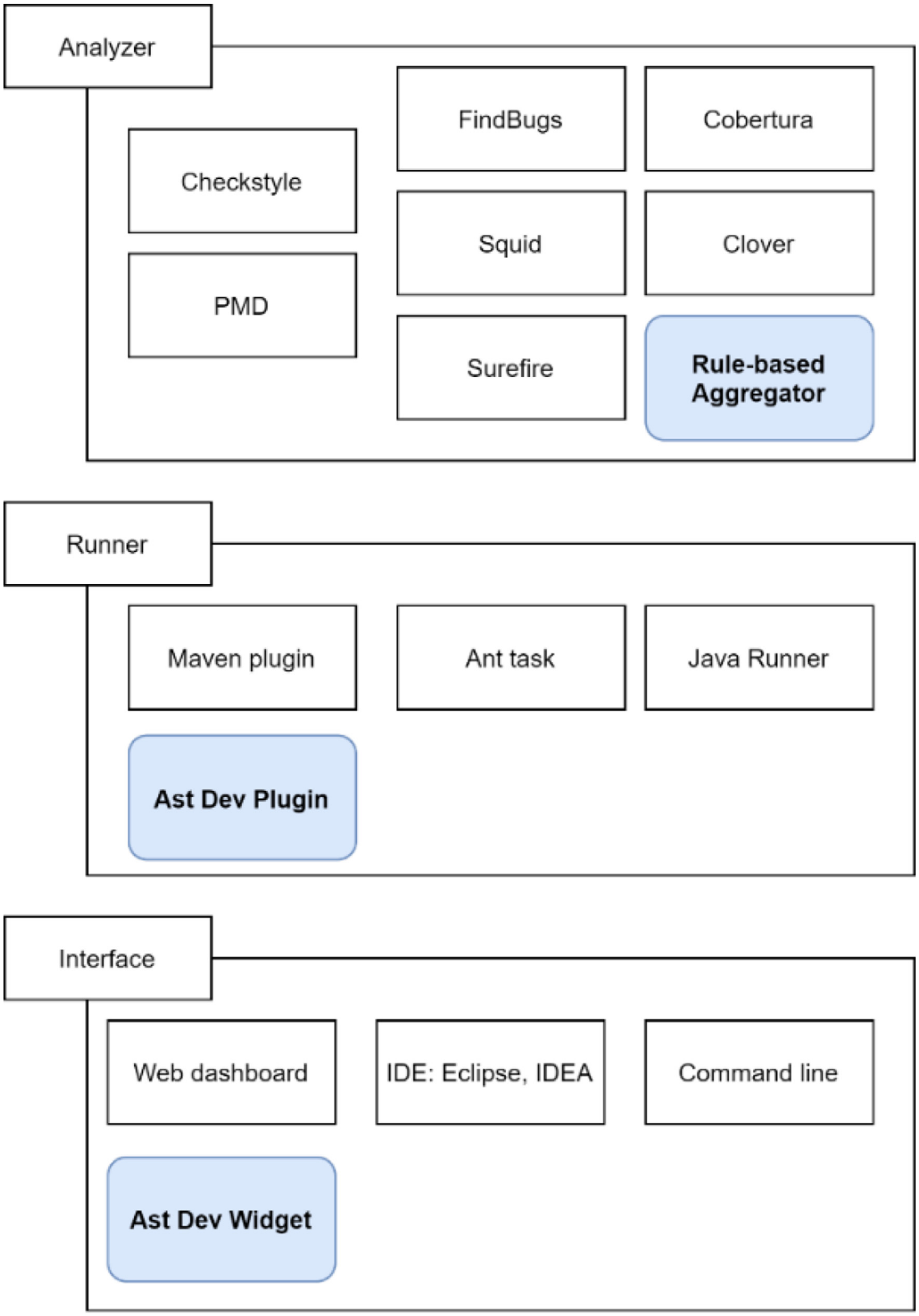

Fig. 5 – An architectural view on modules of the extended Sonar Qube tool.

tool list in 2018. By this time, we still see that using one SAST tool is not enough to cover the whole range of security weaknesses at the implementation phase. This agrees with the observation by Oyetoyan et al. (2018). However, the current SAST tools are rich in their features, e.g. ability to support multiple languages, various visualization options, and customizability. Previous studies reported the best precisions values of SAST tools around 0.45 – 0.7 (Okun et al., 2013; Hofer, 2010; Okun et al., 2012). Our study also reported the precision values of our best tools in this range. We also observed that SAST tools work relatively better in some CWE categories, such as Code quality, Encryption and Randomness, Error Handling, Information Leaks, and Malicious Logic.

In addition to existing studies, we revealed the possibility of combining different SAST tools to achieve better performance. In particular, we have used SonarQube and the base platform and combine the rulesets from other tools. As static analysis uses basically whitebox testing to explore source code, this result shows the potential to improve existing tools, and probably towards a universal security static security ruleset. This result might be interesting for both researchers and practitioners who look for practical improvement of SAST tools that are currently adopting in industry. In this work, we only adopted a simple voting mechanism. Future work would be an investigation of more advanced approaches, i.e. ensembling techniques or white-box analysis of SASTs.

Previous studies reported many challenges in adopting SAST tools during different stages of software project life cycles (Oyetoyan et al., 2018; Okun et al., 2013; Aloraini et al., 2019; Pashchenko, 2017). In this study, we focus on the deployment stage where software from other parties is tested before publishing. This quality check gate is common in all software repository models, such as Apple Store or Google Play. The objective of SAST here is different; we aim at supporting security assessment, not guiding software developers to improve their source code. The performance of the systems are possitvely perceived, as described in RQ3. Within the scope of SOREG, we see that the adopted approach is practically useful and contribute to the overall project scope. However, SASTs, including aggregated approaches are still far from a maturity state. False positives can not be reduced only by combining different SASTs and needed to be managed with the aid of other types of analysis, i.e. DAST:

5.2. *Threats to validity*

Our research also has some limitations. Firstly, we only include open-source SAST tools and only conduct security testing for open-source software. However, as we see from existing research, it might not be too much different in terms of tool performance with commercial SAST tools. Secondly, our research aims at developing a prototype and evaluating the proposed solution, therefore, we did not focus on architectural details and implementation. It could be that the perceived usefulness can be improved when we have a full-scale development of the combined SAST solution. Thirdly, we preliminary tested our SAST tools with a simple software application. The claimed results are mainly based on our experiments with the Juliet test suite. A case study with a large-scale industrial application might provide more insight that is complementary to our findings. Last but not least, the study is conducted in the context of a Vietnamese government project, which would have certain unique characteristics regarding organizational and managerial aspects. However, the research design was conducted separately in Norway and the observation process has been conducted with scientific and professional attitudes. The link to the Juliet test suite is given in this link: https://samate.nist.gov/SRD/testsuite.php. The experiment data is available published at https://docs.google.com/spreadsheets/d/1aH228YZUkDQ_ZsHu6l6Lg-ZMWjhMEpjwgtsTivijpJE/edit?usp=sharing. We tried to report as detail as possible the experiment process so that one can replicate our work in the same condition.

6. Conclusions

We have conducted a case study on a two-year project that develop and evaluate a secured open-source software repository for the Vietnamese government. This paper reports a part of the paper about evaluating and combining SAST tools for security assessment. Among evaluated SAST tools, we have found that Sonarqube and Intellij have the best performance. The combination does give a better performance than a single SAST, depending on the performance metrics. Practically, these SAST tools should be used towards a practical performance and in the combination with triangulated approaches for human-driven vulnerability assessment in real-world projects. In the future work, we will report the next step of the project, with a similar investigation on DAST and the effectiveness of combining SAST tools and DASTs in supporting software security assessment.

Declaration of Competing Interest

None.

CRediT authorship contribution statement

Anh Nguyen-Duc: Conceptualization, Data curation, Writing – original draft, Validation. **Manh Viet Do:** Conceptualization, Investigation, Writing – review & editing, Project administration. **Quan Luong Hong:** Conceptualization, Investigation, Writing – review & editing, Funding acquisition. **Kiem Nguyen Khac:** Software, Investigation, Visualization. **Anh Nguyen Quang:** Software, Investigation, Writing – review & editing.

Acknowledgement

This paper is supported by Vietnam Ministry of Science and Technology under the project “Secured Open source-software Repository for E-Government”, number KC.01.16/16-20. The project is led by MQ Solution 12.

REFERENCES

Source code security analysis tool functional specification version 1.0, National institute of standards and technology, Special Publication 500-268 (2007).

Chess B, McGraw G. Static analysis for security. IEEE Secur. Privacy 2004;2(6):76–9.

Oyetoyan, T.D., Soares Cruzes, D., Jaatun, M.G.: An empirical study on the relationship between software security skills, usage and training needs in agile.

Felderer, M., Buchler, M., Johns, M., Brucker, A.D., Breu, R., Pretschner, A.: Chapter one - security testing: a survey. In: Memon, A. (ed.) Advances in Computers, vol. 101, pp. 1–51. Elsevier, January 2016. 10.1016/bs.adcom. 2015.11.003

McGraw G, Potter B. Software security testing. IEEE Secur. Priv. 2004;2(5):81–5. doi:10.1109/MSP.2004.84.

Ghaffarian SM, Shahriari HR. Software vulnerability analysis and discovery using machine-learning and data-mining techniques: a Survey. ACM Comput. Surv. 2017;50(4) pp. 56:1–56:36, Aug. 2017.

Díaz G, Bermejo JR. Static analysis of source code security: assessment of tools against SAMATE tests. Inf. Softw. Technol. 2013;55(8):1462–76. doi:10.1016/j.infsof.2013.02.005.

Velicheti LMR, Feiock DC, Peiris M, Raje R, Hill JH. Towards modeling the behavior of static code analysis tools. In: Proceedings of the 9th Annual Cyber and Information Security Research Conference. ACM; 2014. p. 17–20.

Oyetoyan, T.D., Milosheska, B., Grini, M., & Soares Cruzes, D. (2018). Myths and facts about static application security testing tools: an action research at telenor digital. In J. Garbajosa, X. Wang, & A. Aguiar (Eds.), Agile Processes in Software Engineering and Extreme Programming (pp. 86–103). Springer International Publishing.

Okun V, Delaitre A, Black PE. Report on the static analysis tool exposition (SATE) IV. Tech. Rep., NIST 2013 January.

Aloraini B, Nagappan M, German DM, Hayashi S, Higo Y. An empirical study of security warnings from static application security testing tools. J. Syst. Softw. 2019;158. doi:10.1016/j.jss.2019.110427.

Pashchenko I. FOSS version differentiation as a benchmark for static analysis security testing tools. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering; 2017. p. 1056–8. doi:10.1145/3106237.3121276.

Dowd M, McDonald J, Schuh J. The Art of Software Security Assessment. Addision-Wesley publications; 2007.

https://fbinfer.com/docs/separation-logic-and-bi-abduction/

Calcagno Cristiano, Distefano Dino, Dubreil Jeremy, Gabi Dominik, Hooimeijer Pieter, Luca Martino, O'Hearn Peter, Papakonstantinou Irene, Purbrick Jim, Rodriguez Dulma. In: Moving Fast with Software Verification. NASA Formal Methods. Lecture Notes in Computer Science, 9058. Cham: Springer; 2015. p. 3–11.

Churchill, Dulma; Distefano, Dino; Luca, Martino; Rhee, Ryan; Villard, Jules. "AL: A New Declarative Language for Detecting Bugs with Infer". Facebook Code Blog Post.

Sergio, de Simone. "Facebook's new AL language aims to simplify static program analysis". InfoQ.

Field T, Muller E, Lau E, Gadriot-Renard H, Vergez C. The case for egovernment: excerpts from the OECD report "The E-Government Imperative. OECD J. Budgeting 2003;3(1):61–96.

Grönlund Å. Electronic Government: Design, Applications and Management: Design, Applications and Management. IGI Global; 2001.

Twizeyimana JD, Andersson A. The public value of E-Government–a literature review. Gov. Inf. Q. 2019;36(2):167–78. doi:10.1016/j.giq.2019.01.001.

Bélanger France, Carter Lemuria. Trust and risk in e-government adoption. J. Strat. Inf. Syst. 2008;17(2):165–76.

Alshehri Mohammed, Drew Steve. In: IADIS International Conference ICT, Society and Human Beings. E-government fundamentals; 2010.

Carlos E, Jiménez Francisco, Falcone Jiao, Feng Héctor, Puyosa Agustí Solanas, González F. e-government: security threats. e-Government 2012;11(21).

Baca D, Carlsson B, Petersen K, Lundberg L. Improving software security with static automated code analysis in an industry setting. Softw. Pract. Exp. 2013;43(3):259–79.

Hofer, T.: Evaluating static source code analysis tools. Technical report (2010)

Okun, V., Delaitre, A., Black, P.E.: NIST SAMATE: static analysis tool exposition (sate) iv, March 2012. https://samate.nist.gov/SATE.html

Nkohkwo QN-A, Islam MS. Challenges to the successful implementation of e-government initiatives in Sub-Saharan Africa: a literature review. Electron. J. e-Government 2013;11(2):253–67.

Charest NRT, Wu Y. Comparison of static analysis tools for Java using the Juliet test suite. In: 11th International Conference on Cyber Warfare and Security; 2016. p. 431–8.

Nguyen Duc, A., & Chirumamilla, A. (2019). Identifying security risks of digital transformation—an engineering perspective. In I.O. Pappas, P. Mikalef, Y.K. Dwivedi, L. Jaccheri, J. Krogstie, & M. Mäntymäki (Eds.), Digital Transformation for a Sustainable Society in the 21st Century (pp. 677–688). Springer International Publishing. 10.1007/978-3-030-29374-1_55

Twizeyimana JD. User-centeredness and usability in e-government: a reflection on a case study in Rwanda. Paper Presented at the Proceedings of the Internationsl Conference on Electronic, 2017.

Dr. Anh Nguyen-Duc is an Associate Professor in the University of South Eastern. He has authored/co-authored more than 50 publications in peer-reviewed journals, conferences, and workshops. Apart from being on the program committees of several international conferences including ICGSE2019, PROFES2019, EASE2019, Dr. Anh Nguyen Duc has also been involved in the organization of several conferences and workshops (ICSOB2019, NOKOBIT2019, NIK2019, ISEC2017, PROFES2016, ICE2016). Particularly, he served as workshop chairs for the International workshop on Software Startup Research in 2015, 2016, and 2017. Prior to joining research and development field, he worked as a software engineer and an IT consultant for several years in Vietnam. His current research interests include software startup research, software security, global software engineering, and empirical software engineering.

Manh Viet Do graduated with a masters degree in IT in Japan, and is the CEO of MQ ICT Solutions. He is the project manager and is responsible for the scientific output of the project.

Quan Luong Hong is the CTO of MQ ICT Solutions. He graduated from Hanoi University of Technology, Vietnam.

Nguyen Khac Kiem is a lecturer of the Institute of Electronics and Telecommunications at Hanoi University of Technology. His research interests include 5G communication, microwave devices and security.

Anh Nguyen Quang is a lecturer of the Faculty of Basic Sciences, Vietnam University of Transport and Communications and a PhD candidate at University of Bordeaux, France. His research interests include educational management and adoption of technology in education.