Perceptions of Software Practitioners Regarding Crypto-API Misuses and Vulnerabilities

ANONYMOUS AUTHOR(S)

CCS Concepts: • **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

Additional Key Words and Phrases: Cryptography misuses, Mining Software Repositories, vulnerabilities

ACM Reference Format:
Anonymous Author(s). 2022. Perceptions of Software Practitioners Regarding Crypto-API Misuses and Vulnerabilities. *J. ACM* 37, 4, Article 111 (August 2022), 18 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

It is becoming increasingly indispensable for developers to rely on cryptography to protect data since software applications are collecting more and more sensitive information. Ferguson et al., [8] says that even a system designed by security experts might be broken a few years later and characterized cryptography as "fiendishly difficult". According to the OWASP Foundation, cryptographic failures correspond to the second most prone cause of vulnerabilities in web applications [9]. As claimed by Nadi et al. [19], the offered cryptography Application Programming Interfaces (APIs) are often relatively complex and not straightforward to use, and application developers are not necessarily cryptography experts. The misuse of cryptography APIs has already been established as a common cause of many security vulnerabilities [6, 7, 11, 19].

To alleviate this problem, several studies employed dynamic and static analysis techniques to identify and investigate cryptographic errors in source code or binaries [2, 3, 17, 20, 21, 24]. Static Application Security Testing (SAST) has been shown to be effective in uncovering security-related bugs early enough in the software development process by applying it directly to the source code or compiled system code, without requiring its execution [18]. Veracode State of Software Security Report [25] shows that Static Application Security Testing (SAST) reported cryptographic issues in 60.4% of the analyzed Applications, which correspond to the top three software weaknesses reported by Static Analysis on Java projects.

In an empirical study, Afrose et al. [2] verified through benchmarks that SAST tools specialized to detect cryptographic misuses (e.g. CryptoGuard [21], CogniCrypt [17]) cover more rules and present higher recall than general-purpose tools such as SpotBugs and Coverity. Previous studies have mostly focused on quantitative methods to evaluate the performance of these SAST tools, often considering their precision and recall [2, 3, 17, 20, 21]. Nevertheless, little is known about how developers react to the warnings reported by SASTs and how these tools could be more effective in helping them to use crypto-APIs correctly. Acar et al. claim that quantitative strategies are valuable for identifying the kinds of mistakes seen most frequently in practice and illustrating the

0004-5411/2022/8-ART111 $15.00
https://doi.org/XXXXXXX.XXXXXXX

"pervasiveness of cryptographic errors" but they cannot disclose the source causes [1]. Moreover, their results usually comprise long lists of raw warnings or absolute values of software metrics that might not provide real insight to the stakeholders of the software products [22]. Hence, the community needs appropriate knowledge extraction studies on top of the results produced by SASTs to shed light on the reasons underlying the developers' success in using cryptography correctly [13].

In an exploratory study, Hazhirpasand et al.[14] first evaluated the perceptions of Java developers regarding crypto misuse. They used CogniCrypt to analyze 489 open source projects that employ JCA classes such as Cipher or MessageDigest and found that a shocking 487 projects had at least one crypto-misuse while only 2 had none. They submitted issues in 216 projects and received 140 feedbacks in which maintainers from 7 projects accepted to fix and 46 rejected by considering the non-security-sensitive context. Zhang et al. [26] have recently investigated how effectively pervasive tools detect crypto-API misuses and how open-source developers perceive the reported findings. In general, developers seem to be defensive when faced with crypto-misuses. Some claim that they are irrelevant since they appear in outdated or test code, whereas some intend not to fix them due to the effort involved. Even when developers agreed with the issues, in 30% of cases, this does not mean they will fix them. Another feedback from developers is that the hints given by the tools are generally not good enough to help developers in the fixing process.

Unlike previous works, we conducted two qualitative studies to deeply evaluate the developers' perceptions of the crypto-API misuses reported by Static Application Security Testing (SAST). (1) The first study was based on three Focus Groups conducted with Java Developers and Security specialists from companies of different domains — a financial institution, a research company from the agricultural field, and a Court and Judge from Brazilian states. The main objective was to understand how the development teams of these institutions use SAST and DAST, and what are the practitioners' opinions concerning the CogniCrypt and CryptoGuard tools. (2) In the second study, we used a survey approach, based on submitting issues in open-source organizations that develop Android applications. The objective in this scenario was to identify the developers' perception of the relevance of warnings generated by the CogniCrypt and CryptoGuard tools. The studies complemented each other, allowing triangulation of some information and generating independent insights. For example, from the focus group study the practitioners ..... In the study with open source communities, we identified...

TO BE FINISHED

2 RELATED WORK

To investigate common obstacles developers face when developing secure Java applications, Nandi et al.[19] have conducted an empirical study. They gathered and analyzed data from different sources. First, they gathered 100 random GitHub repositories that make use of Java Crypto-APIs to investigate what tasks developers try to solve using cryptography. Additionally, they analyzed the top 100 questions regarding cryptography and Java on Stackoverflow to aggregate the main areas of cryptography use developers struggle with. To solidify the results of their analysis, they conducted two developer surveys to learn about their personal struggles and uses of cryptography. Participants struggled with insufficient documentation of the APIs, their difficult design and the lack of abstraction they offer. They generally have issues with using some cryptographic algorithms, and the authors propose using tools that generate boilerplate code for standard use cases of crypto.

In a recent study Hazarpasand et al.[14] extended the research into this field by using the GitHub API to search for projects that use JCA libraries such as Cipher or MessageDigest. These projects were evaluated by CogniCrypt to find Crypto-API misuses and thus in turn learn what parts of JCA developers struggle with the most. CogniCrypt analyzed the use of 15 JCA APIS in 489 projects and found that a shocking 487 projects had at least one crypto-misuse while only 2 had none. The researchers manually reviewed the results to verify them, rejecting 6% of the misuses as false-positives. Developers seem to struggle with more than 50% of the 15 analyzed JCA APIs, seeing that the percentage of misuses when these APIs were used were over 51%. Some APIs such as Cipher, SIgnature, and MessageDigest only had a correct usage of at most 10%, showing that developers struggle with these APIs that are vital to many applications.

Many may consider the topic being explored and all the necessary work done to give developers the tools they need, but the reality is different. Zhang et al. [26] have recently launched an investigation into how effectively pervasive tools detect crypto-API misuses and how the developers perceive the reported findings. They compiled a list of 6 common static analysis tools, including tools developed in the research community as well as tools from the industry. The researchers tested these tools, allowing them to perform against well-known benchmarks MuBench,[4]CryptoBench[2] and the OWASP benchmark[10]. These benchmarks use crafted programs for the evaluation so that the total number of misuses is known and the precision (true misuses vs. false positives) can be calculated. The main takeaway from the evaluation is that there is no clear best performer. The researchers attribute this to the tools' different strengths in intra vs. interprocedural analysis. Zhang et al. followed up on the evaluation with a user study by letting the tools perform analysis on open-source projects and submit the found misuses to developers. In general, developers seem to be defensive when confronted with the found misuses, Some claim that the found misuses are not relevant since they are in outdated or test code, whereas some intend not to fix misuses due to the effort involved. In 30% of the cases, the developers agreed with the researchers, but even then this does not mean that the misuse will be fixed. The hints given by the tools on how to fix the misuses are generally not good enough to help developers fix the misuses.

Ami et al. argue that the comparison and evaluation of static analysis tools with benchmarks are impractical due to the scale of crypto protocols and APIs and the need for tools to be aware of all potential misuses. They propose a mutation-based framework to find implementation or design gaps in static analysis tools. The analysis found 19 flaws in the 9 tools. The flaws found in the tools were mapped to groups which include the incomplete analysis of code, incorrect resolution of values, e.g. parameters, and incorrect analysis of complex inheritance among other things.

3 RESEARCH METHOD

In this section, we present the settings of our study. Recall that this research aims to investigate the developers' perceptions regarding the usage of Static Application Security Testing (SAST) tools and the warnings these tools report about potential misuses of Cryptography APIs on Java and Android projects. More specifically, we aim to verify how software practitioners react to the warnings reported by SAST on the incorrect usage of crypto-APIs.

We adopted the Socio-Technical Grounded Theory (STGT) method [15] that comprises steps and procedures adapted from three different versions of traditional GT — Glaserian [12], Strauss-Corbinian [23], and Constructivist [5]. According to Hoda [15], STGT alternates between rounds of fundamental data collection and analysis processes and the emergence or structured formation of a theoretical model through advanced procedures of data collection, analysis, and theory development. These procedures predominantly employ inductive reasoning while also incorporating elements of deductive and abductive reasoning. STGT allows us to extract knowledge from software practitioners and security experts and establish our theory by grounding our findings according to their experiences.

Achei um pouco confusa essa próxima sentencça. Achava que memoing fazia parte de data analysis. Discutir com Luis

Figure 1 shows an overview of our research method and how we build our theory—in terms of a Data Collection and Analysis (DCA) approach involving the procedures of (theoretical) sampling, data collection, data analysis, and memoing [15].

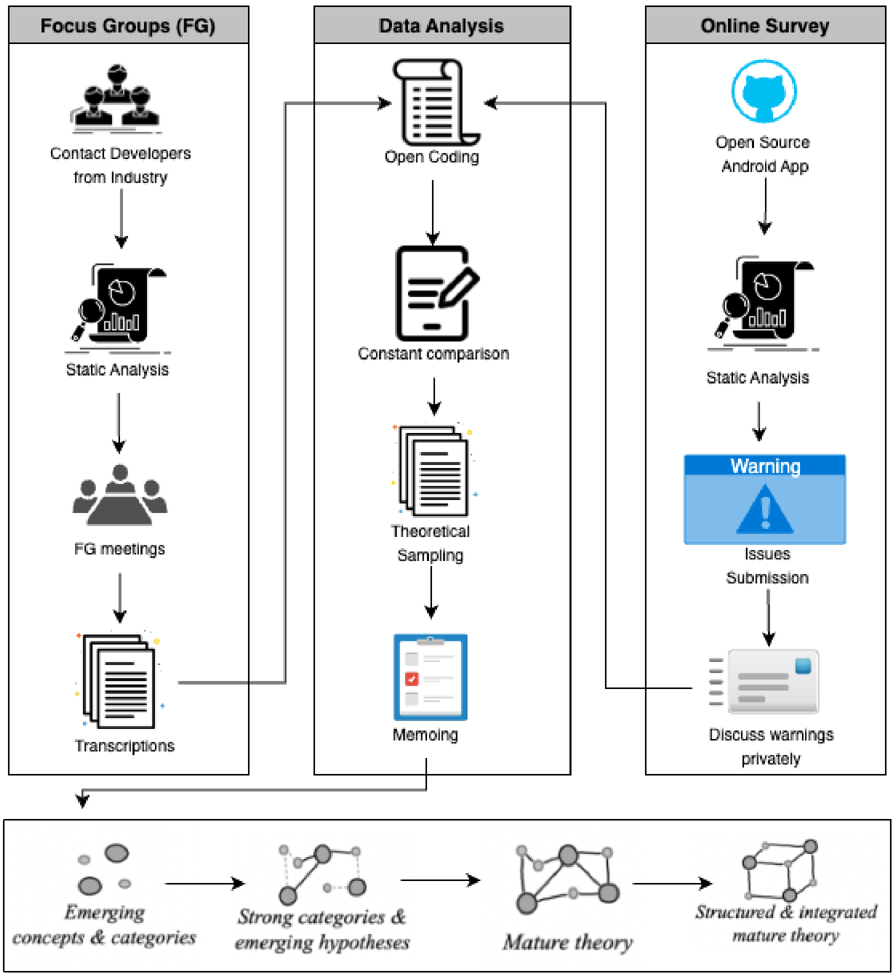

Fig. 1. Research Methodology Overview

3.1 Data Collection

For data collection, we leverage two approaches as shown in Figure 1: Focus Group method [16] with practitioners from the industry and an online survey based on the submission of issues and private Gists to open-source projects.

3.1.1 ***Focus Group with Practitioners from the industry.*** In this step, we contact practitioners from our network that have been working either as software engineers or security specialists to get their perceptions about (a) the use of SAST tools and (b) the warnings these tools generate. To this end, we use the Focus Group method []. Due to some companies' policies, several practitioners declined to participate in our study. Nonetheless, developers working in companies from different domains (research to finance) and IT complexity (dozens to thousands of systems) agreed to participate in this study.

In the first step of our FGs, we ask for the developers from the companies that accepted to participate in our study to execute the analysis with both SAST tools on some of their Java projects and share the reports with us. After receiving the warnings reported by the tools and analyzing the results, in the second step, we schedule the Focus Group (FG) session to discuss with them their perceptions about the relevance of the warnings. For the Focus Group, we invited from the companies not only practitioners that contributed to the development of the projects but also security experts. In the third step, we conduct and record the focus group session using Microsoft Teams. Each session lasted from 50 to 70 minutes. The first phase of the FGs involved only demographic questions to know more details about the participants (development/security experience, role in the company, use of SAST tools) and the projects (project domain, number of contributors, lines of code). Then, we follow the focus group by presenting each warning reported by the tools and thus assessing their perceptions w.r.t. them. Examples of focus group questions:

- In the context of this Class, do you think the use of the Hash *MD5* could reveal a security threat? If yes, what is the severity of this warning?
- Do you think you should fix this issue?
- How clear are the warning notification messages reported by these SAST tools?
- Do you have any recommendations to enhance the quality of these SAST tools?

The first Focus Group was applied with software practitioners from the Courts and Judges of the States, the Federal District and the Territories (TJDFT). In this study, the practitioners selected six relevant Java components from different domains and architecture to perform the analysis using CogniCrypt and CryptoGuard. However, both tools identify crypto API misuses for only one system— a court precatory system developed in Java Enterprise Edition (JEE) and JavaServer Faces (JSF) by another judge institution and that has been lately incorporated by TJDF with some adaptations. According to the developers, they already use SonarQube to detect (and help them fix) software vulnerabilities, hotspots, and code smells on the systems. For this reason, they were not expecting CogniCrypt and CryptoGuard find any crypto API misuse there. After executing the static analyses tools, we asked the practitioners to share the warning reports with us, so that we could prepare the introductory material for the focus group session.

We conducted the second Focus Group with software practitioners from a state-owned research company affiliated with the Brazilian Ministry of Agriculture. The corporation currently employs almost 10 thousand people with more than 24 hundred researchers. In this study, the practitioners selected two relevant Java components to perform the analysis using CogniCrypt and CryptoGuard. The first component is a Java library built for helping developers write systems that need to authenticate with the LDAP[1] and AD[2] mechanisms. The second component is also a Java library built as the reference architecture implementation for systems written in the Java Enterprise Edition (JEE) platform. We helped them to execute both SAST tools on their systems because they had some issues when executing the tools on their environment. After the execution, we selected some warnings to discuss with them in the focus group meeting.

The third Focus Group was executed with software practitioners and security specialists from Banco do Brasil, S.A., the second-largest banking institution in Latin America with more than 87 000 collaborators. Different from previous Focus Groups, we first spend a few weeks at the bank

[1]Lightweight Directory Access Protocol
[2]Windows Active Directory

headquarters in Brasília with the security specialists to understand how their processes with SAST tools work. Then, we asked them to select some JAVA artifacts that were also running in their pipeline, so we could compare the output of the CogniCrypt and CryptoGuard with the Checkmarx, the tool used in the bank, and they selected 48 java components. According to CogniCrypt, 15 projects presented at least one warning over 16 projects that used the JCA — the other projects do not use JCA APIs. CryptoGuard found problems in 23 of the 48 java components that we analyzed. Finally, We selected the most important warnings to discuss with the practitioners and get their perceptions in a Focus Group meeting.

Practitioner	Company	Experience	Role	SAST experience
C1	Court	20	Developer	SonarQube
C2	Court	15	Tech Lead	SonarQube
C3	Court	30	Developer	SonarQube
C4	Court	20	Scrum Master	SonarQube
R1	Research	18	Developer	SonarQube
R2	Research	22	Developer	None
R3	Research	20	Developer	SonarQube
F1	Finance	6	Developer	SonaQube
F2	Finance	12	Developer	SonaQube
F3	Finance	4	IT security engineer	Checkmarx
F4	Finance	10	Security administrator	SonaQube/Checkmarx
F5	Finance	15	IT coordinator	SonaQube/Checkmarx
F6	Finance	17	Cybersecurity specialist	SonaQube/Checkmarx

Table 1. Focus Groups participants demographics.

Table 1 presents all the participants' demographics that contributed to our first study.

3.1.2 ***Open-source Android projects***.

Besides the industry developers, we are also interested in understanding how the open-source community reacts to the warnings reported by crypto-API SASTs — and thus build a more general theory grounded in software practitioners' experiences. As such, we use a second data collection approach — issue submission — complementing the data we collected via focus groups in the first study.

In this study, we drive our project selection procedures considering the domain of Java/Android projects due to the requirement of the SAST tools. Firstly, we execute the static analysis with CogniCrypt [17] and CryptoGuard [21] on all open-source apps of the Money, Security, Connectivity, Phone & SMS, and System categories from F-droid, an installable catalog of FOSS (Free and Open Source Software) applications for the Android platform [1]. Our intuition for selecting those domains is that they might involve privacy and security concerns.

Since the purpose of this study is not to compare the tools but get developers' perceptions about the warnings reported by them, after collecting the misuses of crypto-APIs from the selected projects using both SASTs, we investigate the similarities and differences in their outputs. According to our data analysis, both tools are very similar — with an intersection of around 60% of the warnings.

In the second step, we submit issues to the GitHub repository of the projects reporting that we had possibly found vulnerabilities and documented the warnings in private gists for non-disclosure purposes. If the maintainers were interested in evaluating these gists, we asked for a suitable approach to share them privately. Since we found plenty of warnings — mean of 47.49 per project

[1]https://www.f-droid.org/en/packages/

and median of 18 — we randomly select only five gists for each project to share with them. After they evaluated the initial gists set, we asked whether they were interested in receiving a report with the remaining warnings. Figure 2 shows an example of documented gist that we shared with the developers.

CogniCrypt (report 1)

- Class: **hidden class name**
- Method: `md5`
- Line: 351
- Issue details: Constraint Error
  - Constraint error violating CrySL rule for `java.security.MessageDigest`
  - First parameter (with value MD5) should be one of {SHA-256, SHA-384, SHA-512}

Code

```java
public static String md5(String s) {
  try {
    MessageDigest digest = MessageDigest.getInstance("MD5");
    digest.update(s.getBytes());
    byte[] messageDigest = digest.digest();

    StringBuilder hexString = new StringBuilder();
    for(byte b: messageDigest) {
      hexString.append(Integer.toHexString(0xFF & b));
    }
    return hexString.toString();
  } catch (NoSuchAlgorithmException e) {
    Timber.e(e);
  }
  return "";
}
```

Questions:

(1) How likely might this warning reveal a security threat to this app?
   (a) Very unlikely
   (b) Unlikely
   (c) I cannot evaluate this
   (d) Likely
   (e) Very likely
(2) Are you likely to accept a patch that fixes this particular issue?

Fig. 2. Example of a secret gist sent to an Android project.

In addition to sharing the gists, we asked the developers to rate the perceived security impact of the issues and whether they would accept a patch to fix them. After collecting the responses from the developers, we extracted codes from them to gain insights into how developers dealt with the issues and perceived the input they received from CogniCrypt.

3.2 Data analysis

In the data collection process, we transcribe the feedback gathered from the focus group records and the discussions of the issues. Then, we start the *open coding* process. Here, we are interested in all codes arising at the early stage of the study. Furthermore, we apply *constant comparison* to identify the data key patterns. Besides, we employ *theoretical sampling* — "the ongoing process of assessing the emerging codes, concepts, (sub)categories, and hypotheses, and targeting specific data sources and types for collection in the upcoming iterations that are likely to help identify, develop, and saturate them while filling any theoretical gaps" [15]. Finally, we start the *basic memoing* process, documenting the emerging concepts and categories. The early data analysis stage allows a progressive condensation of the massive portions of initial codes generated during open coding employing constant comparison and memoing into concepts and categories and limiting the focus of analysis.

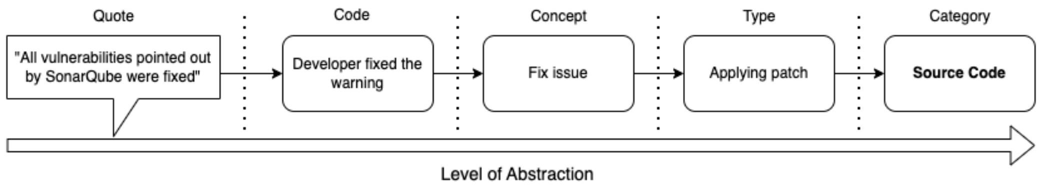
Quote
"All vulnerabilities pointed out by SonarQube were fixed"
Code
Developer fixed the warning
Concept
Fix issue
Type
Applying patch
Category
Source Code
Level of Abstraction

Fig. 3. Levels of abstraction example to generate the SAST usage Theory.

3.3 Selecting Cryptographic Vulnerability Detection Tools

Our focus in this study is to evaluate the perceptions of Java and Android developers due to the existence of advanced SAST tools. To collect the crypto-APIs misuses from Java and Android projects, we selected two open-source research tools that are actively maintained: the CryptoAnalysis component of CryptoGuard [17], hereafter CogniCrypt, and CryptoGuard [21]. According to Afrose et al. [3], these tools are specialized to detect cryptographic misuses, cover more rules and present better performance than general-purpose tools (i.e., SpotBugs, Coverity).

CogniCrypt takes the rules provided in the specification language CrySL as input and performs a static analysis based on the specification of the rules. CrySL [17] is a domain-specific language that allows cryptographic experts to specify the correct usage of cryptographic libraries. We considered the CrySL rules from Java Cryptography Architecture (JCA), the official framework for working with cryptography in Java [19].

CryptoGuard is a static analysis tool that relies on program slicing with novel language-based refinement algorithms [21]. The authors say it significantly reduces the false positive rate which is a typical problem for static analysis. Furthermore, CryptoGuard covers 16 cryptographic rules, including rules for mining possible misuses of the JCA crypto-API.

4 THEORY OF USING SAST TO DETECT CRYPTO-API MISUSES

This section presents the theory that emerged from our exploratory studies. We first report the outcomes of the Focus Groups conducted with Java developers and security specialists from industries of different domains — financial bank, agricultural research corporation and Courts. After that, in section 4.4 we present the results of our second study, the survey conducted with the Android Open-Source community.

4.1 SAST Benefits

Developers argue that **SAST tools evaluate and support code quality**. For instance, the participants mention that some warnings from SAST tools must be fixed before deploying a system in the production environment. To make this happen, it is necessary to configure the Continuous Integration (CI) and Continuous Delivery (CD) pipelines to prevent a system go to production if the scores assigned to the system, either by static and dynamic analysis tools, do not comply with well-defined thresholds.

> **FG - Finance**
> *"When we catch these errors, we stop the development and correct the code. We have a specific score for the package to go into production, and the CD pipeline requires this score to be met."*

> **FG - Finance**
> *"We use SonarQube here in our build pipeline, currently working with Java. So, SonarQube provides various levels of warnings (warning, info). If there is a critical issue in the analysis it performs, it does not allow the build to proceed, so we have to go there and fix it."*

> **FG - Court**
> *"We started using SonarQube more extensively about two years ago. In the past year, we have been actively addressing the vulnerabilities and hotspots that it indicates."*

As a consequence, developers consider that **SAST tools bring more security when deploying the system.** For example, finance company practitioners discuss that eventually a system may go into production even without complying with the thresholds. Nonetheless, this only happens when a technical leader assumes the risks and explicitly authorizes the deployment. This somehow protects the development teams.

> **FG - Finance**
> *"SonarQube provides a score to determine when to deploy a system. It also provides a score to determine who can authorize that deployment (e.g., a technical leader, a system architect, or even a stakeholder). So, it gives us great security because when developers submit a system to deploy with a low score, even with stakeholder approval, the deployment will not happen. The development team needs to fix the system — improving test coverage, and everything necessary to make the system more reliable. Once it reaches a level expected by SonarQube, most of our deployments can be authorized by a technical leader, which brings us more confidence that the code is reasonably okay, more reliable, and we can deploy with greater security and less risk."*

Another benefit developers mention, is that SAST tools help **developers acquire security-specific knowledge**

4.2 SAST Limitations and Improvements Opportunities

According to developers, one standard limitation is that **SAST tools often produce numerous false positive warnings**. For example, the security team at the finance company emphasizes the need for a manual analysis of these warnings to effectively minimize the number of false positives before forwarding them to developers for code fixes. Moreover, before implementing the SAST in the pipeline process, it is necessary to define and configure the rules and thresholds depending on the system security requirements.

> **FG - Finance**
> *"We review the results before sending them to the developer, precisely to remove the number of false positives, which is often very high. So, we perform this task to spare the developer from trying to fix something that is actually a false positive."*

For this reason, developers argue that **tool customization and rule prioritization** are crucial aspects that should be improved in the SAST tools. Depending on the system context, the security requirements are not the same and the tools should be able to adapt to the specificity of each code scenario.

> **FG - Finance**
> *"I believe that the criticality within the tool policies is a key point we discuss and that helps us. What we know is that we cannot classify everything as critical within the tool because it would halt the development process. So we need to determine what to classify as high, medium, and low criticality, as even the medium level deducts points."*

For instance, financial institution specialists say that the analysis of criticism

> **FG - Finance**
> *"We are also working on prioritizing and refining security rules and policies because we know that, when these tools are deployed, they generate around 60% false positives. So, we are analyzing what is happening out there so that we can prioritize rules to manage the security of the SAST process."*

Another concern relates to the difficulties developers face while correcting the warnings reported by these SAST tools. As they say, tools often report **generic warnings with difficult interpretation**, and developers need to search for a solution outside the box, usually in gray literature such as Stack Overflow or Reddit. For instance, when we present some warnings to the participants, most of them struggle to find out what the problem is and how they could fix it. Furthermore, CogniCrypt and CryptoGuard do not label the severity of warnings in their reports.

> **FG - Research**
> *"I need to think a while. We can mark it [the warning] as a false positive. I'll do some more research on this Hostname Verifier to see what the source of the issue is. Perhaps the way it was presented here was not clear to us either."*

> **Android Study**
> *"One thing I did miss is a suggestion of fix, it is super useful when a tool points a problem and a possible solution or at least a link to a page explaining better the problem."*

> **Participant F3**
> *"In practice, I understand that the developer will seek this knowledge within the tool itself, where they have the initial support to make the necessary corrections. Sometimes the information provided by these tools is somewhat generic, and we realize that. Therefore, the developer must leave the environment to investigate and try to understand how to correct."*

> **FG - Finance**
> *"The problem is that sometimes errors arise without any explanation on how to fix them, and then we get stuck. Okay, it is telling you what the error is, you research it, and there is no solution on how to resolve it. Then I feel stagnant, not knowing what to do, and I waste some time. If I manage to solve it, well. If not, it remains to be adjusted in the future. So if there is a way to pinpoint the error and provide suggestions for improvement, that would be perfect."*

> **FG - Finance**
> *"If someone doesn't have much experience with cryptography and needs to implement something, they will struggle a lot when receiving a message like this."*

> **FG - Finance**
> *"Looking at that message, I am already confused, and since I don't understand much about cryptography, it would take me longer to understand. I understood what happened because you explained it earlier, but if I read that message, I wouldn't know if my code or the algorithm is broken with the message: **Found broken crypto schemes**."*

Considering the Android study, we receive many responses suggesting that the SAST tools report crypto-API misuses coming from external libraries. According to the Android study, the inability to differentiate issues from the app from issues from external libraries is one of the main limitations of CogniCrypt and CryptoGuard.

> **Android Study**
> *"I cannot judge this, but I can say with certainty that this is inside zip4j. Please consider reporting this directly to the external library repository. None of the issues reported so far had to do with Catima itself, only libraries."*

Android Study
"All of these warnings are in dependencies of the project I use, so they don't pertain to my code."

Android Study
"In a perfect world, the tool would have a view showing two groups: 1. warnings only related to our own code; 2. warnings in libraries that our code depends on, with an indication whether it is inside a transitive or direct dependency, and with suggestions on which dependency to update to fix the problem."

4.3 Process Improvements

The results of the focus groups reveal categories recommending how SAST tools should be integrated into the development practices of the companies. For example, participants in the focus group with the financial institution recommend a more collaborative culture between the security and development teams, perhaps in alignment with DevSecOps initiatives. Balancing security concerns with the demand for fast software releases presents a challenge. On the one hand, there is a need to emphasize the importance of security to development teams. However, security teams must identify the necessary measures to mitigate threats without impeding developers or causing excessive disruption to daily operations.

FG - Finance
"Usually, the security team is "looked down upon," which hampers the process and makes it bureaucratic. Our biggest challenge is precisely trying to understand the developer's perspective in a way that a security complain doesn't impact the development workflow but also avoids being careless. Take the process of prioritizing SAST policies, for example, what is critical and what is not? We have to consider the developer's side, the manager's side, the supervisor's side. So one of our biggest challenges is not being seen as the "annoying" team and at the same time not being careless, finding an optimal balance. Often, the security team does not know the pressures over the development team, so this understanding of each other's day-to-day is essential to improve this type of process."

FG - Finance
"If there is any high and/or critical vulnerability, it receives full attention. What we always discuss is that we cannot solely focus on security and demand immediate fixes because we need to consider the developer's perspective within the room and strive to find that balance."

It is also necessary to implement *change impact analysis*. Depending on the changes needed to fix a crypto API misuse, many other components (and even systems) might have to change. Even very simple changes to fix a crypto API misuse might have to be acommodate in other systems. If a misuse happens in an authentication component, for instance, changing the algorithms for authentication might require change in every other component that needs to authenticate users. Risk assessments are also necessary, since the costs to fix an issue might be higher than the potential lost with the vulnerability.

> **FG - Finance**
> *"I would have to analyze the systems that depend on my code because if I implemented such a fix [recommended by CogniCrypt], it could impact every other dependent system. So, depending on the level of coupling we have between the systems, fixing this warning wouldn't be a straightforward task. That's how I see it. It might seem like a simple change, but it can become complex depending on the dependencies involved."*

> **FG - Finance**
> *"Certainly, fixing this warning is not a trivial matter, as such a change would have significant impacts. For instance, do you remember that bug in Log4J? Well, the security experts conducted an assessment of all the projects using Log4J and enforced an update. It's not exactly the same as this case, as this one involves encryption, but they managed to map and notify everyone affected by the broken Log4J."*

TODO: continue from here

The participants also agree that these tools should be integrated to CI workflow ...

4.4 Perceptions of Open-source Android developers

After studying the way developers and security professionals in big companies use static analysis tools and perceive the results they produce, we additionally performed an analysis on open-source android apps and contacted the developers to get their input. Most prominently, the result of the analysis showed that a lot of the issues raised by CogniCrypt were not part of the android app itself, rather they lie in 3rd party libraries used by the apps. From the 20 open-source projects we contacted 12 had issues raised by CogniCrypt that are actually issues that lie in the code of a 3rd party library. Developers dealt with these security concerns due to external libraries in mainly 2 different ways: 3 developers immediately updated or said they would update the dependencies in question, while 2 developers said that since the 3rd party libraries are from big tech companies they trust them implicitly and don't think the issues are real problems In 8 cases developers assessed the issues and classified them as no threat to their application. Either because they thought the issue was a false positive of CogniCrypt or because they do not deem the issue as security-critical in the context of their app. Exemplary of this would be a developer dismissing an issue raised over the use of MD5 hashes because they think the hash is not used in a security-relevant context. Surprisingly in only 4 cases, the developers acknowledged the validity of an issue directly, yet only one of these developers tried to create a fix. The others did not intend to fix the issue.

An important finding of this survey was that many of the developers had difficulties completely understanding the issues raised by CogniCrypt. In many cases, it was not clear to the developer why a piece of code was a problem, just by reading the explanation given by CogniCrypt. Further, developers wished that CogniCrypt would give further explanations about the issue and direct hints on how to best fix it. Additionally, some developers raised the suggestion of separating CogniCrypt issues based on whether they are part of the scanned app or a 3rd party library.

5 QUANTITATIVE DATA ANALYSIS ON ANDROID APPLICATIONS

Here we present the quantitative results of our exploratory study on Android Applications. As we said before, we selected 307 Android applications from 6 categories from F-droid [1] to execute

[1]https://www.f-droid.org/en/packages/

the both CogniCrypt and CryptoGuard. From these apps, CogniCrypt found warnings in 195 and CryptoGuard, in 298. Wagner, poderia descrever os resultados das suas análises aqui? (lamaral)

The next table show the distribuition the apps selected between the categories and the quantities of apps without warnings in each of them.

Category	**Number of apps**	**CogniCrypt**	**CryptoGuard**
Connectivity	58	20	3
Finances	90	25	2
Internet	39	7	0
Security	47	16	1
Sms-Phone	18	10	2
System	55	34	0
Total	307	112	8

Table 2. Apps by category without warnings in CogniCrypt and CryptoGuard

As we can see, the number of apps without warning in CogniCrypt is greater than CryptoGuard. The category System is the one with the higher difference between them, 34. Connectivity and Finances categories also has a high number of apps without warnings, 20 and 25, respectivily.

Despite that, the number of warnings founded by CogniCrypt is greater than in CryptoGuard, described in the table below.

Category	**CogniCrypt (a)**	**CryptoGuard (b)**	**Difference (a - b)**
Connectivity	1768 (16.02%)	1124 (22.64%)	644 (10.61%)
Finances	3087 (27.97%)	1687 (33.98%)	1400 (23.06%)
Internet	3407 (30.87%)	916 (18.45%)	2491 (41.02%)
Security	1780 (16.13%)	553 (11.14%)	1227 (20.21%)
Sms-Phone	428 (3.88%)	171 (3.44%)	257 (4.23%)
System	566 (5.13%)	513 (10.33%)	53 (0.87%)
Total	11 036 (100.00%)	4964 (100.00%)	6072 (100.00%)

Table 3. Founded warnings in CogniCrypt and CryptoGuard

As we can notice in the table above, the CryptoGuard execution was able to found warnings 4964 while the CogniCrypt founded 11 036 warnings, a diference of 6072 (or 122.32%) warnings between both. The categories Finances and Internet concentrated 58.8% (6494) of CogniCrypt warnings and categories Finances and Connectivity concentrates 56.6% (2811) of CryptoGuard warnings. The greater difference between the founded warnings was noticed in the categories Internet (2491) and Finances (1400), representing 64.1% of differences.

Considering the finds above, we looked at the average amount of alerts founded per app. The findings are shown in the chart below.

As expected, the average amount of alerts is higher in CogniCrypt than in CryptoGuard. The difference is even greater when we disregard apps without warnings. The category Internet has the greater average amount of warnings in both of tools, with a higher value in CogniCrypt: 106.5 warnings per app. In second and third places we have Security and Sms-Phone in CogniCrypt findings and Internet and Connectivity, considering CryptoGuard founded warnings avergare amount.

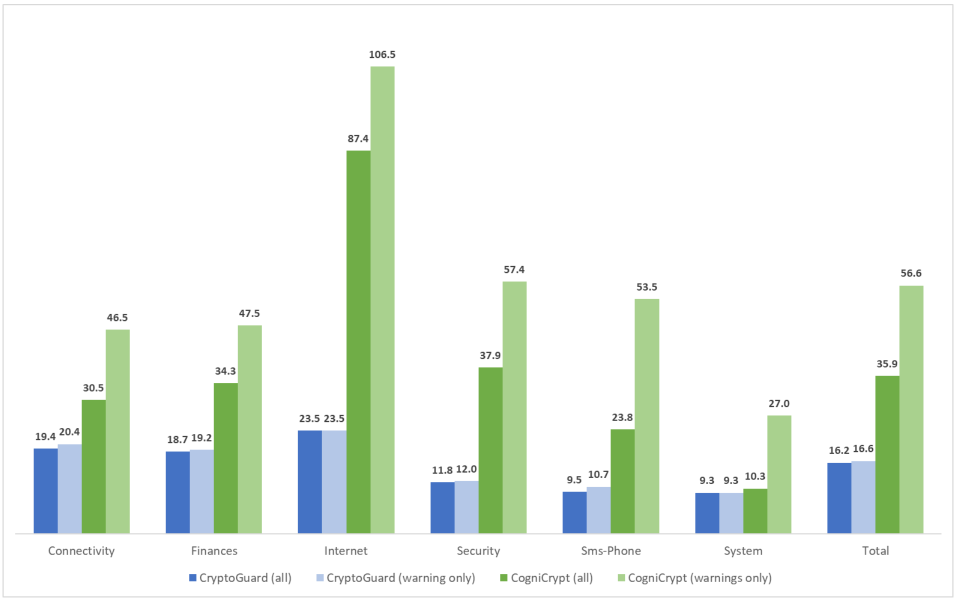

Fig. 4. Average amount of alerts founded per app

5.1 Warnings from Third-party Libraries

The experiment presented in this article earlier showed that when trying to contact the developers behind the code some of them replied to us showing that the piece of code with some kind of misuse or vulnerability presented by either CogniCrypt or CryptoGuard was not their fault and stemmed from external libraries used throughout the code.

Having this concern in mind we have decided to tackle this problem. First, we studied what we could do to identify if the application being analyzed was using an external library with some kind of vulnerability. For that, we have found the article 'Automated Third-Party Library Detection for Android Applications: Are We There Yet?' written by Zhang in which briefly describes 5 of the most used programs to identify if exists code that was imported, them being: LibID, LibRadar, LibScout, LibPecker and ORLIS. Each of them attack the problem in different ways but all of them were rated within four categories. Effectiveness, efficiency/scalability, obfuscation resilience, ease of use. Before I describe them, as Zhang did, I'll be using tpl for the third-party library. The first one, effectiveness evaluates if the tools above can, in fact, explicit if the application on trial have or don't have tpls. In fact, in this study, they noted that the most effective one, LibScout, could only identify 49% of the tpls in a given application with 97% of precision, the second in line was LibId with 45% with only 85% precision, followed by LibRadar with 40% with almost 98% of precision. LibPecker is in fourth and ORLIS is the worst in this category. In all of them, we must not ignore the fact that there were false positives. -Mainly caused for two reasons, tpls being dependent on other tpls and closed version of codes being very similar bitwise which may cause problems for LibScout and LibRadar which uses hierarchy as a supplementary feature to identify tpls. The second criterion, efficiency/scalability, determines whether the solution is viable on large scale or not. Later after reviewing the five tools, this one was one of the main reasons for us choosing libScout as the tool used to scout for tpls. LibRadar, the first one in this category, could run with ease each application in 5 seconds or so. LibScout took 80 seconds on average. ORLIS in the third took 23 minutes. LibId and LibPecker took 4 hours on average and LibId couldn't handle an application with a lot of external libraries as there were memory leaks and the run just crashed. As this research would use several apps in different categories we choose to use the first two for the experiment: LibRadar and LibScout. The third criterion, obfuscation-resilient capability attacks the obfuscation problem in two different ways as there are a lot of ways to obfuscate something. LibPecker won with ease this category exceeding all the other four. As said earlier, due to the time it took to evaluate one app, we decided to use libScout, but this doesn't mean that libScout is bad at finding tpls on obfuscated code, just not the greatest. Finally, the last criterion is ease of use. This category was not considered so deeply, since we scientists must have the ability to not only analyze a new tool but also learn to use it if necessary for some research. Anyway, LibScout and LibRadar outperformed the others competitors.

The second step, use the selected tools to find if in a given application we could identify third-party libraries. We started with ten selected apps using both LibRadar and LibScout. LibRadar and LibScout results for the pilot were not similar. LibScout identified a lot more tpls than LibRadar due to the first one deeply analyzing the application whereas the last one uses a clustering method to categorize if it is or not a tpl. To run the LibRadar we had to set up a Redis server and load a sample so LibRadar can compare if the application on trial uses or not external libraries and the last time the sample was updated with a good cluster example was in 2017. Most of the apps analyzed were from 2017 and beyond. For that reason, libScout became the focus.

In the third step, we have run CogniCrypt and CryptoGuard on five different categories of apps. The categories were: System, Sms, Security, Finances and Connectivity. For each app of the category selected we compared libScout result and CogniCrypt and CryptoGuard results. LibScout results runned on non-obfuscated code mapped a grand majority of libraries used in the apps and throughout comparison with CogniCrypt and CryptoGuard results we used a script made in ruby to verify if the library found by libScout was used in the app. We must not forget that libScout is not a perfect tool and it may find some false positives and besides that, the script only matched the name of the library found by libScout with the name of the library used in the app. For example, if libScout found a library called 'com.google.android.gms:play-services-location' and the app used a library called 'com.google.android.gms:play-services-location-beta' the script would not match them. Nonetheless we have found that libScout found a lot of libraries used in the apps and that the script matched the majority of them. We also had problems when the CogniCrypt or CryptoGuard couldn't map a vulnerability, causing the script to match nothing as CogniCrypt and CryptoGuard results were empty. For that reason we separeted the results in two different files, one with the apps that had a match and another with the apps that didn't. All the procedure to The following table shows an example of the results for the category System for both CogniCrypt and CryptoGuard.

As we can see, the number of apks with both external and native libraries found on CryptoGuard is much higher than on CogniCrypt. However, both CogniCrypt and CryptoGuard found differents results for tpls for the same app. There were cases where CryptoGuard found none tpl or native libraries, while CogniCrypt found some as well as the other way around. For this reason we decided to present each result separately and, as well, decided to use all apps that were found by both tools. There are cases in wich CryptoGuard found at least one tpl where CogniCrypt found none. All cases where both tools couldn't found any tpl or native library were removed from the results.

5.2 Results for CryptoGuard

The following table shows the average and the standard deviation for cryptoguard subset.

For each category on this analysis we have the average and the standard deviation for the number of tpls and native libraries found by CryptoGuard. As we can see, the average number of tpls found by CryptoGuard is similar to the average number of native libraries found. However we can see

CryptoGuard	System	Sms	Security	Finances	Connectivty
Average tpl	1,7903	2,1111	3,7441	5,3888	6,0357
Standard Deviation tpl	1,8955	4,5618	6,6836	11,0257	10,9527
Average native	1,6935	6,1667	3,9534	5,4351	7,4821
Standard Deviation native	3,7858	19,2300	4,3147	12,0786	17,1982

CogniCrypt	System	Sms	Security	Finances	Connectivty
Average tpl	0,9354	4,6667	13,8372	10,7407	4,8571
Standard Deviation tpl	3,0237	12,1993	46,4091	43,5478	12,4318
Average native	4,0322	39,5544	11,4883	9,8889	12,3750
Standard Deviation native	12,6917	40,7012	23,8201	22,3679	39,6780

due to the standard deviation that there are some apps that have a lot of tpls or native libraries where others have none tpl found or none native library found.

5.3 Results for CogniCrypt

The analysis for CogniCrypt is similar to the one for CryptoGuard. However, we can see that the average number of tpls found by CogniCrypt is lower for the first category (system), and much higher for the other four categories. The same happens with the average number of native libraries found by CogniCrypt.

6 DISCUSSION

TO BE FINISHED

7 THREATS TO VALIDITY

8 FINAL REMARKS

TO BE FINISHED

REFERENCES

[1] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L Mazurek, and Christian Stransky. 2017. Comparing the usability of cryptographic apis. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 154–171.

[2] Sharmin Afrose, Sazzadur Rahaman, and Danfeng Yao. 2019. Cryptoapi-bench: A comprehensive benchmark on java cryptographic api misuses. In *2019 IEEE Cybersecurity Development (SecDev)*. IEEE, 49–61.

[3] Sharmin Afrose, Ya Xiao, Sazzadur Rahaman, Barton P Miller, et al. 2021. Evaluation of Static Vulnerability Detection Tools with Java Cryptographic API Benchmarks. *arXiv preprint arXiv:2112.04037* (2021).

[4] Sven Amann, Sarah Nadi, Hoan A Nguyen, Tien N Nguyen, and Mira Mezini. 2016. MUBench: A benchmark for API-misuse detectors. In *Proceedings of the 13th international conference on mining software repositories*. 464–467.

[5] Kathy Charmaz. 2006. *Constructing grounded theory: A practical guide through qualitative analysis*. sage.

[6] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. 2013. An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 73–84.

[7] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. 2012. Why Eve and Mallory love Android: An analysis of Android SSL (in) security. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 50–61.

[8] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. 2011. *Cryptography engineering: design principles and practical applications*. John Wiley & Sons.

[9] The OWASP® Foundation. [n.d.]. . https://owasp.org/Top10/

[10] The OWASP® Foundation. [n.d.]. . https://owasp.org/www-project-benchmark/

[11] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. 2012. The most dangerous code in the world: validating SSL certificates in non-browser software. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 38–49.

[12] Barney G Glaser and Anselm L Strauss. 2017. *Discovery of grounded theory: Strategies for qualitative research*. Routledge.

[13] Mohammadreza Hazhirpasand, Mohammad Ghafari, Stefan Krüger, Eric Bodden, and Oscar Nierstrasz. 2019. The impact of developer experience in using Java cryptography. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–6.

[14] Mohammadreza Hazhirpasand, Mohammad Ghafari, and Oscar Nierstrasz. 2020. Java Cryptography Uses in the Wild. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–6.

[15] Rashina Hoda. 2021. Socio-technical grounded theory for software engineering. *IEEE Transactions on Software Engineering* (2021).

[16] Jyrki Kontio, Johanna Bragge, and Laura Lehtola. 2008. The focus group method as an empirical tool in software engineering. In *Guide to advanced empirical software engineering*. Springer, 93–116.

[17] Stefan Krüger, Johannes Späth, Karim Ali, Eric Bodden, and Mira Mezini. 2019. Crysl: An extensible approach to validating the correct usage of cryptographic apis. *IEEE Transactions on Software Engineering* (2019).

[18] Gary McGraw. 2004. Software security. *IEEE Security & Privacy* 2, 2 (2004), 80–83.

[19] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. 2016. Jumping through hoops: Why do Java developers struggle with cryptography APIs?. In *Proceedings of the 38th International Conference on Software Engineering*. 935–946.

[20] Luca Piccolboni, Giuseppe Di Guglielmo, Luca P Carloni, and Simha Sethumadhavan. 2021. Crylogger: Detecting crypto misuses dynamically. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1972–1989.

[21] Sazzadur Rahaman, Ya Xiao, Sharmin Afrose, Fahad Shaon, Ke Tian, Miles Frantz, Murat Kantarcioglu, and Danfeng Yao. 2019. Cryptoguard: High precision detection of cryptographic vulnerabilities in massive-sized java projects. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2455–2472.

[22] Miltiadis Siavvas, Erol Gelenbe, Dionysios Kehagias, and Dimitrios Tzovaras. 2018. Static analysis-based approaches for secure software development. In *International ISCIS Security Workshop*. Springer, Cham, 142–157.

[23] Anselm Strauss and Juliet Corbin. 1990. *Basics of qualitative research*. Sage publications.

[24] Alexander Trautsch, Steffen Herbold, and Jens Grabowski. 2021. Are automated static analysis tools worth it? An investigation into relative warning density and external software quality. *arXiv preprint arXiv:2111.09188* (2021).

[25] Veracode. 2021. Veracode State of Software Security Report. https://www.veracode.com/sites/default/files/pdf/resources/sossreports/state-of-software-security-v12-nwm.pdf. Accessed: 2022-02-18.

[26] Ying Zhang, Md Mahir Asef Kabir, Ya Xiao, Danfeng Daphne Yao, and Na Meng. 2022. Automatic Detection of Java Cryptographic API Misuses: Are We There Yet. *IEEE Transactions on Software Engineering* (2022), 1–1. https://doi.org/10.1109/TSE.2022.3150302