A Preliminary Conceptualization and Analysis on Automated Static Analysis Tools for Vulnerability Detection in Android Apps

Giammaria Giordano, Fabio Palomba, Filomena Ferrucci
Software Engineering (SeSa) Lab - Department of Computer Science, University of Salerno, Italy
giagiordano@unisa.it, fpalomba@unisa.it, fferrucci@unisa.it

matter*Abstract*—**The availability of dependable mobile apps is a crucial need for over three billion people who use apps daily for any social and emergency connectivity. A key challenge for mobile developers concerns the detection of security-related issues. While a number of tools have been proposed over the years—especially for the ANDROID operating system—we point out a lack of empirical investigations on the actual support provided by these tools; these might guide developers in selecting the most appropriate instruments to improve their apps. In this paper, we propose a preliminary conceptualization of the vulnerabilities detected by three automated static analysis tools such as ANDROBUGS2, TRUESEEING, and INSIDER. We first derive a taxonomy of the issues detectable by the tools. Then, we run the tools against a dataset composed of 6,500 ANDROID apps to investigate their detection capabilities in terms of frequency of detection of vulnerabilities and complementarity among tools. Key findings of the study show that current tools identify similar concerns, but they use different naming conventions. Perhaps more importantly, the tools only partially cover the most common vulnerabilities classified by the Open Web Application Security Project (OWASP) Foundation.**

Index Terms—**Software Vulnerabilities; Android Apps; Automated Static Analysis Tools; Empirical Software Engineering.**

I. Introduction

The last decades and, most notably, the recent years have seen a drastic change in the way people communicate and interact among them. Around 80% of the global population owns a smartphone [2] and about 70% of these smartphones rely on the ANDROID operating system [6]. The diffusion of this operating system (OS) is favored by multiple factors, including the availability and marketing of mobile apps through the online app store [26]. In this context, previous research has pointed out that ANDROID apps can be affected by severe vulnerabilities that can impact both user privacy and security [16], [28], [34]. For this reason, several automated static analysis tools have been proposed to detect security concerns and assist mobile developers in improving their applications [21]. Nevertheless, in our research, we observed a lack of knowledge about the real support provided by these tools. In particular, it is unclear the set of problems that these tools can detect and how they behave when detecting vulnerabilities, e.g., whether their analysis fails in certain cases, the most common vulnerabilities identified, and to what extent different tools cover different vulnerabilities. An improved understanding of these aspects is crucial to let developers be aware of what kind of problems can be currently detected, other than letting them (1) more wisely select the tools to employ, (2) evaluate on complementing more tools, or (3) even understand whether current tools can actually identify vulnerabilities that are becoming more and more popular and harmful nowadays.

This paper performs the first step toward enlarging the body of empirical knowledge on the matter. We focus on three automated static analysis tools, i.e., ANDROBUGS2,[1] TRUESEEING,[2] and INSIDER,[3] to elicit a taxonomy of security-related concerns detectable with these tools. Afterward, we execute the tools on 6,500 free apps to assess the number of vulnerabilities the tools can detect and the complementarity among them.

The main results of the study indicate that in most cases the tools can detect the same vulnerabilities but using a different vocabulary, causing possible misunderstandings. The tools are also complementary, which implies that developers should select tools based on the specific categories of vulnerabilities they would detect. Lastly, the considered tools only partially cover the most widespread vulnerabilities classified by the Open Web Application Security Project (OWASP) Foundation.

To sum up, our paper provides the following contributions:

1) A large-scale empirical investigation into the support provided by three state-of-the-practice tools for the detection of security-related concerns;
2) An empirical analysis of the complementarity among the three considered tools, which might open new research directions connected to their combination;
3) A publicly available replication package [7], which contains all data and scripts employed to address and extend our research questions.

II. Related Work

Researchers have been focusing on the ANDROID platform, as its open-source nature eased the definition of empirical investigations on the matter [14], [15]. For similar reasons, our study revolves around ANDROID; in the remainder of the section, we discuss the literature connected to that.

[1] https://github.com/androbugs2/androbugs2
[2] https://github.com/alterakey/trueseeing
[3] https://github.com/insidersec/insider

First, a significant amount of previous works designed automated techniques to identify vulnerabilities: they establish the level of security of mobile apps with respect to various vulnerability types based on the analysis of various static constructs [32], textual analysis [13], or data flow (e.g., use-def relations [11]). A systematic overview of these approaches has been recently proposed by Li et al. [22].

From an empirical standpoint, existing studies focused on understanding Android vulnerabilities. Linares-Vásquez et al. [25] classified the vulnerabilities affecting the Android OS. In contrast, Gao et al. [18] focused on the evolution of software vulnerabilities from the perspective of mobile apps. Additional studies pertained to third-party libraries [31], [37] and how they might potentially threaten software security aspects of source code. When comparing our work with those just discussed, we can observe that none of them targeted the capabilities of static analysis tools but instead focused on the analysis of aspects connected to the maintenance and evolution of software vulnerabilities.

Finally, a few taxonomies of software vulnerabilities in mobile apps have been previously proposed. Sadeghi et al. [30] conducted a systematic literature review on the research on mobile app security, defining a taxonomy of the vulnerabilities treated by researchers over the years. Qamar et al. [29] and Mirza et al. [27] defined context-specific taxonomies that cover the vulnerabilities affecting the mobile banking domain. These papers relate to the first objective of our study. Nonetheless, our goal is to derive a taxonomy of vulnerabilities from the perspective of the automated static analysis tools to understand what the vulnerabilities that these tools can detect are. Hence, in this case, our study can be seen as complementary.

III. Research Methodology

The *goal* of the empirical study was to assess the current support provided by existing automated static analysis tools in terms of vulnerability detection in Android applications, with the *aim* of providing initial insights on the capabilities of these tools in terms of security issue types detected and their detection capabilities. The *perspective* is of both researchers and practitioners: the former is interested in the capabilities of existing tools to evaluate whether and which aspects should be further improved; the latter are interested in understanding how existing tools can support them in their daily tasks. We set out two main research questions.

First, we analyzed the types of vulnerabilities that current tools can detect through static analysis. Our goal was to elicit a *taxonomy* of the security issue types whose identification is supported by the existing instruments. An improved understanding and investigation of this research angle are required to let researchers be aware of where to invest future research efforts, other than to let practitioners know which tools can be used to detect specific vulnerabilities, hence easing the selection of the proper tools to use in their contexts. This reasoning led to the definition of our first research question (**RQ**$_1$):

RQ$_1$. *What are the vulnerability types identified by existing automated static analysis tools for mobile apps?*

Once we had identified the types of security-related issues whose detection is supported by existing tools, we sought to provide insights into their detection capabilities. The aim is to elaborate on the extent to which existing tools can detect vulnerabilities in the first place and, if so, with which frequency they can detect the vulnerabilities types identified in **RQ**$_1$ and which complementarity exists among them. This perspective is key to understanding the extent to which different tools can collect and provide information on different security-related concerns. From the practitioner's perspective, this analysis would ease further the tool selection process, which might be done by considering the capabilities of the tools. Also, researchers may exploit our findings to assess where additional improvements are needed, e.g., by understanding the characteristics of the tools that should be improved. Therefore we asked our second research question (**RQ**$_2$):

RQ$_2$. *What are the capabilities of existing automated static analysis tools in terms of mobile app analyzability, frequency of detection, and complementarity among them?*

The expected outcome of our analysis is an initial conceptualization and analysis of the current state of the art in vulnerability detection in Android applications. When conducting the empirical study, we followed the empirical software engineering principles and guidelines described by Wohlin et al. [35]. Additionally, in terms of reporting, we employed the *ACM/SIGSOFT Empirical Standards*.[4] For the sake of replicability and reproducibility, we made available in the online appendix [7] datasets, scripts, and the additional analysis that address our research questions.

A. Context Selection

The *context* of the empirical study was composed of automated static analysis tools (**RQ**$_1$) and mobile applications (**RQ**$_2$). To select the tools, we adopted four criteria: They were tools (1) open-source and available on GitHub; (2) that take an `apk` file as input; (3) that perform a static analysis of the source code; and (4) that can be run using the command line. These filters led us to consider:

AndroBugs2. This tool can detect 52 categories of security-related concerns, permission issues, exposure of sensitive information, etc.: these are identified by comparing the analyzed source code against a predefined set of textual and syntax rules, whose violation leads to raising a warning.

Trueseeing. This analyzer can detect 7 types of security issues: *Improper Platform Usage*, *Insecure Data*, *Insecure Communications*, *Insufficient Cryptography*, *Client Code Quality Issues*, *Code Tampering* and *Reverse Engineering*.

Insider. According to the official documentation,[5] the tool

[4] Available at: https://github.com/acmsigsoft/EmpiricalStandards. Given the nature of our study and the currently available empirical standards, we followed the *"General Standard"* and *"Repository Mining"* guidelines.

[5] Insider repository: https://github.com/insidersec/insider.

covers the OWASP mobile Top 10 vulnerabilities and supports multiple programming languages like JAVA, KOTLIN, SWIFT, .NET and others.

When it comes to mobile apps, we aimed at analyzing a large and representative sample of mobile applications, which might let us provide sound and reliable conclusions. While researchers have released various mobile app datasets over the last decade [12], [20], [23], most of them are outdated, contain toy apps, or apps that no longer exist [19]. Therefore, we relied on a public dataset available on KAGGLE,[6] namely *GooglePlayStore dataset.* We selected this dataset for two reasons: (1) The dataset is currently supported by an active community and is continuously updated; (2) The dataset contains over 10,000 real ANDROID apps having different scope and characteristics and is more recent than others (it was released in November 2018). We first considered free and open-source apps from the initial set of applications. This was needed because of the requirements of the static analysis tools selected, which need to decompile the source code of the apps before detecting security threats. We considered the apps that could be freely analyzed to avoid incurring legal issues. Also, we only considered apps available on the *Google Play Store*. These two filters led to a dataset composed of 6,500 applications whose size ranges from 1MB to 99 MB, while the number of installs from a few dozens to 500,000 million. In addition, the apps were fairly equally distributed among all the categories of the GOOGLE PLAY store, meaning that we could analyze apps designed to deal with different objectives and targets.

B. $\textbf{RQ}_1$. A Taxonomy of Vulnerabilities Detected by Existing Static Analysis Tools for Mobile apps

While the description of the tools already indicates the security issues they can detect, the effort in this phase was needed to homogenize names and types of issues identified. Indeed, different tools could detect similar vulnerabilities but name them differently. The goal of $\textbf{RQ}_1$ was to define a unique schema able to represent the issues identifiable with current static analysis tools. Hence, we conducted iterative content analysis sessions [24] involving two software engineering researchers, both authors of this paper (1 Ph.D. student, 1 faculty member), having more than ten years of programming experience (the *inspectors*).

Taxonomy Building Phase. Starting from the list of vulnerabilities detectable by the considered tools, each inspector independently analyzed each item and assigned it to a category based on both the OWASP official documentation and consulting online resources (e.g., papers, websites). Afterward, the inspectors opened a discussion and solved disagreements—this happened in 15 cases (out of 60 vulnerability types). The process led to the definition of a hierarchical taxonomy composed of two layers. The top layer consisted of 11 categories, while the inner layer contained 41 subcategories described in Section IV-A.

Taxonomy Validation Phase. To reduce possible threats to conclusion validity, we decided to validate the defined taxonomy by involving two ANDROID security experts with 5 and 4 years of experience, respectively. These were contacted via e-mail by the first author of the paper, who selected them through his contacts. We provided the developers with a spreadsheet containing a list of 25 randomly chosen vulnerabilities from the total amount of 41 security issues detected by the selected tools. The developers' task was to categorize the security issues according to the taxonomy previously built (which we provided in a PDF format). The developers were either allowed to consult the taxonomy or assign new labels if needed. Once the external developers conclude the task, they send back the spreadsheet annotated with their categorization. As a result, both developers found the taxonomy clear and complete: they always assigned labels contained in the taxonomy without adding other categories.

C. $\textbf{RQ}_2$. On the Detection Capabilities of Existing Static Analysis Tools

The goal of $\textbf{RQ}_2$ was to investigate the behavior of the existing tools more closely. We first executed them against the `apk` files of the considered apps and collected their output—to homogenize the output, we developed an automated parsing tool that converted the output of the tools into `csv` files.

The collected `csv` files were then used to address $\textbf{RQ}_2$. In this respect, we noticed that the tools failed to produce results in some cases. To maximize the number of applications analyzed, the first author attempted to fix the encountered issues manually (e.g., fixing links to external dependencies or changing some versions of some libraries). Nonetheless, we could not address the issues in 20%, 25 %, and 20% cases for ANDROBUGS2, TRUESEEING, and INSIDER, respectively.

For the remaining apps, we computed the number of vulnerabilities—classified according to the taxonomy coming from $\textbf{RQ}_1$—detected by the considered tools. This investigation provides a quantitative measure of the number of vulnerabilities detected by the tools. It provides insights into their detection capabilities concerning the various vulnerability types, hence potentially indicating the strengths and weaknesses of the tools. Finally, we exploited the taxonomy built to evaluate the complementarity of the considered tools. We measured (1) the number of vulnerability types detected by more tools and (2) the number of vulnerability types solely detected by only one of them. In so doing, we considered both the levels of the taxonomy so that we could provide finer-grained insights.

IV. ANALYSIS OF THE RESULTS

This section reports the results of our study. For the sake of clarity, we discuss each research question independently. Afterward, we discuss the overall findings of the study.

[6] The KAGGLE dataset of ANDROID apps: https://www.kaggle.com/lava18/google-play-store-apps.

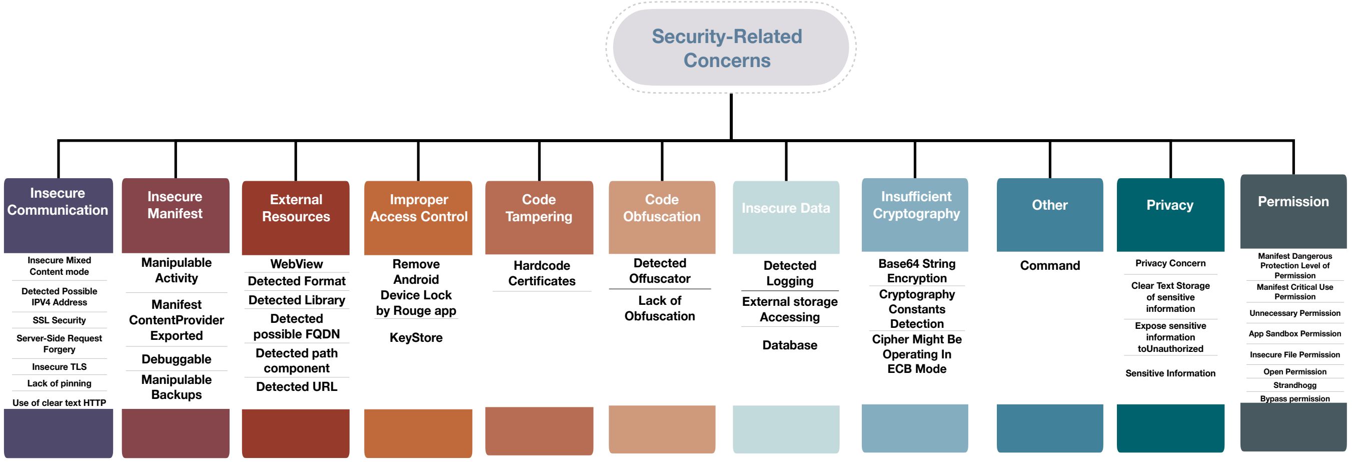

Fig. 1: A Taxonomy of the Security-Related Issues Detected by the Considered Static Analysis Tools

A. *RQ*$_1$. *A Taxonomy of Security-Related Issues Detected by Static Analysis Tools*

Figure 1 overviews the taxonomy of security-related concerns identified by the considered automated static analysis tools. For the sake of space limitation, in the following we only present the high-level categories of the taxonomy, while a complete description of the taxonomy, along with examples, is in our online appendix [7].

A. Insecure Communication [IC]. This warning is generated when a client/server application exchanges information by means of inappropriate protocols [4], for instance, by relying on the *http* protocol. Developers could accidentally select insecure protocols to communicate with other applications or the environment; this might represent a security issue if sensitive data are exchanged.

B. Insecure Manifest [IM]. This category refers to possible issues connected to the insecure use of the `AndroidManifest`. In the manifest, developers declare the application's behavior, including the permissions required. The concern arises when developers accidentally miss the definition of app restrictions, allowing the app to be potentially called by external malicious apps.

C. External Resources [ER]. This issue might arise in cases where mobile applications rely on external resources without putting in place any control over them. For example, the unchecked user or environment inputs represent a threat to security, as malicious users might format the input to create concerns for security.

D. Improper Access Control [IAC]. The application does not apply (or only partially applies) mechanisms to restrict access to resources from an unauthorized user. When those mechanisms are not correctly applied, other users can read sensitive information and execute commands [1]. An example of a successful attack is represented by the case of *Keystore*, which is a private repository that developers use to store sensitive or reserved data. A vulnerability affecting the API allowed malicious users to bypass permissions, leading to privilege escalation without user interaction [8].

E. Code Tampering [CT]. Code tampering is the process conducted by a malicious user to change the app's behavior or the APIs it relies on [10]. An automated static analysis tool could detect this issue when developers implement a potentially untrusted third-party library or other external application whose credibility cannot be verified. An example of this issue concerns the presence of hard-coded certificates, as a mobile app might run without verifying the external component it relies on.

F. Code Obfuscation [CO]. The category refers to the lack of code obfuscation. Developers apply this operation to make it harder for a hacker to access the source code. If code obfuscation is not applied, an attacker could read the source code by decompiling the apk, obtaining the respective Java code in case of native applications, and identifying vulnerabilities to exploit.

G. Insecure Data [ID]. This threat predominantly refers to the data storage and occurs when developers assume that malicious users or malware applications will not have access to the file system. For this reason, they adopt insecure mechanisms to archive private data. Nonetheless, static analysis tools may detect an issue since an user still can perform a root procedure, through which s/he can have access to the entire system and break these mechanisms, hence allowing malware or external attackers to exploit the vulnerability and have access to sensible data [3].

H. Insufficient Cryptography [ICr]. This threat relates to the insufficient mechanisms adopted to preserve personal information [5]. In these cases, developers apply protocols to preserve personal information or sensitive data; however, these might be

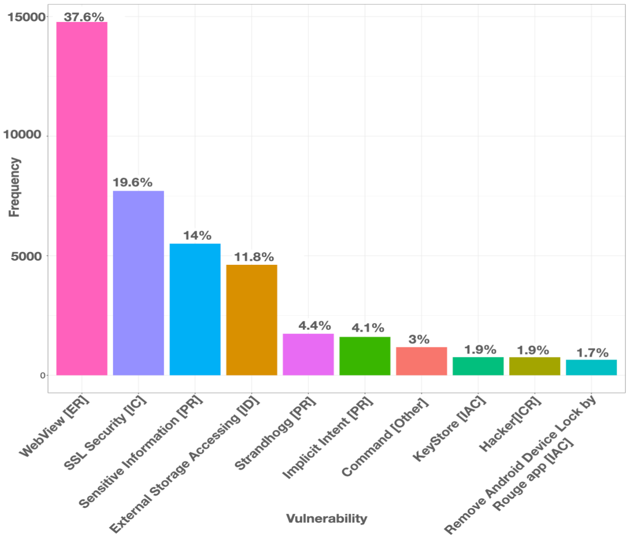

Fig. 2: Top 10 vulnerabilities detected by AndroBugs2.

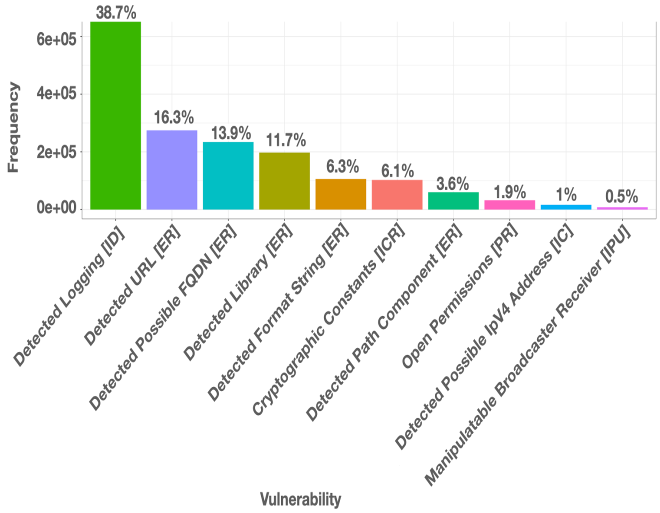

Fig. 3: Top 10 vulnerabilities detected by Trueseeing.

insufficient (e.g., if they select an insecure protocol to encrypt data).

*B. **RQ**$_2$. On the Detection Capabilities of Existing Static Analysis Tools*

Our second research question was concerned with understanding the detection capabilities of the tools in terms of frequency of security issue detection, and complementarity among the considered tools.

Frequency of detection. When determining the frequency of security-related concerns, we analyzed the output of the tools, mapping the vulnerabilities identified onto the taxonomy built in the context of **RQ**$_1$. The results were as follows.

AndroBugs2. Figure 2 shows the top-10 vulnerabilities detected by AndroBugs2. In almost 50% of the cases, the tool identified ’*Web View*’ and ’*SSL Security*’ vulnerabilities: these pertain to the ’*External Resources*’ and ’*Insecure Communication*’ categories of the taxonomy. Looking at the figure, we could then observe that AndroBugs2 could identify various types of vulnerabilities related to different security concerns. While these were detected in fewer cases, the tool seems to support developers in detecting many vulnerabilities. In addition, AndroBugs2 could identify at least one instance of each vulnerability of the taxonomy. On the one hand, further experiments aiming at assessing the accuracy of the insights provided by the tool would be needed. On the other hand, the fact that the tool could detect such a wide range of vulnerabilities might provide indications on the health status of mobile apps, which may be particularly exposed to security issues that threaten their users.

Trueseeing Figure 3 reports the ten most frequent vulnerabilities detected by Trueseeing. As shown, almost 39% of the vulnerabilities found by the tool are connected to the use of logging files, which fall under the ’*Insecure Data*’ category. While logging is typically considered a best practice [36], in some cases, developers log sensitive information, e.g., sensitive keys or URLs. As a consequence, an attacker could potentially exploit logs to damage the app. In fewer cases, Trueseeing identified security-related concerns related to the ’*External Resource*’ category, such as ’*Detected URL*’ (16%) and ’*Detected Possible FQDN*’ (14%). Both vulnerabilities make data available to externals, namely URLs in the former case and Fully Qualified Domain Name (FQDN) in the latter. Other vulnerabilities were detected to a lower extent. Looking at the categories of those vulnerabilities, we can say that Trueseeing identifies a variety of problems, ranging from cryptography to permission issues. Code tampering problems represent the only exception: this is the vulnerability that the tool was unable to detect in our dataset. At the same time, the set of security-related concerns the tool identifies is quite different from those observed with AndroBugs2, suggesting a possible complementarity between the two.

Insider. The behavior of Insider was drastically different from the one of the other tools. In this case, we could detect only two categories of vulnerabilities, namely ’*Exposed to sensitive information to an unauthorized actor*’ (61% of the warnings pertained to this vulnerability) and ’*Clear text storage of sensitive information*’ (39%). Both of them fall under the *Privacy* category of our taxonomy and have to do with sensitive information inappropriately stored within the context of a mobile app. The observed behavior of the tool suggests that it has a close focus on privacy issues, while other vulnerability categories cannot be frequently identified. This is likely the characteristic that makes the tool different from the others considered. In addition, it is important to remark that, differently from the claims made in the official documentation, we could not find any reference to the detection of vulnerabilities listed by OWASP. This likely suggests that the documentation available is outdated. Summing up, different tools seem to focus on different categories of vulnerabilities. It is also worth remarking that we could identify a number of security-related categories that are only rarely detected by the tools. Comparing the categories composing our taxonomy (see Figure 1) with those detected by the tools, we can indeed report that major security categories, like *Code Tampering*, *Improper Platform Usage*, and others, which have been the subject of previous studies in literature [17], [25], are poorly identified by the considered tools. Of course, this might be due to the fact that certain types of vulnerabilities are poorly diffused in ANDROID apps. However, we still point out the need for additional experimentations to assess the support provided by current tools.

Key findings of RQ$_2$ - Frequency.

Different tools detect different security-related concerns with different frequencies. Certain categories of problems are (almost) never detected, possibly suggesting the need for further studies on the actual support provided by the considered tools in practice.

Complementarity among the tools. From the frequency analysis, we discovered that different tools seem to capture different security-related concerns. With our last analysis, we sought to provide additional insights into the complementarity among the tools. We could first observe that INSIDER does not capture any category of problems that the other tools cannot already detect. In this sense, the tool seems to provide fewer benefits in practice, as it not only identifies a lower amount of security concerns but also targets problems that other tools can detect—of course, these observations should be backed up with additional analyses on the accuracy of the recommendations provided to developers. As for ANDROBUGS2 and TRUESEEING, instead, we could observe that the two tools cover pretty different categories of problems. Indeed, only the category of *'Insecure Data'* is in common. These findings suggest that the tools might be combined by developers to enlarge the coverage of security-related concerns.

Similar conclusions could be drawn when considering the security concerns about the second level of the taxonomy. Analyzing those issues, we could discover that only *'Web View'* and *'Manipulable Activity'* vulnerabilities are in common between ANDROBUGS2 and TRUESEEING.

Key findings of RQ$_2$ - Complementarity.

The combination of ANDROBUGS2 and TRUESEEING may provide more extensive coverage of security-related problems. On the contrary, INSIDER does not cover vulnerabilities that cannot be already detected by the other tools.

V. DISCUSSION AND IMPLICATIONS

The results of the empirical study lead to a number of actionable implications for both researchers and practitioners.

- **On the current support provided by static analysis tools.** To better contextualize our findings, we conducted an additional analysis aiming at comparing the support of the tools against the list of the top vulnerabilities identified by the Open Web Application Security Project (OWASP), one of the main security foundations worldwide that periodically produces reports about the most frequent and harmful mobile vulnerabilities.[7] From this analysis, we could observe that the current tools only partially align with the OWASP mobile top-10. While the detection of security-related concerns classified as *'Improper Platform Usage'*, *'Insecure Communication'*, or *'Insufficient Cryptography'* is supported by some of the considered tools, e.g., ANDROBUGS2, a number of other critical issues are still neglected. For example, the categories of *'Client Code Quality'* or *'Extraneous Functionality'* are not considered by any tool. In addition, it is also worth mentioning that the OWASP top-10 considers the issues pertaining to *'Improper Platform Usage'* as the most popular nowadays. Nonetheless, our frequency analysis revealed how this category is not among the most frequently identified, possibly indicating the inability of the tools to deal with this category of security concerns. In other terms, there seems to exist a mismatch between what the tools detect and what they should provide support for. We argue that this mismatch should be further investigated by our research community and tool vendors, which might be interested in providing additional support for ANDROID developers.
- **On the accuracy of current tools.** In our study, we executed three state-of-the-art tools on a dataset of ANDROID apps, analyzing their output from various perspectives. As further elaborated in Section VI, we recognize that our observations might be threatened by the presence of false positives, i.e., wrong indications given by the tools. Part of our future research agenda aims at assessing the impact of false positives on our findings. Yet, our empirical setting allows us to highlight the lack of empirical investigations into the accuracy of static analysis tools for ANDROID apps. Perhaps more worrisome, we point out the lack of datasets that might be used for this purpose. Therefore, we call for more research on the matter and the definition of novel datasets that can be exploited to compare and improve the current state of static analysis in mobile applications.
- **On the combination of more tools.** The results coming from **RQ$_2$** allowed us to assess the complementarity among the three tools considered, which revealed that different security concerns might be identified by different tools. As such, we may argue that more extensive coverage of the issues affecting a mobile app can be achieved by combining multiple tools. On the one hand, this finding can be exploited by practitioners, who might be willing to adopt and run more tools against their code. On the other hand, such a complementarity might be an opportunity

[7] The OWASP Mobile Top-10: https://owasp.org/www-project-mobile-top-10/.

for researchers who might want to devise novel smart techniques able to automatically combine static analysis tools based on the context or the developer's needs.

On the security of mobile apps. While our analysis's main goal was establishing the current support provided by static analysis tools, it also revealed insights into the security of the mobile apps analyzed. Our frequency analysis indicated that mobile apps are frequently affected by security-related concerns. Therefore, it would be useful to better analyze the current state of security in ANDROID. For instance, empirical investigations targeting the vulnerabilities detected by the tools with the objective of elaborating on the reasons behind their introduction might provide key insights for practitioners interested in improving the security profile of their apps.

VI. THREATS TO VALIDITY

Threats to Construct Validity. One of the objectives of our paper was concerned with the categorization of the security-related concerns detected by existing automated static analysis tools. This objective was naturally threatened by the selection and accuracy of the considered tools. Our choice was driven by multiple considerations. In the first place, we have considered only open-source static analyzers that can be called via command line and that take an `apk` file as input. The set of tools analyzed restricts our possibility of providing a comprehensive view of the issues detected by currently available tools. From the accuracy of the tools selected, our work might be threatened by not considering *false positives* and/or *false negatives*. The identification of *false positives* was not possible in our case. This indeed requires knowledge of the application domain and the source code of the considered apps: as such, the identification should have been done by the original developers of the application, but, unfortunately, this was not feasible. For this reason, our results might be partially affected by the presence of *false positives*. At the same time, our analysis could not deal with *false negatives*, namely the actual security-related concerns that the tools did not identify. Of course, this limitation of our study cannot be addressed as the tools could not detect those issues. More in general, our analysis sets a *lower bound* for researchers and practitioners: despite the inherent limitations, our findings still quantify how the currently available vulnerability and malware detectors support developers. Researchers interested in further elaborating on the matter may build on top of our findings, providing an improved view of the capabilities of the tools.

Threats to Conclusion Validity. The major threat in this category concerns the research methodology used to address the research questions. When considering $\mathbf{RQ}_1$, we applied an iterative approach to extract the taxonomy of security-related concerns detected by the existing tools. This task requires a human-intensive effort, is subjective by design, and leads to wrong interpretations. To mitigate this threat, we used (as possible) a systematic approach and two inspectors actively participated in building the taxonomy. In $\mathbf{RQ}_2$, a possible threat refers to the different granularity of the vulnerabilities identified by the considered tools. Part of our future work will investigate the impact of this aspect on our findings more closely.

Threats to External Validity. There are four important considerations. First, we considered three tools, meaning that the coverage of the conclusions is somehow limited. Secondly, we decided not to consider dynamics or hybrid approaches to detect possible vulnerabilities. This was a methodological decision: we were focused on analyzing statics vulnerability tools, so other kinds of methods were considered out of scope. Third, our study considered 6,500 ANDROID apps, and it can be regarded as one of the most extensive empirical investigations up to date [9], [33]. Finally, the empirical investigation considers ANDROID apps: as such, our work might not generalize to apps written using different platforms. Future replications of this work are desirable: to enable them, we released a replication package that would allow further researchers to reproduce our study in other contexts.

VII. CONCLUSION AND FUTURE WORK

In this paper, we conducted a preliminary conceptualization and analysis of the vulnerabilities detected by three automated static analysis tools for ANDROID: ANDROBUGS2, TRUESEEING, and INSIDER. First, using iterative content analysis sessions, we developed a taxonomy of the security-related concerns these tools can detect. Second, we run the three tools against a dataset composed of 6,500 ANDROID applications to collect information about the detection capabilities of the tools. Results indicate that these tools can be used to support developers identified 11 high-level vulnerabilities categories and 41 low-level ones. Although tools can detect many vulnerability categories, we found that they only partially cover the top 10 vulnerabilities listed by OWASP. We demonstrate that practitioners might benefit from combinations of multiple tools. The findings of the paper, along with the implications we could delineate, open several follow-up research directions that involve the definition of empirical studies and automated techniques to better support the detection of vulnerabilities in ANDROID. Our future research agenda will be centered around these novel directions. We aim at deepening our knowledge of the accuracy of currently available static analysis tools.

ACKNOWLEDGEMENT

Fabio is partially funded by the Swiss National Science Foundation through the SNF Projects No. PZ00P2_186090.

REFERENCES

[1] Cwe-284: Improper access control. https://cwe.mitre.org/data/definitions/284.html. Accessed: 2022-01-11.

[2] How many smartphones are in the world? https://www.bankmycell.com/blog/how-many-phones-are-in-the-world. Accessed: 2021-12-13.

[3] M2: Insecure data storage. https://owasp.org/www-project-mobile-top-10/2016-risks/m2-insecure-data-storage. Accessed: 2022-01-11.

[4] M3: Insecure communication. https://owasp.org/www-project-mobile-top-10/2016-risks/m3-insecure-communication. Accessed: 2022-01-11.

[5] M5: Insufficient cryptography. https://owasp.org/www-project-mobile-top-10/2016-risks/m5-insufficient-cryptography. Accessed: 2022-01-11.

[6] Mobile operating systems' market share worldwide from january 2012 to june 2021. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/. Accessed: 2021-12-13.

[7] Online appendix. https://figshare.com/s/5aba382667d406194aea.

[8] Vulnerability details: Cve-2017-13236. https://www.cvedetails.com/cve/CVE-2017-13236/. Accessed: 2022-01-19.

[9] A. Bartel, J. Klein, Y. Le Traon, and M. Monperrus. Automatically securing permission-based software by reducing the attack surface: An application to android. In *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 274–277. IEEE, 2012.

[10] M. Ceccato, P. Tonella, C. Basile, B. Coppens, B. De Sutter, P. Falcarin, and M. Torchiano. How professional hackers understand protected code while performing attack tasks. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, pages 154–164. IEEE, 2017.

[11] F. Chow, S. Chan, S.-M. Liu, R. Lo, and M. Streich. Effective representation of aliases and indirect memory operations in ssa form. In *International Conference on Compiler Construction*, pages 253–267. Springer, 1996.

[12] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, pages 845–854, 2017.

[13] A. Desnos. Android: Static analysis using similarity distance. In *2012 45th Hawaii International Conference on System Sciences*, pages 5394–5403, 2012.

[14] W. Enck, D. Octeau, P. D. McDaniel, and S. Chaudhuri. A study of android application security. In *USENIX security symposium*, volume 2, 2011.

[15] D. Feth and A. Pretschner. Flexible data-driven security for android. In *2012 IEEE Sixth International Conference on Software Security and Reliability*, pages 41–50. IEEE, 2012.

[16] E. Fife and J. Orjuela. The privacy calculus: Mobile apps and user perceptions of privacy and security. *International Journal of Engineering Business Management*, 4(Godište 2012):4–11, 2012.

[17] J. Gajrani, M. Tripathi, V. Laxmi, G. Somani, A. Zemmari, and M. S. Gaur. Vulvet: Vetting of vulnerabilities in android apps to thwart exploitation. *Digital Threats: Research and Practice*, 1(2), may 2020.

[18] J. Gao, L. Li, P. Kong, T. F. Bissyandé, and J. Klein. Understanding the evolution of android app vulnerabilities. *IEEE Transactions on Reliability*, 2019.

[19] F.-X. Geiger and I. Malavolta. Datasets of android applications: a literature review. *arXiv preprint arXiv:1809.10069*, 2018.

[20] F.-X. Geiger, I. Malavolta, L. Pascarella, F. Palomba, D. Di Nucci, and A. Bacchelli. A graph-based dataset of commit history of real-world android apps. In *Proceedings of the 15th International Conference on Mining Software Repositories*, pages 30–33, 2018.

[21] K. Kulkarni and A. Y. Javaid. Open source android vulnerability detection tools: a survey. *arXiv preprint arXiv:1807.11840*, 2018.

[22] L. Li, T. F. Bissyandé, M. Papadakis, S. Rasthofer, A. Bartel, D. Octeau, J. Klein, and L. Traon. Static analysis of android apps: A systematic literature review. *Information and Software Technology*, 88:67–95, 2017.

[23] W. Li, X. Fu, and H. Cai. Androct: Ten years of app call traces in android. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 570–574. IEEE, 2021.

[24] W. Lidwell, K. Holden, and J. Butler. *Universal principles of design, revised and updated: 125 ways to enhance usability, influence perception, increase appeal, make better design decisions, and teach through design*. Rockport Pub, 2010.

[25] M. Linares-Vásquez, G. Bavota, and C. Escobar-Velásquez. An empirical study on android-related vulnerabilities. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 2–13. IEEE, 2017.

[26] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman. A survey of app store analysis for software engineering. *IEEE transactions on software engineering*, 43(9):817–847, 2016.

[27] S. Mirza Abdullah, B. Ahmed, and M. M Ameen. A new taxonomy of mobile banking threats, attacks and user vulnerabilities. *Eurasian Journal of Science and Engineering*, 3(3):12–20, 2018.

[28] S. Mujahid, R. Abdalkareem, and E. Shihab. Studying permission related issues in android wearable apps. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 345–356. IEEE, 2018.

[29] A. Qamar, A. Karim, and V. Chang. Mobile malware attacks: Review, taxonomy & future directions. *Future Generation Computer Systems*, 97:887–909, 2019.

[30] A. Sadeghi, H. Bagheri, J. Garcia, and S. Malek. A taxonomy and qualitative comparison of program analysis techniques for security assessment of android software. *IEEE Transactions on Software Engineering*, 43(6):492–530, 2016.

[31] P. Salza, F. Palomba, D. Di Nucci, A. De Lucia, and F. Ferrucci. Third-party libraries in mobile apps. *Empirical Software Engineering*, 25(3):2341–2377, 2020.

[32] S.-H. Seo, A. Gupta, A. M. Sallam, E. Bertino, and K. Yim. Detecting mobile malware threats to homeland security through static analysis. *Journal of Network and Computer Applications*, 38:43–53, 2014.

[33] H. Wang, Y. Guo, Z. Tang, G. Bai, and X. Chen. Reevaluating android permission gaps with static and dynamic analysis. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2015.

[34] H. Wang, Y. Li, Y. Guo, Y. Agarwal, and J. I. Hong. Understanding the purpose of permission use in mobile apps. *ACM Transactions on Information Systems (TOIS)*, 35(4):1–40, 2017.

[35] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[36] D. Yuan, S. Park, and Y. Zhou. Characterizing logging practices in open-source software. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 102–112. IEEE, 2012.

[37] X. Zhan, L. Fan, T. Liu, S. Chen, L. Li, H. Wang, Y. Xu, X. Luo, and Y. Liu. Automated third-party library detection for android applications: Are we there yet? In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 919–930. IEEE, 2020.