RESEARCH

Software Engineering Techniques for Statically Analyzing Mobile Apps: Research Trends, Characteristics, and Potential for Industrial Adoption

Marco Autili[1*], Ivano Malavolta[2], Alexander Perucci[1], Gian Luca Scoccia[1] and Roberto Verdecchia[2]

Abstract

Mobile platforms are rapidly and continuously changing, with support for new sensors, APIs, and programming abstractions. Static analysis is gaining a growing interest, allowing developers to predict properties about the run-time behavior of mobile apps without executing them. Over the years, literally hundreds of static analysis techniques have been proposed, ranging from structural and control-flow analysis to state-based analysis.

In this paper, we present a systematic mapping study aimed at identifying, evaluating and classifying characteristics, trends and potential for industrial adoption of existing research in static analysis of mobile apps. Starting from over 12,000 potentially relevant studies, we applied a rigorous selection procedure resulting in 261 primary studies along a time span of 9 years. We analyzed each primary study according to a rigorously-defined classification framework. The results of this study give a *solid foundation for assessing existing and future approaches for static analysis of mobile apps, especially in terms of their industrial adoptability*.

Researchers and practitioners can use the results of this study to (i) identify existing research/technical gaps to target, (ii) understand how approaches developed in academia can be successfully transferred to industry, and (iii) better position their (past and future) approaches for static analysis of mobile apps.

Keywords: Software Engineering; Static Analysis; Mobile apps; Systematic mapping study

1 Introduction

Nowadays, the digital media usage time is driven by mobile devices, with smartphone and tablets accounting for 66% of all time spent, against desktop usage which accounts for 34% only [2]. Specifically, more than 80% of mobile minutes in all markets are spent on mobile apps [35]. Indeed, the development of mobile apps is exponentially growing since the establishment of a number of app stores from where to download and install them.

The main key success factors of mobile apps is in fact the distribution model offered by dedicated *app stores*, such as Google Play for Android apps, and the Apple app store for iOS apps. As of today, these stores make available millions of mobile apps of different categories to millions of people, who use them for their everyday activities like purchasing products, messaging, etc. [2]. Clearly, this is a highly competitive business in which even the smallest error may have a tremendous financial impact. Revenue and profit of a mobile app is often proportional to the number of its users [241], who may enjoy using the app (and possibly rate it positively in the store) or dislike it (and possibly abandon it or even leaving a negative review in the store). This implies that improving the level of users satisfaction is fundamental for app developers to both keep existing users active and attract new ones.

Technically, **mobile apps** consist of executable files that are downloaded directly to the end user's device and stored locally. Mobile apps are developed atop the services provided by their underlying mobile platform (e.g., Android). Those services are exposed via a dedicated Application Programming Interface (API) with methods related to communication and messaging, graphics, security. Programming languages and tools for developing mobile apps are platform-specific (e.g., Java code for Android apps, and Swift code for Apple iOS apps), and present many challenges that may hamper the success of a mobile app as a

*Correspondence: marco.autili@univaq.it
[1]University of L'Aquila, L'Aquila, Italy
Full list of author information is available at the end of the article

whole [122, 250]. As empirically emerged in [122], app developers strongly need better analysis and testing support, with a focus on important features like mobility, location services, sensors, as well as different gestures and inputs. Indeed, although it may be assumed that app developers are adhering to development best practices – mainly related to well-established software engineering principles and design patterns – there is still the need of assessing, or even guaranteeing, properties about apps with a certain degree of confidence. Examples of those properties include: low energy consumption, efficient use of computational resources, security, performance, and reliability. Satisfying these needs would allow (i) app developers to raise the level of quality of their products and, potentially, their revenues, (ii) app users to use high-quality products in their everyday activities, and (iii) app store moderators (e.g., Google and Apple) to raise the overall level of quality and trustworthiness of their stores.

Static program analysis allows for predicting (precise or approximated) quantitative and qualitative properties related to the run-time behavior of a program without actually executing it [190]. For instance, static analysis techniques allow for statically inferring cost-related properties (such as the estimation of the maximal number of loop iterations and the related worst-case execution time), as well as properties related to resource consumption [13] (such as memory/heap usage and energy consumption).

Under this perspective, static analysis of mobile apps can be a valuable instrument for both (i) app developers, who can use it to quickly get non-trivial insights about their apps (e.g., subtle security issues, energy hotspots due to some programming antipattern, inefficient use of hardware sensors) and (ii) app store moderators, who can use static analysis for systematically assessing the level of quality of the apps they distribute, possibly identifying those apps with an unacceptable level of quality (e.g., apps with well-known security flaws, apps asking for suspicious permissions, apps with strong energy inefficiencies).

Static analysis of mobile apps is gaining a growing interest in both academia and industry. *Literally hundreds of (often overlapping) kinds of (theoretical and practical) static analysis approaches exist in the literature*, ranging from structural and control-flow analysis, to data-flow and state-based analysis, interval analysis (used in optimizing compilers) and so on [190]. Such approaches exploit static analysis techniques from different perspectives and belong to extremely different research areas of software engineering, such as software analytics, security, testing, verification. Industrial tools are also emerging and being maintained by key players in the technological panorama. For example, Facebook's Infer[1] applies separation logic and bi-abduction for inter-procedural analysis [42] and it is used by Facebook itself, Spotify, Mozilla, the Amazon Web Services division, etc.

The **goal** of this paper is to precisely characterize existing software engineering research on static analysis of mobile apps from three different perspectives, namely: (i) research trends, (ii) the characteristics of the proposed approaches, and (iii) their potential for industrial adoption.

In order to achieve this goal, we applied the systematic mapping study **methodology** [204, 258]. The aim of this methodology is to provide an objective, replicable, and unbiased approach to answer a set of research questions about the state of the art on a given topic. In this paper, we systematically selected 261 primary studies from over 12,000 potentially relevant publications on static analysis of mobile apps. Then, we defined a classification framework for categorizing the selected approaches and rigorously applied it to the 261 primary studies. Finally, we synthesized the obtained data to let emerge a crystal-clear snapshot of the state of the art on static analysis of mobile apps.

The main **contributions** of this study are:

1. a *classification framework* for categorizing, comparing, and evaluating approaches for static analysis of mobile apps according to a number of parameters (e.g., analysis goal, supported platforms, type and number of needed inputs, types of supported analysis);
2. an up-to-date *map* of the state of the art in static analysis of mobile apps;
3. an evaluation of the *potential for industrial adoption* of existing research results on static analysis of mobile apps;
4. a discussion of the *emerging challenges* and their implications for future research on static analysis for mobile apps;
5. a *replication package* for independent replication and verification of this study.

The **audience** of this study is composed of both (i) *researchers* interested in adopting existing static analysis approaches, possibly to further contribute to this research area by targeting (a subset of) the identified research challenges (see Section 9), and (ii) *app developers* interested to critically understand existing research results and thereby to adopt/extend those approaches in the context of their products. The latter point is specially relevant since in this study we also assessed how approaches developed in academia can be successfully transferred and adopted in industrial

[1] `http://fbinfer.com`

projects. As a concrete case, the Infer approach lays its theoretical foundations in academic results developed by researchers from the Imperial College and the Queen Mary University of London [42, 43], and it is now used by top tech companies such as Facebook, Amazon, Spotify, Mozilla, Sky.

The rest of the paper is organized as follows. Section 2 provides background information on the mobile apps ecosystem and static program analysis. Section 3 puts our study in context with respect to related work. The design of our study from a methodological perspective is provided in Section 4. The main results of our study are reported in Sections 5, 6, 7, and 8. Section 9 discusses and puts the achieved results in context by also elaborating on future research challenges. Threats to validity are reported in Section 10. Section 11 closes the paper.

2 Background

This section provides the reader with background notions on the mobile apps ecosystem (Section 2.1) and static program analysis (Section 2.2).

2.1 The mobile apps ecosystem

A *mobile app* (short for *mobile application*) is a computer program designed to run on mobile devices such as smartphones and tablet computers. Mobile apps were originally offered for general productivity and information retrieval, including email, calendar, contacts, stock market and weather information. However, public demand drove rapid expansion into other categories and nowadays, according to a 2017 report, the global app economy is worth 1.3$ trillions and is predicted to grow to 6.3$ trillions in 2021 [9].

Mobile apps fall broadly into three categories: native, web-based, and hybrid [228]. Native apps run on a device's operating system and are required to be adapted for different devices. Web-based apps require a web browser on a mobile device. Hybrid apps are web-based apps hosted inside a native application.

Apps that are not pre-installed are generally distributed to end-users through *app stores*, application distribution platforms first appeared in 2008. Dedicated app stores are typically operated by the owners of the mobile operating systems (such as the Apple App Store[2], Google Play[3], and the Windows Phone Store[4]). Generally, mobile apps are downloaded directly from the distribution platform to a target mobile device. Currently, Android and iOS platforms, the two most prominent mobile operating systems, make up over 99% of smartphone sales worldwide [114].

Still, being relatively new, mobile apps present a wide array of issues and challenges for both end users and developers. On the one hand, when using mobile apps, end users often face issues that stem from poor quality of development (such as apps that exhibit frequent crashes, lack in responsiveness or consume an abnormal amount of energy or memory) or deliberate malicious behavior (such as apps that invade privacy or are unethical [129]). On the other hand, developers face multiple challenges when developing apps for mobile devices such as fragmentation, both across multiple platforms and within the same platform, lack of robust monitoring, analysis and testing tools, as well as having to keep up with frequent platform updates and changes [123].

In the following section, we provide a concise summary of the topic on which our research focusses, namely static program analysis techniques, followed by a concrete example of one of such techniques.

2.2 Static program analysis

The denomination *static program analysis* encloses a set of static compile-time techniques that predict computable approximations of values or behaviors arising at run-time when executing a program [190]. When applied to mobile apps, static program analysis can be an effective instrument for both app developers an app store moderators (e.g., Google, Apple) to predict and evaluate (precise or approximated) quantitative and qualitative properties related to the run-time behavior of mobile apps without actually executing them. Hence, it can be a valuable instrument to create apps with better quality in a world where a low-quality releases can have devastating consequences [124].

In the literature, static analysis of mobile apps has been applied with variety of goals in mind, ranging from malware and privacy leaks detection to detection of bugs in the app source, to reduction of energy and memory consumption [15, 16, 93, 113, 144]. To achieve these goals, researchers have experimented with a variety of different static analysis techniques. Among the ones worth mentioning, there is *data-flow analysis*, in which a program is considered as a graph: nodes are elementary blocks and edges describe how control passes from one block to another [190]. *Taint Analysis* is a special case of data-flow analysis that aims to detect the existence of a data flow from sensitive data sources, often simply referred as *sources*, to untrusted program statements, called *sinks* [113]. *Type Analysis* aims to verify the type safety of a program, i.e., if we can guarantee that the eventual value of any expression in the program will not violate the expression's static type.

[2] https://www.apple.com/it/ios/app-store/
[3] https://play.google.com/
[4] https://www.microsoft.com/store/apps/windows-phone

In other words, type analysis aims to detect type errors in a program source code. *Abstract interpretation* is a sound approximation of the semantics of a program, based on monotonic functions over ordered sets. It is able to extract information about the semantics of a program without performing all the calculations. *Program slicing* aims to compute the set of program statements, referred to as the program slice, which may affect the values at some point of interest, referred to as a slicing criterion.

In some cases, static analysis approaches rely on *additional inputs* other than the program itself (e.g., knowledge bases, code mappings), either to improve the accuracy of the analysis or to perform broader kinds of analyses that would be impossible without. When the analysis makes use of information collected at run-time, while executing the program, we refer to it as *hybrid analysis*.

Approaches for static analysis of mobile apps can be generic or *platform specific*. The latter approaches are able to analyze only apps developed for a specific mobile platform, as the analysis leverages or focuses on programming constructs that are available only on that platform (e.g., *Android Intents*).

Example – In the remaining of this section, we describe CHEX [171], one of the identified primary studies, in order to give a concrete idea about the typical traits and features of a software engineering techniques for static analysis of mobile apps. The main goal of CHEX is to automatically detect component hijacking vulnerabilities, a specific class of security vulnerabilities existing on the Android platform. In this sense, CHEX is *Android-specific*. These vulnerabilities have been modeled from a data-flow analysis perspective, thus enabling their identification via a reachability analysis on custom system dependence graphs. In [171], the authors also devised novel techniques to tackle analysis challenges arising from the Android's programming paradigm, such as multiple app entry points and asynchronous code execution. CHEX has been implemented on top of Dalysis, a generic static analysis framework that the authors built to support many types of analysis on Android app bytecode. CHEX was evaluated on 5,486 real Android apps and correctly identified 254 potential component hijacking vulnerabilities.

3 Related Work on Static Analysis of Mobile Apps

In this section, we discuss other existing studies related to our work. Literature reviews, surveys and mapping studies on either static analysis approaches or analysis methodologies and techniques applied to mobile apps that can be considered as research related to our study.

Based on our knowledge, we found no systematic mapping study (SMS) and only one systematic literature review (SLR) on the specific topic of static analysis of mobile apps [152]. Thus, in the following, we first discuss in more detail the SLR reported in [152], which is a valuable and solid work study closely related to ours. Then, we discuss other works in the literature that, although having different scopes and objectives, can be related to our research.

Similarly to our SMS, the SLR in [152] reviewed publications on approaches involving the use of static analysis for mobile apps. The main difference between the SLR in [152] and our SMS is methodological; as extensively discussed in [204] and [135], SLRs aim at synthesizing evidence with a very specific goal in mind (e.g., which static analysis technique achieves higher accuracy in specific contexts), whereas systematic maps are *primarily concerned with structuring a research area* [204], providing an overview of the direction and intensity of the scientific interest over a specific topic (static analysis for mobile apps in our case), which sub-topics are covered, and relevant research gaps and trends. This difference in aim implies profound methodological differences throughout the whole research protocol, ranging from the nature of the research questions, the broadness of the searchers, and most importantly the synthesised findings.

In the following, we provide an overview of the main methodological differences among our study and the one in [152]. In addition to Android, our study considered also other platforms. As per the search strategy, the main difference is that we performed a manual search of top venues for SE and programming languages, followed by backward snowballing and then forward snowballing; in [152], the authors performed automatic search followed by manual search of top venues for SE, programming languages, security and privacy, and then authors' self-check followed by backward snowballing. Concerning the selections criteria, we considered only peer reviewed work, by excluding studies in the form of editorials and tutorial, as well as short and poster papers, secondary or tertiary studies. In [152], only short papers were excluded. Moreover, differently from them, we accounted for the existence of some kind of evaluation together with the availability of an implementation. As a result, they collected 124 research papers, in the timespan 2011-2015; we have a better coverage made of 261 primary studies in the timespan 2007-2019. Finally, similarly to [152], in our study we perform a vertical analysis of the extracted data (i.e., we perform an in-depth analysis of the extracted data for each parameter of our classification framework); in addition, in our study we also complement the vertical analysis with horizontal

analysis (i.e., we build contingency tables across pairs of parameters and investigate on emerging interesting correlations).

Importantly, in [152], the authors do not consider the potential for industrial adoption of existing research on static analysis of mobile apps, as we do through our research question RQ3. This is a substantial difference that permitted us to identify in the state of the art those approaches to static analysis of mobile apps that are ready for technological transfer and industrial adoption. Another profitable difference is in the nature of the study, SLR versus SMS, and in the target audience. As already introduced, in our SMS we target both researchers and practitioners, such as app developers, who are interested in selecting/choosing existing static analysis approaches, and want to critically understand what they offer and how, in order to opt for their adoption or possible industrial transfer. The SLR in [152] more specifically targets researchers and practitioners that want to propose a new approach to static analysis or to extend existing ones. In this sense, we believe that our work and the work in [152] complement one another, and together they constitute a valuable asset to the academic and industrial world in the wide spectrum of static analysis.

In [87], a survey about static analysis and model checking approaches for searching patterns and vulnerabilities within a software system is reported. The authors examine the proposed algorithms and their effectiveness in finding bugs. A peculiarity of this research is the comparison between static analysis algorithms and mathematical logic languages for model checking.

In [205], the authors report on a survey about static analysis for identifying security issues and vulnerabilities in software systems in general (not specific to mobile apps). For each type of security vulnerability, the authors present both relevant studies and the implementation details of the used static analysis algorithms.

A systematic mapping study is reported in [65]. The study was conducted for classifying and analysing approaches that combine different static and dynamic quality assurance techniques. The study includes a discussion about reported effects, characteristics, and constraints of the various existing techniques.

A literature review about mobile usability models can be found in [105], as a means for validating a specific usability model. Among the main results, from this literature review it emerges that usability is usually measured in terms of three key indicators, namely, effectiveness, efficiency and satisfaction.

Even if some of the above mentioned works are about static analysis, none of them is specifically focussed on the static analysis of mobile apps, and none of them is a systematic literature review.

4 Study Design

This research was organized into three main phases, which are well-established when it comes to systematic literature studies [133, 258]: *planning*, *conducting*, and *documenting*.

Planning. We established the need for performing a review on static analysis of mobile app (Section 3), we identified the main research questions (Section 4.1), and we defined the protocol to be followed by the involved researchers.

Conducting. We performed the mapping study by following all the steps defined in our research protocol, namely: (i) search and selection of primary studies, i.e., the relevant research articles on static analysis methods and techniques of mobile apps (Section 4.2), (ii) extraction of relevant data from each primary study according to a rigorously-defined classification framework (Section 4.3), and (iii) synthesis of main findings emerging from the analysis and summary of the data extracted in the previous activity (Section 4.4).

Documenting. The main activities performed in this phase are: (i) a thorough elaboration of the data extracted in the previous phase, with the main goal of setting the obtained results in their context, (ii) the discussion of possible threats to validity, specially to the ones identified during the definition of the review protocol (in this activity new threats to validity may emerge too), and (iii) the writing of a final report (i.e., this article) describing the performed mapping study.

A complete ***replication package*** is publicly available to allow interested researchers to independently replicate and verify our study[5]. It includes the review protocol, the list of both searched and selected studies, a detailed data extraction form, the raw extracted data, and the R scripts for data analysis.

4.1 Research questions

We formulate the goal of this study by using the Goal-Question-Metric perspectives (i.e., purpose, issue, object, viewpoint [30]). Table 1 shows the result of the above mentioned formulation.

Table 1: Goal of this research

Purpose	Identify, classify, and evaluate
Issue	trends, characteristics and potential for industrial adoption
Object	of existing research in static analysis of mobile apps
Viewpoint	from a researcher's and practitioner's point of view.

[5] https://github.com/sesygroup/mobile-static-analysis-replication-package

The results of this study are targeted to both (i) researchers willing to further contribute to this research area, and (ii) practitioners willing to understand existing research on static analysis approaches of mobile apps and thereby to be able to adopt those solutions that better fit with their needs. We refined our abstract goal into the following research questions:

RQ1: *What are the **research trends** on static analysis of mobile apps?*
Rationale: a multitude of researchers are investigating on static analysis for mobile apps over time with different degrees of independence and different methodologies. By answering this research question, we aim at characterizing the scientific interest on static analysis approaches of mobile apps, the relevant venues where academics are publishing their results on the topic, and their contribution type.

RQ2: *What are the **characteristics** of existing approaches for static analysis of mobile apps?*
Rationale: static analysis of mobile apps is a multi-faceted research topic, where researchers can focus on very different aspects (e.g., energy consumption, security), applying very different research methodologies (e.g., industrial case studies, empirical evaluations), providing different
types of contributions (e.g., tools for automating development activities, techniques for analyzing a specific aspect of the mobile app). By answering this research question, we aim at providing (i) a solid foundation for classifying existing (and future) research on static analysis of mobile apps, and (ii) an understanding of current research trends and gaps in the state of the art on static analysis of mobile apps.

RQ3: *What is the **potential for industrial adoption** of existing research on static analysis of mobile apps?*
Rationale: while it is well known that mobile apps have their roots in industry, many research groups focus on them from an academic perspective.
Therefore, it is natural to ask ourselves how the produced research findings and contributions can be actually transferred back to industry. By answering this research question we aim at assessing how and if the current state of the art on static analysis of mobile apps is ready to be adopted in industry.

4.2 Search and selection process

Our first choice for searching potentially relevant studies was to perform an automatic search on known data sources (e.g., IEEE Xplore, the ACM Digital Library, SCOPUS). However, from the results of a preliminary study [14], we understood that the research topic of mobile static analysis resulted to be extremely heterogeneous; for example, many keywords like "program analysis" resulted to be profoundly overloaded, leading to imprecise and inaccurate automatic search results. In order to prevent biases associated to automatic searches, we adopted two complementary manual search activities. This decision is supported by the evidence that automatic searches and backward snowballing activities lead to similar results, and that the decision on which to prefer is context-specific [116, 256]. Our search strategy was divided into two subsequent and complementary steps. The first step was carried out by manually inspecting all the publications of the top-level software engineering venues. The papers identified through this first step were then subsequently utilized as input for a backward and forward snowballing[6] process [257]. In order to ensure the correctness of the adopted manual approach, the backward snowballing activity was based exclusively on the papers selected from the top-level software engineering venues. Furthermore, the backward snowballing results were further contemplated by adopting a forward snowballing process, that ensured soundness and relevance of the set of the selected primary studies.

Figure 1 shows our search and selection process, whose main steps are detailed in the following. Our search and selection process is designed as a multi-stage process in order to have full control on the number and characteristics of the studies being either selected or excluded during the various stages.

1. Perform initial manual search. We performed a manual search by considering exclusively articles published in the top-level software engineering conferences and international journals according to well-recognized sources in the field [55, 272]. It is important to note that the main aim of this step was not to select *all* primary studies but, as suggested in [256], we aimed at obtaining a good start set of papers for the subsequent snowballing procedure (stage 3), i.e., high-quality relevant papers about static analysis techniques for mobile apps in the field of software engineering. We used the quality of the publication venues as proxy of the quality of the potentially relevant studies. Table 2 shows the considered conferences and journals. The time span of our search ranges from January 2007[7] to December 2019.

[6] Inspection of the studies referenced by a paper (*backward snowballing*) and of the studies referencing it (*forward snowballing*)

[7] Given that the concept of mobile app exists only since 2007

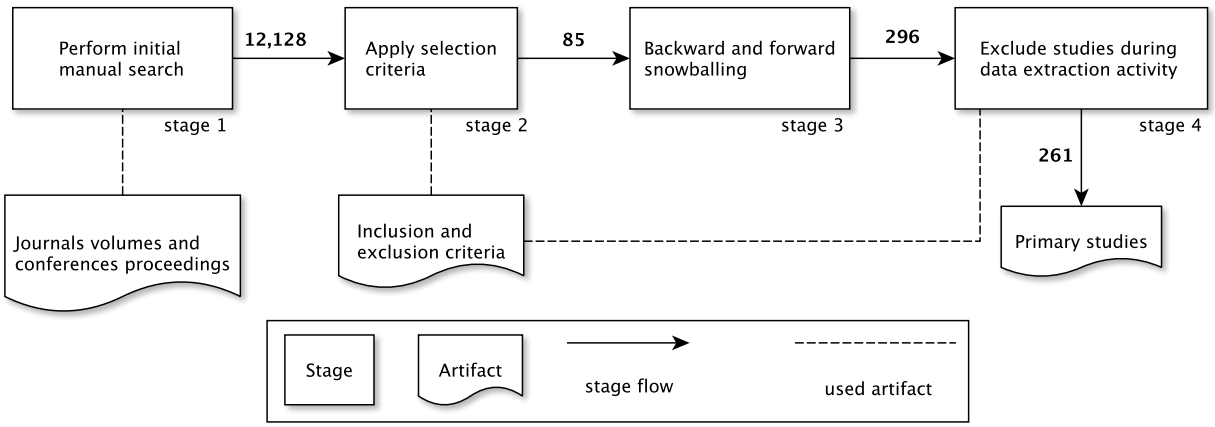

Figure 1: The search and selection process of this study

Table 2: Searched data sources with number of potentially relevant studies

Conferences	#Studies	Journals	#Studies
International Conference on Software Engineering (ICSE)	1,092	IEEE Transactions on Software Engineering (TSE)	798
European Software Engineering Conference (ESEC)\ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)	1,039	ACM Transactions on Software Engineering and Methodology (TOSEM)	267
International Conference on Fundamental Approaches to Software Engineering (FASE)	357	Information and Software Technology (IST)	1,456
IEEE/ACM International Conference on Automated Software Engineering (ASE)	1,012	Automated Software Engineering (ASE journal)	225
ACM SIGPLAN conference on Systems, Programming, Languages and Applications: Software for Humanity (SPLASH)	668	Software Maintenance & Evolution - Research & Practice (JSEP)	568
European Conference on Object-Oriented Programming (ECOOP)	362	Software and Systems Modeling (SoSyM)	637
International Symposium on Software Testing and Analysis (ISSTA)	466	Empirical Software Engineering (ESEJ)	686
		Journal of Systems and Software (JSS)	2,495
Total	4,996	**Total**	7,132

The search was performed by manually screening the DBLP entries of all conference proceedings and journal issues within the considered time span and contextually applying the selection criteria described in stage 2. DBLP is the Computer Science Bibliography from the University of Trier [142] and contains all proceedings and issues of the publication venues listed in Table 2. This step resulted in a total of 12,128 potentially relevant studies distributed across more than 9 years of research in software engineering.

2. Apply selection criteria. Each study was filtered according to a set of well-defined selection criteria. The adopted criteria are detailed in Section 4.2.1. An adaptive reading depth was applied in order to carry out the selection process in a time-efficient and objective manner [203], because it was not necessary to read the full text of approaches that clearly did not qualify. This step resulted in a total of 85 potentially relevant studies. This significant reduction of the number of potentially relevant studies is due to the fact that (i) we considered exclusively top-level venues in the field of software engineering, and (ii) the considered venues are quite general, with static analysis of mobile apps being only one of the many topics of interest of those venues. In order to reduce possible biases, three researchers were involved in this stage of the study, with a fourth researcher playing the role of arbiter in case of conflicts so to 'avoid endless discussions' [292]. The application of the selection criteria lead to an initial set of 85 primary studies.

3. Backward and forward snowballing. In this step, we applied backward and forward snowballing in order to take into account also studies that are published outside the contexts of the conferences and journal considered in the previous step. In particular, this process was carried out by considering the studies selected in the initial search, and subsequently selecting relevant papers among those cited by the initially selected ones. This method is commonly referred to as a *backward snowballing* activity [256].

In addition to the backward snowballing, we also analyzed the researches citing the studies selected through the initial search. This process is usually referred to as a *forward snowballing* activity [256]. Specifically, we included this further literature search method in order to consider also newer studies that, at that time, had not been included in official journal volumes or conference proceedings yet. Regarding the forward snowballing process, the *Google Scholar*[8] bibliographic
database was adopted to retrieve the studies citing the ones selected through the initial search phase.

The final decision about the inclusion of the papers was based on the adherence of the full text of the studies to the predefined selection criteria presented in Section 4.2.1. This step resulted in a total of 296 potentially relevant studies. The total number of potentially relevant studies increased significantly since in this step we considered papers published in all research venues, which by definition are far more than the top-level ones.

4. Exclude studies during data extraction activity. While reading in details each potentially relevant study, we agreed that 35 studies were semantically out of the scope of this research, so they were excluded. This final step led us to the final set of 261 primary studies.

4.2.1 Selection criteria

Following the guidelines for systematic literature review for software engineering [133], in order to reduce the likelihood of biases, we defined a set of inclusion and exclusion criteria beforehand. In the following, we detail the set of inclusion and exclusion criteria that guided the selection of the potentially relevant studies. A potentially relevant study was included if it satisfied *all* the inclusion criterion stated below; whereas, it was discarded if it satisfied *at least one* of the exclusion criteria reported below.

Inclusion criteria

I1) Studies proposing or using a static analysis method or technique for mobile apps.
I2) Studies in which the static analysis method or technique takes as input one or more mobile applications in the form of binary files or source code.
I3) Studies providing some kind of evaluation of the proposed method or technique (e.g., via formal analysis, controlled experiment, exploitation in industry, application to a simple example).

[8] `https://scholar.google.it/`

Exclusion criteria

E1) Studies not describing any implementation of the proposed method or technique.
E2) Secondary or tertiary studies (e.g., systematic literature reviews, surveys).
E3) Studies in the form of editorials, tutorial, short, and poster papers, because they do not provide enough information.
E4) Studies not published in English language.
E5) Studies not peer reviewed.
E6) Studies in which the static analysis method or technique takes as input only store metadata (e.g., user reviews, ratings) or other app artifacts (e.g., manifest files).

4.3 Data extraction

This phase concerns (i) the creation of a classification framework for the primary studies, and (ii) the collection of data from each primary study.

In order to carry out a rigorous data extraction process, as well as to ease the control and the subsequent analysis of the extracted data, a predefined data extraction form was designed prior the data extraction process. The data extraction form is composed of the various categories of the classification framework. The classification framework is composed of three distinct parts, one for each research question of our study[9]. The overview of each part of the classification framework, and respective parameters, is reported in Table 3, whereas the definition and values of each specific parameter is given in Sections 5, 6, and 7.

Table 3: Overview of the classification framework

Research trends (RQ1)	
• Year of publication	• Publication venue
• Publication venue type	• Analysis goal
• Macro analysis goal	• Paper goal
Characteristics (RQ2)	
• Platform specificity	• Implementation
• Static/hybrid approach	• Usage of machine learning
• App artifact	• Additional inputs
• Analysis pre-steps	• Analysis technique
Potential for industrial adoption (RQ3)	
• Target stakeholder	• Tool availability
• Number of analysed apps	• Applied research method
• Industry involvement	

For each primary study, three researchers collaboratively collected a record with the extracted information in the data extraction form for subsequent analysis. As suggested in [258], in order to validate our data extraction strategy, we performed a sensitivity analysis to check whether the results were consistent, independently from the researcher performing the analysis.

[9] For the sake of simplicity, we do not report standard publication information (e.g., study ID, title, search strategy), they are available in the replication package.

More specifically, each of the three researchers considered a random sample of 5 primary studies and analyzed them independently by filling the data extraction form for each of them. Then, each disagreement was discussed and resolved with the intervention of a fourth researcher. Specifically, this process was carried out by jointly inspecting the disagreement items, and subsequently providing references available in the literature fitted to solve the item under discussion. For example, an early disagreement item arose between two researchers on the internal or external nature of a quality attribute. Such item was solved by escalating the item to a fourth researcher, who provided a reference to the relative standard available in the literature [115] and additional examples of both types of attributes.

4.4 Data synthesis

The data synthesis activity involves collating and summarizing the data extracted from the primary studies [134] with the main goal of understanding, analysing, and classifying current research on static analysis of mobile apps.

Our data synthesis was split into two main phases: vertical analysis and horizontal analysis. When performing **vertical analysis**, we analyzed the extracted data to find trends and collect information about each parameter of each category of our classification framework. When performing **horizontal analysis**, we analysed the extracted data to explore possible relations across different parameters of our classification framework. We used contingency tables for evaluating the actual existence of those relations[10].

In both phases, we performed a combination of content analysis (mainly for categorizing and coding the studies under broad thematic categories) and narrative synthesis (mainly for explaining in details and interpreting the findings coming from the content analysis). During the horizontal analysis, we used contingency tables for evaluating the actual existence of inter-parameter relations.

5 Results - Research Trends (RQ1)

5.1 Year of publication

An overview of the year of publication of the primary studies is reported in Figure 2. Overall, the publication rate results to be constantly increasing through time until 2016. In 2017 there were registered a significant decrease in the publication rate (-25 publications with respect to the previous year). The number of publications surges in 2018 (+20 publications) and remains almost constant in 2019. The lack of growth in recent years could indicate that the initial push, tied to the novelty of the topic, has now stopped. However, the coming years will be decisive in confirming or denying this trend.

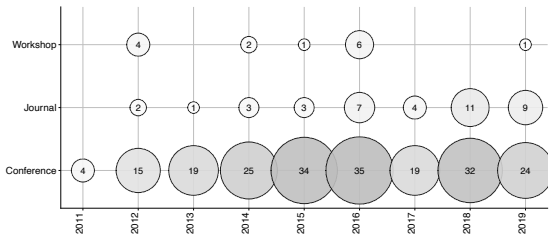

Figure 2: Bubble plot of primary studies by year and venue type

A steep increase of the publication rate can be noticed between the years 2011-2012 and 2015-2016, with a difference of 17 and 10 publications, respectively. We can conjecture that the first steep increase (years 2011-2012) is due to the popularity gained in those years by the Android operating system, with its version 4.0. The appearance of lightweight static analysis approaches for mobile application, e.g., Flowdroid [12], could instead be one of the root causes of the increase of publications between the years 2013 and 2014. No publication was found before the year 2011. Considering that the concept of mobile app originated in 2007, we conjecture that the lack of publications in the years 2007-2011 is attributable to the time required by mobile apps to gain widespread diffusion and, hence, for the topic considered (static analysis methods for mobile) to attract the interest of researchers.

5.2 Publication venue

Studies on static analysis of mobile apps have been published to a certain extent in all the most prominent top-level conferences and journals in software engineering. An overview of the most targeted venues and the papers there published is reported in Figure 3. The ICSE conference results to be the venue in which most studies on this topic were published (31/261), followed by ASE (30/261). Overall, a high heterogeneity can be found in the publication venues, which led to a total number of 112 different venues. Only a small number of venues results to be focused on mobile related topics. The vast majority of targeted venues is on general areas of computing, e.g., software engineering, security, testing and program analysis.

5.3 Publication venue type

As shown in Figure 4, most of the papers were published in conferences (207/261), followed by journals

[10] For our horizontal analysis we applied the same process as the one in [80].

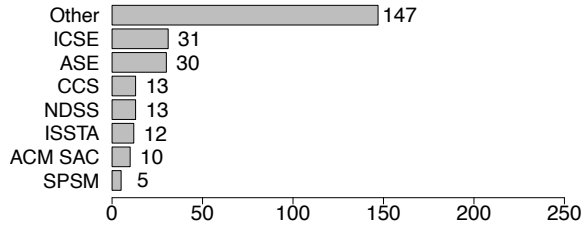

Figure 3: Most targeted publication venues

(40/261) and workshops (14/261). The higher number of conference papers might be due to the high pace of technological advances in the topic. Specifically, researchers may have focussed more on timely publications in conference, rather than targeting journals, which have a (usually) slower publication timeline. Interestingly, as shown in Figure 2, 31 out of 40 journal papers were published from 2016 onwards, which can be considered as an indication of the maturing of static analysis techniques for mobile apps as a scientific topic.

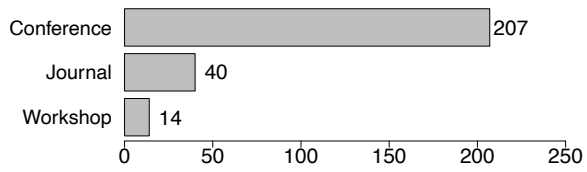

Figure 4: Primary studies by venue type

5.4 Analysis goal

The *analysis goal* represents the principal purposes for which the static analysis approaches were conceived. By carefully analyzing the primary studies, sixteen main analysis goal categories emerged from the keywording process. In Figure 5, the comprehensive mapping of primary studies to analysis goals is reported. The most recurrent goals are: *privacy* (96/261), *malware* (66/261), *inter-component communication* (33/261), *energy* (25/261) and *inter-app communication* (24/261).

From an inspection of the more recurrent goals, we can observe that most of the studies focus either on analysing crucial aspects of the mobile ecosystem (e.g., *privacy* and *malware*) or on improving existing analysis methods (e.g., *inter/intra-component communication*). We can conjecture that this trend may be due to the fast pace of development that usually characterizes mobile application, where new app releases must be quickly developed and tested in order to be published in the app stores. This may lead to a lack of

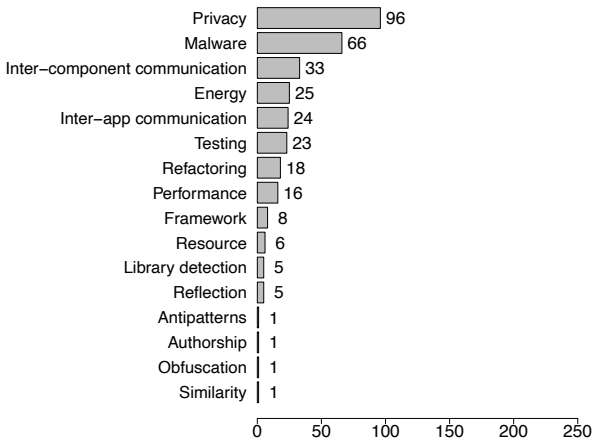

Figure 5: Primary studies by analysis goal (Categories are not mutually exclusive)

interest in analysing less critical software aspects of the app, such as refactoring the code of the app itself or identifying specific code anti-patterns.

Example. The European Union data protection regulations impose restriction on the locations of European users' personal data transfer. In P2, Eskandari et al. investigate whether these regulations are respected by mobile apps, thus safeguarding end users *Privacy*. For this purpose, they developed *PDTLoc*, a static analysis tool that analyzes an app to identify the location of servers to which personal data is transferred.

5.5 Macro analysis goal

The *macro analysis goal* refers to the generic goal considered by the static analyses. The values of this parameter are based on the definition of *internal* and *external* quality attribute provided in the ISO/IEC 25010 standard [115]. Specifically, external quality attributes provide a "black box" view of the software under consideration and address properties related to the execution of the software on hardware and an operating system, e.g., reliability [115]. Internal quality attributes provide a "white box" view of software under consideration and address static properties that typically manifest themselves at development time, e.g., maintainability [115]. So, the macro analysis goal of a primary study can have the following values: (i) *external quality*, if the approach evaluates one or more external quality attribute and (ii) *internal quality*, if the approach evaluates one or more internal quality attributes. In order to identify approaches which explicitly aim to improve existing methods referenced in the literature, we have a third possible value for this parameter called *improving of methodology*; we use such

value if the main goal of the primary study is to improve a static analysis method or technique.

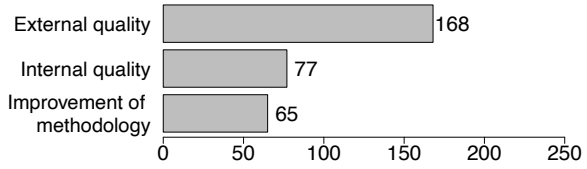

Figure 6: Primary studies by macro analysis goal (Categories are not mutually exclusive)

The macro analysis goals considered by the primary studies are reported in Figure 6. The majority of the primary studies focus on external quality (168/261). A smaller amount of studies focuses on the improvement of static analysis methodologies (77/261) and on internal quality (65/261)[11]. From this data, we conjecture that the high pace of the mobile technological advances and the strong role of end users in the mobile ecosystem are leading researchers to give more importance to external qualities. Research aimed to refine static analysis approaches results to be higher than the ones focusing on internal quality, making us conjecture that the ones considering internal quality are either at an early stage of development or have been less explored than the ones improving the existing methods.

In addition, the distribution of macro analysis goals throughout the years is depicted in Figure 7. Here, we observe that, although studies focusing on external quality have been the majority in each of the considered years, a steady increase in number can be observed for studies that focus on either methodology improvement or internal quality, from 2013 to 2016. Due to the decrease in number of publications occured in 2017, studies that focus on either methodology improvement or internal quality also decrease.

Example. A resource leak is a common bug caused by missing release of resources that require programmers to explicitly release them (e.g., camera and sensors). Although not directly observable by end users, a resource leak might lead to several problems such as performance degradation and occurrences of crashes. *Relda2* (P42), a light-weight static analysis tool for the automatic detection of resource leaks in Android apps, is an example of a primary study aimed at improving an *internal quality* attribute.

[11]It is important to note that these categories are not mutually exclusive, i.e., a paper could be mapped to more than one category if it addresses more than one type of goal.

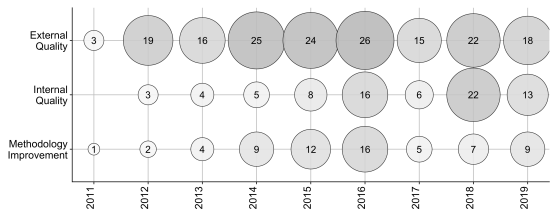

Figure 7: Macro analysis goal by year (Categories are not mutually exclusive)

5.6 Paper goal

This parameter can be of two types, namely: (i) *Quality attribute assessment*, if the research reported in the primary study focuses on assessing a quality attribute of mobile apps (e.g., security); (ii) *Improvement of methodology*, if the research reported in the primary study focuses on improving existing static analyses for mobile apps.

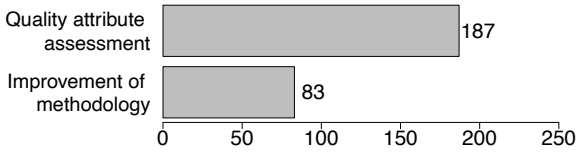

Figure 8: Primary studies by analysis goal (Categories are not mutually exclusive)

The goals taken into account by the primary studies is documented in Figure 8. The majority of the primary studies (187/261) focuses on the assessment of some quality attribute(s) of mobile apps. A lower number instead (83/261) considers the improvement of static analysis techniques. We can conjecture that this trend can be associated to the more "immediate impact", e.g., ease of adoption and real-life utilization by practitioners. From this, we can conjecture that a certain maturity with respect with assessment of apps quality attributes has been achieved (and hence a high presence of such approaches is observable), which is reflected in the reasonable amount of techniques aimed exclusively at improving the existing methods.

Example. *Ripple* (P4) is an incomplete information environment aware static reflection analysis for Android apps. Ripple is an *improvement of methodology*, as it is able to resolve reflective calls more soundly than conventional string inference. It enables more precise taint analyses when used in combination with tools such as FlowDroid (P86).

Main findings on research trends:

- The intensity of research on static analysis for mobile apps has been growing year by year, until 2016, especially after app-specific techniques have been devised (e.g., Flowdroid). Following a slight fluctuation in 2017, the number of publications seems to have recovered and remains stable in most recent years.
- Researchers are targeting primarily conferences (e.g., ASE and ICSE), even if journal publications have been much more targeted in recent years.
- Many of the approaches are focussing on security-related concerns, such as privacy leaks identification and malware detection.
- Approaches for enhancing the modeling and analysis of both inter-component communication (e.g., intent raising across Android activities) and inter-app communication are receiving quite an intensive scientific attention.
- The vast majority of primary studies is targeting the assessment of external quality attributes (e.g., security, energy consumption). Only a smaller portion focussed on assessing internal quality attributes (e.g., maintainability, reusability).
- Reasonable research effort is being devoted to the improvement of the methodology, such as devising more sound static analyses support for more events in the mobile components lifecycles (e.g., Android intents sharing).

6 Results - Characteristics of Approaches (RQ2)

6.1 Platform specificity

This parameter identifies whether the proposed approach is specifically designed for a specific platform (e.g., *Android* or *iOS*) or if it is *generic* and can in principle be applied to any platform. As shown in Figure 9, the vast majority of the approaches (239/261) presents an analysis approach specific for Android; only one study (1/261) presents an approach specific for iOS. A smaller amount of studies (21/261) presents an approach that is *generic*. Possible reasons for this imbalance may be due to the popularity and the open-source nature of the Android platform, which eases the effort required by researchers during the design of new analyses. Furthermore, Android app binaries can be straightforwardly disassembled with off-the-shelf software libraries (e.g., apktool[12], dex2jar[13]), and their internal structure and contained static resources are easily analyzable in an automatic way.

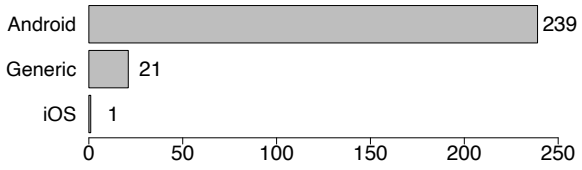

Figure 9: Primary studies by platform specificity

Example. As an example of platform specificity, P20 presents a technique to optimize energy consumption of mobile apps minimizing the number of HTTP requests that they perform. Proposed technique uses static analysis to detect Sequential HTTP Requests Sessions, i.e., sequences of HTTP requests in which generation of the first request implies that the following requests will also be made. Energy savings can be achieved by bundling these requests. The technique is *Generic* and applicable to all major mobile platforms, as mechanisms available to perform HTTP requests are similar across these platforms.

6.2 Implementation

Values for the *implementation* parameter, summarized in Figure 10, were extracted from the primary studies according to whether the implementation used for evaluation purposes is implemented for a specific platform, e.g., Android or iOS, or it is Generic, applicable to apps developed for any platform.

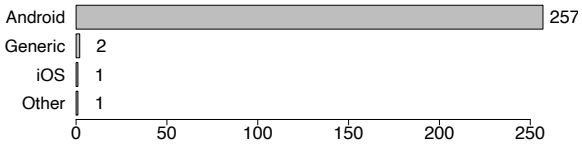

Figure 10: Primary studies by platform implementation

Almost all the studies (257/261) implement the proposed approach exclusively for the Android platform. Two studies present approaches (2/261) having a generic implementation, applicable to any platform. Only one study (1/261) presents an approach that is implemented specifically for the iOS platform. Other less popular platforms are almost completely absent,

[12] http://ibotpeaches.github.io/Apktool

[13] http://github.com/pxb1988/dex2jar

with only one study (1/261) implementing the proposed analysis on TouchDevelop scripts [234]. We speculate that the reason for this disproportion, in addition to the ones already evidenced in the discussion of the *platform specificity* parameter, stem from the fact that some of the most popular static analysis frameworks (e.g., Soot [239] and WALA [209]) are adapted to support analysis of Android apps. The same cannot be said for the other platforms and, hence, researchers interested in performing static analysis on apps designed for those platforms experience a higher barrier to entry as they must develop their own tools, often from scratch.

Example. *PiOS* (P134) studies the privacy threats that applications written for Apple's iOS may pose to users. To this end, the authors leverage static analysis techniques to extract data flows from iOS apps. PiOS is an *iOS*-specific implementation of the proposed technique, that automates the data flow extraction process from binaries resulting from the compilation of Objective-C code.

6.3 Static/Hybrid approach

The *static/hybrid approach* parameter describes whether an approach relies on static analysis only (Static) or utilizes some form of dynamic analysis also (Hybrid).

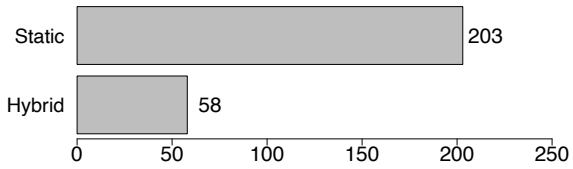

Figure 11: Primary studies by usage of dynamic analysis

Results for the extraction of this parameter are summarized in Figure 11. The preponderance of the studies (203/261) present an approach that relies on static analysis only. Nonetheless, a considerable amount of them (58/261) present an approach that complements static analysis with dynamic analysis. The presence of dynamic analysis in a considerable portion of the studies can be explained by considering that, despite all its drawbacks, dynamic analysis still provides an invaluable contribution for a variety of purposes, such as privacy leaks detection, GUI-modeling, energy profiling. A further discussion on the fields where dynamic analysis is most common can be found in Section 8.

Example. *SmartDroid* (P113) is an *hybrid* analysis technique whose goal is identifying UI-based trigger conditions required to expose the sensitive behavior of Android malwares. As shown in Figure 14, SmartDroid uses static analysis to extract Activity and Function call graphs from the application binaries. Then, guided by the static analysis results, it uses dynamic analysis to interact with the UI and identify UI-based conditions required to trigger sensitive APIs.

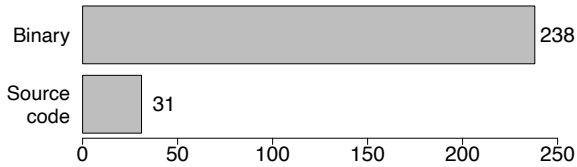

Figure 12: Primary studies by additional app artifacts (Categories not mutually exclusive)

6.4 Usage of machine learning techniques

Values for this parameter are summarized in Figure 13. The possible values identify whether the approach under evaluation complements its analysis with machine learning techniques (Yes) or not (No). A vast majority of the studies (213/261) does not make use of machine learning in the proposed approach. The remaining studies (48/261) perform features extraction from the application source code or other intermediate representations (e.g., a method-level call graph), and applies machine learning techniques on the extracted features. Machine learning techniques are widely used for some specific goals (e.g., malware detection), but their application to others has not been explored yet by researchers.

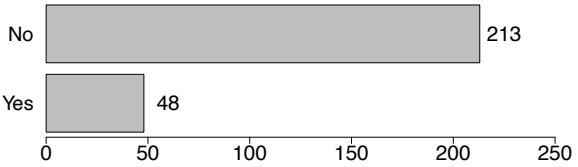

Figure 13: Primary studies by usage of machine learning

Example. An example of usage of *machine learning* coupled with static analysis is P28. In this study, the authors adopts a machine learning approach that leverages the use of data flow application program interfaces (APIs) as classification features to detect Android malware. Static analysis is employed to extract data flow related API-level features, used to train a k-nearest neighbor model for malware classification.

6.5 App artifact

The values of this parameter describe what formats are accepted as input by the selected studies for the apps to be analyzed. As shown in Figure 12, the majority of the studies (238/261) accepts as input apps in the form of binary packages (Binary), i.e., APK (Android PacKage) files for the Android platform or IPA (iPhone Application Archive) packages for the iOS platform.

This implies that the proposed analyses can be performed by a variety of subjects (app store moderators, researchers, security experts), and not only by app developers. Nonetheless, a considerable amount of primary studies (31/261) takes as input the app source code (Source Code), hence targeting app developers and researchers. In those cases, developers can potentially integrate them into their development workflow, e.g., as dedicated analyses integrated into the Android Studio IDE or as specific steps in their continuous integration pipeline. Note that both APK and source code are valid inputs for some of the studies.

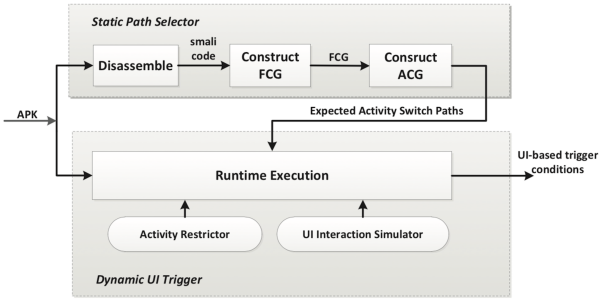

Figure 14: Example of an hybrid analysis technique

6.6 Additional inputs

The possible values for the *additional inputs* parameter, listed in Figure 15, identify what other inputs, if any, are required by the primary studies to perform the proposed analysis (in addition to the app itself). Overall, the majority of primary studies (194/261) is able to perform the analysis without any additional input, whereas 67/261 studies require some additional inputs. We consider this to be a positive trend, as it simplifies the adoption of the proposed techniques by industry and other researchers, additionally enabling batch analysis of a large quantity of apps more easily. Nevertheless, as for P123, in some cases relying on additional inputs is a necessity, e.g., when the app needs to be executed in a controlled, non-random, and non-trivial manner.

When focusing on the studies requiring additional inputs, we can observe that additional inputs are mostly required by techniques that verify whether given policies, rules, or constraints are violated (13/261). This

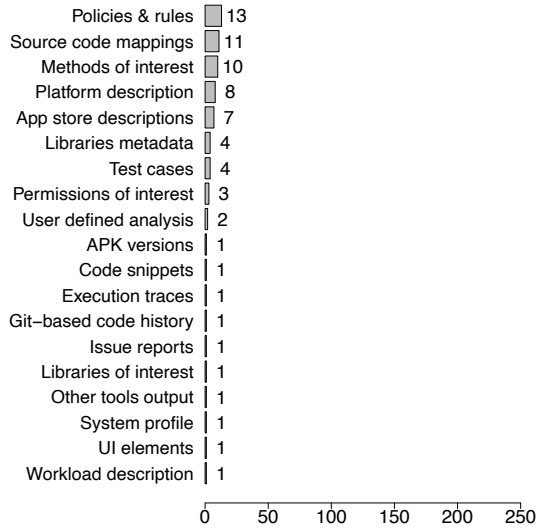

Figure 15: Primary studies by additional input (Categories are not mutually exclusive)

is followed by mappings from the source code of the app to other auxiliary information (11/261) and by techniques that focus on a list of one or more methods leveraging the app source code (10/261). A number of studies (7/261) take as input app descriptions retrieved through app stores, and leverage this information in order to perform ad-hoc analyses. For example, CHABADA [94] aims at automatically identifying malicious apps by evaluating how their implementation differs from their description in the app store. Some proposed techniques take as input the platform (8/261) or system (1/261) profiles for application execution. Other studies (4/261) take as input test cases. This is particularly noteworthy as test cases are artifacts commonly produced during the software development cycle, and how information can be extracted from test artifacts has widely been investigated in the software engineering literature [3, 141]. Other studies (3/261) focus on problems pertaining to system permissions and, consequently, take as input an identifier of the permissions of interest. Two studies (2/261) require as input the specification of a user-defined analysis. Similarly, requires the user to write down some additional code snippets to perform the analysis (1/261). Two studies focus on app evolution and extract change information from multiple APK versions (1/261) or from the Git repository code history (1/261). One study requires a description of the workload to be executed (1/261) and one study requires execution traces (1/261). One study focuses on the behavior triggered by the interaction with user

interface (UI) elements and hence requires as input a list of the latter (1/261). Interestingly, only one study (1/261) leverages information extracted from bug reports to perform the analysis and only one study takes as input information provided by other analysis tools (1/261). It is important to notice that the vast majority of these additional inputs require the knowledge of a developer or a domain expert in order to be reproduced and only a handful can be reproduced by end-users. This makes it harder to reproduce the results and might hinder large-scale adoption.

Example. As an example, Figure 16 presents *eCalc* (P123), a technique involving two main steps that are performed by an Execution Traces Generator and an Analyzer, respectively. The Execution Traces Generator uses test cases for generating execution traces. Although this step requires to execute the software artifact under analysis, the actual analysis step is statically performed by the Analyzer on the execution traces by taking as input a CPU profile, without requiring the execution of the software artifact. This additional input is needed for automatically running and profiling the app under analysis multiple times in order to take into account the well-known phenomenon of energy consumption fluctuations at run-time.

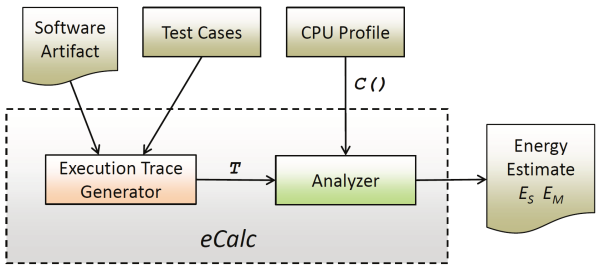

Figure 16: Example of an analysis technique requiring additional inputs

6.7 Analysis pre-steps

The *analysis pre-steps* parameter identifies whether the studies under evaluation require steps that must be executed manually before the analysis can be performed. Results are listed in Figure 17.

The majority of the approaches (192/261) does not require any analysis pre-step. A still considerable amount (69/261) requires some analysis pre-step to be performed manually. Examples of possible pre-steps include, but are not limited to, building models of the platform APIs or libraries used by the application under analysis, collecting execution traces, collecting runtime power consumption measures, creating rule sets or security policies. Similarly to the previous parameter, having to perform manual steps before or during the application of a static analysis approach may hinder its reproducibility and large-scale adoption.

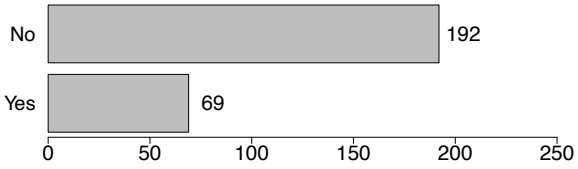

Figure 17: Primary studies by need of analysis pre-steps

Example. *UIPicker* (P71) is a primary study that makes use of preprocesing steps. UIPicker aims to reduce the risks to which users are exposed when using an application by automatically identifying sensitive user inputs. To this end, in its preprocessing module, it extracts the layouts texts and reorganizes them through natural language processing techniques for further usage. This pre-step includes word splitting, redundant content removal and stemming.

6.8 Analysis technique

This parameter identifies the family of static analysis techniques performed by the approaches proposed in the primary studies. Results are summarized in Figure 18.

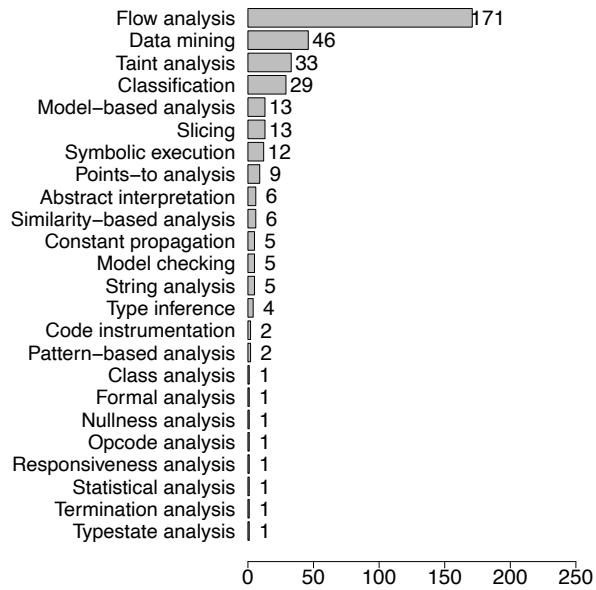

Figure 18: Primary studies by analysis technique (Categories are not mutually exclusive)

A wide variety of static analysis techniques is used in the primary studies, the most common being Flow (171/261). A considerable amount of primary studies limit their analysis to data mining (46/261) to extract relevant information from the application bytecode or source code. Taint Analysis (33/261) follows as the third most adopted analysis technique. Machine learning classification, slicing and model-based analysis are also other relevantly used techniques, each being used in twenty-nine (29/261), thirteen (13/261), and thirteen (13/261) studies, respectively. Other less frequently used techniques are symbolic execution (12/261), points-to analysis (9/261), abstract interpretation (6/261), similarity-based analysis (6/261), constant propagation (5/261), string analysis (5/261), model checking (5/261), type inference (4/261), code instrumentation (2/261), pattern-based analysis (2/261), code-instrumentation (2/261), class analysis (1/261), formal analysis (1/261), opcode analysis (1/261), nullness analysis (1/261), responsiveness analysis (1/261), statistical analysis (1/261), termination analysis (1/261), and typestate analysis (1/261). We speculate that the popularity of Flow and Taint analysis is due to the fact that many of the issues researchers want to detect in mobile apps can be modeled under those analysis paradigms and, as further discussed in Section 8, it appears that researchers identify the technique to be used in a goal-driven fashion. We also believe that, again, researchers are limited by the available frameworks and tools, and choose to focus more on those techniques for which mature tools exist (e.g., Soot).

Example. *AppSealer* (P79) aims to automatically detect and prevent *component hijacking attacks*, a class of vulnerabilities commonly appearing in Android applications. When triggered by attackers, the vulnerable apps can expose sensitive information and compromise data integrity. For this purpose, AppSealer employs a combination of flow analysis and backward slicing. First, flow- and context-sensitive inter-procedural dataflow analysis is performed to track the propagation of sensitive information and detect if it propagates into dangerous data sinks. Then, employing backward slicing, one or more program slices that directly contribute to the dangerous information flow are computed. With the guidance of the computed slices, AppSealer automatically creates patches to deal with the discovered vulnerability, placing guarding statements at affected sinks to block the propagation of dangerous information.

Main findings on characteristics of approaches:

- Being open source pays off from a scientific perspective. The vast majority of the studied approaches is specific to the Android platform, both from a conceptual and implementation perspective. Thanks to its open-source nature, Android gives more control and flexibility, and fuels an ecosystem of accompanying tools and libraries useful for static analysis (avoiding to reinvent the wheel). This is also proved by the fact that Android has been chosen as implementation platform also for generic static analysis approaches.
- Static analysis of mobile apps is widely performed in isolation and by considering only the app to be analyzed (no additional input like test cases or platform profiles). If on the one side this is a confirmation of the fact that static analysis is a very versatile tool for analyzing non-trivial properties of mobile apps, on the other side, researchers may be loosing an opportunity for pushing further by complementing static analysis with other artifacts and/or additional analysis techniques (e.g., like done in the eCalc approach in P123).
- Machine learning techniques seem to be promising and are applied in conjunction with static analysis techniques. Machine learning techniques are widely used for some goals (mainly for security), but they are not yet fully explored in other areas, such as app store analysis [180] or software repository mining.
- Many are the static analysis techniques used by researchers when considering mobile apps, ranging from flow analysis to taint analysis, to type inference and abstract interpretation. The clear winner is flow analysis. We conjecture that this success is mainly due to a combination of factors: (i) as of today, the programming model of mobile platforms is inherently based on a flow of (often asynchronous) messages exchanged between a set of components (e.g., Android activities, iOS views) reacting to events (e.g., a tap of the user, a callback from a sensor request); (ii) flow analysis nicely lends itself to identify and predicate on both intra- and inter-app interactions (a cornerstone capability for security and reliability analyses); (iii) the availability of open-source tools like Soot that developers can use as building blocks for their own approaches.

7 Results - Potential for Industrial Adoption (RQ3)

7.1 Target stakeholder

As shown in Figure 19, app *developers* are the most recurrent stakeholders of static analysis approaches (150/261).

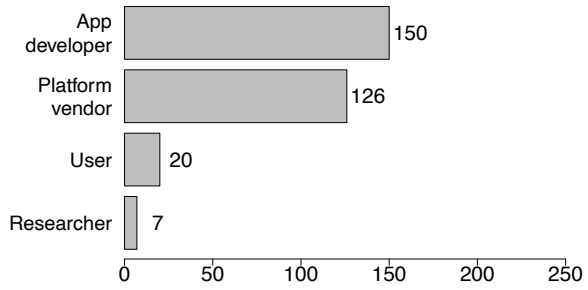

Figure 19: Primary studies by target stakeholder (Categories are not mutually exclusive)

Platform vendors (126/261) like Apple and Google distribute apps via their own dedicated mobile application markets. They can benefit from the use of static analysis approaches in their market places for systematically assessing the level of quality of their distributed apps, possibly identifying those apps with an unacceptable level of quality (e.g., apps with well-known security flaws, apps asking for suspicious permissions, apps with strong energy inefficiencies). Interestingly, some approaches directly target app *users* (20/261), who might use static analyses to better understand how their installed apps behave and for examining and granting explicit information flows within an application. Also, users may be interested in implicit information flows across multiple applications, such as permissions for reading the phone number and sending it over the network. As an example, one of the 12 studies targeting users focuses on debugging energy efficiency of apps in their real context of use. Specifically, in P39 the user can launch an automatically instrumented app to precisely record and report observed energy-related failures in order to assists the developer by automatically localizing the reported defects and suggesting patch locations. Last but not least, 7 primary studies explicitly mention *researchers* as target stakeholders, who can extend and/or apply the proposed techniques (and their results) to their own studies on mobile applications.

Example. *FicFinder* (P22) aims to ease the effort required by *developers* to deal with compatibility issues that might be present in their apps due to the fragmented nature of the Android platform. FicFinder automatically detects compatibility issues by performing static code analysis based on a model that captures Android APIs behavior as well as their associated context by which compatibility issues are triggered. Once detected, FicFinder reports actionable debugging information to developers.

7.2 Tool availability

All the primary studies contribute with a tool implementing the proposed approach. Nonetheless, our results also show that only 97 studies over 261 (see Figure 20) released the tool, making it publicly available for download and adoption. When possible, the availability of a tool supporting the proposed approach is desirable as it surely helps in making the obtained results more credible, reproducible, and replicable by the community.

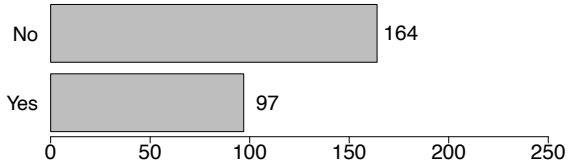

Figure 20: Primary studies by tool availability

7.3 Number of analysed apps

The authors of the analyzed primary studies evaluate and validate their findings by using an input set of applications. The evaluation of this parameter builds on the assumption that approaches evaluated on a larger set of apps are more adoptable in industry since it is less likely that they exhibit unexpected behaviors (specially for corner cases). Here, we categorized the primary studies according to the number of apps used for evaluating them.

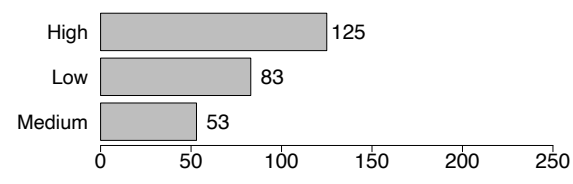

Figure 21: Primary studies by number of evaluated apps

As shown in Figure 21, in the majority of studies (125/261) the number of applications used for evaluating the proposed approach is greater than 1,000, followed by those studies which evaluated their approach by using less than 100 apps (83/261), and those studies

(53/261) which took into account a medium set of apps (between 100 and 1,000). This result is promising in that a relatively good number of approaches was evaluated on a high number of applications, making the scientific community and practitioners reasonably confident about their applicability in industrial contexts. Nevertheless, it is important to note that evaluating an approach on a low number of apps should not be seen as a strongly negative point because it may have been a necessity from an empirical perspective. For example, the number of analyzed apps could depend on the execution time of the analysis tool; if the analysis tool requires a large amount of time for each app (e.g., because including user thinking time), then the input set of applications is inevitably small in order to keep the experiment duration acceptable from a pragmatic perspective.

Example. *AutoPPG* (P15) aims to facilitate the process of writing privacy policies for mobile apps. A privacy policy is a statement informing users how their information will be collected, used, and disclosed. Failing to adhere to privacy policies is can lead to severe consequences, such as the issue of steep fines. AutoPPG conducts static code analysis on mobile apps by extracting their behavior and subsequently relating such behavior to the personal information stored by the end-users. Once the relation between the app behavior and personal data is established, AutoPPG leverages natural language processing techniques to generate a textual description of the fair privacy policy which characterizes the analyzed app. Due to the time consuming nature of manually comparing the statically generated privacy policy with the existing one, the evaluation of AutoPPG was limited to the *low* number of 20 randomly selected apps.

7.4 Applied research method

This parameter represents the type of applied research method used to assess the proposed technique. Possible values of this parameter are *Validation* and *Evaluation*. *Validation* is done in lab contexts using applications specifically created or customized for the purpose of their approach evaluation. *Evaluation* takes place in real-world (industrial) contexts, using exclusively unmodified applications. The latter generally provides a higher level of evidence about the practical applicability of a proposed technique.

From the analysis of the primary studies, it emerged that the majority, during the evaluation phase, use exclusively unmodified applications (see Figure 22) mined from an app market (234/261). In other cases, the applications to be analysed were created for the purpose of the evaluation, or they were customized versions of real apps (46/261). In some cases (e.g., P14, P17, P169, P259), a combination of real and custom applications is used; in these cases, custom apps support the evaluation of the proposed approach to exercise specific aspects of the proposed static analysis approach (e.g., corner cases when building a control flow graph of the app under analysis), which are not fully covered by the mined original apps.

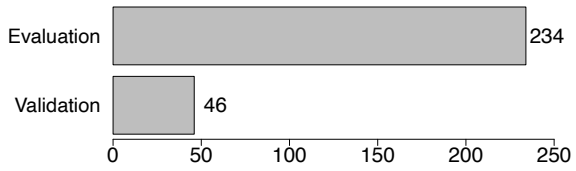

Figure 22: Primary studies by applied research method (Categories are not mutually exclusive)

Overall, the obtained results are promising since approaches evaluated on (a potentially large number of) real apps, in principle, undergo a more realistic investigation with respect to those evaluated on synthetically-built apps. This realism comes also from the fact that apps mined from app stores are developed in real industrial contexts involving practitioners working under real business and organizational constraints (e.g., release deadlines, specific development workflows). Moreover, apps mined from app stores can be totally different from synthetic apps because the former are distributed to and downloaded by real users; it is well known that users play a central role in the success (and indirectly in the development process) of the apps, e.g., by providing publicly accessible app ratings and reviews [180], deciding to uninstall disappointing apps.

Example. In P14, the authors propose two automated static analysis techniques for automatic detection of a privilege-escalation attack known as Android Wicked Delegation (AWiDe). In order to manually verify the correctness of the two detection techniques, apps for evaluation experiments were collected from F-Droid [14], an online repository of free open source Android apps, in order to be able to inspect the app source code. As 70% of collected apps were also published on the Google Play store the study performs both *Validation* and *Evaluation*.

7.5 Industry involvement

Each primary study was classified as (i) *Academia*, if the authors are affiliated exclusively to an academic organization, e.g., university or research center; (ii) *Industry* if the authors are affiliated exclusively to an

[14] https://f-droid.org/en/

industrial organization, e.g., a company, startup, or software house; (iii) *Academia and Industry* if some of the authors are affiliated to an academic organization and some others to an industrial one. As depicted in Figure 23, the vast majority of the authors of our primary studies is academic (231/261), followed by a combination of researchers and industrial practitioners (29/261), and finally 1 contribution involves industrial authors only. The emerged result is quite disappointing, as in almost all of the studies there is no involvement of industrial researchers or practitioners.

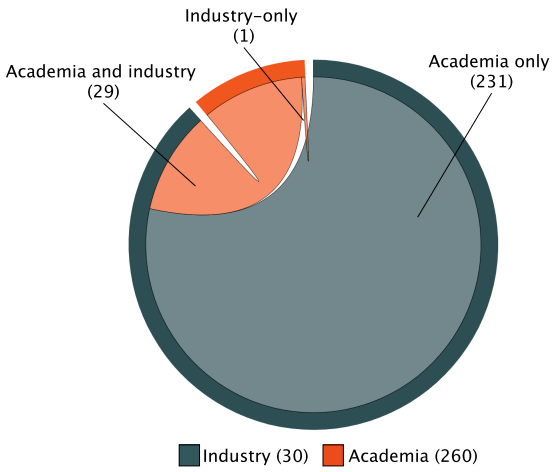

Figure 23: Distribution of industry involvement

In the single industry-only primary study (P91), the authors tackle the problem of Android application collusion. Specifically, they state that existing analysis techniques focused on identifying undesirable behaviors in single-apps neglecting multi-application collusion danger. Therefore, the authors present a collection of tools that provide static information flow analysis across sets of applications, showing a holistic view of all the applications running on a particular device. The techniques proposed in P91 include: (i) static binary single-app analysis, (ii) security lint tool to mitigate the limits of static binary analysis, (iii) multi-app information flow analysis, and (iv) evaluation engine to detect information flows that violate specified security policies. We believe that P91 is a good example of a research study tackling an industrially-relevant problem and proposing an industry-driven solution. Academic researchers could compare with or be inspired by the work in P91 for designing and evaluating the approaches for static analysis of mobile apps of the future.

Main findings on potential for industrial adoption:

- It comes without a surprise that app developers and platform vendors are the most targeted stakeholders. Still, a potentially unexplored venue is related to static analysis targeting the end users of mobile apps, who may have different requirements and needs with respect to the apps currently installed in their devices.
- In the vast majority of primary studies, researchers are not providing any tool implementing their proposed approaches. This result is strongly negative, as it impacts studies replications and comparative evaluations, which are at the basis of the scientific method. We suggest researchers to always provide publicly available implementations of their approaches (when possible); this will help researchers and practitioners in improving the overall quality of research in static analysis of mobile apps.
- The evaluation of the proposed approaches is generally performed on unmodified apps (i.e., experimentation in the wild). The number of apps considered in the evaluation phase is either high (more than 1,000) or low (less than 100).
- As a community, we should encourage new connections between academia and industry in order to potentially improve the knowledge exchanged between them, where (i) research is performed on industrially relevant problems and (ii) new methods, technologies and tools are transferred from academia to industry.

8 Orthogonal Findings

This section reports on the results of our horizontal analysis. It is worth recalling that, in this phase of the study, we (i) built contingency tables for pairs of parameters coming from our vertical analysis, (ii) analyzed each one of them, and (iii) identified perspectives of interest.

Analysis goal - Platform specificity. *Privacy* is the most recurrent analysis goal for all platforms, especially for the Android operating system. The only iOS approach found in the literature is also focusing on privacy. *Malware* results to be the second most studied subject in both Android and generic approaches. Overall, very few studies are platform-independent, and none for the categories *performance*, *inter-app communication*, and *antipatterns*.

We conjecture that the popularity of *privacy* and *malware* analysis goals can be associated to the ubiquity and handling of sensitive data that nowadays characterizes mobile apps. As a consequence, new methods and techniques to address the associated challenges is receiving a growing attention. Indeed, many of the researches focusing on *privacy* rely on a technique, namely, the inspection of the `AndroidManifest.xml`, which is quite simple to implement. This consideration further explains the high occurrences of such studies. Regarding the *performance*, *inter-app communication* and *antipatterns* goals, we hypothesize that such goals can be studied exclusively from a platform-specific point of view due to their tight relationship with the platform on which the app is running.

Analysis goal - Static\Hybrid approach. Except for *frameworks* and *antipatterns*, which result to be supported exclusively by static analysis, the majority of the goal categories are studied through hybrid approaches. Overall, *privacy* results to be the most studied subject in both static and dynamic approaches (74 static approaches and 22 hybrid ones). *Energy consumption* (13 static approaches and 12 hybrid ones) is the second most recurrent goal of hybrid analyses.

We believe that the rationale behind the popularity of hybrid approaches resides in the ability to circumvent weaknesses that arise when using only one kind of analysis, hence making it possible to gather more comprehensive, yet precise, results. As presented in the previous section, the popularity of the *privacy* goal can be justified by the interest of final users, developers and app store vendors to protect sensitive data from unauthorised access. The high number of hybrid approach targeted at the *energy* goal evidences the reliance of such approaches on dynamic methodologies, utilised to exercise the applications under analysis, and gather empirical energy consumption measurements. On the other hand, we conjecture that the lack of usage of dynamic analysis by approaches aimed at the *frameworks* and *antipatterns* goals is due to the nature of these goals, which are more tightly related to source code metrics rather than runtime ones, thus making static analysis techniques more suitable for them.

Analysis goal - App artifact. In general, the vast majority of the approaches require the *APK* package of the mobile application. This has to be attributed to the skewed data gathered for this research, from which most of the approaches result to focus on Android applications. In contrast, the goals that require more often source code are the ones focusing on *refactoring* and *performance*. Additionally, some goals that do not require access to the source code of the application were identified, namely *reflection*, *antipatterns*, *similarity*, *obfuscation*, and *authorship*.

Regarding the goals for which analyses are often performed on source code, we believe that the reason underlying this trend is that these types of analysis require the exact source code of the app under analysis to be carried out properly. Even though Android decompilers and disassemblers do exist, at the time of writing, their precision is not high enough to perform these kind of analysis on packaged applications [192]. On the other hand, when focusing on the analysis goals requiring an *APK* as input, we can notice that for *testing*, *privacy* and *energy consumption* researchers have been focusing on black-box approaches, while neglecting white-box ones (at least partially). For these goals, approaches of the latter kind could be of assistance during development of mobile apps, either notifying developers when they unknowingly insert known antipatterns in their code (e.g., an energy hotspot in the case of *energy consumption* or a privacy leak in the case of *privacy*) or in helping them in performing more efficient testing (in the case of *testing*).

Analysis technique - Analysis pre-steps. Eight out of 24 analysis techniques do not require pre-steps. In fact, *nulness*, *points-to* and *termination* analyses are carried out by inspecting the source code repository of the application, and hence do not require additional tooling or configuration. The remaining 16 analysis techniques require pre-steps of different nature. As expected, most of the analysis techniques needing pre-steps require the manipulation of source code, such as *abstract interpretation* (for which two out of three papers required analysis pre-steps). In general, only three of the 24 identified analysis techniques resulted to require in the majority of the cases analysis pre-steps. This indicates that the vast majority of analysis techniques is executable "as is", i.e., without requiring any additional process before the analysis can be actually carried out.

Target stakeholder - Analysis goal. Approaches targeting *app stores vendors* result to be mostly interested in *malware* (57) and *privacy* (56 studies), followed by *inter-app* and *inter-component communication* (15 and 14 studies respectively). Approaches targeting *developers* also result to be mostly interested in *privacy*-related analyses (48 studies), but also consider more low-level goals, such as *energy consumption* (25 studies), *inter-component communication* (24 studies), and *testing* (23 studies). Approaches targeting *researchers* result to be mostly related to the improvement of the state of the art analysis techniques, hence often considering goals related to *inter-component communication* (4 studies), and *frameworks* (3 studies). As expected, approaches targeting *end users* result to be mostly interested in *privacy* (14 studies), and approaches targeting *app store vendors*

are more interested in *malware* than *developers* (57 against 8 studies). In contrast, approaches targeting *developers* result to be more interested than those targeting app store vendors in analyses related to *testing* (21 against 2 study), *resources* (6 against 0), *refactoring* (16 against 2), *performance* (16 against 0), and *energy* (25 against 0). Again, this indicates that approaches targeting developers are more interested in the quality of the applications than those targeting app store vendors; the latter are mainly focused on ensuring the security of the *end user* by identifying potential *malware* and *privacy* leaks.

Usage of machine learning - Analysis goal. Usage of machine learning techniques is not evenly distributed among all goals. In particular, machine learning techniques are mostly employed for the goal of malware detection: out of 48 studies leveraging machine learning techniques in their analyses, 32 fall into the *malware* goal, the remainder is split among *privacy* (11), *inter-component communication* (4) and *inter-app communication* (2), *energy* (1) and *obfuscator identification* (1) (remember that goals are not mutually exclusive). This trend is traceable to the common techniques utilized to identify malware applications, which mostly often rely on training a classifier on a collected dataset of both benign and malicious applications. It is worth noting that the same machine learning techniques can potentially be applied when targeting other goals, such as *performance* or *energy consumption*; surprisingly, only one of the studies that fall into those goals make use of machine learning. We believe that this is due to the greater effort required for the collection of large datasets when considering these goals.

Industry involvement - Analysis goal. As expected, all analysis goals are considered by academic researchers. *energy* (25/25), *inter-component communication* (26/33), *malware* (59/66), and *privacy* (81/96) are the most targeted goals for academic researchers. In some cases, when the analysis goal concerns *privacy* (15/96), *malware* (6/66), *inter-component communication* (6/33), *inter-app communication* (2/24), *framework* (2/8), *testing* (1/23), *resource* (1/6), and *refactoring* (1/18), academic researchers are supported by industrial professionals.

By analyzing these results, we can conjecture that, although industrial organizations are interested in addressing the issues related to these goals, there is still a lack of industrial involvement when targeting other research goals, such as *energy* and *performance*, that would improve the overall user experience of mobile apps. We argue that researchers should more actively try to involve industry practitioners when working on such goals.

Target stakeholder - Analysis technique. Approaches to be utilized by app stores vendors have a more prominent usage of techniques such as *data mining* (33/46), *taint analysis* (18/33), and *classification* (21/29). This is in line with the most prominent goal of such stakeholder, i.e., identifying malicious applications in order to remove them from their stores. On the contrary, approaches to be utilized by *developers*, which are more interested in the inner workings of the applications, result to be characterized by a higher usage of techniques based on *flow analysis* (108/171). An explanation for this trend is the difference in performances among different static analysis techniques: approaches targeted at *app stores* must be highly scalable, as they have to be executed daily on thousands of apps; approaches targeted at *developers* have less stringent requirements. This evidences that improving the performances of some techniques is a relevant open problem, as they are currently a limiting factor for the kind of analyses that can be performed on *app stores*.

Tool availability - Analysis goal. When dealing with static analysis, automation is a crucial requirement for an approach to be effectively adopted in practice. Although for the majority of the identified analysis goals many different approaches have been proposed, most of them do not have a (released) tool ready for adoption by practitioners. On the one hand, we can argue that addressing goals such as *privacy* and *malware*, may require the realization of a mature supporting tool requiring a development effort that cannot be always afforded. On the other hand, addressing some goals represent more a theoretical interest, with potentially marginal practical impact, such as the study of an analysis *framework* itself. Nonetheless, we encourage researchers to undergo the extra effort required for making their analysis tool available to the research community: not only it makes easier to replicate their results but also analysis types for which a mature tool has been made available have been far more explored by the scientific community (as in the case of *Flowdroid* [12] for *flow analysis*).

9 Discussion and Future Research Challenges

The results presented in the previous sections give a data-driven, objective overview of the current state of the art on static analysis for mobile apps. In this section, we provide our own interpretation of the main points we deem as important challenges for future researchers in this area.

Is there life after Android? When considering the targeted platforms, it is evident that Android is the clear winner, with more than 90% of approaches targeting it. If on the one hand, we could have expected

this result (as of today, Android is the most popular mobile operating system with more than 90% market share [2] and a relatively large number of open-source tools for apps analysis), on the other hand, it makes us wonder what will be the fate of this Android-specific large body of knowledge and tools we researchers are producing in the future. If we look back in time, it is widely recognized that the mobile ecosystem is extremely dynamic, with platforms unpredictably raising and failing in terms of sells of devices, companies acquisitions, users flowing to/from other platforms. For example, 10 years ago, Apple iOS and Symbian were having 38% and 16% of the market share, whereas today they account for less than 14% together[15].

It is encouraging to see that 2 approaches out of 261 are generic (even though the implementation of the majority of them is again Android-specific). We believe that in the future researchers should reason at a higher level of abstraction, and focus more on approaches which are technology-independent, generic, and applicable to different platforms with reasonable effort. It is only in this way that our research results will pass the test of time and will (hopefully) remain relevant also in the future, despite the inevitable technological waves we will be facing. It is important to note that we are not suggesting to totally neglect platform-specific aspects, rather we are proposing to design our own research products to be platform-independent and robust with respect to (future) technologies; among many, researchers might take advantage of the well-known principles of extensibility and separation of concerns, of layered or plugin-based architectures for making their research products applicable in the context of new technologies without disrupting their general principles and base mechanisms. This will also speed up research by helping researchers in avoiding to reinvent the wheel whenever a (potentially applicable) research product will be applied to a new mobile platform.

Analysis goals shall be expanded substantially. The results of our study tell that privacy and malware are the most targeted analysis goals, far more than the others (e.g., performance, energy, resources usage). This is a clear gap that we, as researchers in the area of mobile apps analysis, should be filling in the future.

Given its strong importance for mobile apps, it seems that *performance* is extremely under-explored. Indeed, performance is a fundamental aspect of mobile apps development and is one of the top concerns for both developers and users; indeed, frequent complaints in app stores are about apps' performance, impacting the ratings of the apps and potentially undermining their chances of success [59, 163]. Moreover, *anti-patterns identification and refactoring* are among the least explored analysis goals so far, despite the fact that bug fixing and code re-organization are among the most recurrent activities of mobile apps developers [198]. In this context, P52 can be considered as a reference study about how to propose, design, and evaluate a refactoring method for mobile apps. Specifically, P52 presents a preliminary large-scale formative study about how developers approach asynchronous programming in Android apps. Then, based on the obtained results (e.g., that developers are using the Android AsyncTask construct also for long running operations, potentially leading to memory leaks, lost results, and wasted energy), a tool-based method is proposed for (i) statically identifying usages of the AsyncTask construct which can be automatically improved, and (ii) refactoring those parts of the app via a safe code rewriting algorithm. Finally, an empirical evaluation provides objective and reproducible evidence about the applicability and saved effort of the proposed method.

Users are being left out of the equation. From the results of RQ3, it emerged that only 20 studies consider end users as stakeholders, revealing that researchers are mostly focusing on techniques aimed at assisting developers, store moderators and researchers instead. Although this unbalance is not unexpected, when also considering that the majority of studies focused on privacy as their goal, we can notice a lack of users-first privacy approaches. Indeed, privacy is a subjective property, as different users may have different concerns when judging the trustability of an application. Current solutions fail to address this subjective aspect of privacy, considering all users as equals. In light of these considerations, we can identify one research area currently open and overlooked: the design of more user-centric approaches to privacy, where users are provided with the necessary tools to specify and validate the "personal" requirements to which an application must comply [219, 220].

Developers are being left out of the equation too! Even though when answering RQ3 it emerged that practitioners were involved in 30 studies, it also emerged that almost all approaches have not been evaluated or adopted in an industrial environment. We consider this finding as an indication that practitioners are involved in the technical phases of the study (e.g., elicitation of the requirements for the approaches, analysis steps definition, experiments results evaluation), but not as subjects of the evaluation of the proposed approaches. This situation is in strong contrast with the fact that the most recurrent stakeholders of the proposed approaches are the practitioners

[15] https://www.statista.com/statistics/263453/global-market-share-held-by-smartphone-operating-systems/

themselves. For the future, we strongly advise to close the loop by including practitioners in all the phases of the studies, specially while (i) defining the assumptions, requirements, and usage scenarios of the proposed static analysis approaches, as well as (ii) evaluating the proposed approaches in terms of their usefulness, applicability, and usability. At best, the latter can be performed by applying the case study methodology [258]. This is already happening in other research areas within the software engineering domain, such as software energy efficiency [240], technical debt [181] and software testing [215].

Tools and datasets shall be released and publicly available. An underlying problem which hinders the effective uptake of static analysis of mobile apps research lies in tool availability. In fact, from the results of our research, we evince that only a small portion of tools utilized or developed in the primary studies are available online. This constitutes a serious problem for researchers interested in extending or adapting tools which have been already developed. Additionally, the data utilized in the primary studies (e.g., accurate versioning history of apps used for experimentation) is only seldom available. This potentially slows down investigations, as datasets still have to be created on an *ad-hoc* basis for researches, as the number of already available ones is scarce. In recent times, this trend has been opposed by the constitution of some conference tracks explicitly aimed to make datasets publicly available. Among the most prominent ones are the "Artifact" track of the International Conference on Software Maintenance and Evolution (ICSME), and the "Data Showcase" track of the Mining Software Repositories (MSR) conference. Researches belonging to this tracks range from general purpose datasets, e.g., large versioning datasets focusing on Android applications [89], to context-specific datasets, e.g., to support dynamic analyses of Android applications [41]. Finally, from the findings of our study, we detect a shortcoming shared by many studies of static analysis of mobile apps, namely the impossibility to replicate the reported results. In fact, the absence of structured replication packages, in form of tools and dataset utilized, precludes the possibility to replicate the results reported in the primary studies. This constitutes a major problem affecting not only researchers interested in the field of mobile static analysis, but also the soundness of the studies itself.

10 Threats to Validity

In order to ensure the high quality of the data gathered for this study, a well-defined research protocol was established before carrying out the data collection. The research activities were designed by following a set of well-accepted and revised guidelines for systematic mapping studies [134]. From the formalization of such guidelines, we established the research protocol that was strictly followed all throughout the evolution of the study, as documented in Section 4. In addition, in order to further ensure the adherence to the established protocol and the envisioned quality standards, all the steps of the research (e.g., study design, search and selection, data extraction, data analysis) were carried out in team. This activity was deemed necessary also to lower potential sources of bias by discussing crucial considerations in team. Even by adopting a methodic literature review approach, threats to validity are still unavoidable. The remaining of this section reports on the main threats to validity to our study and how we mitigated them.

External validity refers to conditions that hinder the ability to generalize the results of our research [258]. The major threat of this category is represented by the fact that our primary studies are not representative of the state of the art research on static analysis of mobile applications. In order to mitigate this threat, we adopted a search strategy consisting of a manual search encompassing all the top-level software engineering conferences[16] and international journals[17] according to well known sources in the field. Such process was further extended by executing a backward and forward snowballing process on the selected literature. In order to ensure the quality of the selected researches, we exclusively considered peer-reviewed papers and excluded the so-called grey literature, such as white papers, editorials, etc. We disregard such decision as a significant source of bias, as peer-review processes are a standard requirement for high-quality publications. Finally, we adopted a set of well-defined inclusion and exclusion criteria, which rigorously guided our selection of the literature.

Internal Validity refers to the influences that can affect the design of the study, without the researcher's knowledge [258]. In this regard, we defined *a priori* a rigorous research protocol for the study. The classification framework adopted was established iteratively by strictly following the keywording process and it has been piloted by three researchers in an independent manner. Regarding the synthesis of the collected data, such process was carried out by adopting simple and well-assessed descriptive statistics. Subsequently, during the orthogonal analysis, we performed sanity tests on the extracted data by cross-analyzing different parameters of the established classification framework.

[16] `http://goo.gl/auU7su`

[17] `http://www.webofknowledge.com`

Construct validity refers to the extent to which the primary studies selected are suited to answer our research questions [258]. In order to mitigate such threat, we manually inspected thoroughly the literature published in the top-level software engineering conferences and journals. This procedure was performed by adhering to a rigorous predefined protocol. In addition, the results of such process were expanded by integrating the results gathered through a backward and forward snowballing process. Subsequently, we methodologically selected the identified studies by applying a set of well-documented inclusion and exclusion criteria. This latter process was carried out by three researchers independently. As recommended by Wholin et al. [258], a random sample of eight studies were selected and analyzed by all three researchers in order to ensure that the analyses were aligned.

Conclusion validity refers to issues that might hinder the ability to draw the correct conclusion from the data gathered [258]. In order to minimize the presence such threat, we carefully carried out the data extraction and analysis by strictly adhering to an *a priori* defined protocol. Such protocol was specifically conceived to collect the data necessary to answer our research questions. This enabled us to reduce potential sources of bias resulting from the data extraction and analyses processes. In addition, such methodology guaranteed us that the extracted data was fitted to answer our research questions. In order to further mitigate potential threats to conclusion validity, we adhered to the best practices reported in several well-known guidelines for systematic literature reviews [133, 204, 258]. Such guidelines were strictly followed throughout each phase of our research, and were comprehensively documented in order to make our research approach transparent and replicable.

11 Conclusions

The systematic mapping study reported in this paper permitted us to precisely characterize the most relevant methods and techniques for statically analyzing mobile apps. Starting from over 12,000 potentially relevant studies, we applied a rigorous selection procedure resulting in 261 primary studies along 122 scientific venues and a time span of 9 years.

We rigorously defined a classification framework with the target of identifying, evaluating and classifying the characteristics of existing approaches to the static analysis of mobile apps, while understanding trends and potentials of industrial adoption.

The main findings of this study have been synthesized by performing (i) a combination of content analysis and narrative synthesis (vertical analysis), and (ii) a correspondence analysis via contingency tables (horizontal analysis).

Our study will help researchers and practitioners in identifying the purposes and the limitations of existing research on static analysis of mobile apps. Also, we assessed the potential of research on static analysis of mobile apps, discussing how to foster industrial adoption and technological transfer. The knowledge of the potential of existing methods and techniques constitutes a reference framework in support of researchers and practitioners, such as app developers, who are interested in selecting/choosing existing static analysis approaches, and want to critically understand what they offer and how. In this sense, we can argue that this work constitutes a valuable asset to the academic and industrial world in the wide spectrum of static analysis.

12 Declarations

Availability of data and materials

The datasets analysed during the current study are available in the github repository, `https://github.com/sesygroup/mobile-static-analysis-replication-package`.

Competing interests

The authors declare that they have no competing interests.

Funding

This research was funded by the authors' institutional affiliations.

Author's contributions

The authors equally contributed to the elaboration of this survey. All authors read and approved the final manuscript. Authors are listed in alphabetical order.

Acknowledgements

This research was funded by the authors' institutional affiliations.

13 List of abbreviations

API: Application Programming Interface
SMS: Systematic Mapping Study
SLR: Systematic Literature Review
SE: Software Engineering
RQ: Research Question
APK: Android PacKage
IPA: iPhone Application Archive **IDE**: integrated Development Environment
AWiDe: Android Wicked Delegation
ICSE: International Conference on Software Engineering
ASE: Automated Software Engineering
ICSME: International Conference on Software Maintenance and Evolution
MSR: Mining Software Repositories

Author details

[1]University of L'Aquila, L'Aquila, Italy. [2]Vrije Universiteit Amsterdam, Amsterdam, Netherlands.

References

1. Aafer Y, Du W, Yin H (2013) Droidapiminer: Mining api-level features for robust malware detection in android. In: International conference on security and privacy in communication systems, Springer, pp 86–103
2. Adam Lella, Andrew Lipsman (2017) The U.S. Mobile App Report. ComsCore white paper
3. Agrawal H, Alberi JL, Horgan JR, Li JJ, London S, Wong WE, Ghosh S, Wilde N (1998) Mining system tests to aid software maintenance. Computer 31(7):64–73

4. Ahmad M, Costamagna V, Crispo B, Bergadano F (2017) Teicc: targeted execution of inter-component communications in android. In: Proceedings of the symposium on applied computing, pp 1747–1752
5. Al Rahat T, Feng Y, Tian Y (2019) Oauthlint: an empirical study on oauth bugs in android applications. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, pp 293–304
6. Alam S, Qu Z, Riley R, Chen Y, Rastogi V (2017) Droidnative: Automating and optimizing detection of android native code malware variants. computers & security 65:230–246
7. Allen J, Landen M, Chaba S, Ji Y, Chung SPH, Lee W (2018) Improving accuracy of android malware detection with lightweight contextual awareness. In: Proceedings of the 34th Annual Computer Security Applications Conference, pp 210–221
8. Allix K, Bissyandé TF, Jérome Q, Klein J, State R, Le Traon Y (2016) Empirical assessment of machine learning-based malware detectors for android. Empirical Softw Engg pp 183–211
9. Annie A (2017) App annie's global app economy forecast, last accessed: 27/09/2017. URL `http://go.appannie.com/report-app-economy-forecast-part-two`
10. Arp D, Spreitzenbarth M, Hubner M, Gascon H, Rieck K, Siemens C (2014) Drebin: Effective and explainable detection of android malware in your pocket. In: Ndss, vol 14, pp 23–26
11. Arzt S, Bodden E (2016) Stubdroid: automatic inference of precise data-flow summaries for the android framework. In: Proceedings of the 38th International Conference on Software Engineering, ACM, pp 725–735
12. Arzt S, Rasthofer S, Fritz C, Bodden E, Bartel A, Klein J, Le Traon Y, Octeau D, McDaniel P (2014) Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. Acm Sigplan Notices 49(6):259–269
13. Autili M, Benedetto PD, Inverardi P (2013) A hybrid approach for resource-based comparison of adaptable java applications. Science of Computer Programming 78(8):987–1009,
14. Autili M, Malavolta I, Perucci A, Scoccia GL (2015) Perspectives on static analysis of mobile apps (invited talk). In: Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile, ACM, pp 29–30
15. Avdiienko V, Kuznetsov K, Gorla A, Zeller A, Arzt S, Rasthofer S, Bodden E (2015) Mining apps for abnormal usage of sensitive data. In: Proceedings of the 37th International Conference on Software Engineering-Volume 1, IEEE Press, pp 426–436
16. Azim T, Neamtiu I (2013) Targeted and depth-first exploration for systematic testing of android apps. In: Acm Sigplan Notices, ACM, vol 48, pp 641–660
17. Backes M, Bugiel S, Derr E (2016) Reliable third-party library detection in android and its security applications. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp 356–367
18. Bae S, Lee S, Ryu S (2019) Towards understanding and reasoning about android interoperations. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, pp 223–233
19. Bagheri H, Kang E, Malek S, Jackson D (2015) Detection of design flaws in the android permission protocol through bounded verification. In: International Symposium on Formal Methods, Springer, pp 73–89
20. Bagheri H, Sadeghi A, Garcia J, Malek S (2015) Covert: Compositional analysis of android inter-app permission leakage. IEEE transactions on Software Engineering 41(9):866–886
21. Bagheri H, Sadeghi A, Jabbarvand R, Malek S (2016) Practical, formal synthesis and automatic enforcement of security policies for android. In: Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on, IEEE, pp 514–525
22. Bagheri H, Wang J, Aerts J, Malek S (2018) Efficient, evolutionary security analysis of interacting android apps. In: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, pp 357–368
23. Bai G, Ye Q, Wu Y, Botha H, Sun J, Liu Y, Dong JS, Visser W (2017) Towards model checking android applications. IEEE Transactions on Software Engineering 44(6):595–612
24. Banerjee A, Chong LK, Chattopadhyay S, Roychoudhury A (2014) Detecting energy bugs and hotspots in mobile apps. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, pp 588–598
25. Banerjee A, Guo HF, Roychoudhury A (2016) Debugging energy-efficiency related field failures in mobile apps. In: Proceedings of the International Conference on Mobile Software Engineering and Systems, ACM, pp 127–138
26. Banerjee A, Chong LK, Ballabriga C, Roychoudhury A (2018) Energypatch: Repairing resource leaks to improve energy-efficiency of android apps. IEEE Transactions on Software Engineering 44(5):470–490
27. Barros P, Just R, Millstein S, Vines P, Dietl W, Ernst MD, et al (2015) Static analysis of implicit control flow: Resolving java reflection and android intents (t). In: Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, IEEE, pp 669–679
28. Bartel A, Klein J, Le Traon Y, Monperrus M (2012) Automatically securing permission-based software by reducing the attack surface: An application to android. In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, ACM, pp 274–277
29. Bartel A, Klein J, Monperrus M, Le Traon Y (2014) Static analysis for extracting permission checks of a large scale framework: The challenges and solutions for analyzing android. IEEE Transactions on Software Engineering 40(6):617–632
30. Basili VR, Caldiera G, Rombach HD (1994) The Goal Question Metric Approach. In: Encyclopedia of Software Engineering, vol 2, Wiley, pp 528–532
31. Bastani O, Anand S, Aiken A (2015) Interactively verifying absence of explicit information flows in android apps. In: ACM SIGPLAN Notices, ACM, vol 50, pp 299–315
32. Batyuk L, Herpich M, Camtepe SA, Raddatz K, Schmidt AD, Albayrak S (2011) Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within android applications. In: Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on, IEEE, pp 66–72
33. Baumgärtner L, Graubner P, Schmidt N, Freisleben B (2015) Andro lyze: A distributed framework for efficient android app analysis. In: Mobile Services (MS), 2015 IEEE International Conference on, IEEE, pp 73–80
34. Behrouz RJ, Sadeghi A, Garcia J, Malek S, Ammann P (2015) Ecodroid: An approach for energy-based ranking of android apps. In: Green and Sustainable Software (GREENS), 2015 IEEE/ACM 4th International Workshop on, IEEE, pp 8–14
35. Ben Martin (2017) The Global Mobile Report - comScore's cross-market comparison of mobile trends and behaviours. ComsCore white paper
36. Bianchi A, Corbetta J, Invernizzi L, Fratantonio Y, Kruegel C, Vigna G (2015) What the app is that? deception and countermeasures in the android user interface. In: 2015 IEEE Symposium on Security and Privacy (SP), IEEE, pp 931–948
37. Bosu A, Liu F, Yao D, Wang G (2017) Collusive data leak and more: Large-scale threat analysis of inter-app communications. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, pp 71–85
38. Brucker AD, Herzberg M (2016) On the static analysis of hybrid mobile apps. In: International Symposium on Engineering Secure Software and Systems, Springer, pp 72–88
39. Brutschy L, Ferrara P, Müller P (2014) Static analysis for independent app developers. In: ACM SIGPLAN Notices, ACM, vol 49, pp 847–860
40. Brutschy L, Ferrara P, Tripp O, Pistoia M (2015) Shamdroid: gracefully degrading functionality in the presence of limited resource access. In: ACM SIGPLAN Notices, ACM, vol 50, pp 316–331
41. Cai H, Ryder BG (2017) Artifacts for dynamic analysis of android apps. In: 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017, p 659, , URL `https://doi.org/10.1109/ICSME.2017.36`
42. Calcagno C, Distefano D (2011) Infer: An automatic program verifier for memory safety of c programs. NASA Formal Methods pp 459–465

43. Calcagno C, Distefano D, O'Hearn P, Yang H (2009) Compositional shape analysis by means of bi-abduction. In: ACM SIGPLAN Notices, ACM, vol 44, pp 289–300
44. Calzavara S, Grishchenko I, Maffei M (2016) Horndroid: Practical and sound static analysis of android applications by smt solving. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), IEEE, pp 47–62
45. Cam NT, Phuoc NCH (2017) Nesedroid—android malware detection based on network traffic and sensitive resource accessing. In: Proceedings of the International Conference on Data Engineering and Communication Technology, Springer, pp 19–30
46. Canfora G, Martinelli F, Mercaldo F, Nardone V, Santone A, Visaggio CA (2018) Leila: formal tool for identifying mobile malicious behaviour. IEEE Transactions on Software Engineering 45(12):1230–1252
47. Cao Y, Fratantonio Y, Bianchi A, Egele M, Kruegel C, Vigna G, Chen Y (2015) Edgeminer: Automatically detecting implicit control flow transitions through the android framework. In: NDSS
48. Chan PP, Hui LC, Yiu SM (2012) Droidchecker: analyzing android applications for capability leak. In: Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks, ACM, pp 125–136
49. Chen K, Liu P, Zhang Y (2014) Achieving accuracy and scalability simultaneously in detecting application clones on android markets. In: Proceedings of the 36th International Conference on Software Engineering, pp 175–186
50. Chen KZ, Johnson NM, D'Silva V, Dai S, MacNamara K, Magrino TR, Wu EX, Rinard M, Song DX (2013) Contextual policy enforcement in android applications with permission event graphs. In: NDSS, vol 234
51. Chen X, Chen J, Liu B, Ma Y, Zhang Y, Zhong H (2019) Androidoff: Offloading android application based on cost estimation. Journal of Systems and Software 158:110,418
52. Chen X, Li C, Wang D, Wen S, Zhang J, Nepal S, Xiang Y, Ren K (2019) Android hiv: A study of repackaging malware for evading machine-learning detection. IEEE Transactions on Information Forensics and Security 15:987–1001
53. Chin E, Felt AP, Greenwood K, Wagner D (2011) Analyzing inter-application communication in android. In: Proceedings of the 9th international conference on Mobile systems, applications, and services, ACM, pp 239–252
54. Choi B, Kim J, Cho D, Kim S, Han D (2018) Appx: an automated app acceleration framework for low latency mobile app. In: Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies, pp 27–40
55. Clarivate (2019) Isi web of science, last accessed: 06/11/2019. URL `http://www.webofknowledge.com`
56. Crussell J, Gibler C, Chen H (2013) Andarwin: Scalable detection of semantically similar android applications. In: European Symposium on Research in Computer Security, Springer, pp 182–199
57. Cui X, Yu D, Chan P, Hui LC, Yiu SM, Qing S (2014) Cochecker: Detecting capability and sensitive data leaks from component chains in android. In: Australasian Conference on Information Security and Privacy, Springer, pp 446–453
58. Cui X, Wang J, Hui LC, Xie Z, Zeng T, Yiu SM (2015) Wechecker: efficient and precise detection of privilege escalation vulnerabilities in android apps. In: Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks, pp 1–12
59. Das T, Penta MD, Malavolta I (2016) A quantitative and qualitative investigation of performance-related commits in android apps. In: 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, Raleigh, NC, USA, October 2-7, 2016, pp 443–447, URL `http://www.ivanomalavolta.com/files/papers/ICSME_2016.pdf`
60. Del Vecchio J, Shen F, Yee KM, Wang B, Ko SY, Ziarek L (2015) String analysis of android applications (n). In: Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, IEEE, pp 680–685
61. Demissie BF, Ghio D, Ceccato M, Avancini A (2016) Identifying android inter app communication vulnerabilities using static and dynamic analysis. In: Proceedings of the International Conference on Mobile Software Engineering and Systems, ACM, pp 255–266
62. Demissie BF, Ceccato M, Shar LK (2018) Anflo: Detecting anomalous sensitive information flows in android apps. In: 2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft), IEEE, pp 24–34
63. Egele M, Kruegel C, Kirda E, Vigna G (2011) Pios: Detecting privacy leaks in ios applications. In: NDSS, pp 177–183
64. Egele M, Brumley D, Fratantonio Y, Kruegel C (2013) An empirical study of cryptographic misuse in android applications. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, ACM, pp 73–84
65. Elberzhager F, Münch J, Nha VTN (2012) A systematic mapping study on the combination of static and dynamic quality assurance techniques. Information and Software Technology 54(1):1–15
66. Elish KO, Yao D, Ryder BG (2012) User-centric dependence analysis for identifying malicious mobile apps. In: Workshop on Mobile Security Technologies
67. Ernst MD, Just R, Millstein S, Dietl W, Pernsteiner S, Roesner F, Koscher K, Barros PB, Bhoraskar R, Han S, et al (2014) Collaborative verification of information flow for a high-assurance app store. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM, pp 1092–1104
68. Eskandari M, Kessler B, Ahmad M, de Oliveira AS, Crispo B (2017) Analyzing remote server locations for personal data transfers in mobile apps. Proceedings on Privacy Enhancing Technologies 2017(1):118–131
69. Fahl S, Harbach M, Muders T, Baumgärtner L, Freisleben B, Smith M (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, ACM, pp 50–61
70. Fan L, Su T, Chen S, Meng G, Liu Y, Xu L, Pu G (2018) Efficiently manifesting asynchronous programming errors in android apps. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pp 486–497
71. Fan M, Liu J, Wang W, Li H, Tian Z, Liu T (2017) Dapasa: detecting android piggybacked apps through sensitive subgraph analysis. IEEE Transactions on Information Forensics and Security 12(8):1772–1785
72. Fan M, Liu J, Luo X, Chen K, Tian Z, Zheng Q, Liu T (2018) Android malware familial classification and representative sample selection via frequent subgraph analysis. IEEE Transactions on Information Forensics and Security 13(8):1890–1905
73. Fan M, Luo X, Liu J, Wang M, Nong C, Zheng Q, Liu T (2019) Graph embedding based familial analysis of android malware using unsupervised learning. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, pp 771–782
74. Fazzini M, Xin Q, Orso A (2019) Automated api-usage update for android apps. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp 204–215
75. Feizollah A, Anuar NB, Salleh R, Suarez-Tangil G, Furnell S (2017) Androdialysis: Analysis of android intent effectiveness in malware detection. computers & security 65:121–134
76. Felt AP, Chin E, Hanna S, Song D, Wagner D (2011) Android permissions demystified. In: Proceedings of the 18th ACM conference on Computer and communications security, ACM, pp 627–638
77. Feng R, Meng G, Xie X, Su T, Liu Y, Lin SW (2019) Learning performance optimization from code changes for android apps. In: 2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), IEEE, pp 285–290
78. Feng Y, Anand S, Dillig I, Aiken A (2014) Apposcopy: Semantics-based detection of android malware through static analysis. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, pp 576–587
79. Ferrari A, Gallucci D, Puccinelli D, Giordano S (2015) Detecting energy leaks in android app with poem. In: Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on, IEEE, pp 421–426
80. Franzago M, Di Ruscio D, Malavolta I, Muccini H (2017) Collaborative model-driven software engineering: a classification framework and a research map. IEEE Transactions on Software Engineering

81. Fratantonio Y, Machiry A, Bianchi A, Kruegel C, Vigna G (2015) Clapp: Characterizing loops in android applications. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 687–697
82. Fratantonio Y, Bianchi A, Robertson W, Kirda E, Kruegel C, Vigna G (2016) Triggerscope: Towards detecting logic bombs in android applications. In: Security and Privacy (SP), 2016 IEEE Symposium on, IEEE, pp 377–396
83. Gadient P, Ghafari M, Frischknecht P, Nierstrasz O (2019) Security code smells in android icc. Empirical software engineering 24(5):3046–3076
84. Gao X, Tan SH, Dong Z, Roychoudhury A (2018) Android testing via synthetic symbolic execution. In: 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, pp 419–429
85. Garcia J, Hammad M, Ghorbani N, Malek S (2017) Automatic generation of inter-component communication exploits for android applications. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp 661–671
86. Garcia J, Hammad M, Malek S (2018) Lightweight, obfuscation-resilient detection and family identification of android malware. ACM Transactions on Software Engineering and Methodology (TOSEM) 26(3):1–29
87. Garcıa-Ferreira I, Laorden C, Santos I, Bringas PG (2014) A survey on static analysis and model checking. In: International Joint Conference SOCO'14-CISIS'14-ICEUTE'14: Bilbao, Spain, June 25th-27th, 2014, Proceedings, Springer, vol 299, p 443
88. Gascon H, Yamaguchi F, Arp D, Rieck K (2013) Structural detection of android malware using embedded call graphs. In: Proceedings of the 2013 ACM workshop on Artificial intelligence and security, ACM, pp 45–54
89. Geiger FX, Malavolta I, Pascarella L, Palomba F, Nucci DD, Malavolta I, Bacchelli A (2018) A Graph-based Dataset of Commit History of Real-World Android apps. In: Proceedings of the 15th International Conference on Mining Software Repositories, MSR, ACM, New York, NY, p to appear, URL `http://www.ivanomalavolta.com/files/papers/MSR_2018.pdf`
90. Gibler C, Crussell J, Erickson J, Chen H (2012) Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale. In: International Conference on Trust and Trustworthy Computing, Springer, pp 291–307
91. Gonzalez H, Stakhanova N, Ghorbani AA (2018) Authorship attribution of android apps. In: Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, pp 277–286
92. Gordon MI, Kim D, Perkins JH, Gilham L, Nguyen N, Rinard MC (2015) Information flow analysis of android applications in droidsafe. In: NDSS, vol 15, p 110
93. Gorla A, Tavecchia I, Gross F, Zeller A (2014) Checking app behavior against app descriptions. In: Proceedings of the 36th International Conference on Software Engineering, ACM, pp 1025–1035
94. Gorla A, Tavecchia I, Gross F, Zeller A (2014) Checking app behavior against app descriptions. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, pp 1025–1035
95. Grace M, Zhou Y, Zhang Q, Zou S, Jiang X (2012) Riskranker: scalable and accurate zero-day android malware detection. In: Proceedings of the 10th international conference on Mobile systems, applications, and services, ACM, pp 281–294
96. Grace MC, Zhou Y, Wang Z, Jiang X (2012) Systematic detection of capability leaks in stock android smartphones. In: NDSS, vol 14, p 19
97. Gui J, Li D, Wan M, Halfond WG (2016) Lightweight measurement and estimation of mobile ad energy consumption. In: Green and Sustainable Software (GREENS), 2016 IEEE/ACM 5th International Workshop on, IEEE, pp 1–7
98. Guo C, Zhang J, Yan J, Zhang Z, Zhang Y (2013) Characterizing and detecting resource leaks in android applications. In: Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on, IEEE, pp 389–398
99. Guo C, Ye Q, Dong N, Bai G, Dong JS, Xu J (2016) Automatic construction of callback model for android application. In: Engineering of Complex Computer Systems (ICECCS), 2016 21st International Conference on, IEEE, pp 231–234
100. Hammad M, Garcia J, Malek S (2018) Self-protection of android systems from inter-component communication attacks. In: 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, pp 726–737
101. Hammad M, Bagheri H, Malek S (2019) Deldroid: an automated approach for determination and enforcement of least-privilege architecture in android. Journal of Systems and Software 149:83–100
102. Hanna S, Huang L, Wu E, Li S, Chen C, Song D (2012) Juxtapp: A scalable system for detecting code reuse among android applications. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, pp 62–81
103. Hao S, Li D, Halfond WG, Govindan R (2012) Estimating android applications' cpu energy usage via bytecode profiling. In: Proceedings of the First International Workshop on Green and Sustainable Software, IEEE Press, pp 1–7
104. Hao S, Li D, Halfond WG, Govindan R (2013) Estimating mobile application energy consumption using program analysis. In: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, pp 92–101
105. Harrison R, Flood D, Duce D (2013) Usability of mobile applications: literature review and rationale for a new usability model. Journal of Interaction Science 1(1):1–16
106. He D, Li L, Wang L, Zheng H, Li G, Xue J (2018) Understanding and detecting evolution-induced compatibility issues in android apps. In: 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, pp 167–177
107. He J, Chen T, Wang P, Wu Z, Yan J (2019) Android multitasking mechanism: Formal semantics and static analysis of apps. In: Asian Symposium on Programming Languages and Systems, Springer, pp 291–312
108. Hecht G, Benomar O, Rouvoy R, Moha N, Duchien L (2015) Tracking the software quality of android applications along their evolution (t). In: Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, IEEE, pp 236–247
109. Hoffmann J, Ussath M, Holz T, Spreitzenbarth M (2013) Slicing droids: program slicing for smali code. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing, ACM, pp 1844–1851
110. Holavanalli S, Manuel D, Nanjundaswamy V, Rosenberg B, Shen F, Ko SY, Ziarek L (2013) Flow permissions for android. In: Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering, IEEE Press, pp 652–657
111. Huang J, Zhang X, Tan L, Wang P, Liang B (2014) Asdroid: Detecting stealthy behaviors in android applications by user interface and program behavior contradiction. In: Proceedings of the 36th International Conference on Software Engineering, ACM, pp 1036–1046
112. Huang J, Li Z, Xiao X, Wu Z, Lu K, Zhang X, Jiang G (2015) Supor: Precise and scalable sensitive user input detection for android apps. In: USENIX Security Symposium, pp 977–992
113. Huang W, Dong Y, Milanova A, Dolby J (2015) Scalable and precise taint analysis for android. In: Proceedings of the 2015 International Symposium on Software Testing and Analysis, pp 106–117
114. Inc G (2017) Title of citation, last accessed: 27/09/2017
115. ISO/IEC (2010) Iso/iec 25010 system and software quality models. Tech. rep.
116. Jalali S, Wohlin C (2012) Systematic literature studies: Database searches vs. backward snowballing. In: Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, New York, NY, USA, ESEM '12, pp 29–38
117. Jang Jw, Kang H, Woo J, Mohaisen A, Kim HK (2015) Andro-autopsy: Anti-malware system based on similarity matching of malware and malware creator-centric information. Digital Investigation 14:17–35
118. Jeon J, Micinski KK, Vaughan JA, Fogel A, Reddy N, Foster JS, Millstein T (2012) Dr. android and mr. hide: fine-grained permissions in android applications. In: Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices, ACM, pp 3–14

119. Jia YJ, Chen QA, Lin Y, Kong C, Mao ZM (2017) Open doors for bob and mallory: Open port usage in android apps and security implications. In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P), IEEE, pp 190–203
120. Jiang YZX, Xuxian Z (2013) Detecting passive content leaks and pollution in android applications. In: Proceedings of the 20th Network and Distributed System Security Symposium (NDSS)
121. Jin X, Hu X, Ying K, Du W, Yin H, Peri GN (2014) Code injection attacks on html5-based mobile apps: Characterization, detection and mitigation. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM, pp 66–77
122. Joorabchi ME, Mesbah A, Kruchten P (2013) Real challenges in mobile app development. In: Empirical Software Engineering and Measurement, 2013, pp 15–24
123. Joorabchi ME, Mesbah A, Kruchten P (2013) Real challenges in mobile app development. In: Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on, IEEE, pp 15–24
124. Joorabchi ME, Mesbah A, Kruchten P (2013) Real challenges in mobile app development. In: 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, pp 15–24
125. Junaid M, Liu D, Kung D (2016) Dexteroid: Detecting malicious behaviors in android apps using reverse-engineered life cycle models. computers & security 59:92–117
126. Junaid M, Ming J, Kung D (2018) Statedroid: Stateful detection of stealthy attacks in android apps via horn-clause verification. In: Proceedings of the 34th Annual Computer Security Applications Conference, pp 198–209
127. Keng JCJ (2016) Automated testing and notification of mobile app privacy leak-cause behaviours. In: Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on, IEEE, pp 880–883
128. Keng JCJ, Jiang L, Wee TK, Balan RK (2016) Graph-aided directed testing of android applications for checking runtime privacy behaviours. In: Proceedings of the 11th International Workshop on Automation of Software Test, ACM, pp 57–63
129. Khalid H, Shihab E, Nagappan M, Hassan AE (2015) What do mobile app users complain about? IEEE Software 32(3):70–77
130. Kim CHP, Kroening D, Kwiatkowska M (2016) Static program analysis for identifying energy bugs in graphics-intensive mobile apps. In: Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2016 IEEE 24th International Symposium on, IEEE, pp 115–124
131. Kim J, Yoon Y, Yi K, Shin J, Center S (2012) Scandal: Static analyzer for detecting privacy leaks in android applications. MoST 12:110
132. Kim J, Choi H, Namkung H, Choi W, Choi B, Hong H, Kim Y, Lee J, Han D (2016) Enabling automatic protocol behavior analysis for android applications. In: Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies, ACM, pp 281–295
133. Kitchenham B, Brereton P (2013) A systematic review of systematic review process research in software engineering. Information and software technology 55(12):2049–2075
134. Kitchenham BA, Charters S (2007) Guidelines for performing systematic literature reviews in software engineering. Tech. Rep. EBSE-2007-01, Keele University and University of Durham
135. Kitchenham BA, Budgen D, Brereton OP (2010) The value of mapping studies-a participant-observer case study. In: EASE, vol 10, pp 25–33
136. Klieber W, Flynn L, Bhosale A, Jia L, Bauer L (2014) Android taint flow analysis for app sets. In: Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis, ACM, pp 1–6
137. Koch W, Chaabane A, Egele M, Robertson W, Kirda E (2017) Semi-automated discovery of server-based information oversharing vulnerabilities in android applications. In: Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp 147–157
138. Lai D, Rubin J (2019) Goal-driven exploration for android applications. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, pp 115–127
139. Lee S, Dolby J, Ryu S (2016) Hybridroid: static analysis framework for android hybrid applications. In: Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on, IEEE, pp 250–261
140. Lee YK, Bang JY, Safi G, Shahbazian A, Zhao Y, Medvidovic N (2017) A sealant for inter-app security holes in android. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), IEEE, pp 312–323
141. Lemos OAL, Bajracharya S, Ossher J, Masiero PC, Lopes C (2011) A test-driven approach to code search and its application to the reuse of auxiliary functionality. Information and Software Technology 53(4):294–306
142. Ley M (2002) The dblp computer science bibliography: Evolution, research issues, perspectives. In: International symposium on string processing and information retrieval, Springer, pp 1–10
143. Li D, Hao S, Halfond WG, Govindan R (2013) Calculating source line level energy information for android applications. In: Proceedings of the 2013 International Symposium on Software Testing and Analysis, ACM, pp 78–89
144. Li D, Lyu Y, Gui J, Halfond WG (2016) Automated energy optimization of http requests for mobile applications. In: Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on, IEEE, pp 249–260
145. Li L, Bartel A, Klein J, Le Traon Y (2014) Automatically exploiting potential component leaks in android applications. In: Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on, IEEE, pp 388–397
146. Li L, Allix K, Li D, Bartel A, Bissyandé TF, Klein J (2015) Potential component leaks in android apps: An investigation into a new feature set for malware detection. In: 2015 IEEE International Conference on Software Quality, Reliability and Security, IEEE, pp 195–200
147. Li L, Bartel A, Bissyandé TF, Klein J, Le Traon Y (2015) Apkcombiner: Combining multiple android apps to support inter-app analysis. In: IFIP International Information Security and Privacy Conference, Springer, pp 513–527
148. Li L, Bartel A, Bissyandé TF, Klein J, Le Traon Y, Arzt S, Rasthofer S, Bodden E, Octeau D, McDaniel P (2015) Iccta: Detecting inter-component privacy leaks in android apps. In: Proceedings of the 37th International Conference on Software Engineering-Volume 1, IEEE Press, pp 280–291
149. Li L, Bissyandé TF, Octeau D, Klein J (2016) Droidra: Taming reflection to support whole-program analysis of android apps. In: Proceedings of the 25th International Symposium on Software Testing and Analysis, ACM, pp 318–329
150. Li L, Bissyandé TF, Octeau D, Klein J (2016) Reflection-aware static analysis of android apps. In: Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on, IEEE, pp 756–761
151. Li L, Li D, Bartel A, Bissyandé TF, Klein J, Traon YL (2016) Towards a generic framework for automating extensive analysis of android applications. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing, ACM, pp 1460–1465
152. Li L, Bissyandé TF, Papadakis M, Rasthofer S, Bartel A, Octeau D, Klein J, Le Traon Y (2017) Static analysis of android apps: A systematic literature review. Information and Software Technology
153. Li L, Bissyandé TF, Wang H, Klein J (2018) Cid: Automating the detection of api-related compatibility issues in android apps. In: Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp 153–163
154. Li L, Riom T, Bissyandé TF, Wang H, Klein J, et al (2019) Revisiting the impact of common libraries for android-related investigations. Journal of Systems and Software 154:157–175
155. Li M, Wang W, Wang P, Wang S, Wu D, Liu J, Xue R, Huo W (2017) Libd: Scalable and precise third-party library detection in android markets. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), IEEE, pp 335–346
156. Li W, Jiang Y, Xu C, Liu Y, Ma X, Lü J (2019) Characterizing and detecting inefficient image displaying issues in android apps. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution

and Reengineering (SANER), IEEE, pp 355–365
157. Li Y, Guo Y, Chen X (2016) Peruim: Understanding mobile application privacy with permission-ui mapping. In: Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing, ACM, pp 682–693
158. Li Z, Sun J, Yan Q, Srisa-an W, Bachala S (2018) Grandroid: Graph-based detection of malicious network behaviors in android applications. In: International Conference on Security and Privacy in Communication Systems, Springer, pp 264–280
159. Li Z, Sun J, Yan Q, Srisa-an W, Tsutano Y (2019) Obfusifier: Obfuscation-resistant android malware detection system. In: International Conference on Security and Privacy in Communication Systems, Springer, pp 214–234
160. Liang S, Keep AW, Might M, Lyde S, Gilray T, Aldous P, Van Horn D (2013) Sound and precise malware analysis for android via pushdown reachability and entry-point saturation. In: Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices, ACM, pp 21–32
161. Lin Y, Radoi C, Dig D (2014) Retrofitting concurrency for android applications through refactoring. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, pp 341–352
162. Lin Y, Okur S, Dig D (2015) Study and refactoring of android asynchronous programming (t). In: Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, IEEE, pp 224–235
163. Linares-Vasquez M, Vendome C, Luo Q, Poshyvanyk D (2015) How developers detect and fix performance bottlenecks in android apps. In: 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, pp 352–361
164. Liu A, Guo C, Wang W, Qiu Y, Xu J (2019) Static back-stack transition analysis for android. IEEE Access 7:110,781–110,793
165. Liu J, Wu T, Yan J, Zhang J (2016) Fixing resource leaks in android apps with light-weight static analysis and low-overhead instrumentation. In: 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), IEEE, pp 342–352
166. Liu J, Wu D, Xue J (2018) Tdroid: Exposing app switching attacks in android with control flow specialization. In: 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, pp 236–247
167. Liu Y, Xu C, Cheung SC (2014) Characterizing and detecting performance bugs for smartphone applications. In: Proceedings of the 36th International Conference on Software Engineering, ACM, pp 1013–1024
168. Liu Y, Xu C, Cheung SC, Lu J (2014) Greendroid: Automated diagnosis of energy inefficiency for smartphone applications. IEEE Transactions on Software Engineering (1):1–1
169. Liu Y, Xu C, Cheung SC, Terragni V (2016) Understanding and detecting wake lock misuses for android applications. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, pp 396–409
170. Lortz S, Mantel H, Starostin A, Bähr T, Schneider D, Weber A (2014) Cassandra: Towards a certifying app store for android. In: Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices, ACM, pp 93–104
171. Lu L, Li Z, Wu Z, Lee W, Jiang G (2012) Chex: statically vetting android apps for component hijacking vulnerabilities. In: Proceedings of the 2012 ACM conference on Computer and communications security, ACM, pp 229–240
172. Lyu Y, Li D, Halfond WG (2018) Remove rats from your code: automated optimization of resource inefficient database writes for mobile applications. In: Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp 310–321
173. Ma J, Liu S, Jiang Y, Tao X, Xu C, Lu J (2018) Lesdroid: a tool for detecting exported service leaks of android applications. In: Proceedings of the 26th Conference on Program Comprehension, pp 244–254
174. Ma S, Bertino E, Nepal S, Li J, Ostry D, Deng RH, Jha S (2019) Finding flaws from password authentication code in android apps. In: European Symposium on Research in Computer Security, Springer, pp 619–637
175. Ma Z, Wang H, Guo Y, Chen X (2016) Libradar: Fast and accurate detection of third-party libraries in android apps. In: Proceedings of the 38th international conference on software engineering companion, pp 653–656
176. Mahmood R, Mirzaei N, Malek S (2014) Evodroid: Segmented evolutionary testing of android apps. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp 599–609
177. Mann C, Starostin A (2012) A framework for static detection of privacy leaks in android applications. In: Proceedings of the 27th annual ACM symposium on applied computing, ACM, pp 1457–1462
178. Maqsood HMA, Qureshi KN, Bashir F, Islam NU (2019) Privacy leakage through exploitation of vulnerable inter-app communication on android. In: 2019 13th International Conference on Open Source Systems and Technologies (ICOSST), IEEE, pp 1–6
179. Mariconti E, Onwuzurike L, Andriotis P, De Cristofaro E, Ross G, Stringhini G (2017) Mamadroid: Detecting android malware by building markov chains of behavioral models. In: 24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017
180. Martin W, Sarro F, Jia Y, Zhang Y, Harman M (2017) A survey of app store analysis for software engineering. IEEE transactions on software engineering 43(9):817–847
181. Martini A, Bosch J (2015) The danger of architectural technical debt: Contagious debt and vicious circles. In: Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on, IEEE, pp 1–10
182. Mirzaei N, Bagheri H, Mahmood R, Malek S (2015) Sig-droid: Automated system input generation for android applications. In: 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), IEEE, pp 461–471
183. Mirzaei N, Garcia J, Bagheri H, Sadeghi A, Malek S (2016) Reducing combinatorics in gui testing of android applications. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), IEEE, pp 559–570
184. Mishra A, Kanade A, Srikant Y (2016) Asynchrony-aware static analysis of android applications. In: 2016 ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE), IEEE, pp 163–172
185. Morales R, Saborido R, Khomh F, Chicano F, Antoniol G (2017) Earmo: an energy-aware refactoring approach for mobile apps. IEEE Transactions on Software Engineering 44(12):1176–1206
186. Moran K, Linares-Vásquez M, Bernal-Cárdenas C, Vendome C, Poshyvanyk D (2016) Automatically discovering, reporting and reproducing android application crashes. In: 2016 IEEE international conference on software testing, verification and validation (icst), IEEE, pp 33–44
187. Nan Y, Yang M, Yang Z, Zhou S, Gu G, Wang X (2015) Uipicker: User-input privacy identification in mobile applications. In: USENIX Security Symposium, pp 993–1008
188. Narayanan A, Chandramohan M, Chen L, Liu Y (2018) A multi-view context-aware approach to android malware detection and malicious code localization. Empirical Software Engineering 23(3):1222–1274
189. Narayanan A, Soh C, Chen L, Liu Y, Wang L (2018) apk2vec: Semi-supervised multi-view representation learning for profiling android applications. In: 2018 IEEE International Conference on Data Mining (ICDM), IEEE, pp 357–366
190. Nielson F, Nielson HR, Hankin C (2015) Principles of program analysis. Springer
191. Nirumand A, Zamani B, Tork Ladani B (2019) Vandroid: A framework for vulnerability analysis of android applications using a model-driven reverse engineering technique. Software: Practice and Experience 49(1):70–99
192. Nolan G (2012) Decompiling android. Apress
193. Octeau D, McDaniel P, Jha S, Bartel A, Bodden E, Klein J, Le Traon Y (2013) Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis. Effective Inter-Component Communication Mapping in Android with Epicc: An Essential Step Towards Holistic Security Analysis

194. Octeau D, Luchaup D, Dering M, Jha S, McDaniel P (2015) Composite constant propagation: Application to android inter-component communication analysis. In: Proceedings of the 37th International Conference on Software Engineering-Volume 1, IEEE Press, pp 77–88
195. Octeau D, Jha S, Dering M, McDaniel P, Bartel A, Li L, Klein J, Le Traon Y (2016) Combining static analysis with probabilistic models to enable market-scale android inter-component analysis. In: ACM SIGPLAN Notices, ACM, vol 51, pp 469–484
196. Ongkosit T, Takada S (2014) Responsiveness analysis tool for android application. In: Proceedings of the 2nd International Workshop on Software Development Lifecycle for Mobile, ACM, pp 1–4
197. Pan L, Cui B, Yan J, Ma X, Yan J, Zhang J (2019) Androlic: an extensible flow, context, object, field, and path-sensitive static analysis framework for android. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp 394–397
198. Pascarella L, Geiger FX, Palomba F, Nucci DD, Malavolta I, Bacchelli A (2018) Self-Reported Activities of Android Developers. In: 5th IEEE/ACM International Conference on Mobile Software Engineering and Systems, ACM, New York, NY, p to appear, URL `http://www.ivanomalavolta.com/files/papers/mobilesoft_2018_self.pdf`
199. Pathak A, Jindal A, Hu YC, Midkiff SP (2012) What is keeping my phone awake?: characterizing and detecting no-sleep energy bugs in smartphone apps. In: Proceedings of the 10th international conference on Mobile systems, applications, and services, ACM, pp 267–280
200. Pauck F, Wehrheim H (2019) Together strong: cooperative android app analysis. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp 374–384
201. Payet É, Spoto F (2012) Static analysis of android programs. Information and Software Technology 54(11):1192–1201
202. Peiravian N, Zhu X (2013) Machine learning for android malware detection using permission and api calls. In: 2013 IEEE 25th international conference on tools with artificial intelligence, IEEE, pp 300–305
203. Petersen K, Feldt R, Mujtaba S, Mattsson M (2008) Systematic mapping studies in software engineering. In: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, British Computer Society, Swinton, UK, UK, EASE'08, pp 68–77, URL `http://dl.acm.org/citation.cfm?id=2227115.2227123`
204. Petersen K, Vakkalanka S, Kuzniarz L (2015) Guidelines for conducting systematic mapping studies in software engineering: An update. Information and Software Technology 64:1–18
205. Pistoia M, Chandra S, Fink SJ, Yahav E (2007) A survey of static analysis methods for identifying security vulnerabilities in software systems. IBM Systems Journal 46(2):265–288
206. Radhakrishna A, Lewchenko NV, Meier S, Mover S, Sripada KC, Zufferey D, Chang BYE, Cernỳ P (2018) Droidstar: callback typestates for android classes. In: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), IEEE, pp 1160–1170
207. Rasthofer S, Arzt S, Triller S, Pradel M (2017) Making malory behave maliciously: Targeted fuzzing of android execution environments. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), IEEE, pp 300–311
208. Ravitch T, Creswick ER, Tomb A, Foltzer A, Elliott T, Casburn L (2014) Multi-app security analysis with fuse: Statically detecting android app collusion. In: Proceedings of the 4th Program Protection and Reverse Engineering Workshop, ACM, p 4
209. Research I (2006) T.j. watson libraries for analysis, last accessed: 06/11/2019. URL `http://wala.sourceforge.net/wiki/index.php/Main_Page`
210. Rosen S, Qian Z, Mao ZM (2013) Appprofiler: a flexible method of exposing privacy-related behavior in android applications to end users. In: Proceedings of the third ACM conference on Data and application security and privacy, ACM, pp 221–232
211. Rountev A, Yan D (2014) Static reference analysis for gui objects in android software. In: Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization, ACM, p 143
212. Rubin J, Gordon MI, Nguyen N, Rinard M (2015) Covert communication in mobile applications (t). In: Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, IEEE, pp 647–657
213. Sadeghi A, Jabbarvand R, Malek S (2017) Patdroid: permission-aware gui testing of android. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp 220–232
214. Sadeghi A, Jabbarvand R, Ghorbani N, Bagheri H, Malek S (2018) A temporal permission analysis and enforcement framework for android. In: Proceedings of the 40th International Conference on Software Engineering, pp 846–857
215. Sahaf Z, Garousi V, Pfahl D, Irving R, Amannejad Y (2014) When to automate software testing? decision support based on system dynamics: an industrial case study. In: Proceedings of the 2014 International Conference on Software and System Process, ACM, pp 149–158
216. Sattler F, von Rhein A, Berger T, Johansson NS, Hardø MM, Apel S (2018) Lifting inter-app data-flow analysis to large app sets. Automated Software Engineering 25(2):315–346
217. Scalabrino S, Bavota G, Linares-Vásquez M, Lanza M, Oliveto R (2019) Data-driven solutions to detect api compatibility issues in android: an empirical study. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), IEEE, pp 288–298
218. Scalabrino S, Bavota G, Linares-Vásquez M, Piantadosi V, Lanza M, Oliveto R (2020) Api compatibility issues in android: Causes and effectiveness of data-driven detection techniques. Empirical Software Engineering 25(6):5006–5046
219. Scoccia GL, Malavolta I, Autili M, Di Salle A, Inverardi P (2017) User-centric android flexible permissions. In: Software Engineering Companion (ICSE-C), 2017 IEEE/ACM 39th International Conference on, IEEE, pp 365–367
220. Scoccia GL, Malavolta I, Autili M, Di Salle A, Inverardi P (2019) Enhancing trustability of android applications via user-centric flexible permissions. IEEE Computer Architecture Letters (01):1–1
221. Shan Z, Azim T, Neamtiu I (2016) Finding resume and restart errors in android applications. In: ACM SIGPLAN Notices, ACM, vol 51, pp 864–880
222. Shan Z, Neamtiu I, Samuel R (2018) Self-hiding behavior in android apps: detection and characterization. In: Proceedings of the 40th International Conference on Software Engineering, pp 728–739
223. Shao Y, Luo X, Qian C, Zhu P, Zhang L (2014) Towards a scalable resource-driven approach for detecting repackaged android applications. In: Proceedings of the 30th Annual Computer Security Applications Conference, pp 56–65
224. Sharma A, Nasre R (2019) Qadroid: regression event selection for android applications. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp 66–77
225. Shen F, Vishnubhotla N, Todarka C, Arora M, Dhandapani B, Lehner EJ, Ko SY, Ziarek L (2014) Information flows as a permission mechanism. In: Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, ACM, pp 515–526
226. Sinha L, Bhandari S, Faruki P, Gaur MS, Laxmi V, Conti M (2016) Flowmine: Android app analysis via data flow. In: Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual, IEEE, pp 435–441
227. Slavin R, Wang X, Hosseini MB, Hester J, Krishnan R, Bhatia J, Breaux TD, Niu J (2016) Toward a framework for detecting privacy policy violations in android application code. In: Proceedings of the 38th International Conference on Software Engineering, pp 25–36
228. Software I (2013) Native, web or hybrid mobile-app development. thought leadership white paper.
229. Song H, Lin D, Zhu S, Wang W, Zhang S (2019) Ads-sa: System for automatically detecting sensitive path of android applications based on static analysis. In: International Conference on Smart City and Informatization, Springer, pp 309–322
230. Song W, Zhang J, Huang J (2019) Servdroid: detecting service usage inefficiencies in android applications. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp 362–373

231. Sounthiraraj D, Sahs J, Greenwood G, Lin Z, Khan L (2014) Smv-hunter: Large scale, automated detection of ssl/tls man-in-the-middle vulnerabilities in android apps. In: In Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS'14, Citeseer
232. Su T, Meng G, Chen Y, Wu K, Yang W, Yao Y, Pu G, Liu Y, Su Z (2017) Guided, stochastic model-based gui testing of android apps. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp 245–256
233. Suzuki N, Kamina T, Maruyama K (2016) Detecting invalid layer combinations using control-flow analysis for android. In: Proceedings of the 8th International Workshop on Context-Oriented Programming, ACM, pp 27–32
234. Tillmann N, Moskal M, de Halleux J, Fahndrich M (2011) Touchdevelop: programming cloud-connected mobile devices via touchscreen. In: Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software, ACM, pp 49–60
235. Titze D, Lux M, Schuette J (2017) Ordol: Obfuscation-resilient detection of libraries in android applications. In: 2017 IEEE Trustcom/BigDataSE/ICESS, IEEE, pp 618–625
236. Tiwari A, Groß S, Hammer C (2019) Iifa: modular inter-app intent information flow analysis of android applications. In: International Conference on Security and Privacy in Communication Systems, Springer, pp 335–349
237. Tiwari A, Prakash J, Groß S, Hammer C (2019) Ludroid: A large scale analysis of android–web hybridization. In: 2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM), IEEE, pp 256–267
238. Tsutano Y, Bachala S, Srisa-An W, Rothermel G, Dinh J (2017) An efficient, robust, and scalable approach for analyzing interacting android apps. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), IEEE, pp 324–334
239. Vallée-Rai R, Co P, Gagnon E, Hendren L, Lam P, Sundaresan V (1999) Soot-a java bytecode optimization framework. In: Proceedings of the 1999 conference of the Centre for Advanced Studies on Collaborative research, IBM Press, p 13
240. Verdecchia R, Procaccianti G, Malavolta I, Lago P, Koedijk J (2017) Estimating energy impact of software releases and deployment strategies: The KPMG case study. In: Empirical Software Engineering and Measurement (ESEM), 2017 ACM/IEEE International Symposium on, IEEE, pp 257–266
241. Vu PM, Nguyen TT, Pham HV, Nguyen TT (2015) Mining user opinions in mobile app reviews: A keyword-based approach (t). In: Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, IEEE, pp 749–759
242. Wang H, Guo Y, Ma Z, Chen X (2015) Wukong: A scalable and accurate two-phase approach to android app clone detection. In: Proceedings of the 2015 International Symposium on Software Testing and Analysis, ACM, New York, NY, USA, ISSTA 2015, pp 71–82, , URL `http://doi.acm.org/10.1145/2771783.2771795`
243. Wang H, Guo Y, Tang Z, Bai G, Chen X (2015) Reevaluating android permission gaps with static and dynamic analysis. In: Global Communications Conference (GLOBECOM), 2015 IEEE, IEEE, pp 1–6
244. Wang H, Hong J, Guo Y (2015) Using text mining to infer the purpose of permission use in mobile apps. In: Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, ACM, pp 1107–1118
245. Wang H, Li Y, Guo Y, Agarwal Y, Hong JI (2017) Understanding the purpose of permission use in mobile apps. ACM Transactions on Information Systems (TOIS) 35(4):1–40
246. Wang X, Qin X, Hosseini MB, Slavin R, Breaux TD, Niu J (2018) Guileak: Tracing privacy policy claims on user input data for android applications. In: Proceedings of the 40th International Conference on Software Engineering, pp 37–47
247. Wang Y, Rountev A (2016) Profiling the responsiveness of android applications via automated resource amplification. In: Proceedings of the International Conference on Mobile Software Engineering and Systems, ACM, pp 48–58
248. Wang Y, Rountev A (2017) Who changed you? obfuscator identification for android. In: 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft), IEEE, pp 154–164
249. Wang Y, Wu H, Zhang H, Rountev A (2018) Orlis: Obfuscation-resilient library detection for android. In: 2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft), IEEE, pp 13–23
250. Wasserman AI (2010) Software engineering issues for mobile application development. In: Proceedings of the FSE/SDP workshop on Future of software engineering research, ACM, pp 397–400
251. Watanabe T, Akiyama M, Sakai T, Mori T (2015) Understanding the inconsistencies between text descriptions and the use of privacy-sensitive resources of mobile apps. In: Eleventh Symposium On Usable Privacy and Security ({SOUPS} 2015), pp 241–255
252. Wei F, Roy S, Ou X, et al (2014) Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM, pp 1329–1341
253. Wei L, Liu Y, Cheung SC (2016) Taming android fragmentation: Characterizing and detecting compatibility issues for android apps. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ACM, pp 226–237
254. Wei L, Liu Y, Cheung SC (2017) Oasis: prioritizing static analysis warnings for android apps based on app user reviews. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp 672–682
255. Wei L, Liu Y, Cheung SC (2019) Pivot: learning api-device correlations to facilitate android compatibility issue detection. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, pp 878–888
256. Wohlin C (2014) Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, ACM, New York, NY, USA, EASE '14, pp 38:1–38:10
257. Wohlin C (2014) Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th international conference on evaluation and assessment in software engineering, ACM, p 38
258. Wohlin C, Runeson P, Höst M, Ohlsson M, Regnell B, Wesslén A (2012) Experimentation in Software Engineering. Computer Science, Springer
259. Wong MY, Lie D (2016) Intellidroid: A targeted input generator for the dynamic analysis of android malware. In: NDSS, vol 16, pp 21–24
260. Wong MY, Lie D (2018) Tackling runtime-based obfuscation in android with {TIRO}. In: 27th {USENIX} Security Symposium ({USENIX} Security 18), pp 1247–1262
261. Wu DJ, Mao CH, Wei TE, Lee HM, Wu KP (2012) Droidmat: Android malware detection through manifest and api calls tracing. In: Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on, IEEE, pp 62–69
262. Wu H, Yang S, Rountev A (2016) Static detection of energy defect patterns in android applications. In: Proceedings of the 25th International Conference on Compiler Construction, pp 185–195
263. Wu H, Zhang H, Wang Y, Rountev A (2019) Sentinel: generating gui tests for sensor leaks in android and android wear apps. Software Quality Journal pp 1–33
264. Wu J, Cui T, Ban T, Guo S, Cui L (2015) Paddyfrog: systematically detecting confused deputy vulnerability in android applications. Security and Communication Networks 8(13):2338–2349
265. Wu S, Wang P, Li X, Zhang Y (2016) Effective detection of android malware based on the usage of data flow apis and machine learning. Information and Software Technology 75:17–25
266. Wu T, Yang Y (2016) Capadroid: Detecting capability leak for android applications. In: International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage, Springer, pp 95–104
267. Wu T, Liu J, Deng X, Yan J, Zhang J (2016) Relda2: an effective static analysis tool for resource leak detection in android apps. In: Automated Software Engineering (ASE), 2016 31st IEEE/ACM

International Conference on, IEEE, pp 762–767
268. Wu Y, Li X, Zou D, Yang W, Zhang X, Jin H (2019) Malscan: Fast market-wide mobile malware scanning by social-network centrality analysis. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, pp 139–150
269. Xi S, Yang S, Xiao X, Yao Y, Xiong Y, Xu F, Wang H, Gao P, Liu Z, Xu F, et al (2019) Deepintent: Deep icon-behavior learning for detecting intention-behavior discrepancy in mobile apps. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp 2421–2436
270. Xiao X, Tillmann N, Fahndrich M, De Halleux J, Moskal M, Xie T (2015) User-aware privacy control via extended static-information-flow analysis. Automated Software Engineering 22(3):333–366
271. Xiao X, Wang X, Cao Z, Wang H, Gao P (2019) Iconintent: automatic identification of sensitive ui widgets based on icon classification for android apps. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, pp 257–268
272. Xie T (2002) Software engineering conferences (statistics), last accessed: 06/11/2019. URL `http://taoxie.cs.illinois.edu/seconferences.htm`
273. Xu Z, Fan D, Qin S (2016) State-taint analysis for detecting resource bugs. In: Theoretical Aspects of Software Engineering (TASE), 2016 10th International Symposium on, IEEE, pp 168–175
274. Xu Z, Ren K, Qin S, Craciun F (2018) Cdgdroid: Android malware detection based on deep learning using cfg and dfg. In: International Conference on Formal Engineering Methods, Springer, pp 177–193
275. Yan J, Deng X, Wang P, Wu T, Yan J, Zhang J (2018) Characterizing and identifying misexposed activities in android applications. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pp 691–701
276. Yang S, Yan D, Wu H, Wang Y, Rountev A (2015) Static control-flow analysis of user-driven callbacks in android applications. In: Proceedings of the 37th International Conference on Software Engineering-Volume 1, IEEE Press, pp 89–99
277. Yang S, Wu H, Zhang H, Wang Y, Swaminathan C, Yan D, Rountev A (2018) Static window transition graphs for android. Automated Software Engineering 25(4):833–873
278. Yang W, Prasad MR, Xie T (2013) A grey-box approach for automated gui-model generation of mobile applications. In: International Conference on Fundamental Approaches to Software Engineering, Springer, pp 250–265
279. Yang W, Xiao X, Andow B, Li S, Xie T, Enck W (2015) Appcontext: Differentiating malicious and benign mobile app behaviors using context. In: Proceedings of the 37th International Conference on Software Engineering-Volume 1, IEEE Press, pp 303–313
280. Yang W, Prasad MR, Xie T (2018) Enmobile: Entity-based characterization and analysis of mobile malware. In: Proceedings of the 40th International Conference on Software Engineering, pp 384–394
281. Yang X, Lo D, Li L, Xia X, Bissyandé TF, Klein J (2017) Characterizing malicious android apps by mining topic-specific data flow signatures. Information and Software Technology 90:27–39
282. Yang Z, Yang M (2012) Leakminer: Detect information leakage on android with static taint analysis. In: Software Engineering (WCSE), 2012 Third World Congress on, IEEE, pp 101–104
283. Yang Z, Yang M, Zhang Y, Gu G, Ning P, Wang XS (2013) Appintent: Analyzing sensitive data transmission in android for privacy leakage detection. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, ACM, pp 1043–1054
284. Yerima SY, Sezer S, McWilliams G, Muttik I (2013) A new android malware detection approach using bayesian classification. In: 2013 IEEE 27th international conference on advanced information networking and applications (AINA), IEEE, pp 121–128
285. Yu L, Zhang T, Luo X, Xue L (2015) Autoppg: Towards automatic generation of privacy policy for android applications. In: Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices, ACM, pp 39–50
286. Yu L, Luo X, Qian C, Wang S (2016) Revisiting the description-to-behavior fidelity in android applications. In: Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on, IEEE, vol 1, pp 415–426
287. Yu L, Luo X, Qian C, Wang S, Leung HK (2017) Enhancing the description-to-behavior fidelity in android apps with privacy policy. IEEE Transactions on Software Engineering 44(9):834–854
288. Yu L, Zhang T, Luo X, Xue L, Chang H (2017) Toward automatically generating privacy policy for android apps. IEEE Transactions on Information Forensics and Security 12(4):865–880
289. Zhang C, Wang H, Wang R, Guo Y, Xu G (2018) Re-checking app behavior against app description in the context of third-party libraries. In: SEKE, pp 665–664
290. Zhang D, Wang R, Lin Z, Guo D, Cao X (2016) Iacdroid: Preventing inter-app communication capability leaks in android. In: Computers and Communication (ISCC), 2016 IEEE Symposium on, IEEE, pp 443–449
291. Zhang F, Huang H, Zhu S, Wu D, Liu P (2014) Viewdroid: Towards obfuscation-resilient mobile application repackaging detection. In: Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks, pp 25–36
292. Zhang H, Babar MA (2013) Systematic reviews in software engineering: An empirical investigation. Information and Software Technology 55(7):1341–1354
293. Zhang H, Wu H, Rountev A (2016) Automated test generation for detection of leaks in android applications. In: Proceedings of the 11th International Workshop on Automation of Software Test, ACM, pp 64–70
294. Zhang J, Qin Z, Zhang K, Yin H, Zou J (2018) Dalvik opcode graph based android malware variants detection using global topology features. IEEE Access 6:51,964–51,974
295. Zhang J, Beresford AR, Kollmann SA (2019) Libid: reliable identification of obfuscated third-party android libraries. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp 55–65
296. Zhang M, Yin H (2014) Appsealer: Automatic generation of vulnerability-specific patches for preventing component hijacking attacks in android applications. In: NDSS
297. Zhang M, Yin H (2014) Efficient, context-aware privacy leakage confinement for android applications without firmware modding. In: Proceedings of the 9th ACM symposium on Information, computer and communications security, ACM, pp 259–270
298. Zhang M, Duan Y, Yin H, Zhao Z (2014) Semantics-aware android malware classification using weighted contextual api dependency graphs. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM, pp 1105–1116
299. Zhang M, Duan Y, Feng Q, Yin H (2015) Towards automatic generation of security-centric descriptions for android apps. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp 518–529
300. Zhang Y, Huang G, Liu X, Zhang W, Mei H, Yang S (2012) Refactoring android java code for on-demand computation offloading. ACM Sigplan Notices 47(10):233–248
301. Zhang Y, Tan T, Li Y, Xue J (2017) Ripple: Reflection analysis for android apps in incomplete information environments. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, ACM, pp 281–288
302. Zhang Y, Dai J, Zhang X, Huang S, Yang Z, Yang M, Chen H (2018) Detecting third-party libraries in android applications with high precision and recall. In: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, pp 141–152
303. Zhang Y, Sui Y, Xue J (2018) Launch-mode-aware context-sensitive activity transition analysis. In: Proceedings of the 40th International Conference on Software Engineering, pp 598–608
304. Zhao J, Albarghouthi A, Rastogi V, Jha S, Octeau D (2018) Neural-augmented static analysis of android communication. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp 342–353
305. Zhao Y, Laser MS, Lyu Y, Medvidovic N (2018) Leveraging program analysis to reduce user-perceived latency in mobile applications. In: Proceedings of the 40th International Conference on Software

Engineering, pp 176–186
306. Zhauniarovich Y, Ahmad M, Gadyatskaya O, Crispo B, Massacci F (2015) Stadyna: Addressing the problem of dynamic code updates in the security analysis of android applications. In: Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, ACM, pp 37–48
307. Zheng C, Zhu S, Dai S, Gu G, Gong X, Han X, Zou W (2012) Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications. In: Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices, ACM, pp 93–104
308. Zhongyang Y, Xin Z, Mao B, Xie L (2013) Droidalarm: an all-sided static analysis tool for android privilege-escalation malware. In: Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security, pp 353–358
309. Zhou Y, Wang Z, Zhou W, Jiang X (2012) Hey, you, get off of my market: detecting malicious apps in official and alternative android markets. In: NDSS, vol 25, pp 50–52
310. Zhou Y, Wu L, Wang Z, Jiang X (2015) Harvesting developer credentials in android apps. In: Proceedings of the 8th ACM conference on security & privacy in wireless and mobile networks, pp 1–12
311. Zimmeck S, Wang Z, Zou L, Iyengar R, Liu B, Schaub F, Wilson S, Sadeh NM, Bellovin SM, Reidenberg JR (2017) Automated analysis of privacy requirements for mobile apps. In: NDSS

Appendix

A.1 Research Team

Four researchers were involved in this study, each of them with a specific role within the research team.

- *Principal researcher*: Gian Luca Scoccia, and Roberto Verdecchia, postdocs. They took part in all the activities, i.e., planning the study, conducting it, and reporting;
- *Research methodologist*: Ivano Malavolta, assistant professor with expertise in empirical software engineering, software architecture, and systematic literature reviews; he was mainly involved in (i) the planning phase of the study, and (ii) supporting the principal researchers during the whole study, e.g., by reviewing the data extraction form, selected primary studies, extracted data, produced reports, etc.;
- *Advisor*: Marco Autili, associate professor with many-years expertise in software engineering methods applied to the modeling, verification, analysis and automatic synthesis of complex distributed systems, and application of context-oriented programming and analysis techniques to the development of (adaptable) mobile applications. He took final decisions on conflicts and methodological options, and supported the other researchers during data and findings synthesis activities.

A.2 Primary Studies

Table 4 reports the full list of the 261 primary studies.

Table 4: Primary Studies

ID	Title	Authors	Year
P1	NeSeDroid–Android Malware Detection Based on Network Traffic and Sensitive Resource Accessing [45]	N.T. Cam, N.C.H. Phuoc	2017
P2	Analyzing Remote Server Locations for Personal Data Transfers in Mobile Apps [68]	M. Eskandari, B. Kessler, M. Ahmad, A. Santana de Oliveira, B. Crispo	2017
P3	MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models [179]	E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, G. Stringhini	2017
P4	Ripple: Reflection Analysis for Android Apps in Incomplete Information Environments [301]	Y Zhang, T Tan, Y Li, J Xue	2017
P5	AndroDialysis: Analysis of Android Intent Effectiveness in Malware Detection [75]	A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez, S. Furnell	2017
P6	Profiling the responsiveness of Android applications via automated resource amplification [247]	Y. Wang, A. Rountev	2016
P7	Detecting Invalid Layer Combinations Using Control-Flow Analysis for Android [233]	N. Suzuki, T. Kamina, K. Maruyama	2016
P8	Graph-aided directed testing of Android applications for checking runtime privacy behaviours [128]	J.C.J. Keng, L. Jiang, T.K. Wee, R.K. Balan	2016
P9	Dexteroid: Detecting malicious behaviors in Android apps using reverse-engineered life cycle models [125]	M. Junaid, D. Liu, D. Kung	2016
P10	IacDroid: Preventing Inter-App Communication capability leaks in Android [290]	D. Zhang, R. Wang, Z. Lin, D. Guo, X. Cao	2016
P11	Practical, formal synthesis and automatic enforcement of security policies for android [21]	H. Bagheri, A. Sadeghi, R. Jabbarvand, S. Malek	2016
P12	CapaDroid: Detecting Capability Leak for Android Applications [266]	T. Wu, Y. Yang	2016
P13	Asynchrony-aware static analysis of Android applications [184]	A. Mishra, A. Kanade, Y.N. Srikant	2016
P14	Identifying Android inter app communication vulnerabilities using static and dynamic analysis [61]	B.F. Demissie, D. Ghio, M. Ceccato, A. Avancini	2016
P15	Towards Automatically Generating Privacy Policy for Android Apps [288]	L. Yu, T. Zhang, X. Luo, L. Xue, H. Chang	2016
P16	Revisiting the Description-to-Behavior Fidelity in Android Applications [286]	L. Yu, X. Luo, C. Qian, S. Wang	2016
P17	Triggerscope: Towards detecting logic bombs in android applications [82]	Y. Fratantonio, A. Bianchi, W. Robertson, E. Kirda, C. Kruegel, G. Vigna	2016
P18	Automated test generation for detection of leaks in Android applications [293]	H. Zhang, H. Wu, A. Rountev	2016
P19	Automatic Construction of Callback Model for Android Application [99]	C. Guo, Q. Ye, N. Dong, G. Bai, J.S. Dong, J. Xu	2016
P20	Automated energy optimization of HTTP requests for mobile applications [144]	D. Li, Y. Lyu, J. Gui, W.G.J. Halfond	2016
P21	Understanding and detecting wake lock misuses for Android applications [169]	Y. Liu, C. Xu, S.C. Cheung, V. Terragni	2016
P22	Taming Android fragmentation: characterizing and detecting compatibility issues for Android apps [253]	L. Wei, Y. Liu, S.C. Cheung	2016
P23	Reflection-aware static analysis of Android apps [150]	L. Li, T.F. Bissyandé, D. Octeau, J. Klein	2016
P24	Automated testing and notification of mobile app privacy leak-cause behaviours [127]	J.C.J. Keng	2016
P25	Finding resume and restart errors in Android applications [221]	Z. Shan, T. Azim, I. Neamtiu	2016
P26	DroidRA: Taming Reflection to Support Whole-ProgramAnalysis of Android Apps [149]	L. Li, T.F. Bissyandé, D. Octeau, J. Klein	2016
P27	Empirical assessment of machine learning-based malware detectors for Android [8]	K. Allix, T.F. Bissyandé, Q. Jérome, J. Klein, Y. Le Traon	2016
P28	Effective detection of android malware based on the usage of data flow APIs and machine learning [265]	S. Wu, P. Wang, X. Li, Y. Zhang	2016
P29	On the Static Analysis of Hybrid Mobile Apps [38]	A.D. Brucker, M. Herzberg	2016
P30	Towards a Generic Framework for Automating Extensive Analysis of Android Applications [151]	L. Li, D. Li, A. Bartel, T.F. Bissyandé, J. Klein, Y. Le Traon	2016
P31	Static Program Analysis for Identifying Energy Bugs in Graphics-Intensive Mobile Apps [130]	C.H.P. Kim, D. Kroening, M. Kwiatkowska	2016
P32	Combining static analysis with probabilistic models to enable market-scale android inter-component analysis [195]	D. Octeau, S. Jha, M. Dering, P. McDaniel, A. Bartel, L. Li, Y. Le Traon	2016
P33	DroidNative: automating and optimizing detection of android native code malware variants [6]	S. Alam, Z. Qu, R. Riley, Y. Chen, V. Rastogi	2016
P34	Enabling Automatic Protocol Behavior Analysis for Android Applications [132]	J. Kim, H. Choi, H. Namkung, W. Choi, B. Choi, H. Hong, D. Han	2016
P35	PERUIM: Understanding Mobile Application Privacy with permission-UI Mapping [157]	Y. Li, Y. Guo, X. Chen	2016
P36	HybriDroid: Static analysis framework for Android hybrid applications [139]	S. Lee, J. Dolby, S. Ryu	2016
P37	StubDroid: automatic inference of precise data-flow summaries for the android framework [11]	S. Arzt, E. Bodden	2016
P38	FlowMine: Android app analysis via data flow [226]	L. Sinha, S. Bhandari, P. Faruki, M.S. Gaur, V. Laxmi, M. Conti	2016
P39	Debugging energy-efficiency related field failures in mobile apps [25]	A. Banerjee, H.F. Guo, A. Roychoudhury	2016
P40	State-Taint Analysis for Detecting Resource Bugs [273]	Z. Xu, D. Fan, S. Qin	2016
P41	Fixing Resource Leaks in Android Apps with Light-Weight Static Analysis and Low-Overhead Instrumentation [165]	J. Liu, T. Wu, J. Yan, J. Zhang	2016
P42	Relda2: an effective static analysis tool for resource leak detection in Android apps[267]	T. Wu, J. Liu, X. Deng, J. Yan, J. Zhang	2016
P43	Detecting energy leaks in android app with poem [79]	A. Ferrari, D. Gallucci, D. Puccinelli, S. Giordano	2016
P44	Lightweight measurement and estimation of mobile ad energy consumption [97]	J. Gui, D. Li, M. Wan, W.G.J. Halfond	2016
P45	AppContext: Differentiating Malicious and Benign Mobile App Behaviors Using Context [279]	W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, W. Enck	2015
P46	Mining Apps for Abnormal Usage of Sensitive Data [15]	V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, E. Bodden	2015
P47	CLAPP: characterizing loops in Android applications [81]	Y. Fratantonio, A. Machiry, A. Bianchi, C. Kruegel, G. Vigna	2015
P48	Study and Refactoring of Android Asynchronous Programming [162]	Y. Lin, S. Okur, D. Dig	2015
P49	Tracking the Software Quality of Android Applications Along Their Evolution [108]	G. Hecht, O. Benomar, R. Rouvoy, N. Moha, L. Duchien	2015
P50	Covert Communication in Mobile Applications [212]	J. Rubin, M.I. Gordon, N. Nguyen, M.C. Rinard	2015
P51	Static Window Transition Graphs for Android [277]	S. Yang, H. Zhang, H. Wu, Y. Wang, A. Rountev	2015
P52	Static Analysis of Implicit Control Flow: Resolving Java Reflection and Android Intents [27]	P. Barros, R. Just, S. Millstein, P. Vines, W. Dietl, M. D'Amorim, M.D. Ernst	2015
P53	String Analysis of Android Applications [60]	J. Del Vecchio, F. Shen, K.M. Yee, B. Wang, S.Y. Ko, L. Ziarek	2015
P54	Interactively verifying absence of explicit information flows in Android apps [31]	O. Bastani, S. Anand, A. Aiken	2015
P55	ShamDroid: gracefully degrading functionality in the presence of limited resource access [40]	L. Brutschy, P. Ferrara, O. Tripp, M. Pistoia	2015
P56	WuKong: a scalable and accurate two-phase approach to Android app clone detection [242]	H. Wang, Y. Guo, Z. Ma, X. Chen	2015

P57	Reevaluating Android Permission Gaps with Static and Dynamic Analysis [243]	H. Wang, Y. Guo, Z. Tang, G. Bai, X. Chen	2015
P58	Andro-autopsy: Anti-malware system based on similarity matching of malware and malware creator-centric information [117]	J. Jang, H. Kang, J. Woo, A. Mohaisen, H.K. Kim	2015
P59	EdgeMiner: Automatically Detecting Implicit Control Flow Transitions through the Android Framework [47]	Y. Cao, Y. Fratantonio, A. Bianchi, M. Egele, C. Kruegel, G. Vigna, Y. Chen	2015
P60	What the app is that? deception and countermeasures in the android user interface [36]	A. Bianchi, J. Corbetta, L. Invernizzi, Y. Fratantonio, C. Kruegel, G. Vigna	2015
P61	Scalable and Precise Taint Analysis for Android [113]	W. Huang, Y. Dong, A. Milanova, J. Dolby	2015
P62	AutoPPG: Towards Automatic Generation of Privacy Policy for Android Applications [285]	L. Yu, T. Zhang, X. Luo, L. Xue	2015
P63	Information-Flow Analysis of Android Applications in DroidSafe [92]	M.I. Gordon, D.Kim, J.H. Perkins, L.Gilham, N.Nguyen, M.C. Rinard	2015
P64	StaDynA: Addressing the Problem of Dynamic Code Updates in the Security Analysis of Android Applications [306]	Y. Zhauniarovich, M. Ahmad, O. Gadyatskaya, B. Crispo, F. Massacci	2015
P65	Potential Component Leaks in Android Apps: An Investigation into a New Feature Set for Malware Detection [146]	L. Li, K. Allix, D. Li, A. Bartel, T.F. Bissyandé, J. Klein	2015
P66	Static Control-Flow Analysis of User-Driven Callbacks in Android Applications [276]	S.Yang, D.Yan, H.Wu, Y.Wang, A.Rountev	2015
P67	Composite Constant Propagation: Application to Android Inter-Component Communication Analysis [194]	D. Octeau, D. Luchaup, M. Dering, S. Jha, P.D. McDaniel	2015
P68	IccTA: Detecting Inter-Component Privacy Leaks in Android Apps [148]	L. Li, A. Bartel, T.F. Bissyandé, J.Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, P.D. McDaniel	2015
P69	EcoDroid: An Approach for Energy-based Ranking of Android Apps [34]	R.J. Behrouz, A. Sadeghi, J. Garcia, S. Malek, P. Ammann	2015
P70	Supor: Precise and scalable sensitive user input detection for android apps [112]	J. Huang, Z. Li, X. Xiao, Z. Wu, K. Lu, X. Zhang, G. Jiang	2015
P71	Uipicker: User-input privacy identification in mobile applications [187]	Y. Nan, M. Yang, Z. Yang, S. Zhou, G. Gu, X. Wang	2015
P72	Andro Lyze: A Distributed Framework for Efficient Android App Analysis [33]	L. Baumgärtner, P. Graubner, N. Schmidt, B. Freisleben	2015
P73	Using text mining to infer the purpose of permission use in mobile apps [244]	H. Wang, J. Hong, Y. Guo	2015
P74	Static reference analysis for GUI objects in Android software [211]	A. Rountev, D. Yan	2014
P75	Static analysis for independent app developers [39]	L. Brutschy, P. Ferrara, P. Müller	2014
P76	Cochecker: Detecting capability and sensitive data leaks from component chains in android [57]	X. Cui, D. Yu, P.P.F. Chan, L.C.K. Hui, S.M. Yiu, S. Qing	2014
P77	Android Taint Flow Analysis for App Sets [136]	W. Klieber, L. Flynn, A. Bhosale, L. Jia, L. Bauer	2014
P78	Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps [252]	F. Wei, S. Roy, X. Ou, Robby	2014
P79	AppSealer: Automatic Generation of Vulnerability-Specific Patches for Preventing Component Hijacking Attacks in Android Applications [296]	M. Zhang, H. Yin	2014
P80	Semantics-aware android malware classification using weighted contextual api dependency graphs [298]	M. Zhang, Y. Duan, H. Yin, Z. Zhao	2014
P81	DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket [10]	D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck	2014
P82	Retrofitting concurrency for android applications through refactoring [161]	Y. Lin, C. Radoi, D. Dig	2014
P83	Checking app behavior against app descriptions [93]	A. Gorla, I. Tavecchia, F. Gross, A. Zeller	2014
P84	Information Flows As a Permission Mechanism [225]	F. Shen, N. Vishnubhotla, C. Todarka, M. Arora, B. Dhandapani, E.J. Lehner, S.Y. Ko, L. Ziarek	2014
P85	Greendroid: Automated diagnosis of energy inefficiency for smartphone applications [168]	Y. Liu, C. Xu, S.C. Cheung, J. Lu	2014
P86	FlowDroid: Precise context-,flow-,field-,object-sensitive and lifecycle-aware taint analysis for android apps [12]	S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, P.D. McDaniel	2014
P87	Cassandra: Towards a Certifying App Store for Android [170]	S. Lortz, H. Mantel, A. Starostin, T. Bähr, D. Schneider, A. Weber	2014
P88	Code Injection Attacks on HTML5-based Mobile Apps:Characterization, Detection and Mitigation [121]	X. Jin, X. Hu, K. Ying, W. Du, H. Yin, G. Nagesh Peri	2014
P89	Efficient, context-aware privacy leakage confinement for android applications without firmware modding [297]	M. Zhang, H. Yin	2014
P90	Collaborative Verification of Information Flow for a High-Assurance App Store [67]	M.D. Ernst, R. Just, S. Millstein, W. Dietl, S. Pernsteiner, F. Roesner, K. Koscher, P. Barros, R. Bhoraskar, S. Han, P. Vines, E.X. Wu	2014
P91	Multi-App Security Analysis with FUSE: Statically Detecting Android App Collusion [208]	T. Ravitch, E.R. Creswick, A. Tomb, A. Foltzer, T. Elliott, L. Casburn	2014
P92	Characterizing and detecting performance bugs for smartphone applications [167]	Y. Liu, C. Xu, S.C. Cheung	2014
P93	AsDroid: detecting stealthy behaviors in Android applications by user interface and program behavior contradiction [111]	J. Huang, X. Zhang, L. Tan, P. Wang, B. Liang	2014
P94	Apposcopy: semantics-based detection of Android malware through static analysis [78]	Y. Feng, S. Anand, I. Dillig, A. Aiken	2014
P95	Detecting energy bugs and hotspots in mobile apps [24]	A. Banerjee, L.K. Chong, S. Chattopadhyay, A. Roychoudhury	2014
P96	Static Analysis for Extracting Permission Checks of a Large Scale Framework: The Challenges and Solutions for Analyzing Android [29]	A. Bartel, J. Klein, M. Monperrus, Y. Le Traon	2014
P97	Responsiveness analysis tool for android application [196]	T. Ongkosit, S. Takada	2014
P98	Automatically exploiting potential component leaks in android applications [145]	L. Li, A. Bartel, J. Klein, Y. Le Traon	2014
P99	Effective inter-component communication mapping in android: An essential step towards holistic security analysis [193]	D. Octeau, P.D. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein, Y. Le Traon	2013
P100	DroidAPIMiner: Mining API-level features for robust malware detection in android [1]	Y. Aafer, W. Du, H. Yin	2013
P101	An empirical study of cryptographic misuse in android applications [64]	M. Egele, D. Brumley, Y. Fratantonio, C. Kruegel	2013
P102	Targeted and depth-first exploration for systematic testing of android apps [16]	T. Azim, I. Neamtiu	2013
P103	Sound and precise malware analysis for android via pushdown reachability and entry-point saturation [160]	S. Liang, A.W. Keep, M. Might, S. Lyde, T. Gilray, Liang, S., Keep, A. W., Might, M., Lyde, S., Gilray, T., P. Aldous, D. Van Horn	2013
P104	AppIntent: analyzing sensitive data transmission in android for privacy leakage detection [283]	Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, X.S. Wang	2013
P105	AppProfiler: a flexible method of exposing privacy-related behavior in android applications to end users [210]	S. Rosen, Z. Qian, Z.M. Mao	2013
P106	Flow permissions for android [110]	S. Holavanalli, D. Manuel, V. Nanjundaswamy, B. Rosenberg, F. Shen, S.Y. Ko, L. Ziarek	2013
P107	Slicing Droids: Program Slicing for Smali Code [109]	J. Hoffmann, M. Ussath, T. Holz, M. Spreitzenbarth	2013
P108	A grey-box approach for automated GUI-model generation of mobile applications [278]	W. Yang, M.R. Prasad, T. Xie	2013
P109	Structural detection of android malware using embedded call graphs [88]	H. Gascon, F. Yamaguchi, D. Arp, K. Rieck	2013
P110	Estimating mobile application energy consumption using program analysis [104]	S. Hao, D. Li, W.G.J. Halfond, R. Govindan	2013
P111	Characterizing and detecting resource leaks in Android applications [98]	C. Guo, J. Zhang, J. Yan, Z. Zhang, Y. Zhang	2013
P112	Calculating source line level energy information for Android applications [143]	D. Li, S. Hao, W.G.J. Halfond, R. Govindan	2013
P113	Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications [307]	C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, W. Zou	2012
P114	Why Eve and Mallory love Android: An analysis of Android SSL (in) security [69]	S. Fahl, M. Harbach, T. Muders, M. Smith, L. Baumgärtner, B. Freisleben	2012
P115	User-aware privacy control via extended static-information-flow analysis [270]	X. Xiao, N. Tillmann, M. Fähndrich, J. De Halleux, M. Moskal	2012
P116	Dr. Android and Mr. Hide: fine-grained permissions in android applications [118]	J. Jeon, K. K. Micinski, J. A. Vaughan, A. Fogel, N. Reddy, J. S. Foster, T. D. Millstein	2012
P117	LeakMiner: Detect Information Leakage on Android with Static Taint Analysis [282]	Z. Yang, M. Yang	2012
P118	SCANDAL: Static Analyzer for Detecting Privacy Leaks in Android Applications [131]	J. Kim, Y. Yoon, K. Yi, J. Shin, S. Center	2012
P119	A framework for static detection of privacy leaks in android applications [177]	C. Mann, A. Starostin	2012
P120	DroidChecker: analyzing android applications for capability leak [48]	P.P.F. Chan, L. C. K. Hui, S. M. Yiu	2012
P121	DroidMat: Android Malware Detection through Manifest and API Calls Tracing [261]	D.J. Wu, C.H. Mao, T.E. Wei, H.M. Lee, K.P. Wu	2012
P122	Static analysis of Android programs [201]	E. Payet, F. Spoto	2012
P123	Estimating Android applications' CPU energy usage via bytecode profiling [103]	H. Hao, D. Li, W. G. J. Halfond, R. Govindan	2012
P124	What is keeping my phone awake?: characterizing and detecting no-sleep energy bugs in smartphone apps [199]	A. Pathak, A. Jindal, Y. Charlie Hu, S. P. Midkiff	2012
P125	User-centric dependence analysis for identifying malicious mobile apps [66]	K.O. Elish, D. Yao, B.G. Ryder	2012
P126	AndroidLeaks: automatically detecting potential privacy leaks in android applications on a large scale [90]	C. Gibler, J. Crussell, J. Erickson, H. Chen	2012
P127	Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets. [309]	Y. Zhou, Z. Wang, W. Zhou, X. Jiang	2012
P128	RiskRanker: Scalable and Accurate Zero-day Android Malware Detection [95]	M.C. Grace, Y.Zhou, Q. Zhang, S. Zou, X. Jiang	2012
P129	Chex: statically vetting android apps for component hijacking vulnerabilities [171]	L. Lu, Z. Li, Z. Wu, W. Lee, G. Jiang	2012
P130	Automatically securing permission-based software by reducing the attack surface: An application to android [28]	A. Bartel, J. Klein, Y. Le Traon, M. Monperrus	2012

P131	Android permissions demystified [76]	A. Porter Felt, E. Chin, S. Hanna, D. Song, D. A. Wagner:	2011
P132	Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications [32]	L. Batyuk, M. Herpich, S. A. Çamtepe, K. Raddatz, A. D. Schmidt, S. Albayrak	2011
P133	Analyzing Inter-application Communication in Android [53]	E. Chin, A. Porter Felt, K. Greenwood, D. A. Wagner	2011
P134	PiOS: Detecting Privacy Leaks in iOS Applications [63]	M. Egele, C. Kruegel, E. Kirda, G. Vigna	2011
P135	Energypatch: Repairing resource leaks to improve energy-efficiency of android apps [26]	A. Banerjee , L. K. Chong, C. Ballabriga, A. Roychoudhury	2018
P136	A multi-view context-aware approach to Android malware detection and malicious code localization [188]	A. Narayanan, M. Chandramohan, L. Chen, Y. Liu	2018
P137	AndroidOff:Offloading android application based on cost estimation [51]	X. Chen, J. Chen, B. Liu, Y. Ma, Y. Zhang, H. Zhong	2019
P138	A SEALANT for Inter-App Security Holes in Android [140]	Y.K. Lee, J.Y. Bang, G. Safi, A. Shahbazian, Y. Zhao, N. Medvidovic	2017
P139	LibD: Scalable and Precise Third-party Library Detection in Android Markets [155]	M. Li, W. Wang, P. Wang, S. Wang, D. Wu, J. Liu, R. Xue, W. Huo	2017
P140	Self-Hiding Behavior in Android Apps: Detection and Characterization [222]	Z. Shan, I.G. Neamtiu, S. Raina	2018
P141	Iconintent: automatic identification of sensitive ui widgets based on icon classification for android apps [271]	X. Xiao, X. Wang, Z. Cao, H. Wang, P. Gao	2019
P142	Efficiently Manifesting Asynchronous Programming Errors in Android Apps [70]	L. Fan, T. Su, S. Chen, G. Meng, Y. Liu, L. Xu, G. Pu	2018
P143	Semi-automated Discovery of Server-Based Information Oversharing Vulnerabilities in Android Applications [137]	W. Koch, A. Chaabane, M. Egele, W. Robertson, E. Kirda	2017
P144	LibID: Reliable Identification of Obfuscated Third-Party Android Libraries [295]	J. Zhang, A.R. Beresford, S.A. Kollmann	2019
P145	ADS-SA: System for Automatically Detecting Sensitive Path of Android Applications Based on Static Analysis [229]	H. Song, D. Lin, S. Zhu, W. Wanga, S. Zhang	2019
P146	CDGDroid: Android malware detection based on deep learning using CFG and DFG [274]	Z Xu, K Ren, S Qin, F Craciun	2018
P147	Machine Learning for Android Malware Detection Using Permission and API Calls [202]	N. Peiravian, X. Zhu	2013
P148	IIFA: modular inter-app intent information flow analysis of android applications [236]	A. Tiwari, S. Groß, C. Hammer	2019
P149	Static Back-Stack Transition Analysis for Android [164]	A. Liu, C. Guo, W. Wang, Y. Qiu, J. Xu	2019
P150	Android Malware Familial Classification and Representative Sample Selection via Frequent Subgraph Analysis [72]	M. Fan, J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zheng, T. Liu	2018
P151	StateDroid: Stateful Detection of Stealthy Attacks in Android Apps via Horn-Clause Verification [126]	M. Junaid, J. Ming, D. Kung	2018
P152	Apkcombiner: Combining multiple android apps to support inter-app analysis [147]	L. Li, A. Bartel, T.F. Bissyandé, J. Klein, Y. Le Traon	2015
P153	Andarwin: Scalable detection of semantically similar android applications [56]	J. Crussell, C. Gibler, H. Chen	2013
P154	Appx: an automated app acceleration framework for low latency mobile app [54]	B. Choi, J. Kim, D. Cho, S. Kim, D. Han	2018
P155	Achieving Accuracy and Scalability Simultaneously in Detecting Application Clones on Android Markets [49]	K. Chen, P. Liu, Y. Zhang	2014
P156	Search-Based Energy Testing of Android [56]	R. Jabbarvand, J.W. Lin, S. Malek	2019
P157	TelCC: Targeted Execution of Inter-Component Communications in Android [4]	M. Ahmad, V. Costamagna, B. Crispo, F. Bergadano	2017
P158	API compatibility issues in Android: Causes and effectiveness of data-driven detection techniques [218]	S. Scalabrino, G. Bavota, M. Linares-Vásquez, V. Piantadosi, M. Lanza, R. Oliveto	2018
P159	Androlic: an extensible flow, context, object, field, and path-sensitive static analysis framework for Android [197]	L. Pan, B. Cui, J. Yan, X. Ma, J. Yan, J. Zhang	2019
P160	EvoDroid: Segmented Evolutionary Testing of Android Apps [176]	R. Mahmood, N. Mirzaei, S. Malek	2014
P161	LibRadar: Fast and Accurate Detection of Third-party Libraries in Android Apps [175]	Z. Ma, H. Wang, Y. Guo, X. Chen	2016
P162	Open Doors for Bob and Mallory: Open Port Usage in Android Apps and Security Implications [119]	Y.J. Jia; Q.A. Chen; Y. Lin; C. Kong; Z. Morley Mao	2017
P163	Finding flaws from password authentication code in Android apps [174]	S. Ma, E. Bertino, S. Nepal, J. Li, D. Ostry, R.H. Deng, S. Jha	2019
P164	Detecting Third-Party Libraries in Android Applications with High Precision and Recall [302]	Y. Zhang, J. Dai, X. Zhang, S. Huang, Z. Yang, M. Yang, H. Chen	2018
P165	EARMO: An Energy-Aware Refactoring Approach for Mobile Apps [185]	R. Morales, R. Saborido, F. Khomh, F. Chicano, G. Antoniol	2018
P166	DelDroid : An automated approach for determination and enforcement of least-privilege architecture in android [101]	M. Hammad, H. Bagheri, S. Malek	2019
P167	Lifting inter-app data-flow analysis to large app sets [216]	F. Sattler, A. von Rhein, T. Berger, N.S. Johansson, M.M. Hardø, S. Apel	2018
P168	Leveraging Program Analysis to Reduce User-Perceived Latency in Mobile Applications [305]	Y. Zhao, M.S. Laser, Y. Lyu, N. Medvidovic	2018
P169	Graph Embedding based Familial Analysis of Android Malware using Unsupervised Learning [73]	M. Fan, X. Luo, J. Liu, M. Wang, C. Nong, Q. Zheng, T. Liu	2019
P170	PATDroid: Permission-Aware GUI Testing of Android [213]	A. Sadeghi, R. Jabbarvand, S. Malek	2017
P171	Together Strong: Cooperative Android App Analysis [200]	F. Pauck, H. Wehrheim	2019
P172	Android testing via synthetic symbolic execution [84]	X. Gao; S.H. Tan; Z. Dong, A. Roychoudhury	2018
P173	Self-Protection of Android Systems from Inter-component Communication Attacks [100]	M. Hammad, J. Garcia, S. Malek	2018
P174	MalScan: Fast Market-Wide Mobile Malware Scanning by Social-Network Centrality Analysis [268]	W. Yueming, L. XiaoDi, Z. Deqing, Y. Wei, Z. Xin, J. Hai	2019
P175	Automated API-Usage Update for Android Apps [74]	M. Fazzini, Q. Xin, A. Orso	2019
P176	Leila: formal tool for identifying mobile malicious behaviour [46]	G. Canfora, F. Martinelli, F. Mercaldo, V. Nardone, A. Santone, C.A. Visaggio	2018
P177	Re-checking App Behavior against App Description in the Context of Third-party Libraries [289]	C. Zhang, H. Wang, R. Wang, Y. Guo, G. Xu	2018
P178	VAnDroid: A framework for vulnerability analysis of Android applications using a model-driven reverse engineering technique [191]	A. Nirumand, B. Zamani, B.T. Ladani	2019
P179	AnFlo: Detecting Anomalous Sensitive Information Flows in Android Apps [62]	B. Fisseha Demissie, M. Ceccato, L. Khin Shar	2018
P180	PaddyFrog: systematically detecting confused deputy vulnerability in Android applications [264]	J. Wu, T. Cui, T. Ban, S. Guo, L. Cui	2015
P181	Dalvik Opcode Graph Based Android Malware Variants Detection Using Global Topology Features [294]	J. Zhang, Z. Qin, K. Zhang, H. Yin, J. Zou	2018
P182	Obfusifier: Obfuscation-resistant Android malware detection system		2019
P183	On automatically detecting similar Android apps [159]	Z. Li, J. Sun, Q. Yan, W. Srisa-an, Y. Tsutano	2016
P184	Contextual policy enforcement in android applications with permission event graphs [50]	K.Z. Chen, N.M. Johnson, V. D'Silva, S. Dai, K. MacNamara, T. Magrino, E. Wu, M. Rinard, D. Song	2013
P185	Toward a framework for detecting privacy policy violations in android application code [227]	R. Slavin, X. Wang, M.B. Hosseini, J. Hester, R. Krishnan, J. Bhatia, T.D. Breaux, J. Niu	2016
P186	LUDroid: A large scale analysis of Android–Web hybridization [237]	A. Tiwari, J. Prakash, S. Groß, C. Hammer	2019
P187	COVERT: Compositional Analysis of Android Inter-App Permission Leakage [20]	H. Bagheri, A. Sadeghi, J. Garcia, S. Malek	2015
P188	Data-Driven Solutions to Detect API Compatibility Issues in Android: An Empirical Study [217]	S. Scalabrino, G. Bavota, M. Linares-Vásquez, M. Lanza, R. Oliveto	2019
P189	Harvesting Developer Credentials in Android Apps [310]	Y. Zhou, L. Wu, Z. Wang, X. Jiang	2015
P190	Learning Performance Optimization from Code Changes for Android Apps [77]	R. Feng, G. Meng, X. Xie, T. Su, Y. Liu, S. Lin	2019
P191	Guided, Stochastic Model-Based GUI Testing of Android Apps [232]	T. Su, G. Meng, Y. Chen, K. Wu, W. Yang, Y. Yao, G. Pu, Y. Liu, Z. Su	2017
P192	Understanding the purpose of permission use in mobile apps [245]	H. Wang, Y. Li, Y. Guo, Y. Agarwal, J.I. Hong	2017
P193	Smv-hunter: Large scale, automated detection of ssl/tls man-in-the-middle vulnerabilities in android apps [231]	D. Sounthiraraj, J. Sahs, G. Greenwood, Z. Lin, L. Khan	2014
P194	Android HIV: A Study of Repackaging Malware for Evading Machine-Learning Detection [52]	X.Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, K. Ren	2019
P195	Lightweight, Obfuscation-Resilient Detection and Family Identification of Android Malware [86]	J. Garcia, M. Hammad, S. Malek	2018
P196	Who changed you? Obfuscator identification for Android [248]	Y. Wang, A. Rountev	2017
P197	Enhancing the description-to-behavior fidelity in android apps with privacy policy [287]	L. Yu, X. Luo, C. Qian, S. Wang, H.K.N. Leung	2018
P198	Static window transition graphs for Android [277]	S. Yang, H. Wu, H. Zhang, Y. Wang, C. Swaminathan, D. Yan, A. Rountev	2018
P199	An Efficient, Robust, and Scalable Approach for Analyzing Interacting Android Apps [238]	Y. Tsutano, S. Bachala, W. Srisa-An, G. Rothermel, J. Dinh	2017
P200	A Temporal Permission Analysis and Enforcement Framework for Android [214]	A. Sadeghi, R. Jabbarvand, N. Ghorbani, H. Bagheri, S. Malek	2018
P201	GUILeak: Tracing Privacy Policy Claims on User Input Data for Android Applications [246]	X. Wang, X. Qin, M.B. Hosseini, R. Slavin, T. Breaux, J. Niu	2018
P202	Towards understanding and reasoning about android interoperations [18]	S. Bae, S. Lee, S. Ryu	2019

P203	OASIS: Prioritizing Static Analysis Warnings for Android Apps Based on App User Reviews [254]	L. Wei, Y. Liu, S.C. Cheung	2017
P204	ServDroid: Detecting Service Usage Inefficiencies in Android Applications [230]	W. Song, J. Zhang, J. Huang	2019
P205	TDroid: Exposing App Switching Attacks in Android with Control Flow Specialization [166]	J. Liu, D. Wu, J. Xue	2018
P206	Characterizing and identifying misexposed activities in android applications [275]	J. Yan, X. Deng, P. Wang, T. Wu, J. Yan, J. Zhang	2018
P207	OAUTHLINT: An Empirical Study on OAuth Bugs in Android Applications [5]	T. Al Rahat, Y. Feng, Y. Tian	2019
P208	Remove RATs from Your Code: Automated Optimization of Resource Inefficient Database Writes for Mobile Applications [172]	Y. Lyu, D. Li, W.G.J. Halfond	2018
P209	Systematic detection of capability leaks in stock Android smartphones [96]	M.C. Grace, Y. Zhou, Z. Wang, X. Jiang	2012
P210	Understanding the inconsistencies between text descriptions and the use of privacy-sensitive resources of mobile apps [251]	T. Watanabe, M. Akiyama, T. Sakai, T. Mori	2015
P211	Refactoring Android Java Code for On-Demand Computation Offloading [300]	Y. Zhang, G. Huang, X. Liu, W. Zhang, H. Mei, S. Yang	2012
P212	Jitana: A modern hybrid program analysis framework for android platforms [166]	Y. Tsutano, S. Bachala, W. Srisa-an, G. Rothermel, J. Dihn	2019
P213	DroidAlarm: an all-sided static analysis tool for Android privilege-escalation malware [308]	Y. Zhongyang, Z. Xin, B. Mao, L. Xie	2013
P214	Tackling runtime-based obfuscation in Android with TIRO [260]	M.Y. Wong, D. Lie	2018
P215	Grandroid: Graph-based detection of malicious network behaviors in android applications [158]	Z. Li, J. Sun, Q. Yan, W. Srisa-an, S. Bachala	2018
P216	ViewDroid: Towards obfuscation-resilient mobile application repackaging detection [291]	F. Zhang, H. Huang, S. Zhu, D. Wu, P. Liu	2014
P217	Towards automatic generation of security-centric descriptions for android apps [299]	M. Zhang, Y. Duan, Q. Feng, H. Yin	2015
P218	DeepIntent: Deep icon-behavior learning for detecting intention-behavior discrepancy in mobile apps [269]	S. Xi, S. Yang, X. Xiao, Y. Yao, Y. Xiong, F. Xu, H. Wang, P. Gao, Z. Liu, F. Xu, J. Lu	2019
P219	SENTINEL: generating GUI tests for sensor leaks in Android and Android wear apps [263]	H. Wu, H. Zhang, Y. Wang, A. Rountev	2019
P220	LESDroid - A Tool for Detecting Exported Service Leaks of Android Applications [173]	J. Ma, S. Liu, Y. Jiang, X. Tao, C .Xu, J. Lu	2018
P221	Static detection of energy defect patterns in Android applications [262]	H.Wu, S. Yang, A. Rountev	2016
P222	Wechecker: efficient and precise detection of privilege escalation vulnerabilities in android apps [58]	X. Cui, J. Wang, L.C.K. Hui, Z. Xie, T. Zeng, S. Yiu	2015
P223	Launch-Mode-Aware Context-Sensitive Activity Transition Analysis [303]	Y. Zhang, Y. Sui, J. Xue	2018
P224	SIG-Droid: Automated System Input Generation for Android Applications [182]	N. Mirzaei, H. Bagheri, R. Mahmood, S. Malek	2015
P225	Reducing Combinatorics in GUI Testing of Android Applications [183]	N. Mirzaei, J. Garcia, H. Bagheri, A. Sadeghi, S. Malek	2016
P226	Detection of Design Flaws in the Android Permission Protocol Through Bounded Verification [19]	H. Bagheri, E. Kang, S. Malek, D. Jackson	2015
P227	Reliable third-party library detection in android and its security applications [17]	M. Backes, S. Bugiel, E. Derr	2016
P228	IntelliDroid: A Targeted Input Generator for the Dynamic Analysis of Android Malware [259]	M.Y. Wong, D. Lie	2016
P229	EnMobile: Entity-based Characterization and Analysis of Mobile Malware [280]	W. Yang, M.R. Prasad, T. Xie	2018
P230	Ordol: Obfuscation-Resilient Detection of Libraries in Android Applications [235]	D. Titze, M. Lux, J. Schuette	2017
P231	Towards model checking android applications [23]	G. Bai, Q. Ye, Y. Wu, H. Botha, J. Sun, Y. Liu, J. Dong, W. Visser	2018
P232	Security code smells in Android ICC [83]	P. Gadient, M. Ghafari, P. Frischknecht, O. Nierstrasz	2019
P233	Characterizing malicious Android apps by mining topic-specific data flow signatures [281]	X. Yang, D. Lo, L. Li, X. Xia, T.F. Bissyandé, J. Klein	2017
P234	Making Malory Behave Maliciously: Targeted Fuzzing of Android Execution Environments [207]	S. Rasthofer, S. Arzt, S. Triller, M. Pradel	2017
P235	DroidStar: Callback Typestates for Android Classes [206]	A. Radhakrishna, N.V. Lewchenko, S. Meier, S. Mover, K.C. Sripada, D. Zufferey, B.E. Chang, P. Cherny	2018
P236	PIVOT: Learning API-Device Correlations to Facilitate Android Compatibility Issue Detection [255]	L. Wei, Y. Liu, S.C. Cheung	2019
P237	Automatic Generation of Inter-Component Communication Exploits for Android Applications [85]	J. Garcia, M. Hammad, N. Ghorbani, S. Malek	2017
P238	Neural-augmented static analysis of Android communication [304]	J. Zhao, A. Albarghouthi, V. Rastogi, S. Jha, D. Octeau	2018
P239	Understanding and Detecting Evolution-Induced Compatibility Issues in Android Apps [106]	D. He, L. Li, L. Wang, H. Zheng, G. Li, J. Xue	2018
P240	Goal-Driven Exploration for Android Applications [138]	D. Lai, J. Rubin	2019
P241	CiD: Automating the Detection of API-related Compatibility Issues in Android Apps [153]	L. Li, T.F. Bissyandé, H. Wang, J. Klein	2018
P242	QADroid: Regression Event Selection for Android Applications [224]	A. Sharma, R. Nasre	2019
P243	Automated analysis of privacy requirements for mobile app [311]	S. Zimmeck, Z. Wang, L. Zou, R. Iyengar, B. Liu, F. Schaub, W. Shomir, N.M. Sadeh, S.M. Bellovin, J.R. Reidenberg	2017
P244	apk2vec: Semi-supervised multi-view representation learning for profiling Android applications [189]	A. Narayanan, C. Soh, L. Chen, Y. Liu, L. Wang	2018
P245	Collusive Data Leak and More: Large-scale Threat Analysis of Inter-app Communications [37]	A. Bosu, F. Liu, D. Yao, G. Wang	2017
P246	Efficient, Evolutionary Security Analysis of Interacting Android Apps [22]	H. Bagheri, J. Wang, J. Aerts, S. Malek	2018
P247	Detecting Passive Content Leaks and Pollution in Android Applications [120]	Y.Z.X. Jiang, Z. Xuxian	2013
P248	Dapasa: detecting android piggybacked apps through sensitive subgraph analysis [71]	M. Fan, J. Liu, W. Wang, H. Li, Z. Tian, T. Liu	2017
P249	Orlis: Obfuscation-resilient library detection for Android [249]	Y. Wang, H. Wu, H. Zhang, A. Rountev	2018
P250	Juxtapp: A scalable system for detecting code reuse among android applications [102]	S. Hanna, L. Huang, E. Wu, S. Li, C. Chen, D. Song	2012
P251	A new android malware detection approach using bayesian classification [284]	S.Y. Yerima, S. Sezer, G. McWilliams, I. Muttik	2013
P252	Towards a Scalable Resource-driven Approach for Detecting Repackaged Android Applications [223]	Y. Shao, X. Luo, C. Qian, P. Zhu, L. Zhang	2014
P253	Efficiently Manifesting Asynchronous Programming Errors in Android Apps [70]	L. Fan, T. Su, S. Chen, G. Meng, Y. Liu, L. Xu, G. Pu	2018
P254	Automatically Discovering, Reporting and Reproducing Android Application Crashes [186]	K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas; C. Vendome; D. Poshyvanyk	2016
P255	HornDroid: Practical and Sound Static Analysis of Android Applications by SMT Solving [44]	S. Calzavara, I. Grishchenko, M. Maffei	2016
P256	Android Multitasking Mechanism: Formal Semantics and Static Analysis of Apps [107]	J. He, T. Chen, P. Wang, Z. Wu, J. Yan	2019
P257	Privacy Leakage through Exploitation of Vulnerable Inter-App Communication on Android [178]	H.M.A. Maqsood, K.N. Qureshi, F. Bashir, N.U. Islam	2019
P258	Revisiting the impact of common libraries for android-related investigations [154]	L. Li, T. Riom, T.F. Bissyandé, H. Wang, J. Klein	2019
P259	Authorship attribution of Android apps [91]	H. Gonzalez, N. Stakhanova, A.A. Ghorbani	2018
P260	Improving accuracy of Android malware detection with lightweight contextual awareness [7]	J. Allen, M. Landen, S. Chaba, Y. Ji, S.P.H. Chung, W. Lee	2018
P261	Characterizing and Detecting Inefficient Image Displaying Issues in Android Apps [156]	W. Li, Y. Jiang, C. Xu, Y. Liu, X. Ma, J. Lù	2019