A Taxonomy and Qualitative Comparison of Program Analysis Techniques for Security Assessment of Android Software

Alireza Sadeghi, Hamid Bagheri, *Member, IEEE*, Joshua Garcia, and Sam Malek, *Member, IEEE*

Abstract—In parallel with the meteoric rise of mobile software, we are witnessing an alarming escalation in the number and sophistication of the security threats targeted at mobile platforms, particularly Android, as the dominant platform. While existing research has made significant progress towards detection and mitigation of Android security, gaps and challenges remain. This paper contributes a comprehensive taxonomy to classify and characterize the state-of-the-art research in this area. We have carefully followed the systematic literature review process, and analyzed the results of more than 300 research papers, resulting in the most comprehensive and elaborate investigation of the literature in this area of research. The systematic analysis of the research literature has revealed patterns, trends, and gaps in the existing literature, and underlined key challenges and opportunities that will shape the focus of future research efforts.

Index Terms—Taxonomy and survey, security assessment, android platform, program analysis

1 Introduction

Android, with well over a million apps, has become one of the dominant mobile platforms [1]. Mobile app markets, such as Android Google Play, have created a fundamental shift in the way software is delivered to consumers, with thousands of apps added and updated on a daily basis. The rapid growth of app markets and the pervasiveness of apps provisioned on such repositories have paralleled with an increase in the number and sophistication of the security threats targeted at mobile platforms. Recent studies have indicated mobile markets are harboring apps that are either malicious or vulnerable, leading to compromises of millions of devices.

This is nowhere more evident than in the Android markets, where many cases of apps infected with malwares and spywares have been reported [2]. Numerous culprits are in play here, and some are not even technical, such as the general lack of an overseeing authority in the case of open markets and inconsequential implication to those caught provisioning applications with vulnerabilities or malicious capabilities. The situation is even likely to exacerbate given that mobile apps are poised to become more complex and ubiquitous, as mobile computing is still in its infancy.

In this context, Android's security has been a thriving subject of research in the past few years, since its inception in 2008. These research efforts have investigated the Android security threats from various perspectives and are scattered across several research communities, which has resulted in a body of literature that is spread over a wide variety of domains and publication venues. The majority of surveyed literature has been published in the software engineering and security domains. However, the Android's security literature also overlaps with those of mobile computing and programming language analysis. Yet, there is a lack of a broad study that connects the knowledge and provides a comprehensive overview of the current state-of-the-art about what has already been investigated and what are still the open issues.

This paper presents a comprehensive review of the existing approaches for Android security analysis. The review is carried out to achieve the following objectives:

- To provide a basis taxonomy for consistently and comprehensively classifying Android security assessment mechanisms and research approaches;
- To provide a systematic literature review of the state-of-the-art research in this area using the proposed taxonomy;
- To identify trends, patterns, and gaps through observations and comparative analysis across Android security assessment systems; and
- To provide a set of recommendations for deriving a research agenda for future developments.

We have carefully followed the systematic literature review process, and analyzed the results of 336 research papers published in diverse journals and conferences. Specifically, we constructed a comprehensive taxonomy by performing a "survey of surveys" on related taxonomies and conducting an iterative content analysis over a set of papers collected using reputable literature search engines. We then applied the taxonomy to classify and characterize the

- *A. Sadeghi, J. Garcia, and S. Malek are with the School of Information and Computer Sciences, University of California, Irvine, CA 92612. E-mail: {alirezs1, joshug4, malek}@uci.edu.*
- *H. Bagheri is with the Department of Computer Science and Engineering, University of Nebraska, Lincoln, NE 68588. E-mail: bagheri@unl.edu.*

Manuscript received 18 May 2016; revised 20 Sept. 2016; accepted 30 Sept. 2016. Date of publication 4 Oct. 2016; date of current version 20 June 2017. Recommended for acceptance by P. Tonella.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2016.2615307

state-of-the-art research in the field of Android security. We finally conducted a cross analysis of different concepts in the taxonomy to derive current trends and gaps in the existing literature, and underline key challenges and opportunities that will shape the focus of future research efforts. To the best of our knowledge, this study is the most comprehensive and elaborate investigation of the literature in this area of research.

The rest of the paper is organized as follows: Section 2 overviews the Android framework to help the reader follow the discussions that ensue. Section 3 lists the existing surveys that are directly or indirectly related to the Android security analysis. Section 4 presents the research method and the underlying protocol for the systematic literature review. Section 5 presents a comprehensive taxonomy for the Android security analysis derived from the existing research literature. Section 6 presents a classification of the state-of-the-art research into the proposed taxonomy as well as a cross analysis of different concepts in the taxonomy. Section 7 provides a trend analysis of surveyed research, discusses the observed gaps in the studied literature, and identifies future research directions based on the survey results. Section 8 presents the conclusions.

2 Android Overview

This section provides a brief overview of the Android platform and its incorporated security mechanisms and protection measures to help the reader follow the discussions that ensue.

Android Platform. Android is a platform for mobile devices that includes a Linux OS, system libraries, middleware, and a suite of pre-installed applications. Android applications (apps) are mainly written in the Java programming language by using a rich collection of APIs provided by the Android Software Development Kit (SDK). An app's compiled code alongside data and resources are packed into an archive file, known as an Android application package (APK). Once an APK is installed on an Android device, it runs by using the Android runtime (ART) environment.[1]

Application Components. Android defines four types of components: *Activity* components that provide a user interface, *Service* components that execute processes in the background without user interaction, *Content Provider* components that provide the capability of data sharing across applications, and *Broadcast Receiver* components that respond asynchronously to system-wide announcement messages.

Application Configuration. The manifest is a mandatory configuration file (AndroidManifest.xml) that accompanies each Android app. It specifies, among other things, the principal components that constitute the application, including their types and capabilities, as well as required and enforced permissions. The manifest file values are bound to the Android app at compile-time, and cannot be modified at run-time.

Inter-Component Communication. As part of its protection mechanism, Android insulates applications from each other and system resources from applications via a sandboxing mechanism. Such application insulation that Android depends on to protect applications requires interactions to occur through a message passing mechanism, called inter-component communication (ICC). ICC in Android is mainly conducted by means of *Intent* messages. Component capabilities are specified as a set of *Intent-Filters* that represent the kinds of requests a given component can respond to. An Intent message is an event for an action to be performed along with the data that supports that action. Component invocations come in different flavors, e.g., explicit or implicit, intra- or inter-app, etc. Android's ICC allows for late run-time binding between components in the same or different applications, where the calls are not explicit in the code, rather made possible through event messaging, a key property of event-driven systems. It has been shown that the Android ICC interaction mechanism introduces several security issues [3]. For example, Intent event messages exchanged among components, among other things, can be intercepted or even tampered, since no encryption or authentication is typically applied upon them [4]. Moreover, no mechanism exists for preventing an ICC callee from misrepresenting the intentions of its caller to a third party [5].

Permissions. Enforcing permissions is the other mechanism, besides sandboxing, provided by the Android framework to protect applications. In fact, permissions are the cornerstone for the Android security model. The permissions stated in the app manifest enable secure access to sensitive resources as well as cross-application interactions. When a user installs an app, the Android system prompts the user for consent to requested permissions prior to installation. Should the user refuse to grant the requested permissions to an app, the app installation is canceled. Until recently, no dynamic mechanism was provided by Android for managing permissions after app installation. In the latest release of Android[2], however, Google introduced dynamic permission management that allows users to revoke or grant app permissions at runtime.

Besides required permissions, the app manifest may also include enforced permissions that other apps must have in order to interact with this app. In addition to built-in permissions provided by the Android system to protect various system resources, any Android app can also define its own permissions for the purpose of self-protection.

The current permission model of Android suffers from shortcomings widely discussed in the literature [6], [7], [8]. Some examples of such defects include coarse-grained permissions that violate the principle of least privilege [9], [10], [11], enforcing access control policies at the level of individual apps that causes delegation attacks [4], [12], [13], [14], and the lack of permission awareness that leads to uninformed decisions by end users [15], [16], [17], [18].

3 Related Surveys

Identifying, categorizing and examining mobile malware have been an interesting field of research since the emergence of mobile platforms. Several years before the advent of modern mobile platforms, such as iOS and Android, Dagon et al. [19] provided a taxonomy of mobile malware. Although the threat models were described for old mobile devices, such as PDAs, our article draws certain attributes from this study for the Android security taxonomy that will be

1. ART is the successor of the Dalvik VM, which was Android's runtime environment until version 4.4 KitKat.

2. Android 6 or Marshmallow.

introduced in Section 5. More recently, Felt et al. [20] analyzed the behavior of a set of malware spread over iOS, Android, and Symbian platforms. They also evaluated the effectiveness of techniques applied by the official app markets, such as Apple's App Store and Google's Android Market (now called Google Play), for preventing and identifying such malware. Along the same lines, Suarez-Tangil et al. [21] presented a comprehensive survey on the evolution of malware for smart devices and provided an analysis of 20 research efforts that detect and analyze mobile malware. Amamra et al. [22] surveyed malware detection techniques for smartphones and classified them as signature-based or anomaly-based. Haris et al. [23] surveyed the mobile computing research addressing the privacy issues, including 13 privacy leak detection tools and 16 user studies in mobile privacy. Enck [24] reviewed some of the efforts in smartphone research, including OS protection mechanisms and security analysis techniques. He also discussed the limitations as well as directions for future research.

While the focus of these surveys is mainly on malware for diverse mobile platforms, the area of Android security analysis has not been investigated in detail.

They do not analyze the techniques for Android vulnerability detection. Moreover, they categorize malware detection techniques based only on limited comparison criteria, and several rather important aspects—such as approach positioning, characteristics, and assessment—are ignored. These comparison areas are fully discussed in our proposed taxonomy (see Section 5).

Besides these general, platform-independent malware surveys, we have found quite a number of relevant surveys that describe subareas of Android security, mainly concerned with specific types of security issues in the Android platform. For instance, Chin et al. [3] studied security challenges in Android inter-application communication, and presented several classes of potential attacks on applications. Another example is the survey of Shabtai et al. [2], [25], which provides a comprehensive assessment of the security mechanisms provided by the Android framework, but does not thoroughly study other research efforts for detection and mitigation of security issues in the Android platform. The survey of Zhou et al. [26] analyzes and characterizes a set of 1,260 Android malware. This collection of malware, called Malware Genome, are then used by many other researchers to evaluate their proposed malware detection techniques.

Each of these surveys overview specific domains (e.g., inter-app vulnerabilities [3] or families of Android malware [26], [27]), or certain types of approaches (e.g., techniques relying on dynamic analysis [28], static analysis [29], or machine learning [30] as well as mechanisms targeting the enhancement of the Android security platform [31], [32]). However, none of them provide a comprehensive overview of the existing research in the area of Android security, including but not limited to empirical and case studies, as well as proposed approaches and techniques to identify, analyze, characterize, and mitigate the various security issues in either the Android framework or apps built on top it. Moreover, since a systematic literature review (SLR) is not leveraged, there are always some important approaches missing in the existing surveys. Having compared over 330 related research publications through the proposed taxonomy, this survey, to the best of our knowledge, is the most comprehensive study in this line of research.

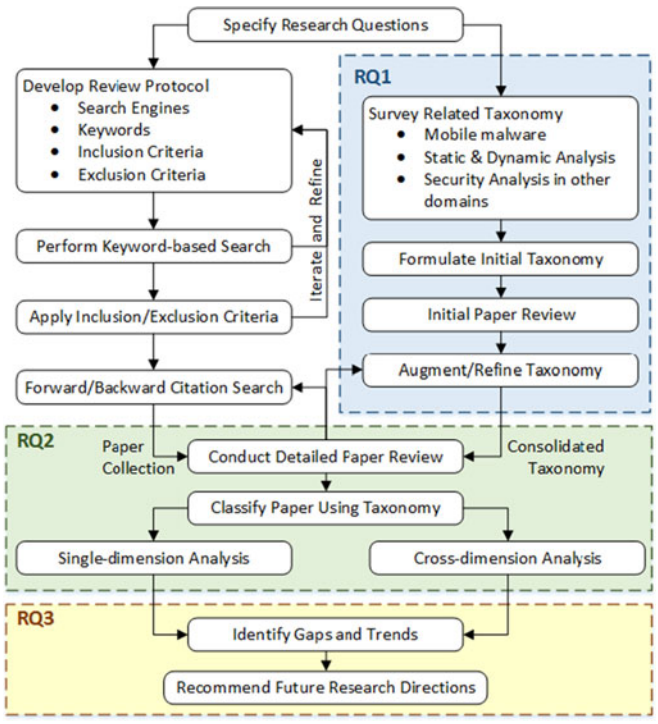
Specify Research Questions
Develop Review Protocol
Search Engines
Keywords
Inclusion Criteria
Exclusion Criteria
Iterate and Refine
Perform Keyword-based Search
Apply Inclusion/Exclusion Criteria
Forward/Backward Citation Search
RQ1
Survey Related Taxonomy
Mobile malware
Static & Dynamic Analysis
Security Analysis in other domains
Formulate Initial Taxonomy
Initial Paper Review
Augment/Refine Taxonomy
RQ2
Paper Collection
Conduct Detailed Paper Review
Consolidated Taxonomy
Classify Paper Using Taxonomy
Single-dimension Analysis
Cross-dimension Analysis
RQ3
Identify Gaps and Trends
Recommend Future Research Directions

Fig. 1. Research process flow and tasks.

4 Research Method

Our survey follows the general guidelines for systematic literature review process proposed by Kitchenham [33]. We have also taken into account the lessons from Brereton et al. [34] on applying SLR to the software engineering domain. The process includes three main phases: planning, conducting, and reporting the review. Based on the guidelines, we have formulated the following research questions, which serve as the basis for the systematic literature review.

- *RQ1:* How can existing research on Android app security analysis be classified?
- *RQ2:* What is the current state of Android security analysis research with respect to this classification?
- *RQ3:* What patterns, gaps, and challenges could be inferred from the current research efforts that will inform future research?

The remainder of this section describes the details of our review process, including the methodology and tasks that we used to answer the research questions (Section 4.1), the detailed SLR protocol including keywords, sources, and selection criteria (Section 4.2), statistics on selected papers based on the protocol (Section 4.3), and finally a short discussion on the threats to validity of our research approach (Section 4.4).

4.1 Research Tasks

To answer the three research questions introduced above, we organized our tasks into a process flow tailored to our specific objectives, yet still adhering to the three-phase SLR process including: planning the review, conducting the review, and reporting the review. The overall process flow is outlined in Fig. 1 and briefly described here.

TABLE 1
Refined Search Keywords

Research Domain (D)	Keywords (K)
Program Analysis	Static (Analysis)*, Dynamic (Analysis)*, Control Flow, Data Flow, Taint, Monitoring, Feature Selection
Security Assessment	Security, Vulnerability/Vulnerable, Malware/Malicious, Virus, Privacy
Android Platform	Android, Mobile, Smartphone, App

First, in the planning phase, we defined the review protocol that includes selection of the search engines, the initial selection of the keywords pertaining to Android security analysis, and the selection criteria for the candidate papers. The protocol is described in detail in Section 4.2.

The initial keyword-based selection of the papers is an iterative process that involves exporting the candidate papers to a "research catalog" and applying the pre-defined inclusion/exclusion criteria on them. In the process, the keyword search expressions and the inclusion/exclusion criteria themselves may also need to be fine-tuned, which would in turn trigger new searches. Once the review protocol and the resulting paper collection were stabilized, our research team also conducted peer-reviews to validate the selections.

For RQ1, in order to define a comprehensive taxonomy suitable for classifying Android security analysis research, we first started with a quick "survey of surveys" on related taxonomies. After an initial taxonomy was formulated, we then used the initial paper review process (focusing on abstract, introduction, contribution, and conclusion sections) to identify new concepts and approaches to augment and refine our taxonomy. The resulting taxonomy is presented in Section 5.

For the second research question (RQ2), we used the validated paper collection and the consolidated taxonomy to conduct a more detailed review of the papers. Each paper was classified using every dimension in the taxonomy, and the results were captured in a research catalog. The catalog, consisting of a set of spreadsheets, allowed us to perform qualitative and quantitative analysis not only in a single dimension, but also across different dimensions in the taxonomy. The analysis and findings are documented in Section 6.[3]

To answer the third research question (RQ3), we analyzed the results from RQ2 and attempted to identify the gaps and trends, again using the taxonomy as a critical aid. The possible research directions are henceforth identified and presented in Section 7.

4.2 Literature Review Protocol

This section provides the details of the review protocol, including our search strategy and inclusion/exclusion criteria.

4.2.1 Search Method

We used reputable literature search engines and databases in our review protocol with the goal of finding high-quality refereed research papers, including journal articles, conference papers, tool demo papers, as well as short papers from respectable venues. The selected search engines consist of IEEE Explore, ACM Digital Library, Springer Link, and ScienceDirect.

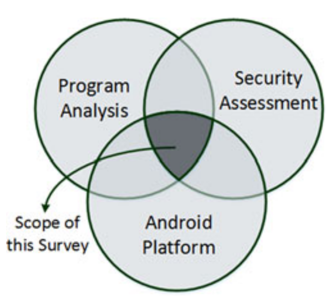

Fig. 2. Scope of this survey.

Given the scope of our literature review, we focused on selected keywords to perform the search on the papers' titles, abstracts, and meta-data, such as keywords and tags. Our search query is formed as a conjunction of three research domains, described in Section 4.2.2 as inclusion criteria, namely, D_1: *Program Analysis*, D_2: *Security Assessment*, and D_3: *Android Platform*. These research domains appear in the literature under different forms and using synonymous words. To retrieve all related papers, each research domain in our search string is represented as a disjunction of corresponding keywords summarized in Table 1. These keywords were continuously refined and extended during the search process. For instance, regarding the security assessment domain, we considered keywords such as, "security", "vulnerability", "malware", "privacy", etc. In summary, our search query is defined as the following formula:

$$query = \bigwedge_{d \in \{D_1, D_2, D_3\}} \left(\bigvee_{keyword \in K_d} keyword \right).$$

Where D_is are the three research domains, and K_d is the set of corresponding keywords specified for domain d in Table 1.

Finally, to eliminate irrelevant publications and also make our search process repeatable, we added a time filter to limit the scope of the search for the papers published from 2008[4] to 2016.[5]

4.2.2 Selection Criteria

Not all the retrieved papers based on the search query fit within the scope of this paper. Therefore, we used the following inclusion and exclusion criteria to further filter the candidate papers.

Inclusion Criteria. As illustrated in Fig. 2, the scope of surveyed research in this study falls at the intersection of three domains:

1) *Program Analysis* domain that includes the techniques used for extracting the models of individual Android apps and/or the Android platform.

3. The research artifacts, including the survey catalog, are available to the public and can be accessed at http://www.ics.uci.edu/~seal/projects/droid-sec-taxonomy

4. The release year of the first version of Android framework.

5. The papers published after January 2016 are not included in this survey.

TABLE 2
Number of Collected Papers at Each Phase of Paper Selection

	Database				
Selection phase	IEEE	ACM	Springer	ScienceDirect	All
Keyword-based search	1,374	938	8,605	2,830	—
Initial filtering	852	721	520	240	—
Merging					2,023
Applying criteria					336

2) *Security Assessment* domain that covers the analysis methods applied on the extracted models to identify the potential security issues among them.
3) *Android Platform* domain that takes into account the special features and challenges involved in the Android platform, its architecture, and security model.

Papers that fall at the intersection of these three domains are included in our review.

Exclusion Criteria. Moreover, we excluded papers that:

1) exclusively developed for platforms other than Android, such as iOS, Windows Mobile, BlackBerry, and Sybmbian (e.g., [35], [36], [37], [38], [39], [40], [41], [42], [43]). However, approaches that cover multiple platforms, including Android, fall within the scope of this survey.
2) focused only on techniques for mitigation of security threats, but not on any security analysis technique. Such techniques attempt to enhance security mechanisms either at the application-level or the level of the Android platform by means of different approaches, such as isolation and sandboxing [44], [45], [46], [47], [48], [49], [50], [51], [52], [53], enhancing permission management [54], [55], [56], [57], [58], [59], anonymity [60], [61], fine-grained or dynamic policy enforcement [10], [62], [63], [64], anti-repackaging [65], [66], [67], [68], [69], security-enhanced communication [70], [71], database and storage [72], [73], [74], cryptography [75], [76], etc. Approaches that consider both detection and protection (e.g., [13], [77], [78], [79]), are included in the survey.
3) performed the analysis *only* on apps meta-data, such as description [80], [81], category [82], signature [83], ranking and reviews [84], [85], resources [86], apk file's meta-data [87], or a combination of these attributes [88], [89], [90]. The analyses running on an app's code, but at opcode level [91], [92], [93], [94], [95] are also excluded.
4) focused only on expanding and enhancing Java program analysis techniques, either static [96], [97], [98], [99], [100], [101] or dynamic [102], [103], for the Android framework. In this survey, however, we included general program-analysis research that, at least, provide a case study or experiment related to security analysis (e.g, [104], [105], [106]).
5) focused solely on low-level monitoring and profiling techniques for identifying security-related anomalies or malware. Such research includes intrusion detection, which performs analysis using hardware signals (e.g., CPU utilization [107], [108], power consumption [109], [110], memory usage [111], [112], network traffic [113], [114], [115], [116], [117], [118], [119], [120], [121], [122], or a combination of multiple sensors [123], [124], [125], [126], [127]). These approaches use mechanisms at a lower level than the Android framework, making them out of scope for this survey.
6) elaborated on a particular attack on the Android framework [14], [128], [129], [130], [131] or apps [132], [133], [134], [135], [136], [137], [138], [139], [140], without describing detection techniques to identify the vulnerabilities that lead to the described security breach.

Fig. 3. Word cloud of the titles of the selected papers.

In addition, the analysis tools that are not accompanied by any peer-reviewed paper were excluded, as most of the taxonomy dimensions are not applicable to such tools. *Dexter* [141] and *DroidBox* [142] are two examples that respectively leverage static and dynamic analysis techniques, but lack any peer-reviewed paper, thus were excluded from this survey.

4.3 Selected Papers

Table 2 provides statistics on each phase of paper collection, illustrated in Fig. 1, for each database.

The first row shows the size of the initial set of papers, selected by keyword-based search over the full paper for each database. Since the search engine of the four databases treat our search query differently, we performed another verification over the initially collected papers, in a consistent manner, based on the same keywords (Row 2). After initial filtering, we merged the search results of all databases into a single repository for further review and filtering (Row 3).

In the fourth row, the number of filtered papers after applying selection criteria is shown. In this stage we applied inclusion and exclusion criteria, enumerated in Section 4.2.2, on the title, abstract and conclusion of the papers selected in the first phase to remove out-of-scope publications. This process led to the selection of 336 papers for this survey—whose titles are illustrated in the form of a word cloud in Fig. 3.

Fig. 4 shows the number of selected papers by publication year. As illustrated in this figure, the number of publications have increased gradually between 2009 and 2011, more than doubled between 2011 and 2012, and hit its peak in 2014.

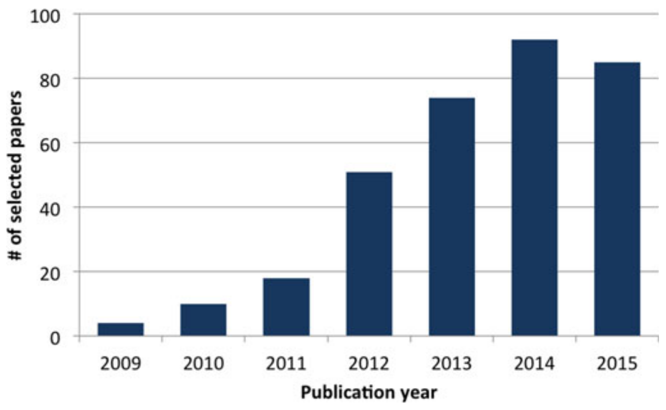

Fig. 4. Distribution of surveyed papers by publication year.

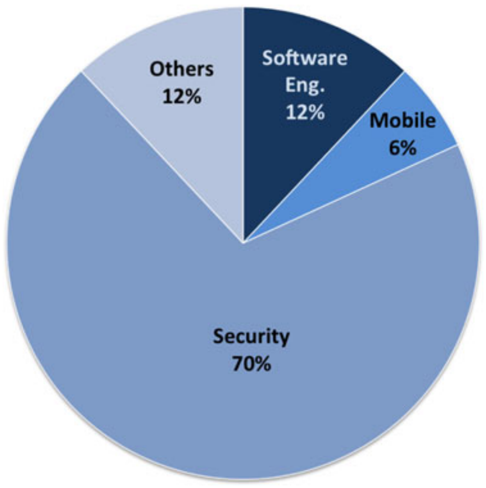

Fig. 5. Distribution of surveyed papers by publication venue.

As shown in Fig. 2, this study covers multidisciplinary research conducted in various domains, such as software engineering (including programming languages), security, and mobility. Consequently, as depicted in Fig. 5, selected papers are also published in different venues related to such domains.

4.4 Threats to Validity

By carefully following the SLR process in conducting this study, we have tried to minimize the threats to the validity of the results and conclusions made in this article. Nevertheless, there are three possible threats that deserve additional discussion.

One important threat is the completeness of this study, that is, whether all of the appropriate papers in the literature were identified and included. This threat could be due to two reasons: (1) some relevant papers were not picked up by the search engines or did not match our keyword search, (2) some relevant papers that were mistakenly omitted, and vice-versa, some irrelevant papers that were mistakenly included. To address these threats, we used multiple search engines, including both scientific and general-purpose search engines. We also adopted an iterative approach for our keyword-list construction. Since different research communities (particularly, software engineering and security) refer to the same concepts using different words, the iterative process allowed us to ensure that a proper list of keywords were used in our search process.

Another threat is the validity of the proposed taxonomy, that is, whether the taxonomy is sufficiently rich to enable proper classification and analysis of the literature in this area. To mitigate this threat, we adopted an iterative content analysis method, whereby the taxonomy was continuously evolved to account for every new concept encountered in the papers. This gives us confidence that the taxonomy provides a good coverage for the variations and concepts that are encountered in this area of research.

Another threat is the objectiveness of the study, which may lead to biased or flawed results. To mitigate this risk, we have tackled the individual reviewer's bias by crosschecking the papers, such that no paper received a single reviewer. We have also tried to base the conclusions on the collective numbers obtained from the classification of papers, rather than individual reviewer's interpretation or general observations, thus minimizing the individual reviewer's bias.

5 Taxonomy

To define an Android security analysis taxonomy for RQ1, we started with selecting suitable dimensions and properties found in existing surveys. The aforementioned studies described in Section 3, though relevant and useful, are not sufficiently specific and systematic enough for classifying the Android security analysis approaches in that they either focus on mobile malware in general, or focus on certain sub-areas, such as Android inter-application vulnerabilities or families of Android malware software, but not on the Android security analysis as a whole.

We thus have defined our own taxonomy to help classify existing work in this area. Nonetheless, the proposed taxonomy is inspired by existing work described in Section 3. The highest level of the taxonomy hierarchy classifies the surveyed research based on the following three questions:

1) *What* are the problems in the Android security being addressed?
2) *How* and with which techniques the problems are solved?
3) How is the validity of the proposed solutions evaluated?

For each question, we derive the sub-dimensions of the taxonomy related to the question, and enumerate the possible values that characterize the studied approaches. The resulting taxonomy hierarchy consists of 21 dimensions and sub-dimensions, which are depicted in Figs. 6, 7, and 8, and explained in the following.

5.1 Approach Positioning (Problem)

The first part of the taxonomy, approach positioning, helps characterize the "WHAT" aspects, that is, the objectives and intent of Android security analysis research. It includes five dimensions, as depicted in Fig. 6.

5.1.1 Analysis Objectives (T1.1)

This dimension classifies the approaches with respect to the goal of their analysis. Thwarting malware apps that compromise the security of Android devices is a thriving

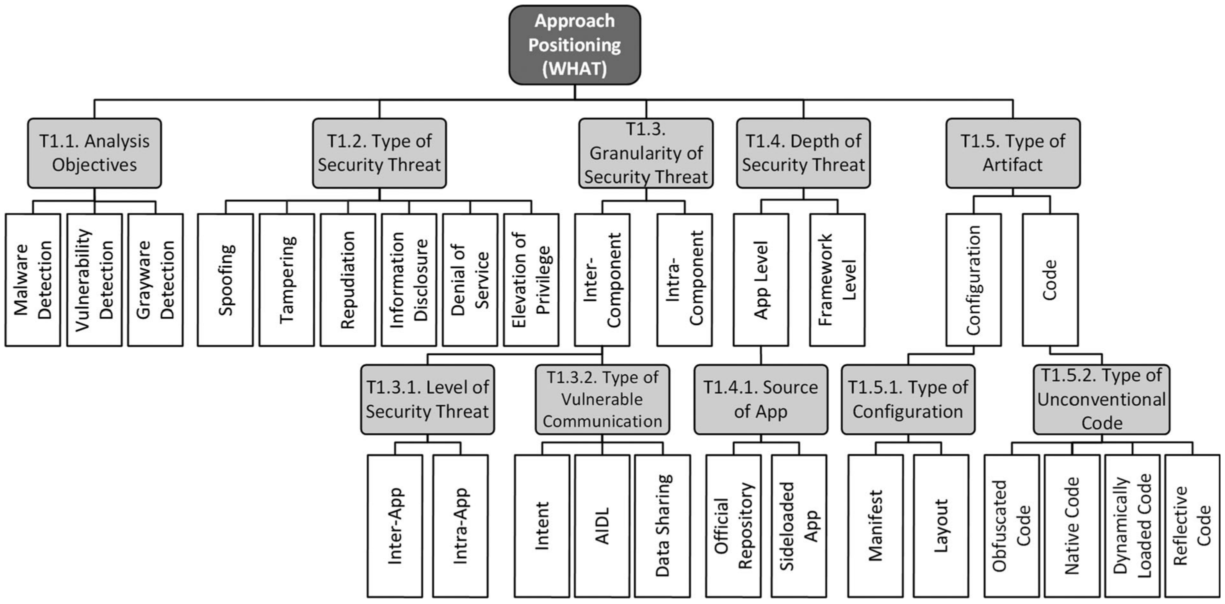

Fig. 6. Proposed taxonomy of android security analysis, problem category.

research area. In addition to detecting malware apps, identifying potential security threats posed by benign Android apps, that legitimately process user's private data (e.g., location information, IMEI, browsing history, installed apps, etc.), has also received a lot of attention in the area of Android security.

Since malware authors exploit the existing vulnerabilities of other apps or the underlying Android framework to breach system security, malware detection techniques and vulnerability identification methods are complementary to each other. In addition to these two kinds of approaches, there exists a third category of techniques intended to detect and mitigate the risk of *grayware*. Grayware, such as advertisement apps and libraries, are not fully malicious but they could violate users' privacy by collecting sensitive information for dubious purposes [20], [21], [143].

5.1.2 Type of Security Threats (T1.2)

This dimension classifies the security threats being addressed in the surveyed research along the Microsoft's threat model, called *STRIDE* [144].

Among existing attack models, we selected STRIDE, as it provides a design-centric model that helps us investigate the security properties of Android system, irrespective of known security attacks, thus allowing us to identify gaps in the literature (e.g., security attacks that have not been observed in Android yet, security attacks that have not received much attention in the literature). Moreover, it recognizes a separate category for each type of security property that is widely referred to in the literature.

Spoofing. Violates the *authentication* security property, where an adversary pretends to be a legitimate entity by properly altering some features that allows it to be recognized as a legitimate entity by the user. An example of this threat in the Android platform is *Intent Spoofing*, where a forged Intent is sent to an exported component, exposing the component to components from other applications (e.g., a malicious application) [3].

App Cloning, *Repackaging* or *Piggybacking* are classified under Spoofing, where malware authors attach malicious code to legitimate apps and advertise them as original apps in app markets to infect users. This technique is quite popular among mobile malware developers; it is used by 86 percent of the Android malware, according to a recent study [26].

Tampering. Affects the *integrity* property and involves a malicious modification of data. *Content Pollution* is an instance of this threat, where an app's internal database is manipulated by other apps [145].

Repudiation. Is in contrast to *non-repudiation* property, which refers to the situation in which entities deny their role or action in a transaction. An example of this security threat occurs when an application tries to hide its malicious behavior by manipulating log data to mislead a security assessment.

Information Disclosure. Compromises the *confidentiality* by releasing the protected or confidential data to an untrusted environment. In mobile devices, sensitive or private information such as device ID (IMEI), device location (GPS data), contact list, etc., might, intentionally or unintentionally, be leaked to an untrusted environment, via different channels as SMS, Internet, Bluetooth, etc.

Denial of Service. (DoS) affects *availability* by denying service to valid users. A common vulnerability in Android apps occurs when a payload of an Intent is used without checking against the null value, resulting in a *null dereference* exception to be thrown, possibly crashing the Android process in which it occurs. This kind of vulnerability has shown to be readily discoverable by an adversary through reverse engineering of the apps [146], which in turn enables launching a denial of service attack. Unauthorized Intent receipt [3], duplicating content provider authorities and permission names [147], battery exhaustion [148], and ransomware [149], [150], are some other examples of DoS attacks targeted at Android apps.

Elevation of Privilege. Subverts the *authorization* and happens when an unprivileged user gains privileged access. An example of the privilege escalation, which is shown to be quite common in the apps on the Android markets [151], happens when an application with less permissions (a non-privileged caller) is not restricted from accessing components of a more privileged application (a privileged callee) [4].

Over-privileged apps are particularly vulnerable to privilege escalation attack, due to the possibility of an attacker successfully injecting malicious code, exploiting the unnecessary permissions [152], [153]. Therefore, we categorize this type of security threat under elevation of privilege.

5.1.3 Granularity of Security Threats (T1.3)

This dimension classifies the approaches based on the granularity of identifiable security threats. In the basic form, a security issue, either vulnerability or malicious behavior, occurs by the execution of a single (vulnerable and/or malicious) component. However, more complicated scenarios are possible, where a security issue may arise from the interaction of multiple components. Accordingly, the existing techniques are classified into two categories: intra-component approaches that only consider security issues in a single component, and inter-component approaches that are able to identify security issues in multiple components. We further classify the inter-component class into subclasses based on two sub-dimensions described below.

Level of Security Threat (T1.3.1). It is possible that interacting vulnerable or malicious components belong to different applications. For example, in an instance of the *app collusion* attack, multiple applications can collude to compromise a security property, such as the user's privacy [4], [154]. Accordingly, security assessment techniques that consider the combination of apps in their analysis (i.e., inter-app) are able to reveal more complicated issues compared to non-compositional approaches (i.e., intra-app).

Type of Vulnerable Communication (T1.3.2). Android platform provides a variety of Inter-Process Communication (*IPC*) mechanisms for app components to communicate among each other, while achieving low levels of coupling. However, due to intrinsic differences with pure Java programming, such communication mechanisms could be easily misimplemented, leading to security issues. From a program analysis perspective, Android communication mechanisms need to be treated carefully, to avoid missing security issues. Our taxonomy showcases three major types of IPC mechanisms that may lead to vulnerable communication:

- As described in Section 2, Intents provide a flexible IPC model for communication among Android components. However, *Intents* are the root of many security vulnerabilities and malicious behaviors.
- *Android Interface Definition Language (AIDL)* is another IPC mechanism in Android that allows client-server RPC-based communication. The implementation of an AIDL interface must be thread-safe to prevent security issues resulting from concurrency problems (e.g., race conditions) [155].
- *Data Sharing* is another mechanism that allows app components to communicate with each other. Among the other methods, using Content Providers is the main technique for sharing data between two applications. However, misusage of such components may lead to security issues, such as passive content leaks (i.e., leaking private data), and content pollution (i.e., manipulating critical data) [145].

5.1.4 Depth of Security Threats (T1.4)

The depth of security threats category reflects if the approach addresses a problem at the application level or the framework level. The former aims at solely analyzing the application software. Third party apps, especially those from an unknown or untrustworthy provenance, pose a security challenge. However, there are some issues, such as overarching design flaws, that require system-wide reasoning, and are not easily attainable by simply analyzing individual parts of the system. Approaches at the framework level include research that focuses on modeling and analyzing the Android platform (e.g., for potential system-level design flaws and issues encountered in the underlying framework).

Source of App (T1.4.1). An application's level of security threat varies based on the source from which its installation package (i.e., apk file) is obtained. As a result, it is important to include a sub-dimension representing the source of the app in our taxonomy, which indicates whether the app is obtained from the official Android repository:

- *Official Repository:* Due to the continuous vetting of the official Android repository (i.e., Google Play), apps installed from that repository are safer than third-party apps.
- *Sideloaded App:* Sideloading, which refers to installing apps from sources other than the official Android repository, exposes a new attack surface for malware. Hence, it is critical for security research to expand their analysis beyond the existing apps in Google Play.

5.1.5 Type of Artifact (T1.5)

Android apps are realized by different kinds of software artifacts at different levels of abstraction, from high-level configuration files (e.g., *Manifest*) to low-level Java source code or native libraries implemented with C or C++. From the security perspective, each artifact captures some aspects essential for security analysis. For instance, while permissions are defined in the manifest file, inter-component messages (i.e., *Intents*) are implemented at the source code level. This dimension of the taxonomy indicates the abstraction level(s) of the extracted models that could lead to identification of a security vulnerability or malicious behavior.

Type of Configuration (T1.5.1). Among different configuration files contributing to the structure of Android app packages (APKs), a few artifacts encode significant security information, most notably, the *manifest* file that contains high-level information such as app components and permissions, as well as the *layout* file that defines the structure of app's user interfaces.

Type of Unconventional Code (T1.5.2). For different reasons, from legitimate to adversarial, developers may incorporate special types of code in their apps. A security assessment technique needs to tackle several challenges for analyzing such unconventional kinds of code. Thus, we further distinguish the approaches based on the special types of code they support, which includes the following:

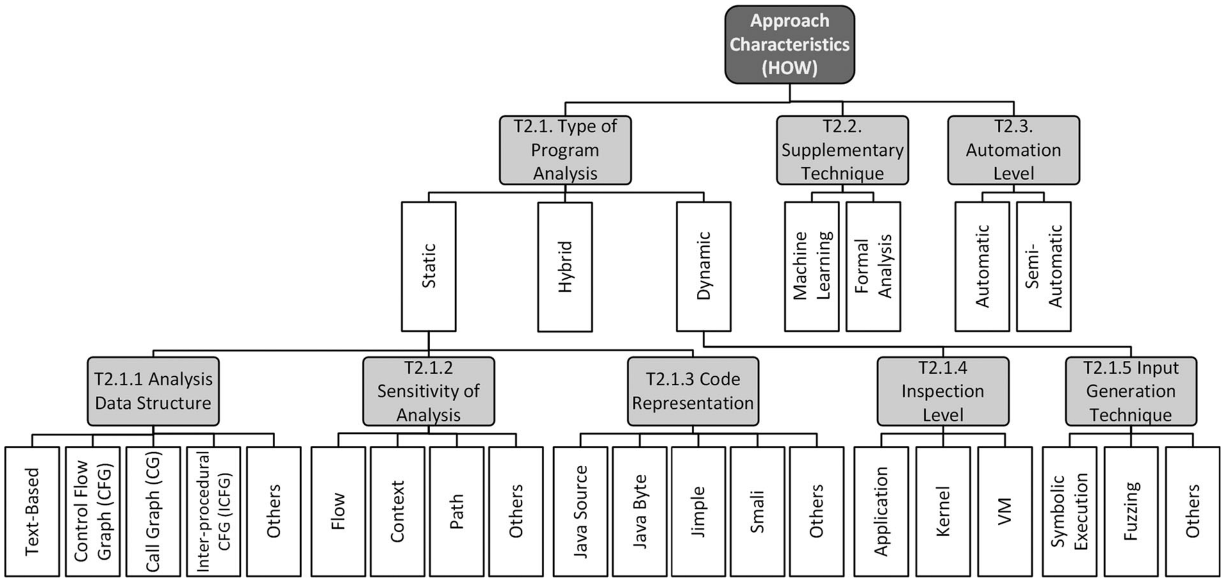

Fig. 7. Proposed taxonomy of android security analysis, solution category.

- *Obfuscated Code:* Benign app developers tend to obfuscate their application to protect the source code from being understood and/or reverse engineered by others. Malware app developers also use obfuscation techniques to hide malicious behaviors and avoid detection by antivirus products. Depending on the complexity of obfuscation, which varies from simple renaming to invoking behavior using reflection, security assessment approaches should tackle the challenges in analyzing the obfuscated apps [138], [156], [157], [158], [159], [160].
- *Native Code:* Beside Java code, Android apps may also consist of native C or C++ code, which is usually used for performance or portability requirements. An analysis designed for Java is not able to support these kinds of apps. To accurately and precisely analyze such apps, they need to be treated differently from non-native apps.
- *Dynamically Loaded Code:* Applications may dynamically load code that is not included in the original application package (i.e., apk file) loaded at installation time. This mechanism allows an app to be updated with new desirable features or fixes. Despite the benefits, this mechanism poses significant challenges to analysis techniques and tools, particularly static approaches, for assessing security threats of Android applications.
- *Reflective Code:* Using Java reflection allows apps to instantiate new objects and invoke methods by their names. If this mechanism is ignored or not handled carefully, it may cause incomplete and/or unsound static analysis. Supporting reflection is a challenging task for a static analysis tool, as it requires precise string and points-to analysis [161].

5.2 Approach Characteristics (Solution)

The second group of the taxonomy dimensions is concerned with classifying the "HOW" aspects of Android security analysis research. It includes three dimensions, as shown in Fig. 7.

5.2.1 Type of Program Analysis (T2.1)

This dimension classifies the surveyed research based on the type of program analysis employed for security assessment. The type of program analysis leveraged in security domain could be *static* or *dynamic*. Static analysis examines the program structure to reason about its potential behaviors. Dynamic analysis executes the program to observe its actual behaviors at runtime.

Each approach has its own strengths and weaknesses. While static analysis is considered to be conservative and sound, dynamic analysis is unsound yet precise [162]. Dynamic analysis requires a set of input data (including events, in event-based systems like Android) to run the application. Since the provided test cases are often likely to be incomplete, parts of the app's code, and thereby its behaviors, are not covered. This could lead to false negatives, i.e., missed vulnerabilities or malicious behaviors in security analysis. Moreover, it has been shown that dynamic approaches could be recognized and deceived by advanced malware, such as what anti-taint tracking techniques do to bypass dynamic taint analyses [163], [164], [165], [166], [167], [168], [169], [170].

On the other hand, by abstracting from the actual behavior of the software, static analysis could derive certain approximations about all possible behaviors of the software. Such an analysis is, however, susceptible to false positives, e.g., a warning that points to a vulnerability in the code which is not executable at runtime.

To better distinguish different approaches with respect to the program analysis techniques they rely on, we suggest sub-dimensions that further classify those two categories (i.e., static and dynamic analyses). Five sub-dimensions are presented below, where the first three (i.e., T2.1.1, T2.1.2, and T2.1.3) classify static analysis techniques and the next two (i.e., T2.1.4, and T2.1.5) are applied to dynamic analyses.

Analysis Data Structures (T2.1.1). In addition to lightweight static analyses that only employ *text-mining* techniques, heavyweight but more accurate static approaches usually leverage a few well-known data structures to abstract the underlying programs. The most frequently encountered data structures are as follows:

- *Control Flow Graph (CFG)* is a directed graph that represents program statements by its nodes, and the flow of control among the statements by the graph's edges.
- *Call Graph (CG)* is a directed graph, in which each node represents a method, and an edge indicates the call of (or return from) a method.
- *Inter-procedural Control Flow Graph (ICFG)* is a combination of CFG and CG that connects separated CFGs using call and return edges.

In addition, variation of these canonical data structures are used for special-purpose analyses. The goal of this dimension is to characterize the analysis based on the usage of these data structures.

Sensitivity of Analysis (T2.1.2). The sensitivities of the analyses vary for different algorithms used by a static analysis technique, leading to tradeoffs among analysis precision and scalability. Thus, this dimension classifies the static approaches based on their sensitivity to the following properties.

- *Flow Sensitive* techniques consider the order of statements and compute separate information for each statement.
- *Context Sensitive* approaches keep track of the calling context of a method call and compute separate information for different calls of the same procedure.
- *Path Sensitive* analyses take the execution path into account, and distinguish information obtained from different paths.

There also exist other levels of sensitivity, such as field- and object-sensitivity, which are discussed less often in the surveyed literature.

Code Representation (T2.1.3). Static analysis algorithms and methods are often implemented on top of off-the-shelf frameworks that perform the analysis on their own intermediate representation (IR) of program code. This dimension classifies the analysis tools based on the used IR (if any), which is translated from apps Dalvik bytecode prior to the analysis.

- *Java Source Code* may be analyzed since Android apps are mostly written in the Java language. This assumption, however, limits the applicability of the analysis to either open-source apps or the developers of an app.
- *Java Bytecode* may be analyzed, which widely broadens the applicability of an approach compared to the first group. Distinct from Java, Android has its own Dalvik bytecode format called Dex, which is executable by the Android virtual machine. As a result, this class of tools needs to retarget Dalvik to Java bytecode prior to the analysis, using APK-to-JAR transformers, such as dex2jar [171], ded [172], and its successor Dare[173].
- *Jimple* is a simplified version of Java bytecode that has a maximum of three components per statement. It is used by the popular static analysis framework Soot [174]. Dexpler [175] is a plugin for the Soot framework that translates Dalvik bytecode to Jimple.
- *Smali* is another intermediate representation, which is used by the popular Android reverse engineering tool, Apktool [176].

Inspection Level (T2.1.4). To capture dynamic behavior of Android apps, analysis techniques monitor the running apps at different levels. This dimension categorizes dynamic analyses based on their inspection level, including:

- *App-level* monitoring approaches trace Java method invocation by weaving the bytecode and injecting log statements inside the original app code or the Android framework. A few approaches achieve this in a more fine-grained manner through instruction-level dynamic analysis, such as data-flow tracking.
- *Kernel-level* monitoring techniques collect system calls, using kernel modules and features such as `strace`, or `ltrace`.
- *Virtual Machine (VM)-level* tools intercept events that occur within emulators. This group of approaches can support several versions of Android. The more recent work in this area supports the interception of Dalvik VM's successor, *Android Runtime* (ART) [177]. However, they are all prone to emulator evasion [28], [164], [178].

Input Generation Technique (T2.1.5). The techniques that employ dynamic analysis for security assessment need to run mobile applications in order to perform the analysis. For this purpose, they require test input data and events that trigger the application under experiment. Security testing is, however, a notoriously difficult task. This is in part because unlike functional testing that aims to show a software system complies with its specification, security testing is a form of negative testing, i.e., showing that a certain (often *a priori* unknown) behavior does not exist.

In addition to manually providing the inputs, which is neither systematic nor scalable, two approaches are often leveraged by the surveyed research: fuzzing and symbolic execution.

- *Fuzz testing or fuzzing* [179] executes the app with random input data. Running apps using inputs generated by Monkey [180], the state-of-the-practice tool for the Android system testing, is an example of fuzz testing.
- *Symbolic execution* [181] uses symbolic values, rather than actual values, as program inputs. It gathers the constraints on those values along each path of the program and with the help of a solver generates inputs for all reachable paths.

5.2.2 Supplementary Techniques (T2.2)

Besides various program analysis techniques, which are the key elements employed by approaches in the surveyed research, other supplementary techniques have also been leveraged to complement the analysis. Among the surveyed research, *Machine Learning* and *Formal Analysis* are the most widely used techniques. In fact, the program analysis either provides the input for, or consumes the output of, the other supplementary techniques. This dimension of the taxonomy determines the techniques other than program analysis (if any) that are employed in the surveyed research.

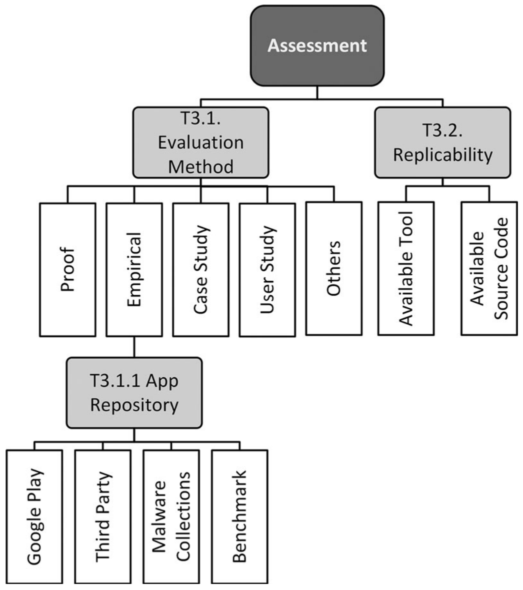

Fig. 8. Proposed taxonomy of android security analysis, assessment category.

5.2.3 Automation Level (T2.3)

The automation level of a security analysis method also directly affects the usability of such techniques. Hence, we characterize the surveyed research with respect to the manual efforts required for applying the proposed techniques. According to this dimension, existing techniques are classified as either automatic or semi-automatic.

5.3 Assessment (Validation)

The third and last section of the taxonomy is about the evaluation of Android security research. Dimensions in this group, depicted in Fig. 8, provide the means to assess the quality of research efforts included in the survey.

The first dimension, evaluation method, captures how, i.e., with which evaluation method, a paper validates the effectiveness of the proposed approach, such as empirical experimentation, formal proof, case studies, user studies, or other methods. Moreover, we further classify the empirical evaluations according to the source of apps they selected for the experiments, including the official Google Play repository, third-party and local repositories, collections of malware, and benchmark apps handcrafted by research groups for the purpose of evaluation.

The other dimension captures the extent to which surveyed research efforts enable a third party to reproduce the results reported by the authors. This dimension classifies replicability of research approaches by considering the availability of research artifacts. For example, whether the approach's underlying platform, tools and/or case studies are publicly available.

6 Survey Results and Analysis

This section presents the results of our literature review to answer the second research question. By using the proposed taxonomy as a consistent point of reference, many insightful observations surface from the survey results. The number of the research papers surveyed will not allow elaboration on each one of them. Rather, we highlight some of them as examples in the observations and analyses below.[6]

6.1 Approach Positioning (Problem)

Tables 3 and 4 provide a summary of the problem-specific aspects that are extracted from our collection of papers included in the survey. Note that the classifications are meant to indicate the primary focus of a research paper. For example, if a certain approach is not mentioned in the *Spoofing* column under the *Type of Security Threat*, it does not necessarily indicate that it absolutely cannot mitigate such threat. Rather, it simply means spoofing is not its primary focus. Furthermore, for some taxonomy categories, such as *Depth of Threat*, a paper may have multiple goals and thus listed several times. On the other hand, several dimensions only apply to a subset of papers surveyed, e.g., *Test Input Generation* only applies to dynamic or hybrid approaches. As a result, percentages presented in the last column of the table may sum up to more or less than 100 percent. In the following, we present the main results for each dimension in the problem category.

6.1.1 Analysis Objective

Security assessment techniques proposed by a number of previous studies could be directly used or extended for various purposes (e.g., detection of malware, grayware, or vulnerabilities). In this survey, to distinguish the main objective(s) of each approach, we consulted the threat model (or adversary model) and also the evaluation goals and results (if any) described in the surveyed papers.

Based on the analysis of the research studies in the literature, it is evident that the majority of Android security approaches have been applied to detection of malicious behaviors, comprising 61 percent of the overall set of papers collected for this literature review. However, sometimes the analysis techniques are not able to determine unequivocally if an application is malicious or benign. Therefore, a number of studied approaches [182], [403], [404], [405], [409], [474] use risk-based analysis to assign each app a level of security risk according to the analysis results (Denoted by *R* in Table 3).

Four percent of efforts in this area are devoted to the analysis of grayware that are less disruptive than malware, but still worrying, particularly from a privacy perspective. Most research efforts on grayware detection target the analysis of advertisement (ad) libraries that are linked and shipped together with the host apps. In fact, a variety of private user data, including a user's call logs, phone numbers, browser bookmarks, and the list of apps installed on a device are collected by ad libraries. Since the required permissions of ad libraries are merged into a hosting app's permissions, it is challenging for users to distinguish, at installation time, the permissions requested by the embedded ad libraries from those actually used by the app [77]. For this reason, *AdRisk* [436] decouples the embedded ad libraries from the host apps and examines the potential unsafe behavior of each library that could result in privacy issues. Other techniques, such as *AdDroid* [77], *AFrame* [477], *AdSplit* [450], and *LayerCake* [439], introduce advertising frameworks with dedicated

6. Throughout this survey (including tables and figures), the approaches without name are shown in the form of "first author's surname_".

TABLE 3
Problem Specific Categorization of the Reviewed Research, Part 1

Dimension		Approaches	%
Analysis Objective	Vulnerability Detection	ADDICTED [508], Amandroid [440], ApkCombiner [256], App-ray [419], AppAudit [451], AppCaulk [380], AppCracker [82], AppFence [207], AppGuard [42], AppProfiler [355], AppSealer [482], Aquifer [305], ASM [202], AuthDroid [432], Bagheri_ [44], Bartel_ [57], Bartsch_ [58], Bifocals [105], Buhov_ [80], Buzzer [88], CMA [389], CoChecker [115], ComDroid [104], ConDroid [379], ContentScope [510], Cooley_ [108], COPES [55], COVERT [45], COVERT_Tool [359], CredMiner [514], CRePE [107], CryptoLint [134], Desnos_ [125], DexDiff [296], DroidAlarm [502], DroidChecker [94], DroidCIA [101], DroidGuard [46], DroidRay [500], Droidsearch [339], Enck_ [142], Epicc [315], FineDroid [488], Flowdroid [35], Gallo_ [171], Geneiatakis_ [175], Grab'nRun [147], Harehunter [15], HornDroid [83], IccTA [257], IPCInspection [161], IVDroid [148], Juxtapp [196], Kantola_ [235], KLD [387], Lintent [79], Lu_ [277], MalloDroid [146], Matsumoto_ [293], Mutchler_ [301], NoFrak [176], NoInjection [226], Onwuzurike_ [318], PaddyFrog [448], PatchDroid [299], PCLeaks [258], PermCheckTool [427], PermissionFlow [372], Poeplau_ [328], PScout [37], QUIRE [127], Ren_ [349], SADroid [195], SCanDroid [168], Scoria [423], SecUP [458], SEFA [449], Smith_ [400], SMV-HUNTER [403], STAMBA [71], Stowaway [157], SUPOR [214], TongxinLi_ [262], Vecchiato_ [424], VetDroid [489], WeChecker [114], Woodpecker [189], Zuo_ [517]	26%
	Malware Detection (R)	A3 [280], A5 [428], AAPL [275], AASandbox [69], Adagio [172], Adebayo_ [19], Afonso_ [20], Amandroid [440], AMDetector [492], Anadroid [265], Ananas [133], AnDarwin [112], AndroidLeaks [178], Andrubis [268], AntiMalDroid [491], Apex [308], ApkCombiner [256], ApkRiskAnalyzer [103]^{R}, App-ray [419], AppAudit [451], AppContext [470], AppFence [207], AppInspector [179], AppIntegrity [425], AppIntent [473], Apposcopy [162], AppProfiler [355], AppsPlayground [342], Aquifer [305], AsDroid [215], ASM [202], AuDroid [326], Aurasium [460], AutoCog [335], AVDTester [211], Bae_ [43], Bal_ [49], Batyuk_ [59], Bianchi_ [65], BlueSeal [206], Brave [321], Brox [284], Canfora_ [85], Canfora3_ [87], Capper [483], Cassandra [274], Cen_ [91], Chabada [184], Chen_ [97], Chen2_ [96], CopperDroid [347], CopperDroid2 [414], COVERT [45], COVERT_Tool [359], Crowdroid [81], Dagger [466], Dai_ [117], DataChest [512], DeepDroid [437], Defensor [319], Dendroid [407], Desnos_ [125], DidFail [246], DNADroid [111], DRACO [63], Drebin [32], DroidADDMiner [264], DroidAnalyst [169], DroidAnalytics [499], DroidAnalyzer [382], DroidAPIMiner [14], DroidBarrier [24], DroidDolphin [450], DroidGuard [46], DroidKin [182], DroidLegacy [124], DroidMat [447], DroidMiner [465], DroidMOSS [507], DroidPAD [279], DroidPermissionMiner [36], DroidRanger [513], DroidRay [500], DroidRisk [438]^{R}, DroidSafe [183], DroidScope [464], Droidsearch [339], DroidSIFT [481], DroidSim [412], DroidTrace [501], DroidTrack [362], Duet [208], EASEAndroid [435], Elish_ [138], Enck_ [142], Fedler_ [153], Fest [490], FineDroid [488], FireDroid [358], FlaskDroid [78], Flowdroid [35], FUSE [346], Gates_ [174]^{R}, Graa_ [186], Graa2_ [185], GroddDroid [16], Ham_ [192], HornDroid [83], Huang_ [210], HunterDroid [479], I-ARM-Droid [122], ICC_Map [137], IccTA [257], IFT [145], IIF [477], IREA [249], Isohara_ [217], Jeon_ [219], Jeong_ [222], Jiao_ [224], Juxtapp [196], Kadir_ [234], Karami_ [236], Kate_ [237], Kim_ [242], Kirin [143], LeakMiner [472], Lee_ [252], Li_ [260], Li2_ [261], Ma_ [283], MADAM [128], Mama [366], Manilyzer [156], Mann_ [289], Marvin [267], MassVet [98], MAST [93], Masud_ [292], MIGDroid [209], Milosevic_ [295], Mobile-Sandbox [404], MobSafe [459], Moonsamy_ [297], Moonsamy2_ [298], Morbs [434], Mudflow [38], MysteryChecker [221], NDroid [333], Nishimoto_ [311], OpSeq [22], Paranoid-Android [330], Patronus [411], pBMDS [455], PCLeaks [258], Pegasus [99], Peiravian_ [324], Peng_ [325]^{R}, Permlyzer [461], PICARD [129], Poeplau_ [328], PREC [203], PUMA2 [198], PuppetDroid [177], Quan_ [336], RAMSES [132], Relda [190], ResDroid [390], Riskmon [227]^{R}, RiskMon2 [228]^{R}, Riskranker [187]^{R}, SAAF [205], Sahs_ [361], Sanz_ [368], Sarma_ [369]^{R}, Sayfullina_ [371], ScanDal [243], SCanDroid [168], Schmidt_ [374], SCSdroid [266], SecUP [458], SFG [27], Shabtai_ [383], Shebaro_ [391], Shen_ [395], SherlockDroid [29], SmartDroid [497], Smith_ [400], Song_ [401], StaDynA [494], Su_ [406], TaintDroid [141], Tchakounte_ [415], TMSVM [453], TraceDroid [422], TreeDroid [119], UID [139], VetDroid [489], ViewDroid [478], Wang_ [436]^{R}, Wognsen_ [445], WuKong [431], Yaase [357], Yerima_ [475]	61%
	Grayware Detection	Achara_ [18], AdDroid [323], AdRisk [188], AndroidLeaks [178], APKLancet [471], AppFence [207], AppProfiler [355], AppsPlayground [342], Han_ [194], LayerCake [352], Leontiadis_ [254], Pedal [270], Seneviratne_ [381], Short_ [398], Wijesekera_ [444]	4%
Type of Security Threat	Spoofing (Cl|Cr)	Amandroid [440], AnDarwin [112]^{Cl}, AppCracker [82]^{Cr}, AppIntegrity [425]^{Cl}, AuthDroid [432]^{Cr}, Bianchi_ [65], Bifocals [105], Buhov_ [80]^{Cr}, Chen_ [97]^{Cl}, ComDroid [104], Compac [439], Cooley_ [108], CredMiner [514]^{Cr}, CryptoLint [134]^{Cl}, DNADroid [111]^{Cl}, DroidCIA [101], DroidKin [182]^{Cl}, DroidMOSS [507]^{Cl}, DroidSim [412]^{cl}, Epicc [315], Gallingani_ [170], HunterDroid [479]^{Cl}, Juxtapp [196]^{Cl}, Kantola_ [235], MalloDroid [146]^{Cr}, MIGDroid [209]^{Cl}, Mutchler_ [301], MysteryChecker [221]^{Cl}, NoFrak [176], NoInjection [226], Onwuzurike_ [318]^{Cr}, PCLeaks [258], PermissionFlow [372], PICARD [129]^{Cl}, Ren_ [349], ResDroid [390]^{Cl}, SCSdroid [266]^{Cl}, SMV-HUNTER [403]^{Cr}, STAMBA [71], WuKong [431]^{Cl}, Zhou_ [506]^{Cl}, Zuo_ [517]^{Cr}	13%
	Tampering	AppCracker [82], AppIntegrity [425], APSET [364], CMA [389], ContentScope [510], Desnos_ [125], DroidCIA [101], DroidFuzzer [474], Harehunter [15], MalloDroid [146], SMV-HUNTER [403], STAMBA [71]	4%
	Repudiation		0%
	Information Disclosure	Achara_ [18], AdDroid [323], AdRisk [188], AdSplit [393], Amandroid [440], AMDetector [492], Ananas [133], AndroidLeaks [178], ApkCombiner [256], APKLancet [471], AppAudit [451], AppCaulk [380], AppFence [207], AppGuard [42], AppInspector [179], AppIntent [473], Apposcopy [162], AppProfiler [355], AppSealer [482], AppsPlayground [342], AsDroid [215], AuDroid [326], Aurasium [460], AutoCog [335], Bagheri_ [44], Bal_ [49], Barbon_ [50], Barros_ [54], Bartsch_ [58], Batyuk_ [59], BayesDroid [420], Berthome_ [62], BlueSeal [206], Brahmastra [64], Brox [284], Capper [483], Cassandra [274], CHEX [276], CMA [389], CoChecker [115], ComDroid [104], ContentScope [510], ConUCON [48], CopperDroid [347], CopperDroid2 [414], COVERT [45], COVERT_Tool [359], DataChest [512], Defensor [319], DexDiff [296], DroidGuard [46], DroidPAD [279], DroidSafe [183], DroidTest [356], DroidTrack [362], Enck_ [142], Epicc [315], Feth_ [164], FineDroid [488], FlaskDroid [78], Flowdroid [35], Graa_ [186], Graa2_ [185], Han_ [194], Harehunter [15], HornDroid [83], ICC_Map [137], IccTA [257], IFT [145], IIF [477], Jia_ [223], KLD [387], Kynoid [377], LazyTainter [442], LeakMiner [472], Lee_ [252], Leontiadis_ [254], Mann_ [289], Matsumoto_ [293], Mobile-Sandbox [404], MobSafe [459], MockDroid [61], MonkeyDroid [282], Morbs [434], MorphDroid [163], MOSES [496], Mudflow [38], Mutchler_ [301], NDroid [333], Nishimoto_ [311], Onwuzurike_ [318], Paupore_ [322], PCLeaks [258], Pegasus [99], Porscha [316], Relda [190], Ren_ [349], SADroid [195], ScanDal [243], SecUP [458], SEFA [449], Seneviratne_ [381], SFG [27], Short_ [398], SmartDroid [497], Smith_ [400], Song_ [401], STAMBA [71], SUPOR [214], TaintDroid [141], TISSA [515], TouchDevelop [454], TrustDroid [493], Uranine [345], VetDroid [489], WeChecker [114], WifiLeaks [17], Wijesekera_ [444], Yaase [357]	35%
	Denial of Service	ASV [212], ComDroid [104], Enck_ [142], Ren_ [349], SecUP [458]	1%
	Elevation of Privilege (O)	ADDICTED [508], AdDroid [323], Ananas [133], AppGuard [42], AppInspector [179], AppsPlayground [342], Aurasium [460], Bagheri_ [44], Bartel_ [57]^{O}, Bartsch_ [58], Bugiel_ [76], CHEX [276], CoChecker [115], ComDroid [104], Compac [439], COPES [55]^{O}, COVERT [45], COVERT_Tool [359], DeepDroid [437], Defensor [319], DroidAlarm [502], DroidBarrier [24], DroidChecker [94], DroidGuard [46], DroidRay [500], DroidTrace [501], Enck_ [142], Epicc [315], FineDroid [488], FlaskDroid [78], FUSE [346], Gallo_ [171], Geneiatakis_ [175], Harehunter [15], IntentFuzzer [467], IPCInspection [161], Johnson_ [232]^{O}, Lee_ [252], Lintent [79], Lu_ [277], MobSafe [459], PaddyFrog [448], PCLeaks [258], Pedal [270], Pegasus [99], PermCheckTool [427]^{O}, PermissionFlow [372], PREC [203], PScout [37]^{O}, QUIRE [127], SADroid [195], SecUP [458], SEFA [449]^{O}, Stowaway [157]^{O}, UID [139], WeChecker [114], Woodpecker [189], Yaase [357]	17%

***R**: Risk-based, **Cl**: Cloning, Repackaging, or Piggybacking, **Cr**: Cryptography Misuse, **O**: Over-privilege apps.*

permissions and APIs that separate privileged advertising functionality from host applications. Also, as a more generic solution, Compac [445] provides fine-grained access control to minimize the privilege of all third-party components.

Android vulnerability analysis has also received attention from a significant portion of existing research efforts (26 percent of the studied papers). Since techniques and methods used for one of the above goals are often applicable to other goals, the target of many surveyed research papers falls in both categories. However, there are some approaches that only target vulnerability detection. Among such approaches, *Woodpecker* [151] tries to identify vulnerabilities in the standard configurations of Android smartphones, i.e., pre-loaded apps in such devices, that may lead to *capability leaks*. A capability (or permission) leak is an instance of a privilege-escalation threat, where some privileged functions (e.g., sending of a text message) is left exposed to apps lacking the required permissions to access those functions.

6.1.2 Type of Security Threat

The Android security approaches studied in this literature review have covered diverse types of security threats. It can be observed from Table 3 that among the *STRIDE* security threats (cf. Section 5.1.2), information disclosure is the most considered threat in Android, comprising 35 percent of the papers. This is not a surprising result, since mobile devices are particularly vulnerable to data leakage [478]. Elevation of privilege (including over-privilege issue marked as *O* in Table 3) is the second class of threats addressed by 17 percent of the overall studied papers. Examples of this class of threats, such as *confused deputy vulnerability* [479], are shown to be quite common in the Android apps on the market [4], [12], [153].

Spoofing has received substantial attention (13 percent), particularly because Android's flexible Intent routing model can be abused in multiple ways, resulting in numerous possible attacks, including *Broadcast injection* and *Activity/Service launch* [3]. Cloning or repackaging, which is a kind of

TABLE 4
Problem Specific Categorization of the Reviewed Research, Part 2

Dimension			Approaches	%
Granularity of Threat	Intra-Comp.		Others *	79%
	Inter-Comp.	Intent	Amandroid [440], Apex [308], ApkCombiner [256], AppAudit [451], AppCaulk [380], AppContext [470], AppIntent [473], Apposcopy [162], APSET [364], AsDroid [215], Bal_ [49], Barros_ [54], Bartsch_ [58], BlueSeal [206], Brahmastra [64], CoChecker [115], ContentScope [510], ConUCON [48], COVERT [45], COVERT_Tool [359], DataChest [512], DidFail [246], DroidAlarm [502], DroidAPIMiner [14], DroidForce [340], DroidGuard [46], DroidSafe [183], DroidSIFT [481], Epicc [315], Feth_ [164], FineDroid [488], FUSE [346], Gallingani_ [170], Han_ [194], HornDroid [83], ICC_Map [137], IccTA [257], IFT [145], IntentFuzzer [467], IPCInspection [161], IVDroid [148], Jeong_ [222], Kantola_ [235], Lintent [79], Morbs [434], Mutchler_ [301], PaddyFrog [448], PCLeaks [258], PermissionFlow [372], QUIRE [127], Ren_ [349], SADroid [195], Shen_ [395], SmartDroid [497], UID [139], WeChecker [114], Woodpecker [189], Xmandroid [75], Zhou_ [506]	18%
		AIDL	ASV [212], BlueSeal [206], CopperDroid [347], CopperDroid2 [414], DataChest [512], Morbs [434], QUIRE [127], Woodpecker [189]	2%
		SharedData	ContentScope [510], Harehunter [15], KLD [387]	1%
	Inter-App		Amandroid [440], ApkCombiner [256], Bal_ [49], Barros_ [54], Bartsch_ [58], Brahmastra [64], COVERT [45], COVERT_Tool [359], DataChest [512], DidFail [246], DroidForce [340], DroidGuard [46], DroidTrack [362], FineDroid [488], FUSE [346], Harehunter [15], IccTA [257], IntentFuzzer [467], IPCInspection [161], Jia_ [223], Morbs [434], PCLeaks [258], PermissionFlow [372], QUIRE [127], SEFA [449], WeChecker [114], Xmandroid [75]	8%
Depth	App Level		Others **	88%
	Framework Level (K)		ADDICTED [508], AdDroid [323], AntiMalDroid [491], Apex [308], Bagheri_ [44], Bartel_ [57], Bifocals [105], Bugiel_ [76], Buzzer [88], COPES [55], CRePE [107], DataChest [512], DexDiff [296], Dr.Android [220], DroidBarrier [24], DroidRay [500], DroidSafe [183], Feth_ [164], FineDroid [488], Gallo_ [171]^{K}, IFT [145], Jung_ [233], Kantola_ [235], Kynoid [377], Morbs [434], MpDroid [417], Nishimoto_ [311], NoFrak [176], PatchDroid [299], Porscha [316], PScout [37], SecUP [458]^{K}, Shebaro_ [391], Smith_ [400], Stowaway [157], TongxinLi_ [262], Vecchiato_ [424], VetDroid [489], WifiLeaks [17]	12%
Type of Artifact	Config.	Manifest	A5 [428], AdDroid [323], AdRisk [188], Amandroid [440], Ananas [133], Androguard [126], Andrubis [268], ApkCombiner [256], APKLancet [471], App-ray [419], AppAudit [451], AppContext [470], AppGuard [42], Apposcopy [162], AppProfiler [355], APSET [364], Aquifer [305], AsDroid [215], AuthDroid [432], AutoCog [335], AVDTester [211], Bagheri_ [44], Barrera2_ [52], Batyuk_ [59], Bianchi_ [65], BlueSeal [206], Brahmastra [64], Capper [483], Cen_ [91], CoChecker [115], ComDroid [104], Compac [439], ContentScope [510], COPES [55], COVERT [45], COVERT_Tool [359], Dai_ [117], DiCerbo_ [92], DidFail [246], Dr.Android [220], Drebin [32], DroidAlarm [502], DroidAnalytics [499], DroidAPIMiner [14], DroidChecker [94], DroidForce [340], DroidFuzzer [474], DroidGuard [46], DroidMat [447], DroidPermissionMiner [36], DroidRanger [513], DroidRay [500], DroidSafe [183], Droidsearch [339], Duet [208], Epicc [315], Flowdroid [35], FUSE [346], Gates_ [174], Geneiatakis_ [175], Harehunter [15], Huang_ [210], ICC_Map [137], IccTA [257], IFT [145], IVDroid [148], Johnson_ [232], Kantola_ [235], Kate_ [237], Kim_ [242], Kirin [143], LeakMiner [472], Lee_ [252], Leontiadis_ [254], Lintent [79], Lu_ [277], Ma_ [283], Malek_ [288], MalloDroid [146], Mama [366], Manilyzer [156], Mann_ [289], Marvin [267], MassVet [98], MAST [93], Matsumoto_ [293], Mobile-Sandbox [404], Moonsamy_ [297], Moonsamy2_ [298], Mudflow [38], Mutchler_ [301], PaddyFrog [448], PCLeaks [258], Pegasus [99], Peiravian_ [324], PermCheckTool [427], PermissionFlow [372], Permlyzer [461], ProfileDroid [441], PUMA2 [198], Relda [190], ResDroid [390], Riskranker [187], SAAF [205], SADroid [195], Sahs_ [361], Sanz_ [368], Sarma_ [369], Sayfullina_ [371], SCanDroid [168], SEFA [449], Shen_ [395], SherlockDroid [29], Short_ [398], SmartDroid [497], Smith_ [400], SMV-HUNTER [403], StaDynA [494], TMSVM [453], TraceDroid [422], TrustDroid [493], UID [139], Wang_ [436], WeChecker [114], Woodpecker [189], Yerima_ [475], Zhou_ [506], Zuo_ [517]	38%
		Layout	Amandroid [440], AsDroid [215], Bianchi_ [65], BlueSeal [206], Brahmastra [64], DataChest [512], Flowdroid [35], FUSE [346], IccTA [257], MassVet [98], Permlyzer [461], ResDroid [390], SUPOR [214], UIPicker [306], WeChecker [114]	4%
	Code (P)	Obfuscated	AnDarwin [112]^{P}, Apposcopy [162], CredMiner [514], Dendroid [407]^{P}, Desnos_ [125]^{P}, DNADroid [111]^{P}, DroidKin [182]^{P}, DroidSIFT [481], DroidSim [412], Graa_ [186], Graa2_ [185]^{P}, IREA [249], Juxtapp [196]^{P}, MassVet [98], OpSeq [22], Pedal [270], ResDroid [390], Shen_ [395], ViewDroid [478]	6%
		Native	Compac [439]^{P}, CopperDroid [347], CopperDroid2 [414], Dagger [466], DeepDroid [437], DroidRanger [513]^{P}, DroidScope [464], FireDroid [358], Flowdroid [35]^{P}, MAST [93]^{P}, Mobile-Sandbox [404]^{P}, NDroid [333], PatchDroid [299], Poeplau_ [328]^{P}, RetroSkeleton [121]^{P}, Riskranker [187]^{P}, VetDroid [489]	5%
		Dynamic	AdRisk [188]^{P}, AppContext [470]^{P}, AppsPlayground [342], ConDroid [379], DroidAPIMiner [14], DroidRanger [513]^{P}, DroidTrace [501], Grab'nRun [147], Poeplau_ [328]^{P}, Riskranker [187]^{P}, StaDynA [494], Yerima_ [475]	4%
		Reflective	AdRisk [188]^{P}, AppAudit [451]^{E}, AppContext [470]^{P}, AppGuard [42], AppsPlayground [342], Barros_ [54], DroidSafe [183]^{P}, DroidSIFT [481], FUSE [346]^{P}, HornDroid [83]^{P}, IFT [145], IIF [477], IREA [249], Pegasus [99], RetroSkeleton [121], Riskranker [187]^{P}, SAAF [205]^{P}, ScanDal [243]^{P}, StaDynA [494], TaintDroid [141], VetDroid [489], Wognsen_ [445]	7%

**: Including all other surveyed papers that are not mentioned as Inter-Comp or Inter-App.*
***: Including all other surveyed papers that are not mentioned as Framework Level.*
***P**: Partial coverage (usually adopting conservative approach and marking all instances of special code as dangerous/suspicious).*
***K** exclusively at Kernel-level.*

spoofing threat, is a common security issue in Android app markets, and hence is addressed by several techniques, including [270], [278], [312], [326]. Note that these techniques are marked as *Cl* in Table 3. Moreover, misusing cryptography techniques, such as failure in the SSL/TLS validation process, might result in *man in the middle* attacks that violate system authentication. Thus, we categorized the techniques attempting to identify cryptography misuse, such as [232], [252], under spoofing. We distinguished these techniques by label *Cr* in Table 3.

Tampering and denial of service issues are also considered in the literature, comprising 4 and 1 percent of the papers, respectively. Among the STRIDE's threats, repudiation is not explicitly studied in the surveyed research. We will revisit this gap in Section 7.

6.1.3 Granularity of Threat

We can observe from Table 4 that the majority of the Android security approaches are intended to detect and mitigate security issues in a single component, comprising 79 percent of the overall papers studied in this literature review, while a comparatively low number of approaches (21 percent) have been applied to inter-component analysis.

The compositional approaches take into account inter-component and/or inter-app communication during the analysis to identify a broader range of security threats that cannot be detected by techniques that analyze a single component in isolation. Among others, *IccTA* [225], [490] performs data leak analysis over a bundle of apps. It first merges multiple apps into a single app, which enables context propagation among components in different apps, and thereby facilitates a precise inter-component taint analysis.

The main challenge with such approaches for compositional analysis is the scalability issue. Because as the number of apps increases, the cost of program analysis grows exponentially. To address the scalability issue intrinsic to compositional analysis, some hybrid approaches are more recently proposed that combine program analysis with other reasoning techniques [205], [280], [491]. For example, COVERT [205], [206] combines static analysis with lightweight formal methods. Through static analysis of each individual app, it first extracts relevant security specifications in an analyzable formal specification language (i.e., Alloy). These app specifications are then combined together and checked as a whole with the aid of a SAT solver for inter-app vulnerabilities.

Intent is the main inter-component communication mechanism in Android and thus, it has been studied and focused more than other ICC mechanism (18 percent compared to 2 and 1 percent). *Epicc* [218] and its successor *IC3* [98] try to precisely infer Intent values, which are necessary information for identifying vulnerable communications. *BlueSeal* [291] and *Woodpecker* [151] briefly discuss AIDL, as another ICC mechanism, and how to incorporate it in control flow graph. Finally, *ContentScope* [145] examines the security threats of using shared data as the third way of achieving ICC.

6.1.4 Depth of Threat

We observe that most approaches perform the analysis at the application-level (88 percent), but about ten percent of the approaches consider the underlying Android framework for analysis (12 percent). The results of analyses carried out at the framework-level are also beneficial in analysis of individual apps, or even revealing the root causes of the vulnerabilities found at the application-level. For example, *PScout* [244] and *Stowaway* [153], through the analysis of the Android framework, obtained permission-API mappings that specify the permissions required to invoke each Android API call. However, due to intrinsic limitations of static and dynamic analyses adopted by PScout and Stowaway, respectively, the generated mappings are incomplete or inaccurate. Addressing this shortcoming, more recent approaches [152], [257] have attempted to enrich the extracted permission mappings. Such permission mappings have then been used by many other approaches, among others, for detecting *over-privileged* apps that violate the "Principle of Least Privilege" [492] (cf. Section 5.1.2).

Among the approaches performing analysis at the framework level, some look into the vulnerabilities of the Android framework that could lead to security breaches of the system, such as design flaws in the permission model [195], security hazards in push-messaging services [255], or security vulnerabilities of the WebView component [197], [235], [236].

Apps installed from arbitrary sources pose a higher security risk than apps downloaded from Google Play. However, regardless of the source of the app, it must be installed using the same mechanism for importing the app's code into the Android platform, i.e., by installing APK files. Nevertheless, to measure the effectiveness of a technique for identifying security threats, researchers need to evaluate the proposed technique using both Google Play and sideloaded apps. We discuss, in detail, the sources of apps used to evaluate Android security analysis techniques in Section 6.3.1.

6.1.5 Type of Artifact

As discussed in Section 5, Android apps are composed of several artifacts at different levels of abstraction, such as high-level configuration files and code implementation. We can observe from Table 4 that most of the studied approaches analyze multiple types of artifacts.

Type of Configuration. Manifest is an XML configuration file, shipped with all Android apps, and includes some high-level architectural information, such as the apps' components, their types, permissions they require, etc. Since a large portion of security-related information are encoded in the apps' manifest files (e.g., required or defined permissions), some techniques only focus on the analysis of this file. Kirin [364], for instance, is among the techniques that only performs the analysis on the app manifest files. By extracting the requested permissions defined in the manifest file and comparing their combination against a set of high-level, blacklist security rules, Kirin is able to identify the apps with potential dangerous functionality, such as information leakage. However, the security policies in Kirin, or similar techniques that are limited to the abstract level of configuration files, may increase the rate of false warnings. For instance, a Kirin's security rule, for mitigating mobile bots that send SMS spam, is stated as *"An application must not have SEND_SMS and WRITE_SMS permission labels [364]"*. As a result, an application requesting these two permissions is flagged as malware, even if there are no data-flow between the parts of code corresponding to these two permissions.

In addition to the manifest file, there are some other resources in the Android application package (a.k.a., *apk* file) that also do not require complicated techniques to be analyzed. One example is the layout file that represents the user interface structure of the apps in an xml format. The layout file can be parsed, among other things, to identify the callback methods registered for GUI widget, which in turn improves the precision of generated call graphs. *CHEX* [456] and *BlueSeal* [291], [493] are among the techniques that leverage layout files for this purpose.

Moreover, the layout file contains information that is critical for security analysis. Password fields, which usually contain sensitive data, are an example of security-critical information embedded in layout files [104]. An example of a technique that leverages this information is *AsDroid* [282]. It examines the layout file to detect stealthy malicious behavior through identifying any contradiction between the actual app behavior and the user interface text initiating that behavior (e.g., the name of a button that was clicked), which denotes the user's expectation of program behavior. Another example is *MassVet* [374] that captures the user interface of apps by encoding layouts in a graph structure called a view graph and then detects repackaged malware by calculating the similarity of view graphs.

Besides manifest and layout files, a few other types of configuration files are processed by a number of analyses. For instance, string resources (i.e., String.xml) are parsed to capture predefined URL strings [234] and to identify the label of sensitive fields [488], or style definition files, among other resources, are leveraged to detect repackaged malware apps [402].

Type of Unconventional Code. In addition to the configuration files, most of the surveyed research perform analysis on apps' code. However, due to analysis challenges, the majority of those techniques (over 80 percent) neglect special types of code, such as obfuscated, native, dynamically loaded, or reflective code, existing in many apps, including malware.

Obfuscation challenges security analysis of application code. For this reason, nearly all of the surveyed static analyses cannot handle heavily obfuscated code. An example of a technique that handles certain obfuscations is *Apposcopy* [280]. It is a static approach that defines a high-level language for semantically specifying malware signatures. *Apposcopy* is evaluated against renaming, string encryption, and control-flow obfuscation.

Besides the type of obfuscations that Apposcopy is resilient to, more sophisticated obfuscations include hiding behaviors through native code, reflection, and dynamic class loading. These types of obfuscation have highly limited support among Android security analysis techniques.

Among the static analysis techniques studied in our survey, none are able to perform analysis directly on native code, which is written in languages other than Java, such as C or C++. However, some approaches [243], [329], [405] can only identify the usage of native code, particularly if it is used in an abnormal way. For instance, *RiskRanker* [405] raises red flags if it finds encrypted native code, or if a native library is stored in a non-standardized place.

Few approaches consider dynamically loaded code, which occurs after app installation. Some static approaches, such as the tool developed by Poeplau et al. [243], are able to identify the attempts to load external code that might be malicious. Nevertheless, more advanced techniques are required to distinguish the legitimate usages of dynamically loaded code from malicious ones. For example, handling of dynamically loaded code that considers an Android component's life-cycle, where a component can execute from multiple entry points, is not considered. As another example, dynamically loaded code that is additionally encrypted poses another challenge to static or hybrid analyses.

Approaches that consider Java reflection can be classified into two categories. One category, adopts a conservative, black-box approach and simply marks all reflective calls as suspicious. An example of such an approach is *AdRisk* [436]. The other thrust of research attempts to resolve reflection using more advanced analysis. For example, *DroidSafe* [331] employs string and points-to analysis to replace reflective calls with direct calls. As another example, *Pegasus* [391] rewrites an app by injecting dynamic checks when reflective calls are made.

As mentioned above, a significant portion of surveyed research that are trying to address special types of code, adopt a conservative approach. That is, instead of analyzing the content of challenging parts of the app code, e.g., called native library or dynamically loaded class, they flag any usage of such code as suspicious. To distinguish those techniques that *partially* analyze native, obfuscated, dynamic, or reflective code, we marked them with P in Table 4.

6.2 Approach Characteristics (Solution)

Tables 5 and 6 present a summary of the solution-specific aspects that are extracted from the collection of papers included in the literature review. In the following, we summarize the main results for each dimension in the solution category.

6.2.1 Type of Program Analysis

Table 5 separates the approaches with respect to the type of program analysis they leverage. As discussed in Section 5, dynamic analysis is unsound but precise, while static analysis is sound yet imprecise. According to their intrinsic properties, each type of analysis has its own merits and is more appropriate for specific objectives. In particular, for security analysis, soundness is considered to be more important than precision, since it is preferred to not miss any potential security threat, even at the cost of generating false warnings. This could explain why the percentage of static analysis techniques (65 percent) surpasses the percentage of approaches that rely on dynamic analysis techniques (49 percent).

SCanDroid [247] and TaintDroid [423] are among the first to explore the use of static and dynamic analysis techniques respectively for Android security assessment. SCanDroid employs static analysis to detect data flows that violate the security policies specified within an app's configuration. TaintDroid leverages dynamic taint analysis to track the data leakage from privacy-sensitive sources to possibly malicious sinks.

In addition to pure static or dynamic approaches, there exist few hybrid approaches that benefit from the advantages of both static and dynamic techniques. These methods usually first apply static analysis to detect potential security issues, and then perform dynamic techniques to improve their precision by eliminating the false warnings. For example, SMV-HUNTER [252] first uses static analysis to identify potentially vulnerable apps to SSL/TLS man-in-the-middle attack, and then uses dynamic analysis to confirm the vulnerability by performing automatic UI exploration.

Despite the fact that Android apps are mainly developed in Java, conventional Java program analysis methods do not work properly on Android apps, mainly due to its particular event-driven programming paradigm. Such techniques, thus, need to be adapted to address Android-specific challenges. Here, we briefly discuss these challenges and the way they have been tackled in the surveyed papers.

Event-Driven Structure. Android is an event-driven platform, meaning that an app's behavior is formed around the events caused by wide usage of callback methods that handle user actions, component's life-cycle, and requests from other apps or the underlying platform. If an analysis fails to handle these callback methods correctly, models derived from Android apps are disconnected and unsound. This problem has been discussed and addressed in several prior efforts. Among others, Yang et al. [99] introduced a program representation, called callback control-flow graph (CCFG), that supports capturing a rich variety of Android callbacks, including life-cycle and user interactions methods. To extract CCFG, a context-sensitive analysis traverses the control-flow of the program and identifies callback triggers along the visited paths.

Multiple Entry Points. Another difference between an Android app and a pure Java program, is the existence of multiple entry points in Android apps. In fact, unlike conventional Java applications with a single *main* method, Android apps comprise several methods that are implicitly called by the Android framework based on the state of the application (e.g., *onResume* to resume a paused app).

The problem of multiple entry points has been considered by a large body of work in this area [104], [225], [291], [365], [456], [493]. For instance, *FlowDroid* [104] models different Android callbacks, including the ones that handle life-cycle, user interface, and system-based events by creating a "dummy" main method that resembles the *main* method of conventional Java applications. Similar to FlowDroid, *IccTA* [225], [490] also generates dummy main methods, but rather than a single method for the whole app, it considers one per component. In addition to handling multiple entry points problem, the way entry points

TABLE 5
Solution Specific Categorization of the Reviewed Research, Part 1

Dimension		Approaches	%
Type of Program Analysis	Static	A3 [280], A5 [428], AAPL [275], AASandbox [69], Achara_ [18], Adagio [172], AdDroid [323], Adebayo_ [19], AdRisk [188], Amandroid [440], AMDetector [492], Anadroid [265], Ananas [133], AnDarwin [112], Androguard [126], AndroidLeaks [178], Andrubis [268], ApkCombiner [256], APKLancet [471], ApkRiskAnalyzer [103], App-ray [419], Apparecium [418], AppAudit [451], AppCaulk [380], AppContext [470], AppCracker [82], AppIntent [473], Apposcopy [162], AppProfiler [355], AsDroid [215], ASV [212], AuthDroid [432], AutoCog [335], AVDTester [211], Bae_ [43], Bagheri_ [44], Barrera2_ [52], Barros_ [54], Bartel_ [57], Bartsch_ [58], Batyuk_ [59], BayesDroid [420], Bianchi_ [65], Bifocals [105], BlueSeal [206], Brahmastra [64], Brave [321], Brox [284], Buhov_ [80], Canfora3_ [87], Capper [483], Cassandra [274], Cen_ [91], Chabada [184], Chen_ [97], Chen2_ [96], CHEX [276], CMA [389], CoChecker [115], ComDroid [104], ConDroid [379], ContentScope [510], COPES [55], COVERT [45], COVERT_Tool [359], CredMiner [514], Dai_ [117], Dendroid [407], Desnos_ [125], DexDiff [296], DiCerbo_ [92], DidFail [246], DNADroid [111], Dr.Android [220], DRACO [63], Drebin [32], DroidADDMiner [264], DroidAlarm [502], DroidAnalytics [499], DroidAnalyzer [382], DroidAPIMiner [14], DroidChecker [94], DroidCIA [101], DroidForce [340], DroidFuzzer [474], DroidGuard [46], DroidKin [182], DroidMat [447], DroidMiner [465], DroidMOSS [507], DroidPermissionMiner [36], DroidRanger [513], DroidRay [500], DroidRisk [438], DroidSafe [183], Droidsearch [339], DroidSIFT [481], DroidSim [412], Duet [208], Elish_ [138], Enck_ [142], Epicc [315], Fedler_ [153], Fest [490], Flowdroid [35], FUSE [346], Gallingani_ [170], Gallo_ [171], Gates_ [174], Geneiatakis_ [175], Graa_ [186], Graa2_ [185], GroddDroid [16], Han_ [194], Harehunter [15], HornDroid [83], Huang_ [210], HunterDroid [479], ICC_Map [137], IccTA [257], IFT [145], IIF [477], IPCInspection [161], IREA [249], IVDroid [148], Jiao_ [224], Johnson_ [232], Juxtapp [196], Kadir_ [234], Kantola_ [235], Kate_ [237], Kim_ [242], Kirin [143], KLD [387], LeakMiner [472], Li_ [260], Lintent [79], Lu_ [277], Ma_ [283], Mahmood_ [286], Malek_ [288], MalloDroid [146], Mama [366], Manilyzer [156], Mann_ [289], Marvin [267], MassVet [98], MAST [93], Masud_ [292], Matsumoto_ [293], MIGDroid [209], Mobile-Sandbox [404], MobSafe [459], MonkeyDroid [282], Moonsamy_ [297], Moonsamy2_ [298], MorphDroid [163], Mudflow [38], Mutchler_ [301], NoInjection [226], Onwuzurike_ [318], OpSeq [22], PaddyFrog [448], Paupore_ [322], PCLeaks [258], Pedal [270], Pegasus [99], Peiravian_ [324], Peng_ [325], PermCheckTool [427], PermissionFlow [372], Permlyzer [461], Poeplau_ [328], ProfileDroid [441], PScout [37], Quan_ [336], RAMSES [132], Relda [190], ResDroid [390], Riskmon [227], RiskMon2 [228], Riskranker [187], SAAF [205], SADroid [195], Sahs_ [361], Sanz_ [368], Sarma_ [369], Sayfullina_ [371], ScanDal [243], SCanDroid [168], Scoria [423], SecUP [458], SEFA [449], Seneviratne_ [381], Shabtai_ [383], Shen_ [395], SherlockDroid [29], SmartDroid [497], Smith_ [400], SMV-HUNTER [403], StaDynA [494], STAMBA [71], SUPOR [214], TMSVM [453], TouchDevelop [454], TraceDroid [422], TrustDroid [493], UID [139], UIPicker [306], ViewDroid [478], Wang_ [436], WeChecker [114], WifiLeaks [17], Wognsen_ [445], Woodpecker [189], WuKong [431], Yerima_ [475], You_ [476], Zhou_ [506], Zuo_ [517]	65%
	Dynamic (E)	A5 [428], AASandbox [69], Achara_ [18], ADDICTED [508], Afonso_ [20], AMDetector [492], Ananas [133], Andrubis [268], AntiMalDroid [491], Apex [308]^E, App-ray [419], AppAudit [451], AppCaulk [380], AppCracker [82], AppFence [207], AppGuard [42], AppInspector [179], AppIntent [473], AppProfiler [355], AppsPlayground [342], APSET [364], Aquifer [305], ASF [40]^E, ASM [202]^E, ASV [212], AuDroid [326]^E, Aurasium [460], AuthDroid [432], AVDTester [211], Bae_ [43], Bal_ [49], Berthome_ [62], Brahmastra [64], Brave [321], Bugiel_ [76], Buzzer [88], Canfora_ [85], Canfora3_ [87], Capper [483]^E, CMA [389], Compac [439]^E, ConDroid [379], ContentScope [510], ConUCON [48]^E, CopperDroid [347], CopperDroid2 [414], CRePE [107], Crowdroid [81], Dagger [466], DataChest [512], DeepDroid [437]^E, Defensor [319], Dr.Android [220], DRACO [63], DroidAnalyst [169], DroidAnalytics [499], DroidBarrier [24], DroidDolphin [450], DroidForce [340]^E, DroidFuzzer [474], DroidGuard [46]^E, DroidLogger [118], DroidPAD [279], DroidRay [500], DroidScope [464], DroidTest [356], DroidTrace [501], DroidTrack [362], EASEAndroid [435], Fedler_ [153], Feth_ [164]^E, FineDroid [488]^E, FireDroid [358], FlaskDroid [78]^E, Geneiatakis_ [175], Graa_ [186], Graa2_ [185], GroddDroid [16], Ham_ [192], HunterDroid [479], I-ARM-Droid [122]^E, IntentFuzzer [467], IPCInspection [161], Isohara_ [217], Jeon_ [219], Jeong_ [222], Jia_ [223]^E, Jiao_ [224], Jung_ [233]^E, Kadir_ [234], Kantola_ [235], Karami_ [236]^E, Kim_ [242], Kynoid [377], LazyTainter [442], Lee_ [252], Leontiadis_ [254], Li2_ [261], MADAM [128], Mahmood_ [286], Malek_ [288], Manilyzer [156], Marvin [267], Masud_ [292], MeadDroid [253], Mobile-Sandbox [404], MobSafe [459], MockDroid [61]^E, Morbs [434], MOSES [496]^E, MpDroid [417]^E, Mutchler_ [301], NDroid [333], Nishimoto_ [311], Onwuzurike_ [318], Paranoid-Android [330], PatchDroid [299], Patronus [411], Paupore_ [322], pBMDS [455], PeBA [60], Pegasus [99]^E, Permlyzer [461], PICARD [129], Porscha [316]^E, PREC [203], ProfileDroid [441], PUMA [365], PuppetDroid [177], Quan_ [336], QUIRE [127], RetroSkeleton [121]^E, Riskmon [227], RiskMon2 [228], Saint [317]^E, Schmidt_ [374], SCSdroid [266], SFG [27], Shebaro_ [391]^E, Short_ [398], SmartDroid [497], SMV-HUNTER [403], StaDynA [494], STAMBA [71], Stowaway [157], Su_ [406], TaintDroid [141], Tchakounte_ [415], TISSA [515]^E, TMSVM [453], TraceDroid [422], TreeDroid [119]^E, UIPicker [306]^E, Uranine [345], VetDroid [489], WifiLeaks [17], Wijesekera_ [444], Xmandroid [75]^E, Yaase [357]^E, Yerima_ [475], You_ [476], Zuo_ [517]	49%
	Hybrid	A5 [428], AASandbox [69], Achara_ [18], AMDetector [492], Ananas [133], Andrubis [268], App-ray [419], AppAudit [451], AppCaulk [380], AppCracker [82], AppIntent [473], AppProfiler [355], ASV [212], AuthDroid [432], AVDTester [211], Bae_ [43], Brahmastra [64], Brave [321], Canfora3_ [87], Capper [483], CMA [389], ConDroid [379], ContentScope [510], Dr.Android [220], DRACO [63], DroidAnalytics [499], DroidForce [340], DroidFuzzer [474], DroidGuard [46], DroidRay [500], Fedler_ [153], Geneiatakis_ [175], Graa_ [186], Graa2_ [185], GroddDroid [16], HunterDroid [479], IPCInspection [161], Jiao_ [224], Kadir_ [234], Kantola_ [235], Kim_ [242], Lee_ [252], Mahmood_ [286], Malek_ [288], Manilyzer [156], Marvin [267], Masud_ [292], Mobile-Sandbox [404], MobSafe [459], Mutchler_ [301], Onwuzurike_ [318], Paupore_ [322], Pegasus [99], Permlyzer [461], ProfileDroid [441], Quan_ [336], Riskmon [227], RiskMon2 [228], SmartDroid [497], SMV-HUNTER [403], StaDynA [494], STAMBA [71], TMSVM [453], TraceDroid [422], UIPicker [306], WifiLeaks [17], Yerima_ [475], You_ [476], Zuo_ [517]	21%
Supplementary Techniques	Machine Learning (P|N)	AAPL [275]^N, Adagio [172], Adebayo_ [19]^P, Afonso_ [20], AnDarwin [112], AntiMalDroid [491], AppContext [470], AutoCog [335]^N, Bae_ [43], Barrera2_ [52], BayesDroid [420]^P, Brave [321], Canfora_ [85], Cen_ [91]^P, Chabada [184]^N, Crowdroid [81], Dai_ [117], Dendroid [407], DiCerbo_ [92], Drebin [32], DroidADDMiner [264], DroidAPIMiner [14], DroidDolphin [450], DroidLegacy [124], DroidMat [447], DroidPAD [279], DroidPermissionMiner [36], DroidSIFT [481], EASEAndroid [435], Fest [490], Gates_ [174]^P, Huang_ [210], Jiao_ [224]^P, Kate_ [237]^P, KLD [387]^P, Ma_ [283]^N, MADAM [128], Mama [366], Marvin [267], MassVet [98], MAST [93], Milosevic_ [295], MobSafe [459], MonkeyDroid [282]^N, Moonsamy_ [297], Moonsamy2_ [298], Mudflow [38], OpSeq [22], pBMDS [455], Pedal [270], Peiravian_ [324], Peng_ [325]^P, PICARD [129]^P, PUMA2 [198], Quan_ [336], RAMSES [132], ResDroid [390], Riskmon [227], RiskMon2 [228], Sahs_ [361], Sanz_ [368], Sarma_ [369], Sayfullina_ [371]^P, Schmidt_ [374], Shabtai_ [383], SherlockDroid [29], Su_ [406], SUPOR [214]^N, TMSVM [453], UIPicker [306]^N, Wang_ [436], WuKong [431], Yerima_ [475]^P, Zhou_ [506]	22%
	Formal Analysis	Apposcopy [162], APSET [364], Armando_ [31], Bagheri_ [44], Barbon_ [50], Cassandra [274], COVERT [45], COVERT_Tool [359], DroidGuard [46], HornDroid [83], Jia_ [223], Lintent [79], Lu_ [277], Mann_ [289], MorphDroid [163], Pegasus [99], SADroid [195], ScanDal [243], Scoria [423], Smith_ [400], Song_ [401], TreeDroid [119]	7%
Auto. Level	Automatic	Others *	93%
	Semi-Automatic	Achara_ [18], AdSplit [393], AndroidLeaks [178], APKLancet [471], Apposcopy [162], AppProfiler [355], Barros_ [54], Batyuk_ [59], Crowdroid [81], Dr.Android [220], DroidForce [340], DroidRay [500], Graa_ [186], Ham_ [192], IFT [145], IREA [249], Isohara_ [217], Mann_ [289], PuppetDroid [177], Scoria [423], Smith_ [400], StaDynA [494], Stowaway [157]	7%

E: *Enforcing security policies (providing a level of protection, in addition to dynamic detection).*
P: *Probabilistic approaches,* ***N***: *Natural language processing (NLP) is used.*
**: Including all other surveyed papers that are not mentioned as Semi-Automatic.*

are discovered is also crucial for a precise analysis. Some approaches [145], [151] simply rely on the domain knowledge, including the Android API documentation, to identify entry points. Some other approaches employ more systematic methods. For instance, *CHEX* describes a sound method to automatically discover different types of app entry points [456]. It iterates over all uncalled framework methods overridden by app, and connects those methods to the corresponding call graph node.

Inter-Component Communication. Android apps are composed of multiple components. The most widely used mechanism provided by Android to facilitate communication between components involves Intent, i.e., a specific type of event message in Android, and Intent Filter. The Android platform then automatically matches an Intent with the proper Intent Filters at runtime, which induce discontinuities in the statically extracted app models. This event-based inter-component communication (ICC) should be treated carefully, otherwise important security issues could be missed. The ICC challenge has received a lot of attention in the surveyed research [3], [146], [218], [225], [311]. *Epicc* [218], among others, is an approach devoted to identify inter-component communications by resolving links between components. It reduces the problem of finding ICCs to an instance of the inter-procedural distributive environment (IDE) problem [503], and then uses an IDE algorithm to solve the ICC resolution problem efficiently.

Modeling the Underlying Framework. In order to reason about the security properties of an app, the underlying Android platform should be also considered and included in the security analysis. However, analyzing the whole Android framework would result in state explosion and scalability issues. Therefore, a precise, yet scalable model, of the Android framework is crucial for efficient security assessment.

Various methods have been leveraged by the surveyed approaches to include the Android framework in their analysis. *Woodpecker* [151] uses a summary of Android built-in classes, which are pre-processed ahead of an app analysis to reduce the analysis costs associated with each app. To enable a more flexible analysis environment, *CHEX* [456] runs in two modes. In one mode, it includes the entire Android framework code in the analysis, and in the other only a partial model of the Android's external behaviors is used. *DroidSafe* [331] attempts to achieve a combination of precision and scalability by generating analysis stubs, abstractions of underlying implementation, which are incomplete for runtime, but complete for the analysis. Finally, to automatically classify Android system APIs as sources and sinks, *SuSi* [504] employs machine learning techniques. Such a list of sources and sinks of sensitive data is then used in a number of other surveyed approaches, including, *FlowDroid* [104], *DroidForce* [480], *IccTA* [225], [490], and *DidFail* [311].

6.2.2 Supplementary Techniques

We observe that most approaches (over 70 percent) only rely on program analysis techniques to assess the security of Android software. Less than 30 percent of the approaches employ other complementary techniques in their analysis. Among them, machine learning and formal analysis techniques are the most widely used, comprising 22 and 7 percent of the overall set of papers collected for this literature review, respectively.

These approaches typically first use some type of program analysis to extract specifications from the Android software that are input to the analysis performed by other supplementary techniques. For example, COVERT, combines formal app models that are extracted through static analysis with a formal specification of the Android framework to check the overall security posture of a system [205].

Machine learning techniques are mainly applied to distinguish between benign and malicious apps. The underlying assumption in this thrust of effort is that abnormal behavior is a good indicator of maliciousness. Examples of this class of research are *CHABADA* [299] and its successor *MUDFLOW* [384], which are both intended to identify abnormal behavior of apps. The focus of CHABADA is to find anomalies between app descriptions and the way APIs are used within the app. MUDFLOW tries to detect the outliers with respect to the sensitive data that flow through the apps.

Natural language processing (NLP) is another supplementary technique employed by CHABADA and a few other approaches (e.g., AAPL[262], AutoCog [285], SUPOR [254], UIPicker [488]), mainly to process apps' meta-data, such as app descriptions, which are expressed in natural language form. Moreover, probabilistic approaches are also leveraged by a number of machine learning-based tools (e.g., [298], [393], [410], [453]) to distinguish malware apps from benign ones, according to the observed probability of extracted features. Research using NLP and probabilistic methods are highlighted by *N* and *P*, respectively, in Table 5.

6.2.3 Automation Level

We observe that most approaches (93 percent) are designed to perform Android security analysis in a completely automated manner, which is promising as it enables wide-scale evaluation of such automated techniques, discussed more in the following section (Section 6.3).

A number of approaches, however, require some manual effort (7 percent); for example, annotating an app's code with labels representing different security concerns. Once the code is annotated manually, an automatic analysis is run to identify the security breaches or attacks in the source code. For instance, *IFT* [353] requires app developers to annotate an app's source code with information-flow type qualifiers, which are fine-grained permission tags, such as INTERNET, SMS, GPS, etc. Subsequently, app repository auditors can employ IFT's type system to check information flows that violate the secure flow policies. Manually applying the annotations affects usability and scalability of such approaches, however, enables a more precise analysis to ensue.

6.2.4 Analysis Data Structures[7]

Almost half of *static* approaches (49 percent) leverage lightweight analysis that only relies on text-based information retrieval techniques. Such approaches treat app's code and configuration as unstructured texts and try to extract security critical keywords and phrases (e.g., permissions, sensitive APIs) for further analysis using supplementary techniques (cf. Section 5.2.2). For instance, *Drebin* [314] extracts sets of strings, such as permissions, app components, and intent filters by parsing the manifest, and API calls, and network addresses from dex code. It then maps those extracted features to a vector space, which is further used for learning-based malware detection.

On the other hand, many techniques take the structure of code into account when extracting the security model of the apps. For this purpose, various data structures that represent apps at an abstract level are commonly used by those analysis techniques. We observe that call graphs (CGs) and control flow graphs (CFGs) are the most frequently used data structure in the surveyed papers.

Taint information are propagated through call graph, among other things, to determine the reachability of various sinks from specific sources. *LeakMiner* [365], *RiskRanker* [405], *TrustDroid* [471], *ContentScope* [145] and *IPC Inspection* [12] are some examples that traverse the call graph for taint analysis. Among others, *ContentScope* traverses CG to find paths form public content provider interfaces to the database function APIs in order to detect database leakage or pollution.

Moreover, generating and traversing the app's CG is also essential in tracking the message (i.e., Intent) transfer among the app's components. *Epicc* [218] and *AsDroid* [282] are among the approaches that use call graph for this purpose. In addition, *PScout* [244] and *PermissionFlow* [242] perform reachability analysis over the CG to map Android permissions to the corresponding APIs.

Control flow graph is also widely used in the surveyed analysis methods. *ContentScope* [145], for example,

7. The percentages reported in Sections 6.2.4, 6.2.5, and 6.2.6 are calculated only for the static techniques.

extracts an app's CFG to obtain the constraints corresponding to potentially dangerous paths. The collected constraints are then fed into a constraint solver to generate inputs corresponding to candidate path executions. Enck et al. [146] have also specified security rules over CFG to enable a control-flow based vulnerability analysis of Android apps.

More advanced and comprehensive program analyses rely on a combination of CFG and CG, a data structure called inter-procedural control flow graph that links the individual CFGs according to how they call each other. *FlowDroid* [104], for example, traverses ICFG to track tainted variables; *Epicc* [218] also performs string analysis over ICFG; *IccTA* [225], [490] detects inter-component data leaks by running data-flow analysis over such a data structure. Since the generated ICFG for the entire application is massive, complicated, and potentially unscalable, a number of approaches leverage a reduced version of ICFG for their analysis. For example, *Woodpecker* [151] locates capability leaks (cf. Section 6.1.1) by traversing a reduced permission-specific ICFG, rather than the generic one.

In addition to such canonical, widely-used data structures, a good portion of existing approaches leverage customized data structures for app analysis. One examples is G*, an ICFG-based graph, in which each call site is represented by two nodes, one before the procedure call and the other after returning [218]. *CHEX* [456] introduces two customized data structures of split data-flow summary (SDS) and permutation data-flow summary (PDS) for its data flow analysis. SDS is a kind of CFG that also considers the notion of split, *"a subset of the app code that is reachable from a particular entry point method"*. PDS is also similar to ICFG, and links all possible permutations of SDS sequences. Another data structure commonly used by app clone detectors, such as *AnDarwin* [270] and *DNADroid* [312], is program dependency graph (PDG). By capturing control and data dependencies between code fragments, a PDG is able to compare similarity between app pairs.

6.2.5 Sensitivity of Analysis

Apart from lightweight, text-based approaches, other static approaches have adopted a level of sensitivity in their analysis. According to our survey, flow-sensitive approaches that consider the program statements sequence, have the highest frequency (23 percent) among the *static* approaches. Following that, 14 percent of static techniques are context-sensitive, that is, they compute the calling context of method calls. Finally, 3 percent of static analyses are path-sensitive, meaning that only a handful of analysis approaches distinguish information obtained from different execution paths. Generally, approaches with higher sensitivity, i.e., considering more program properties for the analysis, generate more accurate results, but they are less scalable in practice.

6.2.6 Code Representation

Different approaches analyze various formats of the Java code, which are broadly distinguishable as source code versus byte code. The applicability of the former group of approaches, such as *SCanDroid* [247], are confined to apps with available source code.

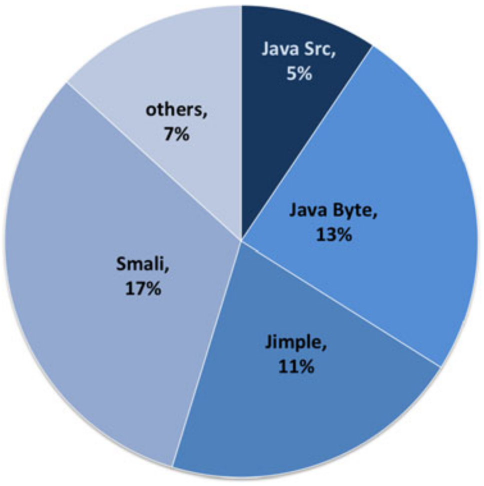

Fig. 9. Distribution of research based on the type of code or intermediate representation (IR) used for analysis.

Most recent approaches, however, support byte-code analysis. Such approaches typically perform a pre-processing step, in which Dalvik byte code, encapsulated in the APK file, is transferred to another type of code or intermediate representation (IR). Fig. 9 shows the distribution of the approaches based on the target IR of the analysis.

According to the diagram, Smali [507] is the most popular intermediate representations, used in 17 percent of those studied approaches that are performing analysis on a type of IR. Also, 13 percent of such approaches, in the pre-processing step, retarget Dalvik byte-code to Java byte-coded JAR files. An advantage of this approach is the ability to reuse pre-developed, off-the-shelf Java analysis libraries and tools. In exchange, APK-to-JAR decompilers suffer from performance overhead and incomplete code coverage.

6.2.7 Inspection Level[8]

Dynamic approaches monitor an app's behavior using different techniques. According to our results, about 35 percent of *dynamic* approaches intercept events that occur within the emulated environments by modifying virtual machines (VMs). VM-based dynamic analyses are further distinguishable by the type of virtual machine they modify: Dalvik VM (e.g., *TaintDroid* [423]) or QEMU VM (e.g., *CopperDroid* [303]). While QEMU-based systems work on a lower level and are able to trace native code, Dalvik-based techniques tend to be more efficient [28]. Therefore, a few tools, such as [272], take advantage of both techniques.

Around 39 percent of studied dynamic analyses weave monitoring code into Android apps or framework APIs to capture app behaviors. Approaches that monitor the framework are marked with *F* in Table 6. Different libraries are developed by the research community to facilitate app-level monitoring, including: APIMonitor

8. The percentages reported in Sections 6.2.7 and 6.2.8 are calculated only for the dynamic techniques.

TABLE 6
Solution Specific Categorization of the Reviewed Research, Part 2

Dimension			Approaches	% (%)*
Static	Analysis Data Structure	Text-based	A3 [280], AASandbox [69], Achara_ [18], AdDroid [323], Adebayo_ [19], AMDetector [492], Ananas [133], Andrubis [268], ApkRiskAnalyzer [103], App-ray [419], AppCracker [82], AppProfiler [355], AuthDroid [432], AutoCog [335], Bae_ [43], Bagheri_ [44], Barrera2_ [52], Batyuk_ [59], BayesDroid [420], Bifocals [105], Brave [321], Buhov_ [80], Canfora3_ [87], Cassandra [274], Cen_ [91], Chabada [184], Chen2_ [96], Dai_ [117], DiCerbo_ [92], Dr.Android [220], DRACO [63], Drebin [32], DroidAnalyzer [382], DroidFuzzer [474], DroidKin [182], DroidMat [447], DroidMOSS [507], DroidPermissionMiner [36], DroidRay [500], DroidRisk [438], Droidsearch [339], Duet [208], Fedler_ [153], Fest [490], Gallo_ [171], Gates_ [174], Geneiatakis_ [175], Huang_ [210], IREA [249], Jiao_ [224], Johnson_ [232], Juxtapp [196], Kadir_ [234], Kantola_ [235], Kate_ [237], Kim_ [242], Kirin [143], KLD [387], Lintent [79], Lu_ [277], Ma_ [283], Malek_ [288], MalloDroid [146], Mama [366], Manilyzer [156], Mann_ [289], Marvin [267], MAST [93], Masud_ [292], Matsumoto_ [293], Mobile-Sandbox [404], MobSafe [459], Moonsamy_ [297], Moonsamy2_ [298], Onwuzurike_ [318], OpSeq [22], Paupore_ [322], Pedal [270], Peiravian_ [324], Peng_ [325], PermCheckTool [427], Permlyzer [461], ProfileDroid [441], Quan_ [336], RAMSES [132], ResDroid [390], Riskmon [227], RiskMon2 [228], Sahs_ [361], Sanz_ [368], Sarma_ [369], Sayfullina_ [371], Scoria [423], SecUP [458], Seneviratne_ [381], Shabtai_ [383], SherlockDroid [29], Smith_ [400], SMV-HUNTER [403], STAMBA [71], TraceDroid [422], UIPicker [306], Wang_ [436], WifiLeaks [17], WuKong [431], Yerima_ [475], You_ [476]	49% (32%)
		Control Flow Graph (CFG)	A5 [428], Adagio [172], Amandroid [440], Anadroid [265], Androguard [126], APKLancet [471], Apparecium [418], AppCaulk [380], AppContext [470], Apposcopy [162], AsDroid [215], ASV [212], AVDTester [211], Barros_ [54], Bianchi_ [65], BlueSeal [206], Brahmastra [64], Capper [483], CMA [389], CoChecker [115], ComDroid [104], ContentScope [510], COVERT [45], COVERT_Tool [359], CryptoLint [134], Dendroid [407], Desnos_ [125], DexDiff [296], DroidAlarm [502], DroidChecker [94], DroidCIA [101], DroidForce [340], DroidGuard [46], DroidMiner [465], DroidSIFT [481], DroidSim [412], Elish_ [138], Enck_ [142], Epicc [315], Flowdroid [35], Gallingani_ [170], Graa_ [186], Graa2_ [185], GroddDroid [16], Harehunter [15], IccTA [257], IIF [477], IVDroid [148], LeakMiner [472], MassVet [98], MonkeyDroid [282], MorphDroid [163], NoInjection [226], PaddyFrog [448], PCLeaks [258], PermissionFlow [372], Poeplau_ [328], Riskranker [187], SAAF [205], SADroid [195], Sahs_ [361], ScanDal [243], SEFA [449], TMSVM [453], TouchDevelop [454], UID [139], WeChecker [114], Wognsen_ [445], Woodpecker [189], Zhou_ [506]	31% (21%)
		Call Graph (CG)	Adagio [172], Amandroid [440], Androguard [126], AndroidLeaks [178], APKLancet [471], Apparecium [418], AppAudit [451], AppCaulk [380], AppContext [470], AppIntent [473], Apposcopy [162], AppSealer [482], AsDroid [215], ASV [212], Bartel_ [57], Bianchi_ [65], BlueSeal [206], Brahmastra [64], Brox [284], Capper [483], CMA [389], CoChecker [115], ContentScope [510], COPES [55], COVERT [45], COVERT_Tool [359], CryptoLint [134], DroidAlarm [502], DroidChecker [94], DroidCIA [101], DroidGuard [46], DroidRanger [513], DroidSafe [183], DroidSIFT [481], Epicc [315], Flowdroid [35], FUSE [346], Gallingani_ [170], HunterDroid [479], IccTA [257], IPCInspection [161], LeakMiner [472], Mahmood_ [286], Malek_ [288], MonkeyDroid [282], MorphDroid [163], Mudflow [38], Mutchler_ [301], NoInjection [226], PaddyFrog [448], PCLeaks [258], PermissionFlow [372], Poeplau_ [328], PScout [37], Relda [190], SADroid [195], SEFA [449], StaDynA [494], TouchDevelop [454], TrustDroid [493], UID [139], WeChecker [114], Woodpecker [189], Zuo_ [517]	28% (19%)
		Inter-procedural CFG (ICFG)	Amandroid [440], AppContext [470], Apposcopy [162], Capper [483], COVERT [45], COVERT_Tool [359], CryptoLint [134], DroidChecker [94], DroidGuard [46], Epicc [315], Flowdroid [35], Gallingani_ [170], IccTA [257], LeakMiner [472], MonkeyDroid [282], MorphDroid [163], PCLeaks [258], PermissionFlow [372], Poeplau_ [328], SEFA [449], TouchDevelop [454], UID [139], WeChecker [114], Woodpecker [189]	10% (7%)
	Sensitivity of Analysis	Flow	AAPL [275], AdRisk [188], Amandroid [440], Apparecium [418], AppContext [470], Apposcopy [162], AppSealer [482], AsDroid [215], Barros_ [54], Bartsch_ [58], Bianchi_ [65], BlueSeal [206], Brox [284], Capper [483], CHEX [276], CMA [389], ComDroid [104], ContentScope [510], COVERT [45], COVERT_Tool [359], CredMiner [514], CryptoLint [134], DidFail [246], DroidADDMiner [264], DroidAlarm [502], DroidAPIMiner [14], DroidChecker [94], DroidForce [340], DroidGuard [46], DroidSIFT [481], Elish_ [138], Enck_ [142], Epicc [315], Flowdroid [35], Han_ [194], Harehunter [15], HornDroid [83], IccTA [257], IFT [145], LeakMiner [472], MorphDroid [163], NoInjection [226], PCLeaks [258], Pegasus [99], PermissionFlow [372], Poeplau_ [328], Riskranker [187], ScanDal [243], SCanDroid [168], SEFA [449], SUPOR [214], UID [139], WeChecker [114]	23% (16%)
		Context	AAPL [275], Amandroid [440], ApkCombiner [256], AppContext [470], AppIntent [473], Apposcopy [162], AppSealer [482], Brox [284], Capper [483], CHEX [276], COVERT [45], COVERT_Tool [359], DidFail [246], DroidADDMiner [264], DroidForce [340], DroidGuard [46], DroidSafe [183], DroidSIFT [481], Epicc [315], Flowdroid [35], Han_ [194], IccTA [257], IFT [145], NoInjection [226], PCLeaks [258], Pegasus [99], PermissionFlow [372], ScanDal [243], SCanDroid [168], SUPOR [214], UID [139], WeChecker [114]	14% (10%)
		Path	ConDroid [379], ContentScope [510], DroidAnalytics [499], DroidForce [340], Gallingani_ [170], Woodpecker [189]	3% (2%)
	Code Representation	Java Source	Bartel_ [57], IFT [145], IVDroid [148], Lu_ [277], Mann_ [289], Matsumoto_ [293], PermCheckTool [427], PScout [37], SCanDroid [168], Smith_ [400], TouchDevelop [454]	5% (3%)
		Java Byte	AnDarwin [112], AndroidLeaks [178], AppAudit [451], AppCracker [82], AppIntent [473], AppProfiler [355], AppSealer [482], AsDroid [215], Bagheri_ [44], Capper [483], Cen_ [91], Chen2_ [96], CoChecker [115], ComDroid [104], Desnos_ [125], DNADroid [111], DroidChecker [94], Duet [208], Enck_ [142], IPCInspection [161], KLD [387], LeakMiner [472], Lee_ [252], Malek_ [288], Onwuzurike_ [318], Pedal [270], Pegasus [99], PermissionFlow [372], Permlyzer [461], Relda [190], UID [139]	13% (9%)
		Jimple	A5 [428], AppContext [470], Apposcopy [162], Bartsch_ [58], BlueSeal [206], Brahmastra [64], COPES [55], COVERT [45], COVERT_Tool [359], DidFail [246], DroidForce [340], DroidGuard [46], DroidSafe [183], Elish_ [138], Epicc [315], Flowdroid [35], Geneiatakis_ [175], Harehunter [15], IccTA [257], MonkeyDroid [282], Mudflow [38], Mutchler_ [301], PCLeaks [258], Shen_ [395], WeChecker [114]	11% (7%)
		Smali	AASandbox [69], AdRisk [188], AdSplit [393], Ananas [133], ApkCombiner [256], APKLancet [471], Apparecium [418], Aurasium [460], AVDTester [211], Batyuk_ [59], Chabada [184], Chen_ [97], CHEX [276], CMA [389], ContentScope [510], CredMiner [514], Dr.Android [220], DroidLegacy [124], DroidMOSS [507], Harehunter [15], IREA [249], Johnson_ [232], Ma_ [283], Mobile-Sandbox [404], MobSafe [459], PaddyFrog [448], ProfileDroid [441], SAAF [205], SADroid [195], Seneviratne_ [381], SmartDroid [497], SMV-HUNTER [403], SUPOR [214], TrustDroid [493], ViewDroid [478], Wognsen_ [445], Woodpecker [189], WuKong [431], Yerima_ [475], Zuo_ [517]	17% (12%)
Dynamic	Inspection Level	Application (F)	Achara_ [18], Ananas [133], Androguard [126], AntiMalDroid [491]^{F}, Apex [308], APKLancet [471], AppCaulk [380], AppGuard [42], AppIntent [473], AppProfiler [355], AppSealer [482], APSET [364], ASM [202], Aurasium [460], AutoCog [335], Bagheri_ [44], Bartsch_ [58], Berthome_ [62], Brahmastra [64], Buzzer [88], Capper [483], CMA [389], ConDroid [379], ContentScope [510], CRePE [107], Dr.Android [220], DroidAnalytics [499], DroidDolphin [450], DroidForce [340], DroidFuzzer [474], DroidGuard [46], DroidLogger [118], DroidRay [500], DroidTrack [362], Feth_ [164]^{F}, FineDroid [488]^{F}, Graa_ [186], GroddDroid [16], HunterDroid [479], I-ARM-Droid [122], ICC_Map [137], IPCInspection [161], Jeon_ [219], Jung_ [233]^{F}, Kantola_ [235], Lee_ [252], Li2_ [261], Lintent [79], Lu_ [277], Mahmood_ [286], Matsumoto_ [293], MobSafe [459], Morbs [434], MpDroid [417]^{F}, Mutchler_ [301], Nishimoto_ [311]^{F}, PaddyFrog [448], Paupore_ [322], Pegasus [99], Permlyzer [461], Porscha [316]^{F}, PUMA [365], PUMA2 [198], QUIRE [127], SADroid [195], SFG [27]^{F}, Shebaro_ [391]^{F}, Short_ [398], SmartDroid [497], Smith_ [400], SMV-HUNTER [403], TruStore [495], Uranine [345], WifiLeaks [17]^{F}, Wijesekera_ [444], Xmandroid [75], You_ [476]	39% (23%)
		Kernel	ADDICTED [508], Afonso_ [20], Ananas [133], Andrubis [268], AppsPlayground [342], Aquifer [305], ASF [40], ASM [202], AVDTester [211], Bugiel_ [76], Canfora_ [85], Canfora3_ [87], Compac [439], CopperDroid [347], CopperDroid2 [414], Crowdroid [81], Dagger [466], DataChest [512], DeepDroid [437], Defensor [319], DroidBarrier [24], DroidScope [464], FineDroid [488], FlaskDroid [78], Ham_ [192], Isohara_ [217], Jeong_ [222], Karami_ [236], MADAM [128], Mobile-Sandbox [404], Paranoid-Android [330], pBMDS [455], PeBA [60], PICARD [129], PREC [203], ProfileDroid [441], Quan_ [336], Schmidt_ [374], SCSdroid [266], Su_ [406], Tchakounte_ [415], TMSVM [453], TraceDroid [422]	27% (13%)
		VM	AASandbox [69], Afonso_ [20], Androguard [126], Andrubis [268], AppAudit [451], AppFence [207], AppInspector [179], AppsPlayground [342], Aquifer [305], ASF [40], ASM [202], ASV [212], AuDroid [326], Aurasium [460], AVDTester [211], Bagheri_ [44], Bal_ [49], Bugiel_ [76], Compac [439], ConUCON [48], CopperDroid [347], CopperDroid2 [414], CRePE [107], DataChest [512], DeepDroid [437], DexDiff [296], Dr.Android [220], DroidPAD [279], DroidRay [500], DroidScope [464], FireDroid [358], FlaskDroid [78], Graa_ [186], IPCInspection [161], Kantola_ [235], Kynoid [377], LazyTainter [442], Lee_ [252], Leontiadis_ [254], MADAM [128], Marvin [267], MeadDroid [253], Mobile-Sandbox [404], Morbs [434], MOSES [496], NDroid [333], Paranoid-Android [330], PatchDroid [299], PeBA [60], Quan_ [336], QUIRE [127], Saint [317], StaDynA [494], TaintDroid [141], TISSA [515], TraceDroid [422], TreeDroid [119], TruStore [495], VetDroid [489], Yaase [357], Zuo_ [517]	35% (18%)
	Input Generation	Fuzzing (H)	A5 [428]^{H}, Afonso_ [20], Ananas [133], Andrubis [268], AppsPlayground [342]^{H}, AVDTester [211], Canfora_ [85], ContentScope [510], CopperDroid [347], CopperDroid2 [414]^{H}, Dagger [466], DroidDolphin [450], DroidFuzzer [474], DroidScope [464], DroidTest [356], DroidTrack [362], HunterDroid [479], IntentFuzzer [467], Jiao_ [224], Karami_ [236]^{H}, LazyTainter [442], Mahmood_ [286], Malek_ [288]^{H}, Mobile-Sandbox [404], MobSafe [459], MOSES [496], NDroid [333], Permlyzer [461], PUMA [365]^{H}, RetroSkeleton [121], SmartDroid [497], SMV-HUNTER [403], TaintDroid [141], UIPicker [306], Uranine [345], VetDroid [489], Zuo_ [517]^{H}	23% (11%)
		Symbolic Exec.	AppInspector [179], AppIntent [473], ConDroid [379], DroidAnalytics [499], Malek_ [288], You_ [476]	4% (2%)

***H**: Heuristics-based Approaches, **F**: Android Framework level.*
** The last column of this table should be read as follows: Percentage among static/dynamic approaches (Percentage among all approaches).*

developed and used in *DroidBox* [142], a Soot-based library proposed by [102], and SIF [103], a selective instrumentation framework.

Finally, about 26 percent of surveyed dynamic techniques capture app behavior through monitoring system calls, using loadable kernel modules (e.g., *ANANAS* [269]) or debugging tools such as `strace` (e.g., *Crowdroid* [304]). Most of the kernel-level techniques are able to trace native code, but they are usually not compatible with multiple versions of Android [28].

To overcome the shortcomings and limitations pertaining to certain monitoring levels, a number of tools leverage a combination of different inspection techniques. According to our survey, around 22 percent of the studied dynamic approaches perform monitoring at multiple levels. For instance, through monitoring both the Linux kernel and Dalvik VM, *DroidScope* [332], a dynamic malware analyzer, is able to identify anomalies in app behaviors.

6.2.8 Input Generation Technique

The Android security assessment approaches that rely on dynamic analysis require test input data and events to drive the execution of apps.

We can observe from Table 6 that most of such approaches use fuzz testing, comprising 23 percent of the dynamic approaches studied for this literature review. Fuzzing is a form of negative testing that feeds malformed and unexpected input data to a program with the objective of revealing security vulnerabilities. For example, it has been shown that an SMS protocol fuzzer is highly effective in finding severe security vulnerabilities in all three major smartphone platforms [508]. In the case of Android, fuzzing found a security vulnerability triggered by simply receiving a particular type of SMS message, which not only kills the phone's telephony process, but also kicks the target device off the network [508].

Despite the individual success of fuzzing as a general method of identifying vulnerabilities, fuzzing has traditionally been used as a brute-force mechanism. Using fuzzing for testing is generally a time consuming and computationally expensive process, as the space of possible inputs to any real-world program is often unbounded. Existing fuzzing tools, such as Android's Monkey [180], generate purely random test case inputs, and thus are often ineffective in practice.

To improve the efficiency of fuzzing techniques, a number of approaches [261], [281], [303] have devised heuristics that guide a fuzzer to cover more segments of app code in an intelligent manner. For instance, by providing meaningful inputs for text boxes by using contextual information, *AppsPlayground* [281] avoids redundant test paths. This in turn enables a more effective exploration of the app code.

A comparatively low number of dynamic approaches (4 percent) employ symbolic execution, mainly to improve the effectiveness of generated test inputs. For example, AppInspector [277] applies concolic execution, which is the combination of symbolic and concrete execution. It switches back and forth between symbolic and concrete modes to enable analysis of apps that communicate with remote parties. Scalability is, however, a main concern with symbolic execution techniques. More recently, some approaches try to improve the scalability of symbolic execution. For instance, AppIntent [279] introduces a guided symbolic execution that narrows down the space of execution paths to be explored by considering both the app call graph and the Android execution model. Symbolic execution is also used for feasible path refinement. Among others, Woodpecker [151] models each execution path as a set of dependent program states, and marks a path "feasible" if each program point follows from the preceding ones.

6.3 Assessment (Validation)

We used reputable sites in our review protocol (cf. Section 4), which resulted in the discovery of high-quality refereed research papers from respectable venues. To develop better insights into the quality of the research papers surveyed, here we use Evaluation Method (T 3.1) and Replicability (T 3.2), which are the two validation dimensions in the taxonomy.

Table 7 presents a summary of the validation-specific aspects that are extracted from the collection of papers included in the literature review. In the following, we summarize the main results for each dimension in this category.

6.3.1 Evaluation Method

The first part of Table 7 depicts the share of different evaluation methods in assessing the quality of Android security analysis approaches. Most of the approaches have used empirical techniques to assess the validity of their ideas using a full implementation of their approach (e.g., Chabada [299], CHEX [456], Epicc [218], and COVERT [205]). Some research efforts (28 percent) have developed a proof-of-concept prototype to perform limited scale case studies (e.g., SCanDroid [247] and SmartDroid [419]). A limited number (2 percent) of approaches (e.g., Chaudhuri et al. [509]) have provided mathematical proofs to validate their ideas.

Availability of various Android app repositories, such as the Google Play Store [510], is a key enabling factor for the large-scale empirical evaluation witnessed in the Android security research. Fig. 10 shows the distribution of surveyed research based on the number of selected apps that are used in the experiments. We observe that most of the experiments (72 percent) have been conducted over sets of more than one hundred apps.

Fig. 11 depicts the distribution of app repositories used in the evaluations of surveyed research. We observe that the Google Play Store, the official and largest repository of Android applications, is the most popular app repository, used by 85 percent of the papers with an empirical evaluation. There are several other third-party repositories, such as F-Droid open source repository [511], used by 24 percent of the evaluation methods. A number of malware repositories (such as [26], [314], [317], [512], [513]) are also widely used in assessing approaches designed for detecting malicious apps (45 percent). Finally, about 14 percent of the evaluations use hand-crafted benchmark suites, such as [514], [515], in their evaluation. A benefit of apps comprising such benchmarks is that the ground-truth for them is known, since they are manually seeded with known vulnerabilities and malicious behavior, allowing researchers to easily assess and compare their techniques in terms of the number of issues that are correctly detected.

Finally, a few papers (2 percent) assess their proposed approach by conducting controlled experiments on a set of users, either app developers (e.g., measuring development overhead in [353]), or app consumers (e.g., studying user reactions in [444]).

TABLE 7
Assessment Specific Categorization of the Reviewed Research

Dimension			Approaches	%
Evaluation	Proof		Apposcopy [162], Bagheri_ [44], Cassandra [274], HornDroid [83], ScanDal [243], SCanDroid [168], Scoria [423], Smith_ [400]	2%
	Empirical	Google Play	AAPL [275], AASandbox [69], Adagio [172], AdDroid [323], Adebayo_ [19], AdRisk [188], Amandroid [440], AMDetector [492], Ananas [133], AnDarwin [112], AndroidLeaks [178], AntiMalDroid [491], ApkCombiner [256], ApkRiskAnalyzer [103], App-ray [419], Apparecium [418], AppAudit [451], AppCaulk [380], AppContext [470], AppCracker [82], AppFence [207], AppGuard [42], AppInspector [179], AppIntegrity [425], AppIntent [473], Apposcopy [162], AppProfiler [355], AppSealer [482], AppsPlayground [342], APSET [364], Aquifer [305], AsDroid [215], AuDroid [326], AutoCog [335], AVDTester [211], Bae_ [43], Bagheri_ [44], Bartsch_ [58], Batyuk_ [59], BayesDroid [420], Bianchi_ [65], Bifocals [105], BlueSeal [206], Capper [483], Cen_ [91], Chabada [184], Chen_ [97], Chen2_ [96], CHEX [276], CMA [389], ComDroid [104], ConDroid [379], ContentScope [510], COPES [55], COVERT [45], CredMiner [514], CryptoLint [134], Dagger [466], Dai_ [117], DataChest [512], DNADroid [111], Dr.Android [220], DRACO [63], DroidADDMiner [264], DroidAnalytics [499], DroidAnalyzer [382], DroidAPIMiner [14], DroidChecker [94], DroidCIA [101], DroidDolphin [450], DroidForce [340], DroidGuard [46], DroidKin [182], DroidMat [447], DroidMiner [465], DroidRanger [513], DroidRay [500], DroidRisk [438], Droidsearch [339], DroidSIFT [481], DroidSim [412], DroidTest [356], DroidTrace [501], Duet [208], Enck_ [142], Epicc [315], Flowdroid [35], FUSE [346], Gallingani_ [170], Gates_ [174], Han_ [194], Harehunter [15], HornDroid [83], HunterDroid [479], I-ARM-Droid [122], ICC_Map [137], IccTA [257], IntentFuzzer [467], IVDroid [148], Jiao_ [224], Johnson_ [232], Juxtapp [196], Kantola_ [235], Kate_ [237], Kirin [143], LeakMiner [472], Leontiadis_ [254], Ma_ [283], MalloDroid [146], Manilyzer [156], Marvin [267], MassVet [98], MAST [93], MeadDroid [253], MockDroid [61], Moonsamy2_ [298], MorphDroid [163], MpDroid [417], Mudflow [38], Mutchler_ [301], NDroid [333], NoFrak [176], NoInjection [226], Onwuzurike_ [318], OpSeq [22], Patronus [411], PCLeaks [258], Pedal [270], Pegasus [99], Peiravian_ [324], Peng_ [325], PermCheckTool [427], PermissionFlow [372], Permlyzer [461], Poeplau_ [328], PREC [203], ProfileDroid [441], PUMA [365], PUMA2 [198], PuppetDroid [177], Quan_ [336], RAMSES [132], Relda [190], Ren_ [349], ResDroid [390], RetroSkeleton [121], Riskmon [227], RiskMon2 [228], Riskranker [187], SAAF [205], Sarma_ [369], ScanDal [243], SEFA [449], Seneviratne_ [381], Shabtai_ [383], Shen_ [395], SherlockDroid [29], Short_ [398], SMV-HUNTER [403], StaDynA [494], Stowaway [157], SUPOR [214], TaintDroid [141], TISSA [515], TMSVM [453], TongxinLi_ [262], UID [139], UIPicker [306], Uranine [345], VetDroid [489], Wang_ [436], WeChecker [114], WifiLeaks [17], Wognsen_ [445], Woodpecker [189], Xmandroid [75], Zhou_ [506], Zuo_ [517]	53%
		Third Party	Adagio [172], Afonso_ [20], AnDarwin [112], AndroidLeaks [178], APKLancet [471], AppCracker [82], AppIntegrity [425], AppProfiler [355], APSET [364], AsDroid [215], Aurasium [460], Bagheri_ [44], Barros_ [54], Chen_ [97], CHEX [276], CoChecker [115], ContentScope [510], COVERT [45], CredMiner [514], DNADroid [111], Drebin [32], DroidAnalytics [499], DroidAnalyzer [382], DroidAPIMiner [14], DroidChecker [94], DroidGuard [46], DroidLegacy [124], DroidMiner [465], DroidMOSS [507], DroidRanger [513], DroidSIFT [481], FUSE [346], HunterDroid [479], Isohara_ [217], Jiao_ [224], Juxtapp [196], Kim_ [242], MassVet [98], Mobile-Sandbox [404], MobSafe [459], Moonsamy_ [297], Moonsamy2_ [298], PaddyFrog [448], Relda [190], Riskranker [187], WuKong [431], Zhou_ [506]	14%
		Malware Collections	A5 [428], Adagio [172], Adebayo_ [19], Afonso_ [20], Amandroid [440], AMDetector [492], Ananas [133], ApkCombiner [256], APKLancet [471], ApkRiskAnalyzer [103], AppAudit [451], AppContext [470], AppIntent [473], Apposcopy [162], AppsPlayground [342], AsDroid [215], Aurasium [460], Bae_ [43], Bianchi_ [65], BlueSeal [206], Brave [321], Brox [284], Cen_ [91], Chabada [184], Chen2_ [96], CopperDroid [347], CopperDroid2 [414], COVERT [45], Dagger [466], Dai_ [117], Dendroid [407], DRACO [63], DroidADDMiner [264], DroidAlarm [502], DroidAnalytics [499], DroidAPIMiner [14], DroidDolphin [450], DroidGuard [46], DroidKin [182], DroidLegacy [124], DroidLogger [118], DroidMat [447], DroidMiner [465], DroidPAD [279], DroidPermissionMiner [36], DroidRisk [438], DroidScope [464], Droidsearch [339], DroidSIFT [481], DroidSim [412], DroidTrace [501], EASEAndroid [435], Elish_ [138], Flowdroid [35], FUSE [346], Gates_ [174], GroddDroid [16], Ham_ [192], Han_ [194], IccTA [257], IIF [477], IREA [249], Jeong_ [222], Jiao_ [224], Juxtapp [196], Kadir_ [234], Karami_ [236], Kate_ [237], Kim_ [242], Lee_ [252], Ma_ [283], Mama [366], Manilyzer [156], Marvin [267], MAST [93], MIGDroid [209], Mobile-Sandbox [404], MpDroid [417], Mudflow [38], OpSeq [22], Patronus [411], Pegasus [99], Peiravian_ [324], Peng_ [325], PREC [203], PUMA2 [198], PuppetDroid [177], Quan_ [336], RAMSES [132], ResDroid [390], SAAF [205], Sanz_ [368], Sarma_ [369], SherlockDroid [29], StaDynA [494], TMSVM [453], UID [139], VetDroid [489], Wang_ [436], Xmandroid [75]	30%
		Benchmark	AAPL [275], Amandroid [440], Anadroid [265], ApkCombiner [256], APKLancet [471], AppAudit [451], BayesDroid [420], CoChecker [115], DidFail [246], DroidBarrier [24], DroidGuard [46], DroidPAD [279], DroidSafe [183], DroidScope [464], Enck_ [142], FireDroid [358], Flowdroid [35], FUSE [346], HornDroid [83], IccTA [257], IFT [145], Kynoid [377], MorphDroid [163], MOSES [496], NDroid [333], QUIRE [127], Smith_ [400], WeChecker [114]	8%
	Case Study		Achara_ [18], AdRisk [188], AdSplit [393], Amandroid [440], Androguard [126], Apex [308], APKLancet [471], AppCaulk [380], AppCracker [82], AppGuard [42], AppIntent [473], AppsPlayground [342], Aquifer [305], AsDroid [215], ASM [202], ASV [212], Aurasium [460], AuthDroid [432], Bagheri_ [44], Bartsch_ [58], Batyuk_ [59], BlueSeal [206], Buhov_ [80], Buzzer [88], Canfora3_ [87], ComDroid [104], Compac [439], ContentScope [510], COVERT [45], COVERT_Tool [359], CredMiner [514], CRePE [107], Dagger [466], Defensor [319], Desnos_ [125], DexDiff [296], DidFail [246], DNADroid [111], DroidAlarm [502], DroidChecker [94], DroidCIA [101], DroidGuard [46], DroidRanger [513], DroidScope [464], DroidTrace [501], Enck_ [142], FineDroid [488], FlaskDroid [78], Flowdroid [35], FUSE [346], Gallo_ [171], Graa_ [186], Graa2_ [185], Harehunter [15], HornDroid [83], IIF [477], Jeong_ [222], Jia_ [223], Juxtapp [196], Kadir_ [234], KLD [387], LayerCake [352], Li2_ [261], Lintent [79], Lu_ [277], Mann_ [289], Morbs [434], NDroid [333], PCLeaks [258], Pegasus [99], PICARD [129], Poeplau_ [328], PuppetDroid [177], Riskmon [227], RiskMon2 [228], Riskranker [187], SCanDroid [168], Scoria [423], SecUP [458], SEFA [449], SFG [27], Shebaro_ [391], Shen_ [395], SmartDroid [497], Smith_ [400], Song_ [401], STAMBA [71], SUPOR [214], TaintDroid [141], Tchakounte_ [415], TreeDroid [119], VetDroid [489], Woodpecker [189]	28%
	User Study (D)		AppProfiler [355], Grab'nRun [147]^D, IFT [145]^D, MalloDroid [146], Riskmon [227], RiskMon2 [228], WifiLeaks [17], Wijesekera_ [444]	2%
Replicability	Available Tool		A5 [428], Adagio [172], Amandroid [440], Androguard [126], AndroTotal [285], Andrubis [268], ApkCombiner [256], Apparecium [418], AppContext [470], AppGuard [42], AppProfiler [355], Aquifer [305], ASM [202], Aurasium [460], AutoCog [335], Barros_ [54], Brahmastra [64], Chabada [184], ComDroid [104], CopperDroid [347], CopperDroid2 [414], COVERT [45], COVERT_Tool [359], Dendroid [407], Desnos_ [125], DidFail [246], DroidForce [340], DroidSafe [183], DroidScope [464], Enck_ [142], Epicc [315], FlaskDroid [78], Flowdroid [35], FUSE [346], Geneiatakis_ [175], Grab'nRun [147], IccTA [257], IFT [145], Kirin [143], LayerCake [352], Lintent [79], MalloDroid [146], Marvin [267], Mobile-Sandbox [404], MockDroid [61], Morbs [434], Mudflow [38], NoInjection [226], PermCheckTool [427], PScout [37], SCanDroid [168], StaDynA [494], Stowaway [157], TaintDroid [141], TraceDroid [422], Wognsen_ [445]	17%
	Available Source Code		A5 [428], Adagio [172], Amandroid [440], Androguard [126], ApkCombiner [256], Apparecium [418], AppContext [470], Aquifer [305], ASM [202], Barros_ [54], Desnos_ [125], DidFail [246], DroidSafe [183], DroidScope [464], Enck_ [142], FlaskDroid [78], Flowdroid [35], Geneiatakis_ [175], Grab'nRun [147], IccTA [257], Kirin [143], LayerCake [352], Lintent [79], MalloDroid [146], MockDroid [61], Morbs [434], NoInjection [226], PermCheckTool [427], PScout [37], SCanDroid [168], StaDynA [494], TaintDroid [141], Wognsen_ [445]	10%

D: App Developer.

6.3.2 Replicability

The evaluation of security research is generally known to be difficult. Making the results of experiments reproducible is even more difficult. Table 7 shows the availability of the executable artifacts, as well as the corresponding source code and documentations in the surveyed papers. According to Table 7, overall only 17 percent of published research have made their artifacts publicly available. The rest have not made their implementations, prototypes, tools, and experiments available to other researchers.

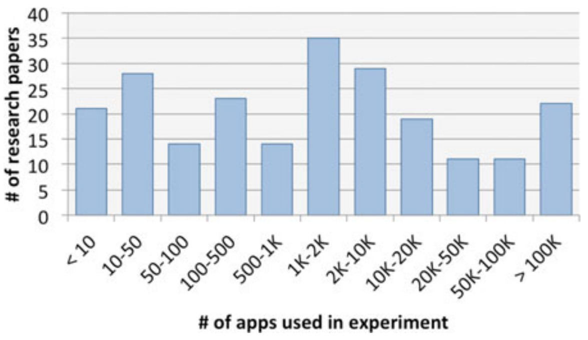

Fig. 10. Distribution of surveyed research based on the number of apps used in their experiments.

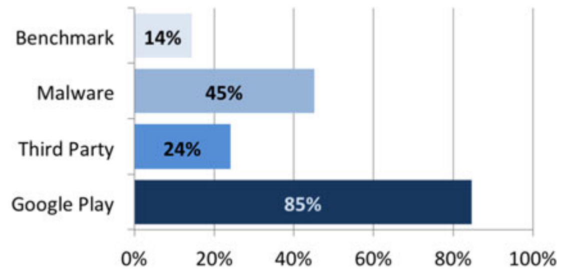

Fig. 11. Distribution of app repositories used in the empirical evaluations.

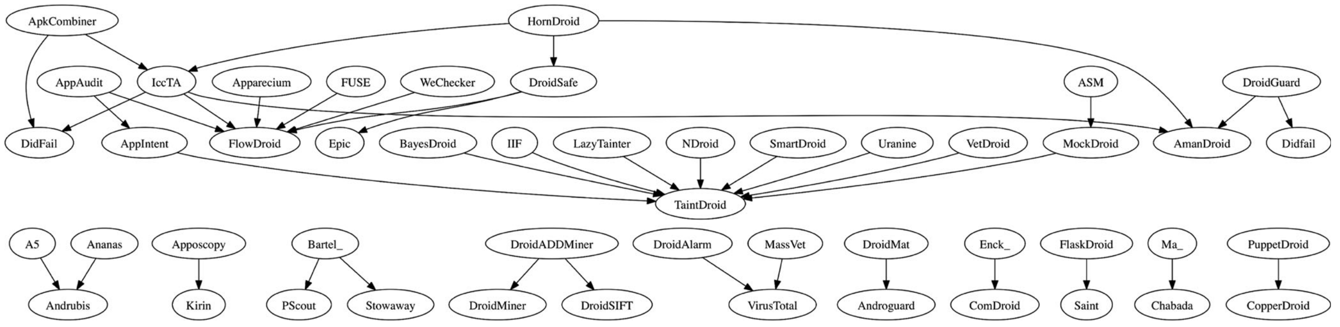

Fig. 12. Comparison graph: X → Y means research method X has quantitatively compared itself to method Y.

Having such artifacts publicly available enables, among other things, quantitative comparisons of different approaches. Fig. 12 depicts the comparison relationships found in the evaluation of the studied papers. In this graph, the nodes with higher fan-in (i.e., incoming edges) represent the tools that are widely used in evaluation of other research efforts. For instance, Enck et al. [423] provided a stable, well-documented monitoring tool, *TaintDroid*, which is widely used in the literature as the state-of-the-art dynamic analysis for evaluating the effectiveness of the newly proposed techniques.

Similarly, making a research tool available, particularly in the form of source code, enables other researchers to expand the tool and build more advanced techniques on top of it. Fig. 13 illustrates the dependency relationships found in the implementation of the surveyed papers. In this graph, the nodes with higher fan-in represent the tools that are widely used to realize the other research efforts. For instance, *FlowDroid* [104], with six incoming edges, has an active community of developers and a discussion group—and is widely used in several research papers surveyed in our study.

6.4 Cross Analysis

In this section, we extend our survey analysis across the different taxonomy dimensions. Given the observations from the reviewing process, we develop the following cross-analysis questions (CQs):

- *CQ1.* What types of program analysis have been used for each security assessment objectives?
- *CQ2.* What types of program analysis have been used for detecting each of the STRIDE's security threats?
- *CQ3.* Is there a relationship between the granularity of security threats and the type of employed program analysis techniques?
- *CQ4.* Is there a relationship between the depth of security threats, i.e., app-level versus framework-level, and the type of analysis techniques employed in the surveyed research?
- *CQ5.* Which evaluation methods are used for different objectives and types of analysis?
- *CQ6.* How reproducible are the surveyed research based on the objectives and types of analysis?
- *CQ7.* Is there a relationship between the availability of research artifacts and their respective citation numbers?

CQ1. Analysis Objectives and Types of Program Analysis. As shown in Fig. 14ⓐ, static and dynamic analyses have been used for identifying both malicious behavior and vulnerabilities. However, static approaches are more frequently leveraged for detecting vulnerable apps rather than malware (58 versus 50 percent), while dynamic techniques have more application in malware detection compared to vulnerability analysis (36 versus 27 percent). Hybrid approaches, though at lower scales, have also been used (15-16 percent) for both purposes.

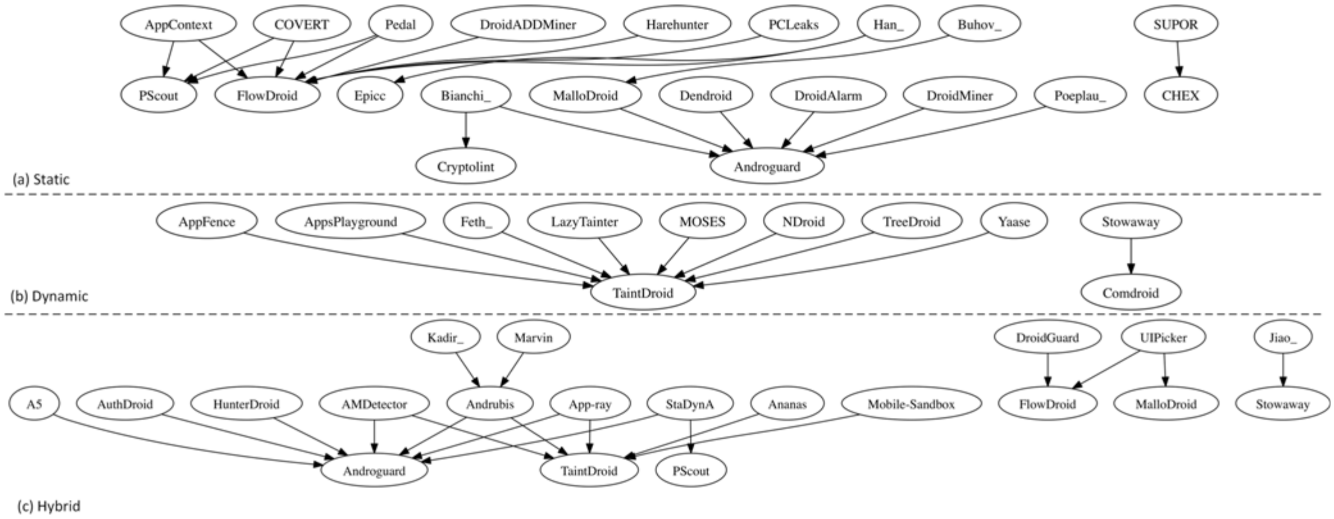

Fig. 13. Dependency graph: X → Y means research method X is built on top of method Y.

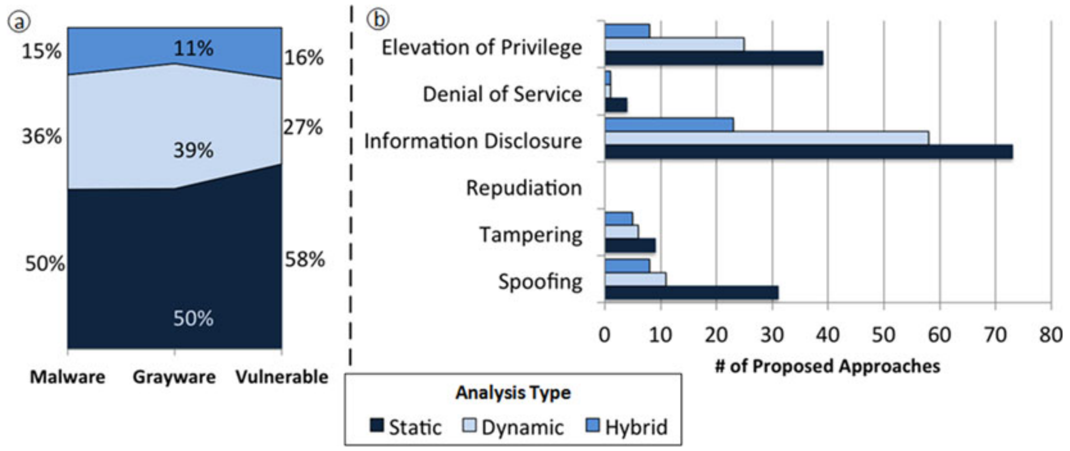

Fig. 14. Types of program analysis that have been used for detecting ⓐ different security assessment objectives (i.e., malware versus vulnerability detection), and ⓑ different security threats.

CQ2. STRIDE's Security Threats and Type of Program Analysis. According to Fig. 14ⓑ, none of the analysis types (i.e., static, dynamic, hybrid) are intended to identify the repudiation security thread (see discussion and gap analysis in Section 7). Moreover, according to this figure, a limited number of research efforts have been devoted to identifying Denial of Service attacks. Finally, the cross analysis results show that static approaches, compared to the other types of program analysis, has been widely used for detecting various security threats, particularly spoofing, where the number of static techniques is almost three times higher than the number of dynamic or hybrid approaches. As discussed before, one reason is that for security analysis soundness is usually considered to be more important than precision, since it is preferred to not miss any security threat, even at the cost of generating false warnings.

CQ3. Granularity of Security Threats and Type of Analysis Techniques. In Fig. 15ⓐ, we observe a similar distribution pattern in use of different analysis types (i.e., static, dynamic, hybrid) for capturing security threats at different levels of granularity (i.e, intra-component, inter-component, inter-app). In general, for identifying security threats in a single component, or between multiple component in a single or multiple apps, static analysis techniques are the most common methods (about 50 percent) used by the state-of-the-art approaches, followed by dynamic analysis (35 percent), and hybrid techniques (15 percent).

CQ4. Depth of Security Threats and Type of Analysis Techniques. The depth of security threats also exhibit a relation with the type of analysis techniques (cf., Fig. 15ⓑ). We observe that the dynamic approaches are employed more often for analysis at the framework-level (55 percent). One reason is that dynamic approaches can employ runtime modules, such as monitors, which are deployed in the Android framework, thereby enabling tracking otherwise implicit relations between system API calls and the Android permissions. Such *runtime* framework-level activity monitoring is not possible using static analysis techniques. Moreover, due to the large size of Android framework (over ten million lines of code), dynamic techniques are more scalable and less-expensive for framework-level monitoring.

CQ5. Evaluation Method versus the Objectives and Types of Analysis. We observe a similar distribution pattern in use of different evaluation methods across various types of analysis and also analysis objectives, except that user study is more popular in grayware analysis, compared to the other objectives. One reason is that the privacy concerns of end users are critical in assessing grayware, such as ad libraries. In general, empirical evaluation is the most widely used, followed by the case study and user study methods and formal proofs (cf., Fig. 16).

CQ6. Reproducibility versus the Objectives and Types of Analysis. As shown in Fig. 17, the research artifacts intended to identify security vulnerabilities are more likely to be available in comparison to those designed for malware/

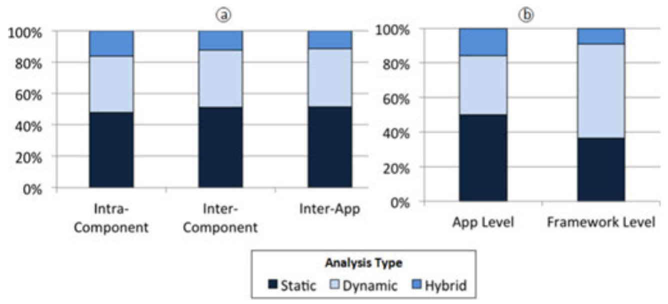

Fig. 15. ⓐ Granularity and ⓑ depth (level) of each type of program analysis.

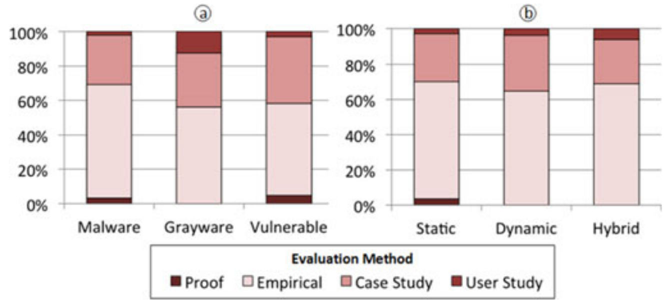

Fig. 16. Approach validation versus ⓐ research objectives and ⓑ types of analysis.

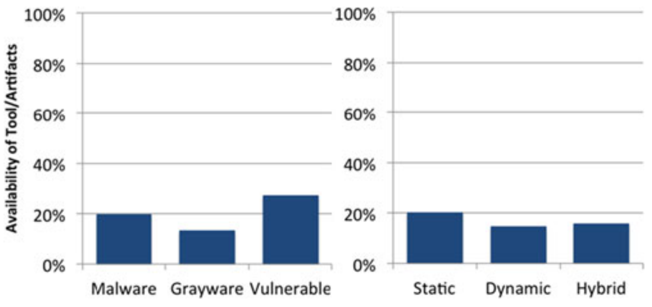

Fig. 17. Availability of tools/artifacts based on the ⓐ objective and, ⓑ type of analysis.

grayware detection (27 versus 20 percent/13 percent). Moreover, availability ratio of the tools performing different types of analysis (i.e., static, dynamic, and hybrid) are all close and under 20 percent, which restricts the other researchers from reproducing, and potentially adopting, achievements in this thrust of research.

CQ7. Artifact Availability and Citation Count. To investigate this research question, we ranked the surveyed papers based on their citation counts. Since older papers have a higher chance of getting more citations, we also provided the same ranking for each year, separately, from 2009 to 2015. Afterwards, we checked the artifact availability of highly cited papers. The summary of our findings are provided in Table 8, which indicates that papers with publicly available artifacts get more citations. 100 percent of overall top-5 cited, and 88 percent of top-cited papers of each year, have available artifacts.

7 Discussion and Directions for Future Research

To address the third research question (RQ3), in this section, we first provide a trend analysis of surveyed research, and then discuss the observed gaps in the studied literature that can help to direct future research efforts in this area.

Based on the results of our literature review (cf., Section 6), it is evident that Android security has received a lot of attention in recently published literature, due mainly to the popularity of Android as a platform of choice for mobile devices, as well as increasing reports of its vulnerabilities. We also observe important trends in the past decade, as reflected by the results of the literature review. Fig. 18 shows some observed trends in Android security analysis research.

TABLE 8
Artifact Availability of Highly Cited Research Papers

Rank	Year	Tool	# of Citations*	Availability
		Top cited papers—overall		
1	2010	TaintDroid[423]	1,563	✓
2	2011	Stowaway[153]	745	✓
3	2009	Kirin[364]	625	✓
4	2011	Enck_[146]	599	✓
5	2011	ComDroid[3]	502	✓
		Top cited papers—yearly		
1	2009	Kirin[364]	625	✓
1	2010	TaintDroid[423]	1,563	✓
1	2011	Stowaway[153]	745	✓
1	2012	DroidRanger[329]	429	×
1	2013	AppsPlayground[281]	129	✓
1	2014	Flowdroid[104]	225	✓
1	2015	IccTA[225]	21	✓

By the end of 2015.

- According to Fig. 18ⓐ, malicious behavior detection not only has attracted more attention, compared to vulnerability identification, but also research in malware analysis tends to grow at an accelerated rate.
- As illustrated in Fig. 18ⓑ, static analysis techniques dominate security assessment in the Android domain. Dynamic and hybrid analysis techniques are also showing modest growth, as they are increasingly applied to mitigate the limitations of pure static analysis (e.g., to reason about dynamically loaded code, and obfuscated code).
- The more recent approaches reviewed in this survey have used larger collections of apps in their evaluation (cf., Fig. 18ⓒ). Such large-scale empirical evaluation in the Android security research is promising, and can be attributed to the meteoric rise of the numbers of apps provisioned on publicly available app markets that in some cases provide free or even open-source apps.

Despite considerable research efforts devoted to mitigating security threats in mobile platforms, we are still witnessing a significant growth in the number of security attacks targeting these platforms [516]. Therefore, our first and foremost recommendation is to increase convergence and collaboration among researchers in this area from software engineering, security, mobility, and other related

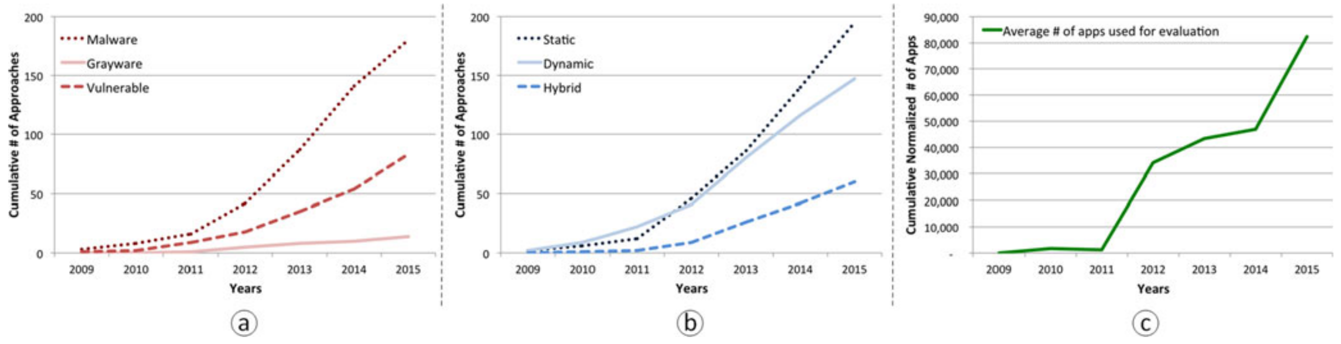

Fig. 18. Observed trends in Android security analysis research with respect to ⓐ objectives of the analysis, ⓑ type of analysis, and ⓒ number of apps used in the evaluation (normalized by dividing the number of apps to the number of publications in each year).

communities to achieve the common goal of addressing these mobile security threats and attacks.

More specifically, the survey—through its use of our proposed taxonomy—has revealed research gaps in need of further study. To summarize, future research needs to focus on the following to stay ahead of today's advancing security threats:

- *Pursue integrated and hybrid approaches that span not only static and dynamic analyses, but also other supplementary analysis techniques:* Recall from Table 5 that only 29 percent of approaches leverage supplementary techniques, which are shown to be effective in identifying modern malicious behaviors or security vulnerabilities.
- *Move beyond fuzzing for security test input generation:* According to Section 6.2.8, only 8 percent of test input generation techniques use a systematic technique (i.e., symbolic execution or heuristic-based fuzzing), as opposed to brute-force fuzzing. Fuzzing is inherently limited in its abilities to execute vulnerable code. Furthermore, such brute-force approaches may fail to identify malicious behavior that may be hidden behind obfuscated code or code that requires specific conditions to execute.
- *Continue the paradigm shift from basic single app analysis to overall system monitoring, and exploring compositional vulnerabilities:* Recall from Sections 6.1.3 and 6.1.4, and Table 4, that the majority of the existing body of research is limited to the analysis of single apps in isolation. However, malware exploiting vulnerabilities of multiple benign apps in tandem on the market are increasing. Furthermore, identifying some security vulnerabilities requires a holistic analysis of the Android framework. For example, consider the analysis of the Android permission protocol to check whether it satisfies the security requirement of preventing unauthorized access [195]. Ensuring that the system achieves such security goals, however, is a challenging task, inasmuch as it can be difficult to predict all the ways in which a malicious application may attempt to misuse the system. Identifying such attacks, indeed, requires system-wide reasoning, and cannot be easily achieved by analysis of individual parts of the system in isolation.
- *Construct techniques capable of analyzing ICC beyond Intents:* Only 3 percent of papers, as shown in Table 4, consider ICCs involving data sharing using Content Providers and AIDL. These mechanisms are, thus, particularly attractive vectors for attackers to utilize, due to the limited analyses available. Consequently, research in that space can help strengthen countermeasures against such threats.
- *Consider dynamically loaded code that is not bundled with installed packages:* Recall from Table 4 that a highly limited amount of research (4 percent) analyzes the security implications of externally loaded code. This Android capability can be easily exploited by malware developers to evade security inspections at installation time.
- *Analyze code of different forms and from different languages:* Besides analyzing Java and its basic constructs, future research should analyze other code constructs and languages used to construct Android apps, such as native C/C++ code or obfuscated code. The usage of complicated obfuscation techniques and/or native libraries for hiding malicious behavior are continually growing. Recall from Section 6.1.5 and Table 4 that only 5-6 percent of surveyed approaches consider obfuscated or native code, where most of those approaches do not perform analysis on the content of such code.
- *Improve the precision of analysis:* Recall from Section 6.2.5 and Table 6 that a low percentage (3-23 percent) of static analysis techniques use high precision sensitivities, leading to high false positives. Moreover, in parallel to enhancing precision, a practical analysis is also needed to scale up to large and complicated apps.
- *Consider studying Android repudiation:* The SLR process returned no results for Android repudiation, as shown in Table 3. Consequently, there is a need for studies that target such threats, particularly in terms of potential weaknesses in the way Android app ecosystem handles digital signatures and certificates. However, repudiation also has a major legal component [517], which may require expertise not held by researchers in software security, software engineering, or computer science. Properly addressing this gap may require inter-disciplinary research.
- *Promote collaboration in the research community:* To that end, we recommend making research results more reproducible. This goal can be achieved through increased sharing of research artifacts. Recall from Table 7 that less than 20 percent of surveyed papers have made their research artifacts available publicly. At the same time, Fig. 12 shows that few approaches conduct quantitative comparisons, mainly due to unavailability of prior research artifacts. Papers that make their artifacts available publicly are able to make a bigger impact, as measured by the citation count (recall Table 8). We hope this will provide another impetus for the research community to publicly share their tools and artifacts. To further aid in achieving reproducibility, we also advocate the development of common evaluation platforms and benchmarks. Recall from Fig. 11 that only 14 percent of studied approaches considered benchmarks for their evaluation. A benchmark of apps with known set of issues allows the research community to compare strengths and weaknesses of their techniques using the same dataset, thus fostering progress in this area of research.

8 Conclusion

In parallel with the growth of mobile applications and consequently the rise of security threats in mobile platforms, considerable research efforts have been devoted to assess the security of mobile applications. Android, as the dominant mobile platform and also the primary target of mobile malware threats, has been in the focus of much research. Existing research has made significant progress towards detection and mitigation of Android security.

This article proposed a comprehensive taxonomy to classify and characterize research efforts in this area. We have carefully followed the systematic literature review process, resulting in the most comprehensive and elaborate investigation of the literature in this area of research, comprised of 336 papers published from 2008 to the beginning of 2016. Based on the results of our literature review, it is evident that Android security has received much attention in recently published literature, due mainly to the popularity of Android as a platform of choice for mobile devices, as well as increasing reports of its vulnerabilities and malicious apps. The research has revealed patterns, trends, and gaps in the existing literature, and underlined key challenges and opportunities that will shape the focus of future research efforts.

In particular, the survey showed the current research should advance from focusing primarily on single app assessment to a more broad and deep analysis that considers combinations of multiple apps and Android framework, and also from pure static or dynamic to hybrid analysis techniques. We also identified a gap in the current research with respect to special vulnerable features of the Android platform, such as native or dynamically loaded code. Finally, we encourage researchers to publicly share their developed tools, libraries, and other artifacts to enable the community to compare and evaluate their techniques and build on prior advancements. We believe the results of our review will help to advance the much needed research in this area and hope the taxonomy itself will become useful in the development and assessment of new research directions.

ACKNOWLEDGMENTS

This work was supported in part by awards CCF-1252644, CNS-1629771 and CCF-1618132 from the National Science Foundation, D11AP00282 from the Defense Advanced Research Projects Agency, W911NF-09-1-0273 from the Army Research Office, HSHQDC-14-C-B0040 from the Department of Homeland Security, and FA95501610030 from the Air Force Office of Scientific Research.

REFERENCES

[1] R. Cozza, I. Durand, and A. Gupta, “Market share: Ultramobiles by region, OS and form factor, 4Q13 and 2013,” Gartner Inc., Stamford, CT, USA, Feb. 2014.

[2] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, “Google Android: A comprehensive security assessment,” *IEEE Secur. Privacy*, vol. 8, no. 2, pp. 35–44, Mar./Apr. 2010.

[3] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, “Analyzing inter-application communication in Android,” presented at the 9th Int. Conf. Mobile Syst. Appl. Services, Bethesda, MD, USA, Jun. 28–Jul. 01, 2011.

[4] L. Davi, A. Dmitrienko, A. Sadeghi, and M. Winandy, “Privilege escalation attacks on Android,” presented at the 13th Int. Conf. Inf. Security, Boca Raton, FL, USA, Oct. 25–28, 2010.

[5] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. S. Wallach, “Quire: Lightweight provenance for smart phone operating systems,” presented at the 20th USENIX Security Symp., San Francisco, CA, USA, Aug. 8–12, 2011.

[6] W. Shin, S. Kwak, S. Kiyomoto, K. Fukushima, and T. Tanaka, “A small but non-negligible flaw in the Android permission scheme,” presented at the IEEE Int. Symp. Policies Distrib. Syst. Netw., Fairfax, VA, USA, Jul. 21–23, 2010.

[7] Z. Fang, W. Han, and Y. Li, “Permission based Android security: Issues and countermeasures,” *Comput. Secur.*, vol. 43, pp. 205–218, 2014.

[8] A. Egners, U. Meyer, and B. Marschollek, “Messing with Android’s permission model,” presented at the 11th IEEE Int. Conf. Trust Secur. Privacy Comput. Commun., Liverpool, U. K., Jun. 25–27, 2012.

[9] S. Bugiel, S. Heuser, and A.-R. Sadeghi, “Flexible and fine-grained mandatory access control on Android for diverse security and privacy policies,” presented at the 22th USENIX Secur. Symp., Washington, DC, USA, Aug. 14–16, 2013.

[10] S. Smalley and R. Craig, “Security enhanced (SE) Android: Bringing flexible MAC to Android,” presented at the 20th Annu. Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 24–27, 2013.

[11] J. Jeon, et al., “Dr. Android and Mr. Hide: Fine-grained permissions in Android applications,” presented at the ACM Workshop Secur. Privacy Smartphones Mobile Devices, Raleigh, NC, USA, Oct. 19, 2012.

[12] A. P. Felt, S. Hanna, E. Chin, H. J. Wang, and E. Moshchuk, “Permission re-delegation: Attacks and defenses,” presented at the 20th USENIX Secur. Symp., San Francisco, CA, USA, Aug. 8–12, 2011.

[13] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, and A.-R. Sadeghi, “XManDroid: A new Android evolution to mitigate privilege escalation attacks,” Technische Universitt Darmstadt, Darmstadt, Germany, Tech. Rep. TR-2011–04, 2011.

[14] C. Cao, Y. Zhang, Q. Liu, and K. Wang, “Function escalation attack,” presented at the 10th Int. Conf. Secur. Privacy Commun. Netw., Beijing, China, Sep. 24–26, 2014.

[15] P. G. Kelley, S. Consolvo, L. F. Cranor, J. Jung, N. M. Sadeh, and D. Wetherall, “A conundrum of permissions: Installing applications on an Android smartphone,” presented at the Workshops Financial Cryptography Data Secur., Kralendijk, Bonaire, Mar. 2, 2012.

[16] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, “Android permissions: User attention, comprehension, and behavior,” presented at the Symp. Usable Privacy Secur., Washington, DC, USA, Jul. 11–13, 2012.

[17] H. Zhu, H. Xiong, Y. Ge, and E. Chen, “Mobile app recommendations with security and privacy awareness,” presented at the 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, New York, NY, USA, Aug. 24–27, 2014.

[18] N. Wang, B. Zhang, B. Liu, and H. Jin, “Investigating effects of control and ads awareness on Android users’ privacy behaviors and perceptions,” presented at the 17th Int. Conf. Human-Comput. Interaction Mobile Devices Services, Copenhagen, Denmark, Aug. 24–27, 2015.

[19] D. Dagon, T. Martin, and T. Starner, “Mobile phones as computing devices: The viruses are coming!” *IEEE Pervasive Comput.*, vol. 3, no. 4, pp. 11–15, Oct.-Dec. 2004.

[20] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, “A survey of mobile malware in the wild,” presented at the 1st ACM Workshop Secur. Privacy Smartphones Mobile Devices, Chicago, IL, USA, Oct. 17, 2011.

[21] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and A. Ribagorda, “Evolution, detection and analysis of malware for smart devices,” *IEEE Commun. Surveys Tutorials*, vol. 16, no. 2, pp. 961–987, Apr.–Jun. 2014.

[22] A. Amamra, C. Talhi, and J.-M. Robert, “Smartphone malware detection: From a survey towards taxonomy,” presented at the 7th Int. Conf. Malicious Unwanted Softw., Fajardo, PR, USA, Oct. 16–18, 2012.

[23] M. Haris, H. Haddadi, and P. Hui, “Privacy leakage in mobile computing: Tools, methods, and characteristics,” *arXiv:1410.4978 [cs]*, Oct. 2014.

[24] W. Enck, “Defending users against smartphone apps: Techniques and future directions,” presented at the 7th Int. Conf. Inf. Syst. Secur., Kolkata, India, Dec. 15–19, 2011.

[25] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, and S. Dolev, “Google Android: A state-of-the-art review of security mechanisms,” *arXiv:0912.5101 [cs]*, Dec. 2009.

[26] Y. Zhou and X. Jiang, “Dissecting Android malware: Characterization and evolution,” presented at the IEEE Symp. Secur. Privacy, San Francisco, California, USA, 21–23 May 2012.

[27] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan, “Android security: A survey of issues, malware penetration, and defenses,” *IEEE Commun. Surveys Tutorials*, vol. 17, no. 2, pp. 998–1022, Mar.–Apr. 2015.

[28] S. Neuner, V. van der Veen, M. Lindorfer, M. Huber, G. Merzdovnik, M. Mulazzani, and E. Weippl, “Enter sandbox: Android sandbox comparison,” *arXiv:1410.7749*, Oct. 2014.

[29] S. Schmeelk, J. Yang, and A. V. Aho, "Android malware static analysis techniques," presented at the 10th Annu. Cyber Inf. Secur. Res. Conf., Oak Ridge, TN, USA, Apr. 7–9, 2015.

[30] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, "A review on feature selection in mobile malware detection," *Digit. Investigation*, vol. 13, pp. 22–37, 2015.

[31] Sufatrio, D. J. J. Tan, T.-W. Chua, and V. L. L. Thing, "Securing Android: A survey, taxonomy, and challenges," *ACM Comput. Survey*, vol. 47, no. 4, 2015, Art. no. 58.

[32] B. Rashidi and C. Fung, "A survey of Android security threats and defenses," *J. Wireless Mobile Netw. Ubiquitous Comput. Dependable Appl.*, vol. 6, no. 3, pp. 3–35, 2015.

[33] B. Kitchenham, "Procedures for performing systematic reviews," Keele Univ., Keele, UK, Tech. Rep. TR/SE-0401, vol. 33, 2004, Art. no. 2004.

[34] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *J. Syst. Softw.*, vol. 80, no. 4, pp. 571–583, Apr. 2007.

[35] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "PIoS: Detecting privacy leaks in IoS applications," presented at the Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 6–9, 2011.

[36] Y. Agarwal and M. Hall, "Protectmyprivacy: Detecting and mitigating privacy leaks on IoS devices using crowdsourcing," presented at the 11th Annu. Int. Conf. Mobile Syst. Appl. Services, Taipei, Taiwan, Jun. 25–28, 2013.

[37] A. Bose, X. Hu, K. G. Shin, and T. Park, "Behavioral detection of malware on mobile handsets," presented at the 6th Int. Conf. Mobile Syst. Appl. Services, Breckenridge, CO, USA, Jun. 17–20, 2008.

[38] A.-D. Schmidt, J. H. Clausen, S. A. Çamtepe, and S. Albayrak, "Detecting symbian OS malware through static function call analysis," presented at the 4th Int. Conf. Malicious Unwanted Softw., Montral, Quebec, Canada, Oct. 13–14, 2009.

[39] B. Livshits and J. Jung, "Automatic mediation of privacy-sensitive resource access in smartphone applications," presented at the 22th USENIX Secur. Symp., Washington, DC, USA, Aug. 14–16, 2013.

[40] J. Cheng, S. H. Y. Wong, H. Yang, and S. Lu, "Smartsiren: Virus detection and alert for smartphones," presented at the 5th Int. Conf. Mobile Syst. Appl. Services, San Juan, Puerto Rico, Jun. 11–13, 2007.

[41] B. Liu, S. Nath, R. Govindan, and J. Liu, "Decaf: Detecting and characterizing ad fraud in mobile apps," presented at the 11th USENIX Symp. Netw. Syst. Des. Implementation, Seattle, WA, USA, Apr. 2–4, 2014.

[42] T. Werthmann, R. Hund, L. Davi, A.-R. Sadeghi, and T. Holz, "PSIoS: Bring your own privacy & security to IoS devices," presented at the 8th ACM Symp. Inf. Comput. Commun. Secur., Hangzhou, China, May 8–10, 2013.

[43] M. Bucicoiu, L. Davi, R. Deaconescu, and A.-R. Sadeghi, "XIoS: Extended application sandboxing on IoS," presented at the 10th ACM Symp. Inf. Comput. Commun. Secur., Singapore, Apr. 14–17, 2015.

[44] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi, and B. Shastry, "Practical and lightweight domain isolation on Android," presented at the 1st ACM Workshop Secur. Privacy Smartphones Mobile Devices, Chicago, IL, USA, Oct. 17, 2011.

[45] M. Conti, E. Fernandes, J. Paupore, A. Prakash, and D. Simionato, "Oasis: Operational access sandboxes for information security," presented at the 4th ACM Workshop Secur. Privacy Smartphones Mobile Devices, Scottsdale, AZ, USA, Nov. 3–7, 2014.

[46] C. Wu, Y. Zhou, K. Patel, Z. Liang, and X. Jiang, "Airbag: Boosting smartphone resistance to malware infection," presented at the 21st Annu. Netw. Distrib. Syst. Secur. Symp., San Diego, California, USA, Feb. 23–26, 2014.

[47] M. Backes, S. Bugiel, C. Hammer, O. Schranz, and P. V. Styp-Rekowsky, "Boxify: Full-fledged app sandboxing for stock Android," presented at the 24th USENIX Secur. Symp., Washington, DC, USA, Aug. 12–14, 2015.

[48] Y. Zhou, K. Patel, L. Wu, Z. Wang, and X. Jiang, "Hybrid user-level sandboxing of third-party Android apps," presented at the 10th ACM Symp. Inf. Comput. Commun. Secur., Singapore, Apr. 14–17, 2015.

[49] M. Lange, S. Liebergeld, A. Lackorzynski, A. Warg, and M. Peter, "L4android: A generic operating system framework for secure smartphones," presented at the 1st ACM Workshop Secur. Privacy Smartphones Mobile Devices, Chicago, IL, USA, Oct. 17, 2011.

[50] M. Sun and G. Tan, "Nativeguard: Protecting Android applications from third-party native libraries," presented at the 7th ACM Conf. Secur. Privacy Wireless Mobile Netw., Oxford, U. K., Jul. 23–25, 2014.

[51] R. Fedler, M. Kulicke, and J. Schtte, "Native code execution control for attack mitigation on Android," presented at the ACM Workshop Secur. Privacy Smartphones Mobile Devices, Berlin, Germany, Nov. 8, 2013.

[52] X. Li, G. Bai, B. Thian, Z. Liang, and H. Yin, "A light-weight software environment for confining Android malware," presented at the IEEE 8th Int. Conf. Softw. Secur. Rel., San Francisco, CA, USA, Jun. 30–Jul. 2, 2014.

[53] L. Yan, Y. Guo, and X. Chen, "Splitdroid: Isolated execution of sensitive components for mobile applications," presented at the 11th Int. Conf. Secur. Privacy Commun. Netw., Dallas, TX, USA, Oct. 26–29, 2015.

[54] A. P. Felt, S. Egelman, M. Finifter, D. Akhawe, and D. Wagner, "How to ask for permission," presented at the 7th USENIX Workshop Hot Topics Secur., Bellevue, WA, USA, Aug. 7, 2012.

[55] P. G. Kelley, L. F. Cranor, and N. M. Sadeh, "Privacy as part of the app decision-making process," presented at the ACM SIGCHI Conf. Human Factors Comput. Syst., Paris, France, Apr. 27–May 2, 2013.

[56] M. Harbach, M. Hettig, S. Weber, and M. Smith, "Using personal examples to improve risk communication for security & privacy decisions," presented at the CHI Conf. Human Factors Comput. Syst., Toronto, ON, Canada, Apr. 26–May 1, 2014.

[57] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. J. Wang, and C. Cowan, "User-driven access control: Rethinking permission granting in modern operating systems," presented at the IEEE Symp. Secur. Privacy, San Francisco, CA, USA, May 21–23, 2012.

[58] M. Kato and S. Matsuura, "A dynamic countermeasure method to Android malware by user approval," presented at the 37th Annu. IEEE Comput. Softw. Appl. Conf., Kyoto, Japan, Jul. 22–26, 2013.

[59] C. S. Gates, J. Chen, N. Li, and R. W. Proctor, "Effective risk communication for Android apps," *IEEE Trans. Dependable Secure Comput.*, vol. 11, no. 3, pp. 252–265, May/Jun. 2014.

[60] B. Shebaro, O. Oluwatimi, D. Midi, and E. Bertino, "Identidroid: Android can finally wear its anonymous suit," *Trans. Data Privacy*, vol. 7, no. 1, pp. 27–50, Apr. 2014.

[61] S. M. Kywe, C. Landis, Y. Pei, J. Satterfield, Y. Tian, and P. Tague, "Privatedroid: Private browsing mode for Android," presented at the 13th IEEE Int. Conf. Trust Secur. Privacy Comput. Commun., Beijing, China, Sep. 24–26, 2014.

[62] D. Wang, H. Yao, Y. Li, H. Jin, D. Zou, and R. H. Deng, "CICC: A fine-grained, semantic-aware, and transparent approach to preventing permission leaks for Android permission managers," presented at the 8th ACM Conf. Secur. Privacy Wireless Mobile Netw., New York, NY, USA, Jun. 22–26, 2015.

[63] F. Rohrer, Y. Zhang, L. Chitkushev, and T. Zlateva, "Dr Baca: Dynamic role based access control for Android," presented at the Annu. Comput. Secur. Appl. Conf., New Orleans, LA, USA, Dec. 9–13, 2013.

[64] Y. Fratantonio, A. Bianchi, W. K. Robertson, M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "On the security and engineering implications of finer-grained access controls for Android developers and users," presented at the 12th Int. Conf. Detection Intrusions Malware Vulnerability Assessment, Milan, Italy, Jul. 9–10, 2015.

[65] W. Zhou, Z. Wang, Y. Zhou, and X. Jiang, "Divilar: Diversifying intermediate language for anti-repackaging on Android platform," presented at the 4th ACM Conf. Data Appl. Secur. Privacy, San Antonio, TX, USA, Mar. 3–5, 2014.

[66] W. Zhou, X. Zhang, and X. Jiang, "Appink: Watermarking Android apps for repackaging deterrence," presented at the 8th ACM Symp. Inf. Comput. Commun. Secur., Hangzhou, China, May 8–10, 2013.

[67] C. Ren, K. Chen, and P. Liu, "Droidmarking: Resilient software watermarking for impeding Android application repackaging," presented at the ACM/IEEE Int. Conf. Automated Softw. Eng., Vasteras, Sweden, Sep. 15–19, 2014.

[68] M. Protsenko, S. Kreuter, and T. Müller, "Dynamic self-protection and tamperproofing for Android apps using native code," presented at the 10th Int. Conf. Availability Rel. Secur., Toulouse, France, Aug. 24–27, 2015.

[69] J. Jang, H. Ji, J. Hong, J. Jung, D. Kim, and S. K. Jung, “Protecting Android applications with steganography-based software watermarking,” presented at the 28th Annu. ACM Symp. Appl. Comput., Coimbra, Portugal, Mar. 18–22, 2013.

[70] M. Backes, S. Bugiel, and S. Gerling, “Scippa: System-centric IPC provenance on Android,” presented at the 30th Annu. Comput. Secur. Appl. Conf., New Orleans, LA, USA, Dec. 8–12, 2014.

[71] Q. Zhou, D. Wang, Y. Zhang, B. Qin, A. Yu, and B. Zhao, “Chaindroid: Safe and flexible access to protected Android resources based on call chain,” presented at the 12th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./11th IEEE Int. Symp. Parallel Distrib. Process. Appl./12th IEEE Int. Conf. Ubiquitous Comput. Commun., Melbourne, Australia, Jul. 16–18, 2013.

[72] S. Mutti, E. Bacis, and S. Paraboschi, “SeSQLite: Security enhanced SQLite: Mandatory access control for Android databases,” presented at the 31st Annu. Comput. Secur. Appl. Conf., Los Angeles, CA, USA, Dec. 7–11, 2015.

[73] Q. Do, B. Martini, and K.-K. R. Choo, “Enforcing file system permissions on Android external storage: Android file system permissions (AFP) prototype and owncloud,” presented at the 13th IEEE Int. Conf. Trust Secur. Privacy Comput. Commun., Beijing, China, Sep. 24–26, 2014.

[74] D. Schreckling, J. Posegga, and D. Hausknecht, “Constroid: Data-centric access control for Android,” presented at the ACM Symp. Appl. Comput., Riva, Trento, Italy, Mar. 26–30, 2012.

[75] D. Barrera, D. McCarney, J. Clark, and P. C. van Oorschot, “Baton: Certificate agility for Android’s decentralized signing infrastructure,” presented at the 7th ACM Conf. Secur. Privacy Wireless Mobile Netw., Oxford, U. K., Jul. 23–25, 2014.

[76] H. Deng, Q. Wu, B. Qin, W. Susilo, J. K. Liu, and W. Shi, “Asymmetric cross-cryptosystem re-encryption applicable to efficient and secure mobile access to outsourced data,” presented at the 10th ACM Symp. Inf. Comput. Commun. Secur., Singapore, Apr. 14–17, 2015.

[77] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner, “Addroid: Privilege separation for applications and advertisers in Android,” presented at the 7th ACM Symp. Inf. Comput. Commun. Secur., Seoul, Korea, May 2–4, 2012.

[78] C. Mann and A. Starostin, “A framework for static detection of privacy leaks in Android applications,” presented at the ACM Symp. Appl. Comput., Riva, Trento, Italy, Mar. 26–30, 2012.

[79] M. Sun, M. Zheng, J. C. S. Lui, and X. Jiang, “Design and implementation of an Android host-based intrusion prevention system,” presented at the 30th Annu. Comput. Secur. Appl. Conf., New Orleans, LA, USA, Dec. 8–12, 2014.

[80] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, “Whyper: Towards automating risk assessment of mobile applications,” presented at the 22th USENIX Secur. Symp., Washington, DC, USA, Aug. 14–16, 2013.

[81] M. Zhang, Y. Duan, Q. Feng, and H. Yin, “Towards automatic generation of security-centric descriptions for Android apps,” presented at the 22nd ACM SIGSAC Conf. Comput. Commun. Secur., Denver, CO, USA, Oct. 12–6, 2015.

[82] A. Sadeghi, N. Esfahani, and S. Malek, “Mining the categorized software repositories to improve the analysis of security vulnerabilities,” presented at the 17th Int. Conf. Fundamental Approaches Softw. Eng. Held Eur. Joint Conf. Theory Practice Softw., Grenoble, France, Apr. 5–13, 2014.

[83] D. Barrera, J. Clark, D. McCarney, and P. C. V. Oorschot, “Understanding and improving app installation security mechanisms through empirical analysis of Android,” presented at the Workshop Secur. Privacy Smartphones Mobile Devices, Raleigh, NC, USA, Oct. 19, 2012.

[84] Z. Xie and S. Zhu, “Grouptie: Toward hidden collusion group discovery in app stores,” presented at the 7th ACM Conf. Secur. Privacy Wireless Mobile Netw., Oxford, U. K., Jul. 23–25, 2014.

[85] Z. Xie and S. Zhu, “Appwatcher: Unveiling the underground market of trading mobile app reviews,” presented at the 8th ACM Conf. Secur. Privacy Wireless Mobile Netw., New York, NY, USA, Jun. 22–26, 2015.

[86] S. Jiao, Y. Cheng, L. Ying, P. Su, and D. Feng, “A rapid and scalable method for Android application repackaging detection,” presented at the 11th Int. Conf. Inf. Secur. Practice Experience, Beijing, China, May 5–8, 2015.

[87] K. Allix, Q. Jrome, T. F. Bissyand, J. Klein, R. State, and Y. L. Traon, “A forensic analysis of Android malware-how is malware written and how it could be detected?” presented at the IEEE 38th Annu. Comput. Softw. Appl. Conf., Vasteras, Sweden, Jul. 21–25, 2014.

[88] Y. Ng, H. Zhou, Z. Ji, H. Luo, and Y. Dong, “Which Android app store can be trusted in China?” presented at the IEEE 38th Annu. Comput. Softw. Appl. Conf., Vasteras, Sweden, Jul. 21–25, 2014.

[89] P. Teufl, M. Ferk, A. Fitzek, D. Hein, S. Kraxberger, and C. Orthacker, “Malware detection by applying knowledge discovery processes to application metadata on the Android market (Google play),” *Secur. Commun. Netw.*, vol. 9, no. 5, pp. 389–419, 2013.

[90] M. Lindorfer, et al., “Andradar: Fast discovery of Android applications in alternative markets,” presented at the 11th Int. Conf. Detection Intrusions Malware Vulnerability Assessment, Egham, U.K., Jul. 10–11, 2014.

[91] P. Faruki, V. Ganmoor, V. Laxmi, M. S. Gaur, and A. Bharmal, “Androsimilar: Robust statistical feature signature for Android malware detection,” presented at the 6th Int. Conf. Secur. Inf. Netw., Aksaray, Turkey, Nov. 26–28, 2013.

[92] B. Sanz, I. Santos, X. Ugarte-Pedrero, C. Laorden, J. Nieves, and P. G. Bringas, “Anomaly detection using string analysis for Android malware detection,” presented at the Int. Joint Conf. SOCO-CISIS-ICEUTE, Salamanca, Spain, Sep. 11–13, 2013.

[93] H. Gonzalez, A. A. Kadir, N. Stakhanova, A. J. Alzahrani, and A. A. Ghorbani, “Exploring reverse engineering symptoms in Android apps,” presented at the 8th Eur. Workshop Syst. Secur., Bordeaux, France, Apr. 21, 2015.

[94] G. Canfora, A. D. Lorenzo, E. Medvet, F. Mercaldo, and C. A. Visaggio, “Effectiveness of opcode ngrams for detection of multi family Android malware,” presented at the 10th Int. Conf. Availability Rel. Secur., Toulouse, France, Aug. 24–27, 2015.

[95] M. M. John, P. Vinod, and K. A. Dhanya, “Hartley’s test ranked opcodes for Android malware analysis,” presented at the 8th Int. Conf. Secur. Inf. Netw., Sochi, Russian Federation, Sep. 8–10, 2015.

[96] S. Blackshear, A. Gendreau, and B.-Y. E. Chang, “Droidel: A general approach to Android framework modeling,” presented at the 4th ACM SIGPLAN Int. Workshop State Art Program Anal., Portland, OR, USA, Jun. 15–17, 2015.

[97] Y. Cao, et al., “Edgeminer: Automatically detecting implicit control flow transitions through the Android framework,” presented at the 22nd Annu. Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 8–11, 2014.

[98] D. Octeau, D. Luchaup, M. Dering, S. Jha, and P. McDaniel, “Composite constant propagation: Application to Android inter-component communication analysis,” presented at the 37th IEEE/ACM Int. Conf. Softw. Eng., Florence, Italy, May 16–24, 2015.

[99] S. Yang, D. Yan, H. Wu, Y. Wang, and A. Rountev, “Static control-flow analysis of user-driven callbacks in Android applications,” presented at the 37th IEEE/ACM Int. Conf. Softw. Eng., Florence, Italy, May 16–24, 2015.

[100] D. Li, Y. Lyu, M. Wan, and W. G. J. Halfond, “String analysis for Java and Android applications,” presented at the 2015 10th Joint Meeting Found. Softw. Eng., Bergamo, Italy, Aug. 30-Sep. 4, 2015.

[101] S. Blackshear, B.-Y. E. Chang, and M. Sridharan, “Thresher: Precise refutations for heap reachability,” presented at the ACM SIGPLAN Conf. Program. Language Des. Implementation, Seattle, WA, USA, Jun. 16–19, 2013.

[102] S. Arzt, S. Rasthofer, and E. Bodden, “Instrumenting Android and Java applications as easy as abc,” presented at the 4th Int. Conf. Runtime Verification, Rennes, France, Sep. 24–27, 2013.

[103] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, “SIF: A selective instrumentation framework for mobile applications,” presented at the 11th Annu. Int. Conf. Mobile Syst. Appl. Services, Taipei, Taiwan, Jun. 25–28, 2013.

[104] S. Arzt, et al., “Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps,” presented at the ACM SIGPLAN Conf. Program. Language Des. Implementation, Edinburgh, U. K., Jun. 9–11, 2014.

[105] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, and Y. L. Traon, “Apkcombiner: Combining multiple Android apps to support inter-app analysis,” presented at the 30th IFIP TC 11 Int. Conf. ICT Syst. Secur. Privacy Protection, Hamburg, Germany, May 26–28, 2015.

[106] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, and G. A. Maranon, “Puma: Permission usage to detect malware in Android,” presented at the Int. Joint Conf. CISIS-ICEUTE-SOCO, Ostrava, Czech Republic, Sep. 5–7, 2012.

[107] N. Zhang, K. Yuan, M. Naveed, X.-Y. Zhou, and X. Wang, “Leave me alone: App-level protection against runtime information gathering on Android,” presented at the 2015 IEEE Symp. Secur. Privacy, San Jose, CA, USA, May 17–21, 2015.

[108] Y. Zhang, K. Huang, Y. Liu, K. Chen, L. Huang, and Y. Lian, “Timing-based clone detection on Android markets,” presented at the 10th Int. ICST Conf. Secur. Privacy Commun. Netw., Beijing, China, Sep. 24–26, 2014.

[109] J. Hoffmann, S. Neumann, and T. Holz, “Mobile malware detection based on energy fingerprints a dead end?” in *Research in Attacks, Intrusions, and Defenses*. Berlin, Germany: Springer, 2013, pp. 348–368.

[110] B. Dixon, Y. Jiang, A. Jaiantilal, and S. Mishra, “Location based power analysis to detect malicious code in smartphones,” presented at the 1st ACM Workshop Secur. Privacy Smartphones Mobile Devices, Chicago, IL, USA, Oct. 17, 2011.

[111] B. Amos, H. A. Turner, and J. White, “Applying machine learning classifiers to dynamic Android malware detection at scale,” presented at the 9th Int. Wireless Commun. Mobile Comput. Conf., Sardinia, Italy, Jul. 1–5, 2013.

[112] R. S. Khune and J. Thangakumar, “A cloud-based intrusion detection system for Android smartphones,” in *Proc. Int. Conf. Radar Commun. Comput.*, 2012, pp. 180–184.

[113] X. Kou and Q. Wen, “Intrusion detection model based on Android,” in *Proc. 4th IEEE Int. Conf. Broadband Netw. Multimedia Technol.*, 2011, pp. 624–628.

[114] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen, “Investigating user privacy in Android ad libraries,” presented at the Workshop Mobile Secur. Technol., San Francisco, CA, May 24, 2012.

[115] A. S. Shamili, C. Bauckhage, and T. Alpcan, “Malware detection on mobile devices using distributed machine learning,” presented at the 20th Int. Conf. Pattern Recognit., Istanbul, Turkey, Aug. 23–26, 2010.

[116] Y. Song and U. Hengartner, “Privacyguard: A VPN-based platform to detect information leakage on Android devices,” presented at the 5th Annu. ACM CCS Workshop Secur. Privacy Smartphones Mobile Devices, Denver, Colorado, USA, Oct. 12, 2015.

[117] L. Vigneri, J. Chandrashekar, I. Pefkianakis, and O. Heen, “Taming the Android appstore: Lightweight characterization of Android applications,” *arXiv:1504.06093*, Apr. 2015.

[118] J. Crussell, R. Stevens, and H. Chen, “Madfraud: Investigating ad fraud in Android applications,” presented at the 12th Annu. Int. Conf. Mobile Syst. Appl. Services, Bretton Woods, NH, USA, Jun. 16–19, 2014.

[119] S. Schrittwieser, et al., “Guess who’s texting you? evaluating the security of smartphone messaging applications,” presented at the 19th Annu. Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 5–8, 2012.

[120] N. Boggs, W. Wang, S. Mathur, B. Coskun, and C. Pincock, “Discovery of emergent malicious campaigns in cellular networks,” presented at the Annu. Comput. Secur. Appl. Conf., New Orleans, LA, USA, Dec. 9–13, 2013.

[121] Y. Nadji, J. T. Giffin, and P. Traynor, “Automated remote repair for mobile malware,” presented at the 27th Annu. Comput. Secur. Appl. Conf., Orlando, FL, USA, Dec. 5–9, 2011.

[122] M. Aresu, D. Ariu, M. Ahmadi, D. Maiorca, and G. Giacinto, “Clustering Android malware families by http traffic,” presented at the 10th Int. Conf. Malicious Unwanted Softw., Fajardo, PR, USA, Oct. 20–22, 2015.

[123] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, “Andromaly: A behavioral malware detection framework for Android devices,” *J. Intell. Inf. Syst.*, vol. 38, no. 1, pp. 161–190, 2012.

[124] Z. Xu and S. Zhu, “Semadroid: A privacy-aware sensor management framework for smartphones,” presented at the 5th ACM Conf. Data Appl. Secur. Privacy, San Antonio, TX, USA, Mar. 2–4, 2015.

[125] H.-S. Ham and M.-J. Choi, “Analysis of Android malware detection performance using machine learning classifiers,” in *Proc. Int. Conf. ICT Convergence*, 2013, pp. 490–495.

[126] C. Marforio, H. Ritzdorf, A. Francillon, and S. Capkun, “Analysis of the communication between colluding applications on modern smartphones,” presented at the 28th Annu. Comput. Secur. Appl. Conf., Orlando, FL, USA, Dec. 3–7, 2012.

[127] M. Bierma, E. Gustafson, J. Erickson, D. Fritz, and Y. R. Choe, “Andlantis: Large-scale Android dynamic analysis,” *arXiv:1410.7751*, Oct. 2014.

[128] X. Zhang and W. Du, “Attacks on Android clipboard,” presented at the 11th Int. Conf. Detection Intrusions Malware Vulnerability Assessment, Egham, U.K., Jul. 10–11, 2014.

[129] T. Luo, H. Hao, W. Du, Y. Wang, and H. Yin, “Attacks on webview in the Android system,” presented at the 27th Annu. Comput. Secur. Appl. Conf., Orlando, FL, USA, Dec. 5–9 2011.

[130] K. Hamandi, A. Chehab, I. H. Elhajj, and A. I. Kayssi, “Android SMS malware: Vulnerability and mitigation,” presented at the 27th Int. Conf. Adv. Inf. Netw. Appl. Workshops, Barcelona, Spain, Mar. 25–28, 2013.

[131] J. Xiao, H. Huang, and H. Wang, “Kernel data attack is a realistic security threat,” presented at the 11th Int. Conf. Secur. Privacy Commun. Netw., Dallas, TX, USA, Oct. 26–29, 2015.

[132] Q. A. Chen, Z. Qian, and Z. M. Mao, “Peeking into your app without actually seeing it: UI state inference and novel Android attacks,” presented at the 23rd USENIX Secur. Symp., San Diego, CA, USA, Aug. 20–22, 2014.

[133] C. Mulliner, W. K. Robertson, and E. Kirda, “Virtualswindle: An automated attack against in-app billing on Android,” presented at the 9th ACM Symp. Inf. Comput. Commun. Secur., Kyoto, Japan, Jun. 3–6, 2014.

[134] I. Polakis, S. Volanis, E. Athanasopoulos, and E. P. Markatos, “The man who was there: Validating check-ins in location-based services,” presented at the Annu. Comput. Secur. Appl. Conf., New Orleans, LA, USA, Dec. 9–13, 2013.

[135] J.-F. Lalande and S. Wendzel, “Hiding privacy leaks in Android applications using low-attention raising covert channels,” presented at the Int. Conf. Availability Rel. Secur., Regensburg, Germany, Sep. 2–6, 2013.

[136] D. Maier, T. Müller, and M. Protsenko, “Divide-and-conquer: Why Android malware cannot be stopped,” presented at the 9th Int. Conf. Availability Rel. Secur., Fribourg, Switzerland, Sep. 8–12, 2014.

[137] G. Canfora, F. Mercaldo, G. Moriano, and C. A. Visaggio, “Composition-malware: Building Android malware at run time,” presented at the 10th Int. Conf. Availability Rel. Secur., Toulouse, France, Aug. 24–27, 2015.

[138] M. Protsenko and T. Müller, “Pandora applies non-deterministic obfuscation randomly to Android,” presented at the 8th Int. Conf. Malicious Unwanted Softw.: ”The Americas”, Fajardo, PR, USA, Oct. 22–24, 2013.

[139] H. Huang, S. Zhu, P. Liu, and D. Wu, “A framework for evaluating mobile app repackaging detection algorithms,” presented at the 6th Int. Conf. Trust Trustworthy Comput., London, UK, Jun. 17–19, 2013.

[140] B. Ma, “How we found these vulnerabilities in Android applications,” presented at the 10th ICST Int. Conf. Secur. Privacy Commun. Netw., Beijing, China, Sep. 24–26, 2014.

[141] Dexter. [Online]. Available: http://dexter.dexlabs.org/, accessed 2016.

[142] Droidbox. [Online]. Available: https://code.google.com/p/droidbox/, accessed 2016.

[143] I. Shklovski, S. D. Mainwaring, H. H. Skúladóttir, and H. Borgthorsson, “Leakiness and creepiness in app space: Perceptions of privacy and mobile app use,” presented at the CHI Conf. Human Factors Comput. Syst., Toronto, ON, Canada, Apr. 26–May 1, 2014.

[144] F. Swiderski and W. Snyder, *Threat Modeling*. Redmond, WA, USA: Microsoft Press, 2004.

[145] Y. Zhou and X. Jiang, “Detecting passive content leaks and pollution in Android applications,” presented at the 20th Annu. Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 24–27, 2013.

[146] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, “A study of Android application security,” presented at the 20th USENIX Secur. Symp., San Francisco, CA, USA, Aug. 8–12, 2011.

[147] R. Johnson, M. Elsabagh, A. Stavrou, and V. Sritapan, “Targeted DoS on Android: How to disable Android in 10 seconds or less,” presented at the 10th Int. Conf. Malicious Unwanted Softw., Fajardo, PR, USA, Oct. 20–22, 2015.

[148] T. Martin, M. Hsiao, D. S. Ha, and J. Krishnaswami, “Denial-of-service attacks on battery-powered mobile computers,” presented at the 2nd IEEE Int. Conf. Pervasive Comput. Commun., Orlando, FL, USA, Mar. 14–17, 2004.

[149] N. Andronio, S. Zanero, and F. Maggi, "Heldroid: Dissecting and detecting mobile ransomware," presented at the 18th Int. Symp. Res. Attacks Intrusions Defenses, Kyoto, Japan, Nov. 2–4, 2015.

[150] T. Yang, Y. Yang, K. Qian, D. C.-T. Lo, Y. Qian, and L. Tao, "Automated detection and analysis for Android ransomware," presented at the 17th IEEE Int. Conf. High Performance Comput. Commun., 7th IEEE Int. Symp. Cyberspace Safety Secur., and 12th IEEE Int. Conf. Embedded Softw. Syst., New York, NY, USA, Aug. 24–26, 2015.

[151] M. C. Grace, Y. Zhou, Z. Wang, and X. Jiang, "Systematic detection of capability leaks in stock Android smartphones," presented at the 19th Annu. Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 5–8, 2012.

[152] A. Bartel, J. Klein, M. Monperrus, and Y. L. Traon, "Static analysis for extracting permission checks of a large scale framework: The challenges and solutions for analyzing Android," *IEEE Trans. Softw. Eng.*, vol. 40, no. 6, pp. 617–632, Jun. 2014.

[153] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," presented at the 18th ACM Conf. Comput. Commun. Secur., Chicago, Illinois, USA, Oct. 17–21, 2011.

[154] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.-R. Sadeghi, and B. Shastry, "Towards taming privilege-escalation attacks on Android," presented at the 19th Annu. Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 5–8, 2012.

[155] Android interface definition language. [Online]. Available: http://developer.android.com /guide/components/aidl.html, accessed 2016.

[156] V. Rastogi, Y. Chen, and X. Jiang, "Droidchameleon: Evaluating Android anti-malware against transformation attacks," presented at the 8th ACM Symp. Inf. Comput. Commun. Secur., Hangzhou, China, May 8–10, 2013.

[157] V. Rastogi, Y. Chen, and X. Jiang, "Catch me if you can: Evaluating Android anti-malware against transformation attacks," *IEEE Trans. Inf. Forensics Secur.*, vol. 9, no. 1, pp. 99–108, Jan. 2014.

[158] M. Zheng, P. P. C. Lee, and J. C. S. Lui, "Adam: An automatic and extensible platform to stress test Android anti-virus systems," presented at the 9th Int. Conf. Detection Intrusions Malware Vulnerability Assessment, Heraklion, Crete, Greece, Jul. 26–27, 2012.

[159] F. C. Freiling, M. Protsenko, and Y. Zhuang, "An empirical evaluation of software obfuscation techniques applied to Android apks," presented at the 10th Int. ICST Conf. Secur. Privacy Commun. Netw., Beijing, China, Sep. 24–26, 2014.

[160] F. Nasim, B. Aslam, W. Ahmed, and T. Naeem, "Uncovering self code modification in Android," presented at the 1st Int. Conf. Codes Cryptology Inf. Secur., Rabat, Morocco, May 26–28, 2015.

[161] B. Livshits, J. Whaley, and M. S. Lam, "Reflection analysis for Java," in *Programming Languages and Systems*. Berlin, Germany: Springer, 2005, pp. 139–160.

[162] M. D. Ernst, "Static and dynamic analysis: Synergy and duality," in *Proc. ACM-SIGPLAN-SIGSOFT Workshop Program Anal. Softw. Tools Eng.*, Washington, DC, USA, June 7–8, 2004.

[163] G. Sarwar, O. Mehani, R. Boreli, and M. A. Kaafar, "On the effectiveness of dynamic taint analysis for protecting against private information leaks on Android-based devices," presented at the 10th Int. Conf. Secur. Cryptography, Reykjavík, Iceland, Jul. 29–31, 2013.

[164] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis, "Rage against the virtual machine: Hindering dynamic analysis of Android malware," presented at the 7th Eur. Workshop Syst. Secur., Amsterdam, The Netherlands, Apr. 13, 2014.

[165] J. Qiu, B. Yadegari, B. Johannesmeyer, S. Debray, and X. Su, "A framework for understanding dynamic anti-analysis defenses," presented at the 4th Program Protection Reverse Eng. Workshop, New Orleans, LA, USA, Dec. 9, 2014.

[166] T. Vidas and N. Christin, "Evading Android runtime analysis via sandbox detection," presented at the 9th ACM Symp. Inf. Comput. Commun. Secur., Kyoto, Japan, Jun. 3–6, 2014.

[167] S. Rasthofer, I. Asrar, S. Huber, and E. Bodden, "How current Android malware seeks to evade automated code analysis," presented at the 9th Int. Conf. Inf. Secur. Theory Practice, Heraklion, Greece, Aug. 24–25, 2015.

[168] S. Arzt, K. Falzon, A. Follner, S. Rasthofer, E. Bodden, and V. Stolz, "How useful are existing monitoring languages for securing Android apps?" presented at the Softw. Eng. Workshopband (inkl. Doktorandensymposium), Fachtagung des GI-Fachbereichs Softwaretechnik, Aachen, Germany, Feb. 26–1 Mar., 2013.

[169] Y. Jing, Z. Zhao, G.-J. Ahn, and H. Hu, "Morpheus: Automatically generating heuristics to detect Android emulators," presented at the 30th Annu. Comput. Secur. Appl. Conf., New Orleans, LA, USA, Dec. 8–12, 2014.

[170] P. Faruki, A. Bharmal, V. Laxmi, M. S. Gaur, M. Conti, and M. Rajarajan, "Evaluation of Android anti-malware techniques against Dalvik bytecode obfuscation," presented at the 13th IEEE Int. Conf. Trust Secur. Privacy Comput. Commun., Beijing, China, Sep. 24–26, 2014.

[171] Dex2Jar. [Online]. Available: https://code.google.com/p/dex2jar/, accessed 2016.

[172] D. Octeau, W. Enck, and P. McDaniel, "The ded decompiler," Netw. Secur. Res. Center, Dept. Comput. Sci. Eng., Pennsylvania State Univ. University Park, PA, USA, Tech. Rep. NAS-TR-0140–2010, 2010.

[173] D. Octeau, S. Jha, and P. McDaniel, "Retargeting Android applications to Java bytecode," presented at the 20th ACM SIGSOFT Symp. Found. Softw. Eng., Cary, NC, USA, Nov. 11–16, 2012.

[174] R. Vallée-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan, "Soot—a Java bytecode optimization framework," presented at the Conf. Centre Adv. Studies Collaborative Res., Mississauga, ON, Canada, Nov. 8–11, 1999.

[175] A. Bartel, J. Klein, Y. Le Traon, and M. Monperrus, "Dexpler: Converting Android Dalvik bytecode to jimple for static analysis with soot," in *Proc. ACM SIGPLAN Int. Workshop State Art Java Program Anal.*, 2012, pp. 27–38.

[176] Apktool. [Online]. Available: http://ibotpeaches.github.io/Apktool/, accessed 2016.

[177] V. Costamagna and C. Zheng, "Artdroid: A virtual-method hooking framework on Android art runtime," presented at the 1st Int. Workshop Innovations Mobile Privacy Security, co-located Int. Symp. Eng. Secure Softw. Syst., London, U.K., Apr. 6, 2016.

[178] S. Mutti, et al., "Baredroid: Large-scale analysis of Android apps on real devices," presented at the 31st Annu. Comput. Secur. Appl. Conf., Los Angeles, CA, USA, Dec. 7–11, 2015.

[179] P. Godefroid, M. Y. Levin, and D. A. Molnar, "Automated whitebox fuzz testing," presented at the Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 10–13, 2008.

[180] Android monkey. [Online]. Available: http://developer.android.com/guide/developing/tools/monkey.html/, accessed 2016.

[181] J. C. King, "Symbolic execution and program testing," *Commun. ACM*, vol. 19, no. 7, pp. 385–394, 1976.

[182] X.-Y. Zhou, Y. Lee, N. Zhang, M. Naveed, and X. Wang, "The peril of fragmentation: Security hazards in Android device driver customizations," presented at the 2014 IEEE Symp. Secur. Privacy, Berkeley, CA, USA, May 18–21, 2014.

[183] F. Wei, S. Roy, X. Ou, and Robby, "Amandroid: A precise and general inter-component data flow analysis framework for security vetting of Android apps," presented at the 2014 ACM SIGSAC Conf. Comput. Commun. Secur., Scottsdale, AZ, USA, Nov. 3–7, 2014.

[184] D. Titze, P. Stephanow, and J. Schütte, "App-ray: User-driven and fully automated Android app security assessment," Fraunhofer AISEC TechReport, 2013.

[185] M. Xia, L. Gong, Y. Lyu, Z. Qi, and X. Liu, "Effective real-time Android application auditing," presented at the 2015 IEEE Symp. Secur. Privacy, San Jose, CA, USA, May 17–21, 2015.

[186] J. Schütte, D. Titze, and J. M. D. Fuentes, "Appcaulk: Data leak prevention by injecting targeted taint tracking into Android apps," presented at the 13th IEEE Int. Conf. Trust Secur. Privacy Comput. Commun., Beijing, China, Sep. 24–26, 2014.

[187] F. Cai, H. Chen, Y. Wu, and Y. Zhang, "Appcracker: Widespread vulnerabilities in user and session authentication in mobile apps," ShanghaiTech Univ., Shanghai, China, Tech. Rep. SHTech/SIST-2014–1, 2014.

[188] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These aren't the droids you're looking for: Retrofitting Android to protect data from imperious applications," presented at the 18th ACM Conf. Comput. Commun. Security, Chicago, Illinois, USA, Oct. 17–21, 2011.

[189] M. Backes, S. Gerling, C. Hammer, M. Maffei, and P. V. Styp-Rekowsky, "Appguard-enforcing user requirements on Android apps," presented at the 19th Int. Conf. Tools Algorithms Construction Anal. Syst., held Eur. Joint Conf. Theory Practice Softw., Rome, Italy, Mar. 16–24, 2013.

[190] S. Rosen, Z. Qian, and Z. M. Mao, "Appprofiler: A flexible method of exposing privacy-related behavior in Android applications to end users," presented at the 3rd ACM Conf. Data Appl. Secur. Privacy, San Antonio, TX, USA, Feb. 18–20, 2013.

[191] M. Zhang and H. Yin, "Appsealer: Automatic generation of vulnerability-specific patches for preventing component hijacking attacks in Android applications," presented at the 21st Annu. Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 23–26, 2014.

[192] A. Nadkarni and W. Enck, "Preventing accidental data disclosure in modern operating systems," presented at the ACM SIGSAC Conf. Comput. Commun. Secur., Berlin, Germany, Nov. 4–8, 2013.

[193] S. Heuser, A. Nadkarni, W. Enck, and A.-R. Sadeghi, "ASM: A programmable interface for extending Android security," presented at the 23rd USENIX Secur. Symp., San Diego, CA, USA, Aug. 20–22, 2014.

[194] H. Wang, et al., "Vulnerability assessment of OAuth implementations in Android applications," presented at the 31st Annu. Comput. Secur. Appl. Conf., Los Angeles, CA, USA, Dec. 7–11, 2015.

[195] H. Bagheri, E. Kang, S. Malek, and D. Jackson, "Detection of design flaws in the Android permission protocol through bounded verification," presented at the 20th Int. Symp. FM 2015: Formal Methods, Oslo, Norway, Jun. 24–26, 2015.

[196] S. Bartsch, B. J. Berger, M. Bunke, and K. Sohr, "The transitivity-of-trust problem in Android application interaction," presented at the Int. Conf. Availability Rel. Secur., Regensburg, Germany, Sep. 2–6, 2013.

[197] E. Chin and D. Wagner, "Bifocals: Analyzing webview vulnerabilities in Android applications," presented at the 14th Int. Workshop Inf. Secur. Appl., Jeju Island, Korea, Aug. 19–21, 2013.

[198] D. Buhov, M. Huber, G. Merzdovnik, E. R. Weippl, and V. Dimitrova, "Network security challenges in Android applications," presented at the 10th Int. Conf. Availability Rel. Secur., Toulouse, France, Aug. 24–27, 2015.

[199] C. Cao, N. Gao, P. Liu, and J. Xiang, "Towards analyzing the input validation vulnerabilities associated with Android system services," presented at the 31st Annu. Comput. Secur. Appl. Conf., Los Angeles, CA, USA, Dec. 7–11, 2015.

[200] S. Shao, G. Dong, T. Guo, T. Yang, and C. Shi, "Modelling analysis and auto-detection of cryptographic misuse in Android applications," presented at the IEEE 12th Int. Conf. Dependable Autonomic Secure Comput., Dalian, China, Aug. 24–27, 2014.

[201] X. Cui, D. Yu, P. P. F. Chan, L. C. K. Hui, S.-M. Yiu, and S. Qing, "Cochecker: Detecting capability and sensitive data leaks from component chains in Android," presented at the 19th Australasian Conf. Inf. Secur. Privacy, Wollongong, NSW, Australia, Jul. 7–9, 2014.

[202] J. Schütte, R. Fedler, and D. Titze, "Condroid: Targeted dynamic analysis of Android applications," presented at the 29th IEEE Int. Conf. Adv. Inf. Netw. Appl., Gwangju, South Korea, Mar. 24–27, 2015.

[203] B. Cooley, H. Wang, and A. Stavrou, "Activity spoofing and its defense in Android smartphones," presented at the 12th Int. Conf. Appl. Cryptography Netw. Secur., Lausanne, Switzerland, Jun. 10–13, 2014.

[204] A. Bartel, J. Klein, Y. Le Traon, and M. Monperrus, "Automatically securing permission-based software by reducing the attack surface: An application to Android," presented at the IEEE/ACM Int. Conf. Automated Softw. Eng., Essen, Germany, Sep. 3–7, 2012.

[205] H. Bagheri, A. Sadeghi, J. Garcia, and S. Malek, "Covert: Compositional analysis of Android inter-app permission leakage," *IEEE Trans. Softw. Eng.*, vol. 41. no. 9, pp. 866–886, Sep. 2015.

[206] A. Sadeghi, H. Bagheri, and S. Malek, "Analysis of Android inter-app security vulnerabilities using covert," presented at the 37th IEEE/ACM Int. Conf. Softw. Eng., Florence, Italy, May 16–24, 2015.

[207] Y. Zhou, L. Wu, Z. Wang, and X. Jiang, "Harvesting developer credentials in Android apps," presented at the 8th ACM Conf. Secur. Privacy Wireless Mobile Netw., New York, NY, USA, Jun. 22–26, 2015.

[208] M. Conti, V. T. N. Nguyen, and B. Crispo, "Crepe: Context-related policy enforcement for Android," presented at the 13th Int. Conf. Inf. Secur., Boca Raton, FL, USA, Oct. 25–28, 2010.

[209] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in Android applications," presented at the 2013 ACM SIGSAC Conf. Comput. Commun. Secur., Berlin, Germany, Nov. 4–8, 2013.

[210] A. Desnos, "Android: Static analysis using similarity distance," presented at the 45th Hawaii Int. Conf. Syst. Sci., Maui, HI, USA, Jan. 4–7, 2012.

[211] M. Mitchell, G. Tian, and Z. Wang, "Systematic audit of third-party Android phones," presented at the 4th ACM Conf. Data Appl. Secur. Privacy, San Antonio, TX, USA, Mar. 3–5, 2014.

[212] Y. Zhongyang, Z. Xin, B. Mao, and L. Xie, "Droidalarm: An all-sided static analysis tool for Android privilege-escalation malware," presented at the 8th ACM Symp. Inf. Comput. Commun. Secur., Hangzhou, China, May 8–10, 2013.

[213] P. P. Chan, L. C. Hui, and S. M. Yiu, "Droidchecker: Analyzing Android applications for capability leak," presented at the 5th ACM Conf. Secur. Privacy Wireless Mobile Netw., Tucson, AZ, USA, Apr. 16–18, 2012.

[214] Y.-L. Chen, H.-M. Lee, A. B. Jeng, and T.-E. Wei, "Droidcia: A novel detection method of code injection attacks on html5-based mobile apps," presented at the IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, Aug. 20–22, 2015.

[215] H. Bagheri, A. Sadeghi, R. Jabbarvand, and S. Malek, "Automated dynamic enforcement of synthesized security policies in Android," Dept. Comput. Sci., George Mason Univ., Fairfax, VA, USA, Tech. Rep. GMU-CS-TR-2015–5, 2015.

[216] M. Zheng, M. Sun, and J. C. S. Lui, "Droidray: A security evaluation system for customized Android firmwares," presented at the 9th ACM Symp. Inf. Comput. Commun. Secur., Kyoto, Japan, Jun. 3–6, 2014.

[217] S. Rasthofer, et al., "Droidsearch: A tool for scaling Android app triage to real-world app stores," presented at the Sci. Inf. Conf., London, U.K., Jul. 28–30, 2015.

[218] D. Octeau, P. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein, and Y. L. Traon, "Effective inter-component communication mapping in Android: An essential step towards holistic security analysis," presented at the 22th USENIX Secur. Symp., Washington, DC, USA, Aug. 14–16, 2013.

[219] Y. Zhang, M. Yang, G. Gu, and H. Chen, "Finedroid: Enforcing permissions with system-wide application execution context," presented at the 11th Int. Conf. Secur. Privacy Commun. Netw., Dallas, TX, USA, Oct. 26–29, 2015.

[220] R. Gallo, et al., "Security and system architecture: Comparison of Android customizations," presented at the 8th ACM Conf. Secur. Privacy Wireless Mobile Netw., New York, NY, USA, Jun. 22–26, 2015.

[221] D. Geneiatakis, I. N. Fovino, I. Kounelis, and P. Stirparo, "A permission verification approach for Android mobile applications," *Comput. Secur.*, vol. 49, pp. 192–205, 2015.

[222] L. Falsina, Y. Fratantonio, S. Zanero, C. Kruegel, G. Vigna, and F. Maggi, "Grab 'n run: Secure and practical dynamic code loading for Android applications," presented at the 31st Annu. Comput. Secur. Appl. Conf., Los Angeles, CA, USA, Dec. 7–11, 2015.

[223] Y. Aafer, et al., "Hare hunting in the wild Android: A study on the threat of hanging attribute references," presented at the 22nd ACM SIGSAC Conf. Comput. Commun. Secur., Denver, CO, USA, Oct. 12–6, 2015.

[224] S. Calzavara, I. Grishchenko, and M. Maffei, "HornDroid: Practical and sound static analysis of Android applications by SMT solving," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2016, pp. 47–62.

[225] L. Li, et al., "ICCTA: Detecting inter-component privacy leaks in Android apps," presented at the 37th IEEE/ACM Int. Conf. Softw. Eng., Florence, Italy, May 16–24, 2015.

[226] Z. Fang, Q. Liu, Y. Zhang, K. Wang, and Z. Wang, "IVDroid: Static detection for input validation vulnerability in Android inter-component communication," presented at the 11th Int. Conf. Inf. Secur. Practice Experience, Beijing, China, May 5–8, 2015.

[227] S. Hanna, L. Huang, E. X. Wu, S. Li, C. Chen, and D. Song, "Juxtapp: A scalable system for detecting code reuse among Android applications," presented at the 9th Int. Conf. Detection Intrusions Malware Vulnerability Assessment, Heraklion, Crete, Greece, Jul. 26–27, 2012.

[228] D. Kantola, E. Chin, W. He, and D. Wagner, "Reducing attack surfaces for intra-application communication in Android," presented at the Workshop Secur. Privacy Smartphones Mobile Devices, Raleigh, NC, USA, Oct. 19, 2012.

[229] H. Shahriar and H. M. Haddad, "Content provider leakage vulnerability detection in Android applications," presented at the 7th Int. Conf. Secur. Inf. Netw., Glasgow, Scotland, U.K., Sep. 9–11, 2014.

[230] M. Bugliesi, S. Calzavara, and A. Spanó, “Lintent: Towards security type-checking of Android applications,” presented at the Formal Techn. Distrib. Syst.-Joint IFIP WG 6.1 Int. Conf., held 8th Int. Federated Conf. Distrib. Comput. Techn., Florence, Italy, Jun. 3–5, 2013.

[231] Z. Lu and S. Mukhopadhyay, “Model-based static source code analysis of Java programs with applications to Android security,” presented at the 36th Annu. IEEE Comput. Softw. Appl. Conf., Izmir, Turkey, Jul. 16–20, 2012.

[232] S. Fahl, M. Harbach, T. Muders, L. Baumgörtner, B. Freisleben, and M. Smith, “Why Eve and Mallory love Android: An analysis of Android SSL (in)security,” presented at the ACM Conf. Comput. Commun. Secur., Raleigh, NC, USA, Oct. 16–18, 2012.

[233] S. Matsumoto and K. Sakurai, “A proposal for the privacy leakage verification tool for Android application developers,” presented at the 7th Int. Conf. Ubiquitous Inf. Manage. Commun., Kota Kinabalu, Malaysia, Jan. 17–19, 2013.

[234] P. Mutchler, A. Doup, J. Mitchell, C. Kruegel, and G. Vigna, “A large-scale study of mobile web app security,” in *Proc. Mobile Secur. Technol. Workshop*, San Jose, California, USA, May 21, 2015.

[235] M. Georgiev, S. Jana, and V. Shmatikov, “Breaking and fixing origin-based access control in hybrid web/mobile application frameworks,” presented at the 21st Annu. Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 23–26, 2014.

[236] X. Jin, X. Hu, K. Ying, W. Du, H. Yin, and G. N. Peri, “Code injection attacks on html5-based mobile apps: Characterization, detection and mitigation,” presented at the 2014 ACM SIGSAC Conf. Comput. Commun. Secur., Scottsdale, AZ, USA, Nov. 3–7, 2014.

[237] L. Onwuzurike and E. D. Cristofaro, “Danger is my middle name: Experimenting with SSL vulnerabilities in Android apps,” presented at the 8th ACM Conf. Secur. Privacy Wireless Mobile Netw., New York, NY, USA, Jun. 22–26, 2015.

[238] J. Wu, T. Cui, T. Ban, S. Guo, and L. Cui, “Paddyfrog: Systematically detecting confused deputy vulnerability in Android applications,” *Secur. Commun. Netw.*, vol. 8, no. 13, pp. 2338–2349, 2015.

[239] C. Mulliner, J. Oberheide, W. K. Robertson, and E. Kirda, “Patchdroid: Scalable third-party security patches for Android devices,” presented at the Annu. Comput. Secur. Appl. Conf., New Orleans, LA, USA, Dec. 9–13, 2013.

[240] L. Li, A. Bartel, J. Klein, and Y. L. Traon, “Automatically exploiting potential component leaks in Android applications,” presented at the 13th IEEE Int. Conf. Trust Secur. Privacy Comput. Commun., Beijing, China, Sep. 24–26, 2014.

[241] T. Vidas, N. Christin, and L. Cranor, “Curbing Android permission creep,” in *Proc. Web Secur. Privacy Workshop*, 2011, pp. 91–96.

[242] D. Sbirlea, M. Burke, S. Guarnieri, M. Pistoia, and V. Sarkar, “Automatic detection of inter-application permission leaks in Android applications,” *IBM J. Res. Develop.*, vol. 57, no. 6, pp. 10:1–10:12, Nov. 2013.

[243] S. Poeplau, Y. Fratantonio, A. Bianchi, C. Kruegel, and G. Vigna, “Execute this! analyzing unsafe and malicious dynamic code loading in Android applications,” presented at the 21st Annu. Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 23–26, 2014.

[244] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, “Pscout: Analyzing the Android permission specification,” presented at the ACM Conf. Comput. Commun. Secur., Raleigh, NC, USA, Oct. 16–18, 2012.

[245] C. Ren, Y. Zhang, H. Xue, T. Wei, and P. Liu, “Towards discovering and understanding task hijacking in Android,” presented at the 24th USENIX Secur. Symp., Washington, DC, USA, Aug. 12–14, 2015.

[246] Z. Han, L. Cheng, Y. Zhang, S. Zeng, Y. Deng, and X. Sun, “Systematic analysis and detection of misconfiguration vulnerabilities in Android smartphones,” presented at the 13th IEEE Int. Conf. Trust Secur. Privacy Comput. Commun., Beijing, China, Sep. 24–26, 2014.

[247] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, SCanDroid: Automated security certification of Android applications, Department of Computer Science, University of Maryland, College Park, Tech. Rep. CS-TR-4991, 2009.

[248] R. Vanciu and M. Abi-Antoun, “Finding architectural flaws using constraints,” presented at the 28th IEEE/ACM Int. Conf. Automated Softw. Eng., Silicon Valley, CA, USA, Nov. 11–15, 2013.

[249] L. Xing, X. Pan, R. Wang, K. Yuan, and X. Wang, “Upgrading your Android, elevating my malware: Privilege escalation through mobile OS updating,” presented at the IEEE Symp. Secur. Privacy, Berkeley, CA, USA, May 18–21, 2014.

[250] L. Wu, M. Grace, Y. Zhou, C. Wu, and X. Jiang, “The impact of vendor customizations on Android security,” presented at the ACM SIGSAC Conf. Comput. Commun. Secur., Berlin, Germany, Nov. 4–8, 2013.

[251] E. Smith and A. Coglio, “Android platform modeling and Android app verification in the ACL2 theorem prover,” presented at the 7th Int. Conf. Verified Softw.: Theories Tools Experiments, San Francisco, CA, USA, Jul. 18–19, 2015.

[252] D. Sounthiraraj, J. Sahs, G. Greenwood, Z. Lin, and L. Khan, “SMV-hunter: Large scale, automated detection of SSL/TLS man-in-the-middle vulnerabilities in Android apps,” in *21st Ann. Network Distributed Syst. Sec. Symp.*, NDSS 2014, San Diego, California, USA, Feb. 23–26, 2014.

[253] S. Bojjagani and V. N. Sastry, “Stamba: Security testing for Android mobile banking apps,” presented at the Advances Signal Process. Intell. Recognit. Syst. 2nd Int. Symp. Signal Process. Intell. Recognit. Syst., Trivandrum, India, Dec. 16–19, 2015.

[254] J. Huang, et al., “Supor: Precise and scalable sensitive user input detection for Android apps,” presented at the 24th USENIX Secur. Symp., Washington, DC, USA, Aug. 12–14, 2015.

[255] T. Li, et al., “Mayhem in the push clouds: Understanding and mitigating security hazards in mobile push-messaging services,” presented at the ACM SIGSAC Conf. Comput. Commun. Secur., Scottsdale, AZ, USA, Nov. 3–7, 2014.

[256] D. Vecchiato, M. Vieira, and E. Martins, “A security configuration assessment for Android devices,” presented at the 30th Annu. ACM Symp. Appl. Comput., Salamanca, Spain, Apr. 13–17, 2015.

[257] Y. Zhang, et al., “Vetting undesirable behaviors in Android apps with permission use analysis,” presented at the ACM SIGSAC Conf. Comput. Commun. Secur., Berlin, Germany, Nov. 4–8, 2013.

[258] X. Cui, J. Wang, L. C. K. Hui, Z. Xie, T. Zeng, and S.-M. Yiu, “Wechecker: Efficient and precise detection of privilege escalation vulnerabilities in Android apps,” presented at the 8th ACM Conf. Secur. Privacy Wireless Mobile Netw., New York, NY, USA, Jun. 22–26, 2015.

[259] C. Zuo, J. Wu, and S. Guo, “Automatically detecting SSL error-handling vulnerabilities in hybrid mobile web apps,” presented at the 10th ACM Symp. Inf. Comput. Commun. Secur., Singapore, Apr. 14–17, 2015.

[260] Z. Luoshi, N. Yan, W. Xiao, W. Zhaoguo, and X. Yibo, “A3: Automatic analysis of Android malware,” in *Proc. 1st Int. Workshop Cloud Comput. Inf. Secur.*, 2013.

[261] T. Vidas, J. Tan, J. Nahata, C. L. Tan, N. Christin, and P. Tague, “A5: Automated analysis of adversarial Android applications,” presented at the 4th ACM Workshop Secur. Privacy Smartphones Mobile Devices, Scottsdale, AZ, USA, Nov. 3–7, 2014.

[262] K. Lu, et al., “Checking more and alerting less: Detecting privacy leakages via enhanced data-flow analysis and peer voting,” presented at the 22nd Annu. Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 8–11, 2014.

[263] T. Bläsing, L. Batyuk, A.-D. Schmidt, S. A. Amtepe, and S. Albayrak, “An Android application sandbox system for suspicious software detection,” presened at the 5th Int. Conf. Malicious Unwanted Softw., Nancy, France, Oct. 19–20, 2010.

[264] H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck, “Structural detection of Android malware using embedded call graphs,” presented at the ACM Workshop Artif. Intell. Secur., Berlin, Germany, Nov. 4, 2013.

[265] O. S. Adebayo and N. AbdulAziz, “Android malware classification using static code analysis and apriori algorithm improved with particle swarm optimization,” in *Proc. 4th World Congr. Inf. Commun. Technol.*, 2014, pp. 123–128.

[266] V. M. Afonso, M. F. D. Amorim, A. R. A. Grgio, G. B. Junquera, and P. L. d. Geus, “Identifying Android malware using dynamically obtained features,” *J. Comput. Virology Hacking Techn.*, vol. 11, no. 1, pp. 9–17, 2015.

[267] S. Zhao, X. Li, G. Xu, L. Zhang, and Z. Feng, “Attack tree based Android malware detection with hybrid analysis,” presented at the 13th IEEE Int. Conf. Trust Secur. Privacy Comput. Commun., Beijing, China, Sep. 24–26, 2014.

[268] S. Liang, et al., "Sound and precise malware analysis for Android via pushdown reachability and entry-point saturation," presented at the ACM Workshop Secur. Privacy Smartphones Mobile Devices, Berlin, Germany, Nov. 8, 2013.

[269] T. Eder, M. Rodler, D. Vymazal, and M. Zeilinger, "Ananas—a framework for analyzing Android applications," presented at the 2013 Int. Conf. Availability Rel. Secur., Regensburg, Germany, Sep. 2–6, 2013.

[270] J. Crussell, C. Gibler, and H. Chen, "Andarwin: Scalable detection of semantically similar Android applications," presented at the 18th Eur. Symp. Res. Comput. Secur. Comput. Secur., Egham, U.K., Sep. 9–13, 2013.

[271] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Androidleaks: Automatically detecting potential privacy leaks in Android applications on a large scale," in *Proc. 5th Int. Conf. Trust Trustworthy Comput.*, 2012, pp. 291–307.

[272] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. van der Veen, and C. Platzer, "ANDRUBIS-1,000,000 apps later: A view on current Android malware behaviors," in *Proc. 3rd Int. Workshop Building Analysis Datasets Gathering Experience Returns Secur.*, 2014, pp. 3–17.

[273] M. Zhao, F. Ge, T. Zhang, and Z. Yuan, "AntiMalDroid: An efficient SVM-based malware detection framework for Android," presented at the 2nd Int. Conf. Inf. Comput. Appl, Qinhuangdao, China, Oct. 28–31, 2011.

[274] M. Nauman, S. Khan, and X. Zhang, "Apex: Extending Android permission model and enforcement with user-defined runtime constraints," presented at the 5th ACM Symp. Inf. Comput. Commun. Secur., Beijing, China, Apr. 13–16, 2010.

[275] S. Cheng, S. Luo, Z. Li, W. Wang, Y. Wu, and F. Jiang, "Static detection of dangerous behaviors in Android apps," presented at the 5th Int. Symp. Cyberspace Safety Secur., Zhangjiajie, China, Nov. 13–15, 2013.

[276] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "Appcontext: Differentiating malicious and benign mobile app behaviors using context," presented at the 37th IEEE/ACM Int. Conf. Softw. Eng., Florence, Italy, May 16–24, 2015.

[277] P. Gilbert, B.-G. Chun, L. P. Cox, and J. Jung, "Vision: Automated security validation of mobile apps at app markets," in *Proc. Second Int. Workshop Mobile Cloud Comput. Services*, 2011, pp. 21–26.

[278] T. Vidas and N. Christin, "Sweetening Android lemon markets: Measuring and combating malware in application marketplaces," presented at the 3rd ACM Conf. Data Appl. Secur. Privacy, San Antonio, TX, USA, Feb. 18–20, 2013.

[279] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, "Appintent: Analyzing sensitive data transmission in Android for privacy leakage detection," presented at the ACM SIGSAC Conf. Comput. Commun. Security, Berlin, Germany, Nov. 4–8, 2013.

[280] Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: Semantics-based detection of Android malware," presented at the 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng., Hong Kong, China, Nov. 16–22, 2014.

[281] V. Rastogi, Y. Chen, and W. Enck, "Appsplayground: Automatic security analysis of smartphone applications," presented at the 3rd ACM Conf. Data Appl. Secur. Privacy, San Antonio, TX, USA, Feb. 18–20, 2013.

[282] J. Huang, X. Zhang, L. Tan, P. Wang, and B. Liang, "Asdroid: Detecting stealthy behaviors in Android applications by user interface and program behavior contradiction," presented at the 36th Int. Conf. Softw. Eng., Hyderabad, India, May 31–Jun. 7, 2014.

[283] G. Petracca, Y. Sun, T. Jaeger, and A. Atamli, "Audroid: Preventing attacks on audio channels in mobile devices," presented at the 31st Annu. Comput. Secur. Appl. Conf., Los Angeles, CA, USA, Dec. 7–11, 2015.

[284] R. Xu, H. Sadi, and R. Anderson, "Aurasium: Practical policy enforcement for Android applications," presented at the 21th USENIX Secur. Symp., Bellevue, WA, USA, Aug. 8–10, 2012.

[285] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, "Autocog: Measuring the description-to-permission fidelity in Android applications," presented at the ACM SIGSAC Conf. Comput. Commun. Secur., Scottsdale, AZ, USA, Nov. 3–7, 2014.

[286] H. Huang, K. Chen, C. Ren, P. Liu, S. Zhu, and D. Wu, "Towards discovering and understanding unexpected hazards in tailoring antivirus software for Android," presented at the 10th ACM Symp. Inf. Comput. Commun. Secur., Singapore, Apr. 14–17, 2015.

[287] C. Bae, J. Jung, J. Nam, and S. Shin, "A collaborative approach on behavior-based Android malware detection," presented at the 11th Int. Conf. Secur. Privacy Commun. Netw., Dallas, TX, USA, Oct. 26–29, 2015.

[288] G. Bal, "Revealing privacy-impacting behavior patterns of smartphone applications," in *Proc. Mobile Secur. Technol. Workshop*, San Francisco, California, USA, May 24, 2012.

[289] L. Batyuk, M. Herpich, S. A. Camtepe, K. Raddatz, A.-D. Schmidt, and S. Albayrak, "Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications," presented at the 6th Int. Conf. Malicious Unwanted Softw., Fajardo, Puerto Rico, USA, Oct. 18–19, 2011.

[290] A. Bianchi, J. Corbetta, L. Invernizzi, Y. Fratantonio, C. Kruegel, and G. Vigna, "What the app is that? deception and countermeasures in the Android user interface," presented at the IEEE Symp. Secur. Privacy, San Jose, CA, USA, May 17–21, 2015.

[291] S. Holavanalli, et al., "Flow permissions for Android," presented at the 28th IEEE/ACM Int. Conf. Automated Softw. Eng., Silicon Valley, CA, USA, Nov. 11–15, 2013.

[292] A. Paturi, M. Cherukuri, J. Donahue, and S. Mukkamala, "Mobile malware visual analytics and similarities of attack toolkits (malware gene analysis)," presented at the 2013 Int. Conf. Collaboration Technol. Syst., San Diego, CA, USA, May 20–24, 2013.

[293] S. Ma, et al., "Detecting GPS information leakage in Android applications," presented at the IEEE Global Commun. Conf., Atlanta, GA, USA, Dec. 9–13, 2013.

[294] G. Canfora, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Detecting Android malware using sequences of system calls," presented at the 3rd Int. Workshop Softw. Develop. Lifecycle Mobile, Bergamo, Italy, Aug. 31–Sep. 4, 2015.

[295] G. Canfora, F. Mercaldo, and C. A. Visaggio, "A classifier of malicious Android applications," presented at the Int. Conf. Availability Rel. Secur., Regensburg, Germany, Sep. 2–6, 2013.

[296] M. Zhang and H. Yin, "Efficient, context-aware privacy leakage confinement for Android applications without firmware modding," presented at the 9th ACM Symp. Inf. Comput. Commun. Secur., Kyoto, Japan, Jun. 3–6, 2014.

[297] S. Lortz, H. Mantel, A. Starostin, T. Bähr, D. Schneider, and A. Weber, "Cassandra: Towards a certifying app store for Android," presented at the 4th ACM Workshop Secur. Privacy Smartphones Mobile Devices, Scottsdale, AZ, USA, Nov. 3–7, 2014.

[298] L. Cen, C. S. Gates, L. Si, and N. Li, "A probabilistic discriminative model for Android malware detection with decompiled source code," *IEEE Trans. Depend. Secure Comput.*, vol. 12, no. 4, pp. 400–412, Jul./Aug. 2015.

[299] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," presented at the 36th Int. Conf. Softw. Eng., Hyderabad, India, May 31–Jun. 7, 2014.

[300] K. Chen, P. Liu, and Y. Zhang, "Achieving accuracy and scalability simultaneously in detecting application clones on Android markets," presented at the 36th Int. Conf. Softw. Eng., Hyderabad, India, May 31–Jun. 7, 2014.

[301] C.-M. Chen, J.-M. Lin, and G. H. Lai, "Detecting mobile application malicious behaviors based on data flow of source code," presented at the Int. Conf. Trustworthy Syst. Appl., Taichung, Taiwan, Jun. 9–10, 2014.

[302] A. Reina, A. Fattori, and L. Cavallaro, "A system call-centric analysis and stimulation technique to automatically reconstruct Android malware behaviors," in *Proc. ACM Eur. Workshop Syst. Secur.*, 2013.

[303] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "Copperdroid: Automatic reconstruction of Android malware behaviors," presented at the 22nd Annu. Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 8–11, 2014.

[304] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-based malware detection system for Android," presented at the 1st ACM Workshop Secur. Privacy Smartphones Mobile Devices, Chicago, IL, USA, Oct. 17, 2011.

[305] C. Yang, G. Yang, A. Gehani, V. Yegneswaran, D. Tariq, and G. Gu, "Using provenance patterns to vet sensitive behaviors in Android apps," presented at the 11th Int. Conf. Secur. Privacy Commun. Netw., Dallas, TX, USA, Oct. 26–29, 2015.

[306] G. Dai, J. Ge, M. Cai, D. Xu, and W. Li, "SVM-based malware detection for Android applications," presented at the 8th ACM Conf. Secur. Privacy Wireless Mobile Netw., New York, NY, USA, Jun. 22–26, 2015.

[307] Y. Zhou, K. Singh, and X. Jiang, "Owner-centric protection of unstructured data on smartphones," presented at the 7th Int. Conf. Trust Trustworthy Comput., Heraklion, Crete, Greece, Jun. 30–Jul. 2, 2014.

[308] X. Wang, K. Sun, Y. Wang, and J. Jing, "Deepdroid: Dynamically enforcing enterprise policy on Android devices," presented at the 22nd Annu. Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 8–11, 2014.

[309] X. Pan, Y. Zhongyang, Z. Xin, B. Mao, and H. Huang, "Defensor: Lightweight and efficient security-enhanced framework for Android," presented at the 13th IEEE Int. Conf. Trust Secur. Privacy Comput. Commun., Beijing, China, Sep. 24–26, 2014.

[310] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and J. B. Alís, "Dendroid: A text mining approach to analyzing and classifying code structures in Android malware families," *Expert Syst. Appl.*, vol. 41, no. 4, pp. 1104–1117, 2014.

[311] W. Klieber, L. Flynn, A. Bhosale, L. Jia, and L. Bauer, "Android taint flow analysis for app sets," presented at the 3rd ACM SIGPLAN Int. Workshop State Art Java Program Analysis, Edinburgh, U.K., Jun. 12, 2014.

[312] J. Crussell, C. Gibler, and H. Chen, "Attack of the clones: Detecting cloned applications on Android markets," presented at the 17th Eur. Symp. Res. Comput. Secur, Pisa, Italy, Sep. 10–12, 2012.

[313] S. Bhandari, R. Gupta, V. Laxmi, M. S. Gaur, A. Zemmari, and M. Anikeev, "Draco: Droid analyst combo an Android malware analysis framework," presented at the 8th Int. Conf. Secur. Inf. Netw., Sochi, Russian Federation, Sep. 8–10, 2015.

[314] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of Android malware in your pocket," presented at the 21st Annu. Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 23–26, 2014.

[315] Y. Li, T. Shen, X. Sun, X. Pan, and B. Mao, "Detection, classification and characterization of Android malware using API data dependency," presented at the 11th Int. Conf. Secur. Privacy Commun. Netw., Dallas, TX, USA, Oct. 26–29, 2015.

[316] J. Gajrani, J. Sarswat, M. Tripathi, V. Laxmi, M. S. Gaur, and M. Conti, "A robust dynamic analysis system preventing sandbox detection by Android malware," presented at the 8th Int. Conf. Secur. Inf. Netw., Sochi, Russian Federation, Sep. 8–10, 2015.

[317] M. Zheng, M. Sun, and J. C. S. Lui, "Droid analytics: A signature based analytic system to collect, extract, analyze and associate Android malware," presented at the 12th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./11th IEEE Int. Symp. Parallel Distrib. Process. Appl./12th IEEE Int. Conf. Ubiquitous Comput. Commun., Melbourne, Australia, Jul. 16–18, 2013.

[318] S.-H. Seo, A. Gupta, A. M. Sallam, E. Bertino, and K. Yim, "Detecting mobile malware threats to homeland security through static analysis," *J. Netw. Comput. Appl.*, vol. 38, pp. 43–53, 2014.

[319] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in Android," presented at the 9th Int. ICST Conf. Secur. Privacy Commun. Netw., Sydney, NSW, Australia, Sep. 25–28, 2013.

[320] H. M. J. Almohri, D. D. Yao, and D. G. Kafura, "DroidBarrier: Know what is executing on your Android," presented at the 4th ACM Conf. Data Appl. Secur. Privacy, San Antonio, TX, USA, Mar. 03–05, 2014.

[321] W.-C. Wu and S.-H. Hung, "DroidDolphin: A dynamic Android malware detection framework using big data and machine learning," presented at the Conf. Res. Adaptive Convergent Syst., Towson, Maryland, USA, Oct. 5–8, 2014.

[322] H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "DroidKin: Lightweight detection of Android apps similarity," presented at the 10th Int. ICST Conf. Secur. Privacy Commun. Netw., Beijing, China, Sep. 24–26, 2014.

[323] L. Deshotels, V. Notani, and A. Lakhotia, "DroidLegacy: Automated familial classification of Android malware," presented at the 3rd ACM SIGPLAN Program Protection Reverse Eng. Workshop, San Diego, CA, USA, Jan. 25, 2014.

[324] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "DroidMat: Android malware detection through manifest and API calls tracing," presented at the 7th Asia Joint Conf. Inf. Secur., Kaohsiung, Taiwan, Aug. 9–10, 2012.

[325] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. A. Porras, "DroidMiner: Automated mining and characterization of fine-grained malicious behaviors in Android applications," presented at the 19th Eur. Symp. Res. Comput. Secur., Wroclaw, Poland, Sep. 7–11, 2014.

[326] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party Android marketplaces," presented at the 2nd ACM Conf. Data Appl. Secur. Privacy, San Antonio, TX, USA, Feb. 7–9, 2012.

[327] W. Luo, S. Xu, and X. Jiang, "Real-time detection and prevention of Android SMS permission abuses," presented at the 1st Int. workshop Secur. Embedded Syst. Smartphones, Hangzhou, China, May 8, 2013.

[328] A. M. Aswini and P. Vinod, "Droid permission miner: Mining prominent permissions for Android malware analysis," in *Proc. 5th Int. Conf. Appl. Digit. Inf. Web Technol.*, 2014, pp. 81–86.

[329] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets," in *19th Ann. Netw. Dist. Syst. Sec. Symp.*, NDSS 2012, San Diego, California, USA, Feb. 5–8, 2012.

[330] Y. Wang, J. Zheng, C. Sun, and S. Mukkamala, "Quantitative security risk assessment of Android permissions and applications," presented at the 27th Annu. IFIP WG 11.3 Conf. Data Appl. Secur. Privacy., Newark, NJ, USA, Jul. 15–17, 2013.

[331] M. I. Gordon, D. Kim, J. Perkins, L. Gilham, N. Nguyen, and M. Rinard, "Information-flow analysis of Android applications in DroidSafe," presented at the 22nd Annu. Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 8–11, 2014.

[332] L. K. Yan and H. Yin, "DroidScope: Seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis," presented at the 21th USENIX Secur. Symp., Bellevue, WA, USA, Aug. 8–10, 2012.

[333] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware Android malware classification using weighted contextual API dependency graphs," presented at the ACM SIGSAC Conf. Comput. Commun. Secur., Scottsdale, AZ, USA, Nov. 3–7, 2014.

[334] X. Sun, Y. Zhongyang, Z. Xin, B. Mao, and L. Xie, "Detecting code reuse in Android applications using component-based control flow graph," presented at the 29th IFIP TC 11 Int. Conf. ICT Syst. Secur. Privacy Protection, Marrakech, Morocco, Jun. 2–4, 2014.

[335] M. Zheng, M. Sun, and J. C. S. Lui, "DroidTrace: A ptrace based Android dynamic analysis system with forward execution capability," presented at the Int. Wireless Commun. Mobile Comput. Conf., Nicosia, Cyprus, Aug. 4–8, 2014.

[336] S. Sakamoto, K. Okuda, R. Nakatsuka, and T. Yamauchi, "DroidTrack: Tracking and visualizing information diffusion for preventing information leakage on Android," *J. Internet Services Inf. Secur.*, vol. 4, no. 2, pp. 55–69, 2014.

[337] W. Hu, D. Octeau, P. D. McDaniel, and P. Liu, "Duet: Library integrity verification for Android applications," presented at the 7th ACM Conf. Secur. Privacy Wireless Mobile Netw., Oxford, U.K., Jul. 23–25, 2014.

[338] R. Wang, et al., "EASEAndroid: Automatic policy analysis and refinement for security enhanced Android via large-scale semi-supervised learning," presented at the 24th USENIX Secur. Symp., Washington, D.C., USA, Aug. 12–14, 2015.

[339] K. O. Elish, D. Yao, and B. G. Ryder, "User-centric dependence analysis for identifying malicious mobile apps," in *IEEE Mob. Sec. Tech. (MoST), in conjunction with the IEEE Symp. Sec. Privacy*, MoST 2012, San Francisco, California, USA, May 24, 2012.

[340] R. Fedler, M. Kulicke, and J. Schtte, "An antivirus API for Android malware recognition," presented at the 8th Int. Conf. Malicious Unwanted Softw.: "The Americas", Fajardo, PR, USA, Oct. 22–24, 2013.

[341] K. Zhao, D. Zhang, X. Su, and W. Li, "Fest: A feature extraction and selection tool for Android malware detection," presented at the IEEE Symp. Comput. Commun., Larnaca, Cyprus, Jul. 6–9, 2015.

[342] G. Russello, A. B. Jimenez, H. Naderi, and W. v. d. Mark, "FireDroid: Hardening security in almost-stock Android," presented at the Annu. Comput. Secur. Appl. Conf., New Orleans, LA, USA, Dec. 9–13, 2013.

[343] T. Ravitch, E. R. Creswick, A. Tomb, A. Foltzer, T. Elliott, and L. Casburn, "Multi-app security analysis with fuse: Statically detecting Android app collusion," presented at the 4th Program Protection Reverse Eng. Workshop, New Orleans, LA, USA, Dec. 9, 2014.

[344] C. S. Gates, et al., "Generating summary risk scores for mobile applications," *IEEE Trans. Depend. Secure Comput.*, vol. 11, no. 3, pp. 238–251, May/June 2014.

[345] M. Graa, N. Cuppens-Boulahia, F. Cuppens, and A. R. Cavalli, "Protection against code obfuscation attacks based on control dependencies in Android systems," presented at the IEEE 8th Int. Conf. Softw. Secur. Rel., San Francisco, CA, USA, Jun. 30–Jul. 2, 2014.

[346] M. Graa, N. Cuppens-Boulahia, F. Cuppens, and A. R. Cavalli, "Detecting control flow in smarphones: Combining static and dynamic analyses," in *Proc. 4th Int. Conf. Cyberspace Safety Secur.* 2012, pp. 33–47.

[347] A. Abraham, R. Andriatsimandefitra, A. Brunelat, J.-F. Lalande, and V. V. T. Tong, "GroddDroid: A gorilla for triggering malicious behaviors," presented at the 10th Int. Conf. Malicious Unwanted Softw., Fajardo, PR, USA, Oct. 20–22, 2015.

[348] Y. J. Ham, D. Moon, H.-W. Lee, J. D. Lim, and J. N. Kim, "Android mobile application system call event pattern analysis for determination of malicious attack," *Int. J. Secur. Appl.*, vol. 8, no. 1, pp. 231–246, 2014.

[349] C.-Y. Huang, Y.-T. Tsai, and C.-H. Hsu, "Performance evaluation on permission-based detection for Android malware," *Adv. Intell. Syst. Appl.*, vol. 2, pp. 111–120, 2013.

[350] L. Zhang, Y. Zhang, and T. Zang, "Detecting malicious behaviors in repackaged Android apps with loosely-coupled payloads filtering scheme," presented at the 10th Int. ICST Int. Conf. Secur. Privacy Commun. Netw., Beijing, China, Sep. 24–26, 2014.

[351] B. Davis, B. Sanders, A. Khodaverdian, and H. Chen, "I-ARM-droid: A rewriting framework for in-app reference monitors for Android applications," *Mobile Secur. Technol.*, San Francisco, California, USA, May 24, 2012.

[352] K. O. Elish, D. Yao, and B. G. Ryder, "On the need of precise inter-app ICC classification for detecting Android malware collusions," in *IEEE Mobile Secur. Technol. (MoST), in Conjunction IEEE Symp. Secur. Privacy*, MoST 2015, San Jose, California, USA, May 21, 2015.

[353] M. D. Ernst, et al., "Collaborative verification of information flow for a high-assurance app store," presented at the ACM SIGSAC Conf. Comput. Commun. Secur., 2014, pp. 1092–1104.

[354] W. You, B. Liang, J. Li, W. Shi, and X. Zhang, "Android implicit information flow demystified," presented at the 10th ACM Symp. Inf. Comput. Commun. Secur., Singapore, Apr. 14–17, 2015.

[355] M. Kühnel, M. Smieschek, and U. Meyer, "Fast identification of obfuscation and mobile advertising in mobile malware," presented at the IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, Aug. 20–22, 2015.

[356] T. Isohara, K. Takemori, and A. Kubota, "Kernel-based behavior analysis for Android malware detection," presented at the 7th Int. Conf. Comput. Intell. Secur., Sanya, Hainan, China, Dec. 3–4, 2011.

[357] C. Jeon, W. Kim, B. Kim, and Y. Cho, "Enhancing security enforcement on unmodified Android," presented at the 28th Annu. ACM Symp. Appl. Comput., Coimbra, Portugal, Mar. 18–22, 2013.

[358] Y. Jeong, H.-T. Lee, S. Cho, S. Han, and M. Park, "A kernel-based monitoring approach for analyzing malicious behavior on Android," presented at the Symp. Appl. Comput., Gyeongju, Republic of Korea, Mar. 24–28, 2014.

[359] H. Jiao, X. Li, L. Zhang, G. Xu, and Z. Feng, "Hybrid detection using permission analysis for Android malware," presented at the 10th Int. Conf. Secur. Privacy Commun. Netw., Beijing, China, Sep. 24–26, 2014.

[360] A. F. A. Kadir, N. Stakhanova, and A. A. Ghorbani, "Android Botnets: What URLs are telling us," presented at the 9th Int. Conf. Netw. Syst. Secur., New York, NY, USA, Nov. 3–5, 2015.

[361] M. Karami, M. Elsabagh, P. Najafiborazjani, and A. Stavrou, "Behavioral analysis of Android applications using automated instrumentation," presented at the 7th Int. Conf. Softw. Secur. Rel., Gaithersburg, MD, USA, 18–20 Jun. 2013.

[362] P. M. Kate and S. V. Dhavale, "Two phase static analysis technique for Android malware detection," presented at the 3rd Int. Symp. Women Comput. Informat., Kochi, India, Aug. 10–13, 2015.

[363] D.-U. Kim, J. Kim, and S. Kim, "A malicious application detection framework using automatic feature extraction tool on Android market," in *Proc. 3rd Int. Conf. Comput. Sci. Inf. Technol.* 2013.

[364] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," presented at the ACM Conf. Comput. Commun. Secur., Chicago, IL, USA, Nov. 9–13, 2009.

[365] Z. Yang and M. Yang, "LeakMiner: Detect information leakage on Android with static taint analysis," presented at the 3rd World Congr. Softw. Eng., Hong Kong, China, 2012.

[366] S.-H. Lee and S.-H. Jin, "Warning system for detecting malicious applications on Android system," *Int. J. Comput. Commun. Eng.*, vol. 2, no. 3, 2013, Art. no. 324.

[367] Q. Li and X. Li, "Android malware detection based on static analysis of characteristic tree," presented at the Int. Conf. Cyber-Enabled Distrib. Comput. Knowl. Discovery, Xi'an, China, Sep. 17–19, 2015.

[368] S. Li, J. Chen, T. Spyridopoulos, P. Andriotis, R. Ludwiniak, and G. Russell, "Real-time monitoring of privacy abuses and intrusion detection in Android system," presented at the 3rd Int. Conf. Human Aspects Inf. Secur. Privacy Trust, Los Angeles, CA, USA, Aug. 2–7, 2015.

[369] S. Ma, S. Wang, D. Lo, R. H. Deng, and C. Sun, "Active semi-supervised approach for checking app behavior against its description," presented at the 39th IEEE Annu. Comput. Softw. Appl. Conf., Taichung, Taiwan, Jul. 1–5, 2015.

[370] G. Dini, F. Martinelli, A. Saracino, and D. Sgandurra, "MADAM: A multi-level anomaly detector for Android malware," presented at the 6th Int. Conf. Math. Methods Models Archit. Comput. Netw. Secur., St. Petersburg, Russia, Oct. 17–19, 2012.

[371] B. Sanz, "MAMA: Manifest analysis for malware detection in Android," *Cybern. Syst.*, vol. 44, no. 6/7, pp. 469–488, 2013.

[372] S. Feldman, D. Stadther, and B. Wang, "Manilyzer: Automated Android malware detection through manifest analysis," presented at the 11th IEEE Int. Conf. Mobile Ad Hoc Sensor Syst., Philadelphia, PA, USA, Oct. 28–30, 2014.

[373] M. Lindorfer, M. Neugschwandtner, and C. Platzer, "Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis," presented at the 39th IEEE Annu. Comput. Softw. Appl. Conf., Taichung, Taiwan, Jul. 1–5, 2015.

[374] K. Chen, et al., "Finding unknown malice in 10 seconds: Mass vetting for new threats at the Google-play scale," presented at the 24th USENIX Secur. Symp., Washington, DC, USA, Aug. 12–14, 2015.

[375] S. Chakradeo, B. Reaves, P. Traynor, and W. Enck, "Mast: Triage for market-scale mobile malware analysis," presented at the 6th ACM Conf. Secur. Privacy Wireless Mobile Netw., Budapest, Hungary, Apr. 17–19, 2013.

[376] M. Z. Mas'ud, S. Sahib, M. F. Abdollah, S. R. Selamat, R. Yusof, and R. Ahmad, "Profiling mobile malware behaviour through hybrid malware analysis approach," presented at the 9th Int. Conf. Inf. Assurance Secur., Gammarth, Tunisia, Dec. 4–6, 2013.

[377] W. Hu, J. Tao, X. Ma, W. Zhou, S. Zhao, and T. Han, "MigDroid: Detecting app-repackaging Android malware via method invocation graph," presented at the 23rd Int. Conf. Comput. Commun. Netw., Shanghai, China, Aug. 4–7, 2014.

[378] J. Milosevic, A. Dittrich, A. Ferrante, and M. Malek, "A resource-optimized approach to efficient early detection of mobile malware," presented at the 9th Int. Conf. Availability Rel. Secur., Fribourg, Switzerland, Sep. 8–12, 2014.

[379] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann, "Mobile-sandbox: Having a deeper look into Android applications," presented at the 28th Annu. ACM Symp. Appl. Comput., Coimbra, Portugal, Mar. 18–22, 2013.

[380] J. Xu, "Mobsafe: Cloud computing based forensic analysis for massive mobile applications using data mining," *Tsinghua Sci. Technol.*, vol. 18, no. 4, pp. 418–427, 2013.

[381] V. Moonsamy, J. Rong, and S. Liu, "Mining permission patterns for contrasting clean and malicious Android applications," *Future Generation Comput. Syst.*, vol. 36, pp. 122–132, 2014.

[382] V. Moonsamy, J. Rong, S. Liu, G. Li, and L. M. Batten, "Contrasting permission patterns between clean and malicious Android applications," presented at the 9th Int. ICST Conf. Secur. Privacy Commun. Netw., Sydney, NSW, Australia, Sep. 25–28, 2013.

[383] R. Wang, L. Xing, X. Wang, and S. Chen, "Unauthorized origin crossing on mobile platforms: Threats and mitigation," presented at the ACM SIGSAC Conf. Comput. Commun. Secur., Berlin, Germany, Nov. 4–8, 2013.

[384] V. Avdiienko, et al., "Mining apps for abnormal usage of sensitive data," presented at the 37th IEEE/ACM Int. Conf. Softw. Eng., Florence, Italy, May 16–24, 2015.

[385] J. Jeong, D. Seo, C. Lee, J. Kwon, H. Lee, and J. Milburn, "Mysterychecker: Unpredictable attestation to detect repackaged malicious applications in Android," presented at the 9th Int. Conf. Malicious Unwanted Softw.: Americas, Fajardo, PR, USA, Oct. 28–30, 2014.

[386] C. Qian, X. Luo, Y. Shao, and A. T. S. Chan, "On tracking information flows through JNI in Android applications," presented at the 44th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw., Atlanta, GA, USA, Jun. 23–26, 2014.

[387] Y. Nishimoto, N. Kajiwara, S. Matsumoto, Y. Hori, and K. Sakurai, "Detection of Android API call using logging mechanism within Android framework," presented at the 9th Int. ICST Conf. Secur. Privacy Commun. Netw., Sydney, NSW, Australia, Sep. 25–28, 2013.

[388] A. Ali-Gombe, I. Ahmed, G. G. Richard III, and V. Roussev, "Opseq: Android malware fingerprinting," presented at the 5th Program Protection Reverse Eng. Workshop, Los Angeles, CA, USA, Dec. 8, 2015.

[389] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, "Paranoid Android: Versatile protection for smartphones," presented at the 26th Annu. Comput. Secur. Appl. Conf., Austin, Texas, USA, Dec. 6–10, 2010.

[390] L. Xie, X. Zhang, J.-P. Seifert, and S. Zhu, "pBMDS: A behavior-based malware detection system for cellphone devices," presented at the 3rd ACM Conf. Wireless Netw. Secur., Hoboken, NJ, USA, Mar. 22–24, 2010.

[391] K. Z. Chen, et al., "Contextual policy enforcement in Android applications with permission event graphs," presented at the 20th Annu. Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 24–27, 2013.

[392] N. Peiravian and X. Zhu, "Machine learning for Android malware detection using permission and API calls," presented at the IEEE 25th Int. Conf. Tools Artif. Intell., Herndon, VA, USA, Nov. 4–6, 2013.

[393] H. Peng, et al., "Using probabilistic generative models for ranking risks of Android apps," presented at the ACM Conf. Comput. Commun. Secur., Raleigh, NC, USA, Oct. 16–18, 2012.

[394] W. Xu, F. Zhang, and S. Zhu, "Permlyzer: Analyzing permission usage in Android applications," presented at the IEEE 24th Int. Symp. Softw. Rel. Eng., Pasadena, CA, USA, Nov. 4–7, 2013.

[395] G. Dini, F. Martinelli, A. Saracino, and D. Sgandurra, "Probabilistic contract compliance for mobile applications," presented at the Int. Conf. Availability Rel. Secur., Regensburg, Germany, Sep. 2–6, 2013.

[396] T.-H. Ho, D. J. Dean, X. Gu, and W. Enck, "PREC: Practical root exploit containment for Android devices," presented at the 4th ACM Conf. Data Appl. Secur. Privacy, San Antonio, TX, USA, Mar. 3–5, 2014.

[397] S. Hao, B. Liu, S. Nath, W. G. J. Halfond, and R. Govindan, "PUMA: Programmable UI-automation for large-scale dynamic analysis of mobile apps," presented at the 12th Annu. Int. Conf. Mobile Syst. Appl. Services, Bretton Woods, NH, USA, Jun. 16–19, 2014.

[398] A. Gianazza, F. Maggi, A. Fattori, L. Cavallaro, and S. Zanero, "PuppetDroid: A user-centric UI exerciser for automatic dynamic analysis of similar Android applications," *arXiv:1402.4826*, Feb. 2014.

[399] D. Quan, L. Zhai, F. Yang, and P. Wang, "Detection of Android malicious apps based on the sensitive behaviors," presented at the 13th IEEE Int. Conf. Trust Secur. Privacy Comput. Commun., Beijing, China, Sep. 24–26, 2014.

[400] L. Dolberg, Q. Jérôme, J. François, R. State, and T. Engel, "RAMSES: Revealing Android malware through string extraction and selection," presented at the 10th Int. ICST Conf. Secur. Privacy Commun. Netw., Beijing, China, Sep. 24–26, 2014.

[401] C. Guo, J. Zhang, J. Yan, Z. Zhang, and Y. Zhang, "Characterizing and detecting resource leaks in Android applications," presented at the 28th IEEE/ACM Int. Conf. Automated Softw. Eng., Silicon Valley, CA, USA, Nov. 11–15, 2013.

[402] Y. Shao, X. Luo, C. Qian, P. Zhu, and L. Zhang, "Towards a scalable resource-driven approach for detecting repackaged Android applications," presented at the 30th Annu. Comput. Secur. Appl. Conf., New Orleans, LA, USA, Dec. 8–12, 2014.

[403] Y. Jing, G.-J. Ahn, Z. Zhao, and H. Hu, "RiskMon: Continuous and automated risk assessment of mobile applications," presented at the 4th ACM Conf. Data Appl. Secur. Privacy, San Antonio, TX, USA, Mar. 3–5, 2014.

[404] Y. Jing, G.-J. Ahn, Z. Zhao, and H. Hu, "Towards automated risk assessment and mitigation of mobile applications," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 5, pp. 571–584, Sep./Oct. 2015.

[405] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: Scalable and accurate zero-day Android malware detection," presented at the 10th Int. Conf. Mobile Syst. Appl. Services, Ambleside, U. K., Jun. 25–29, 2012.

[406] J. Hoffmann, M. Ussath, T. Holz, and M. Spreitzenbarth, "Slicing droids: Program slicing for smali code," presented at the 28th Annu. ACM Symp. Appl. Comput., Coimbra, Portugal, Mar. 18–22, 2013.

[407] J. Sahs and L. Khan, "A machine learning approach to Android malware detection," presented at the Eur. Intell. Secur. Informat. Conf., Odense, Denmark, Aug. 22–24, 2012.

[408] B. Sanz, I. Santos, X. Ugarte-Pedrero, C. Laorden, J. Nieves, and P. G. Bringas, "Instance-based anomaly method for Android malware detection," presented at the 10th Int. Conf. Secur. Cryptography, Reykjavk, Iceland, Jul. 29–31, 2013.

[409] B. P. Sarma, N. Li, C. S. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Android permissions: A perspective combining risks and benefits," presented at the 17th ACM Symp. Access Control Models Technol., Newark, NJ, USA, Jun. 20–22, 2012.

[410] L. Sayfullina, et al., "Efficient detection of zero-day Android malware using normalized Bernoulli Naive Bayes," presented at the IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, Aug. 20–22, 2015.

[411] J. Kim, Y. Yoon, K. Yi, J. Shin, and S. Center, "SCANDAL: Static analyzer for detecting privacy leaks in Android applications," in *IEEE Mobile Secur. Technol. (MoST), Conjunction IEEE Sympo. Secur. Privacy,* MoST 2012, San Francisco, California, USA, May 24, 2012.

[412] A.-D. Schmidt, et al., "Static analysis of executables for collaborative malware detection on Android," presented at the IEEE Int. Conf. Commun., Dresden, Germany, Jun. 14–18, 2009.

[413] Y.-D. Lin, Y.-C. Lai, C.-H. Chen, and H.-C. Tsai, "Identifying Android malicious repackaged applications by thread-grained system call sequences," *Comput. Secur.*, vol. 39, pp. 340–350, 2013.

[414] R. Andriatsimandefitra and V. V. T. Tong, "Capturing Android malware behaviour using system flow graph," presented at the 8th Int. Conf. Netw. Syst. Secur., Xi'an, China, Oct. 15–17, 2014.

[415] A. Shabtai, Y. Fledel, and Y. Elovici, "Automated static code analysis for classifying Android applications using machine learning," presented at the Int. Conf. Comput. Intell. Secur., Nanning, China, Dec. 11–14, 2010.

[416] B. Shebaro, O. Oluwatimi, and E. Bertino, "Context-based access control systems for mobile devices," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 2, pp. 150–163, Mar./Apr. 2015.

[417] T. Shen, Y. Zhongyang, Z. Xin, B. Mao, and H. Huang, "Detect Android malware variants using component based topology graph," presented at the 13th IEEE Int. Conf. Trust Secur. Privacy Comput. Commun., Beijing, China, Sep. 24–26, 2014.

[418] L. Apvrille and A. Apvrille, "Identifying unknown Android malware with feature extractions and classification techniques," presented at the IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, Aug. 20–22, 2015.

[419] C. Zheng, et al., "SmartDroid: An automatic system for revealing UI-based trigger conditions in Android applications," presented at the Workshop Secur. Privacy Smartphones Mobile Devices, Raleigh, NC, USA, Oct. 19, 2012.

[420] F. Song and T. Touili, "Model-checking for Android malware detection," presented at the 12th Asian Symp. Program. Languages Syst., Singapore, Nov. 17–19, 2014.

[421] Y. Zhauniarovich, M. Ahmad, O. Gadyatskaya, B. Crispo, and F. Massacci, "StaDyna: Addressing the problem of dynamic code updates in the security analysis of Android applications," presented at the 5th ACM Conf. Data Appl. Secur. Privacy, San Antonio, TX, USA, Mar. 2–4, 2015.

[422] X. Su, M. Chuah, and G. Tan, "Smartphone dual defense protection framework: Detecting malicious applications in Android markets," presented at 8th Int. Conf. Mobile Ad-hoc Sensor Netw., Chengdu, China, Dec. 14–16, 2012.

[423] W. Enck, et al., "TaintDroid: An information flow tracking system for real-time privacy monitoring on smartphones," in *Proc. 9th USENIX Conf. Operating Syst. Des. Implementation*, 2010, pp. 393–407.

[424] F. Tchakount and P. Dayang, "System calls analysis of malwares on Android," *Int. J. Sci. Tecnology*, vol. 2, pp. 669–674, 2013.

[425] X. Xiao, X. Xiao, Y. Jiang, and Q. Li, "Detecting mobile malware with TMSVM," presented at the 10th Int. ICST Conf. Secur. Privacy Commun. Netw., Beijing, China, Sep. 24–26, 2014.

[426] V. van der Veen, H. Bos, and C. Rossow, "Dynamic analysis of Android malware," Ph.D. dissertation, Dept. Comput. Sci., VU Univ. Amsterdam, Netherlands, 2013.

[427] M. Dam, G. L. Guernic, and A. Lundblad, "TreeDroid: A tree automaton based approach to enforcing data processing policies," presented at the ACM Conf. Comput. Commun. Secur., Raleigh, NC, USA, Oct. 16–18, 2012.

[428] K. O. Elish, D. D. Yao, B. G. Ryder, and X. Jiang, "A static assurance analysis of Android applications," Department of Computer Science, Virginia Polytechnic Institute & State University, Tech. Rep. TR-13-03, 2013.

[429] F. Zhang, H. Huang, S. Zhu, D. Wu, and P. Liu, "ViewDroid: Towards obfuscation-resilient mobile application repackaging detection," presented at the 7th ACM Conf. Secur. Privacy Wireless Mobile Netw., Oxford, U. K., Jul. 23–25, 2014.

[430] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, "Exploring permission-induced risk in Android applications for malicious application detection," *IEEE Trans. Inf. Forensics Secur.*, vol. 9, no. 11, pp. 1869–1882, Nov. 2014.

[431] E. R. Wognsen, H. S. Karlsen, M. C. Olesen, and R. R. Hansen, "Formalisation and analysis of Dalvik bytecode," *Sci. Comput. Program.*, vol. 92, pp. 25–55, 2014.

[432] H. Wang, Y. Guo, Z. Ma, and X. Chen, "WuKong: A scalable and accurate two-phase approach to Android app clone detection," presented at the Int. Symp. Softw. Testing Anal., Baltimore, MD, USA, Jul. 12–17, 2015.

[433] G. Russello, B. Crispo, E. Fernandes, and Y. Zhauniarovich, "YAASE: Yet another Android security extension," presented at the IEEE 3rd Int. Conf. Privacy Secur. Risk Trust and IEEE 3rd Int. Conf. Social Comput., Boston, MA, USA, Oct. 9–11, 2011.

[434] S. Y. Yerima, S. Sezer, G. McWilliams, and I. Muttik, "A new Android malware detection approach using Bayesian classification," presented at the 27th IEEE Int. Conf. Adv. Inf. Netw. Appl., Barcelona, Spain, Mar. 25–28, 2013.

[435] J. P. Achara, J.-D. Lefruit, V. Roca, and C. Castelluccia, "Detecting privacy leaks in the RATP app: How we proceeded and what we found," *J. Comput. Virology Hacking Techn.*, vol. 10, no. 4, pp. 229–238, 2014.

[436] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," presented at the 5th ACM Conf. Secur. Privacy Wireless Mobile Netw., Tucson, AZ, USA, Apr. 16–18, 2012.

[437] W. Yang, J. Li, Y. Zhang, Y. Li, J. Shu, and D. Gu, "APKLancet: Tumor payload diagnosis and purification for Android applications," presented at the 9th ACM Symp. Inf. Comput. Commun. Secur., Kyoto, Japan, Jun. 3–6, 2014.

[438] H. Han, R. Li, J. Hu, and M. Qiu, "Context awareness through reasoning on private analysis for Android application," presented at the IEEE 2nd Int. Conf. Cyber Secur. Cloud Comput., New York, NY, USA, Nov. 3–5, 2015.

[439] F. Roesner and T. Kohno, "Securing embedded user interfaces: Android and beyond," presented at the 22th USENIX Secur. Symp., Washington, DC, USA, Aug. 14–16, 2013.

[440] I. Leontiadis, C. Efstratiou, M. Picone, and C. Mascolo, "Don't kill my ads!: Balancing privacy in an ad-supported mobile application market," presented at the Workshop Mobile Comput. Syst. Appl., San Diego, CA, USA, Feb. 28–29, 2012.

[441] B. Liu, B. Liu, H. Jin, and R. Govindan, "Efficient privilege de-escalation for ad libraries in mobile apps," presented at the 13th Annu. Int. Conf. Mobile Syst. Appl. Services, Florence, Italy, May 19–22, 2015.

[442] S. Seneviratne, H. Kolamunna, and A. Seneviratne, "A measurement study of tracking in paid mobile applications," presented at the 8th ACM Conf. Secur. Privacy Wireless Mobile Netw., New York, NY, USA, Jun. 22–26, 2015.

[443] A. Short and F. Li, "Android smartphone third party advertising library data leak analysis," presented at the 11th IEEE Int. Conf. Mobile Ad Hoc Sensor Syst., Philadelphia, PA, USA, Oct. 28–30, 2014.

[444] P. Wijesekera, A. Baokar, A. Hosseini, S. Egelman, D. Wagner, and K. Beznosov, "Android permissions remystified: A field study on contextual integrity," presented at the 24th USENIX Secur. Symp., Washington, DC, USA, Aug. 12–14, 2015.

[445] Y. Wang, S. Hariharan, C. Zhao, J. Liu, and W. Du, "Compac: Enforce component-level access control in Android," presented at the 4th ACM Conf. Data Appl. Secur. Privacy, San Antonio, TX, USA, Mar, 3–5, 2014.

[446] D. Gallingani, R. Gjomemo, V. N. Venkatakrishnan, and S. Zanero, "Static detection and automatic exploitation of intent message vulnerabilities in Android applications," presented at the BlackHat, Las Vegas, USA, 2014.

[447] W. Zhou, Y. Zhou, M. C. Grace, X. Jiang, and S. Zou, "Fast, scalable detection of "piggybacked" mobile applications," presented at the 3rd ACM Conf. Data Appl. Secur. Privacy, San Antonio, TX, USA, Feb. 18–20, 2013.

[448] S. Salva and S. R. Zafimiharisoa, "Data vulnerability detection by security testing for Android applications," presented at the Inf. Secur. South Africa, Johannesburg, South Africa, Aug. 14–16, 2013.

[449] H. Ye, S. Cheng, L. Zhang, and F. Jiang, "DroidFuzzer: Fuzzing the Android apps with intent-filter tag," presented at the 11th Int. Conf. Advances Mobile Comput. Multimedia, Vienna, Austria, Dec. 2–4, 2013.

[450] S. Shekhar, M. Dietz, and D. S. Wallach, "AdSplit: Separating smartphone advertising from applications," presented at the 21th USENIX Secur. Symp., Bellevue, WA, USA, Aug. 8–10, 2012.

[451] G. Barbon, A. Cortesi, P. Ferrara, M. Pistoia, and O. Tripp, "Privacy analysis of Android apps: Implicit flows and quantitative analysis," presented at the 14th IFIP TC 8 Int. Conf. Comput. Inf. Syst. Ind. Manage., Warsaw, Poland, Sep. 24–26, 2015.

[452] P. Barros, et al., "Static analysis of implicit control flow: Resolving Java reflection and Android intents," presented at the 30th IEEE/ACM Int. Conf. Automated Softw. Eng., Lincoln, NE, USA, Nov. 9–13, 2015.

[453] O. Tripp and J. Rubin, "A Bayesian approach to privacy enforcement in smartphones," presented at the 23rd USENIX Secur. Symp., San Diego, CA, USA, Aug. 20–22, 2014.

[454] P. Berthomé, T. Fécherolle, N. Guilloteau, and J.-F. Lalande, "Repackaging Android applications for auditing access to private data," presented at the 7th Int. Conf. Availability Rel. Secur., Prague, Czech Republic, Aug. 20–24, 2012.

[455] R. Bhoraskar, et al., "Brahmastra: Driving apps to test the security of third-party components," presented at the 23rd USENIX Secur. Symp., San Diego, CA, USA, Aug. 20–22, 2014.

[456] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "ChEX: Statically vetting Android apps for component hijacking vulnerabilities," in *ACM Conf. Comput. Commun. Secur.*, CCS'12, Raleigh, NC, USA, Oct. 16–18, 2012.

[457] G. Bai, L. Gu, T. Feng, Y. Guo, and X. Chen, "Context-aware usage control for Android," presented at the 6th Iternational ICST Conf. Secur. Privacy Commun. Netw., Singapore, Sep. 7–9, 2010.

[458] S. T. A. Rumee and D. Liu, "DroidTest: Testing Android applications for leakage of private information," presented at the 16th Int. Conf. Inf. Secur., Dallas, Texas, USA, Nov. 13–15, 2013.

[459] D. Feth and A. Pretschner, "Flexible data-driven security for Android," presented at the 6th Int. Conf. Softw. Secur. Rel., Gaithersburg, Maryland, USA, Jun. 20–22, 2012.

[460] L. Jia, et al., "Run-time enforcement of information-flow properties on Android," presented at the 18th Eur. Symp. Res. Comput. Secur., Egham, U.K., Sep. 9–13, 2013.

[461] D. Schreckling, J. Posegga, J. Köstler, and M. Schaff, "Kynoid: Real-time enforcement of fine-grained, user-defined, and data-centric security policies for Android," presented at the 6th IFIP WG 11.2 Int. Workshop Inf. Secur. Theory Practice Secur. PrivacyTrust Comput. Syst. Ambient Intell. Ecosystems, Egham, U.K., Jun. 20–22, 2012.

[462] Z. Wei and D. Lie, "LazyTainter: Memory-efficient taint tracking in managed runtimes," presented at the 4th ACM Workshop Secur. Privacy Smartphones Mobile Devices, Scottsdale, AZ, USA, Nov. 3–7, 2014.

[463] A. R. Beresford, A. C. Rice, N. Skehin, and R. Sohan, "MockDroid: Trading privacy for application functionality on smartphones," presented at the 12th Workshop Mobile Comput. Syst. Appl., Phoenix, AZ, USA, Mar. 1–3, 2011.

[464] K. Ma, M. Liu, S. Guo, and T. Ban, "MonkeyDroid: Detecting unreasonable privacy leakages of Android applications," presented at the 22nd Int. Conf. Neural Inf. Process., Istanbul, Turkey, Nov. 9–12, 2015.

[465] P. Ferrara, O. Tripp, and M. Pistoia, "MorphDroid: Fine-grained privacy verification," presented at the 31st Annu. Comput. Secur. Appl. Conf., Los Angeles, CA, USA, Dec. 7–11, 2015.

[466] Y. Zhauniarovich, G. Russello, M. Conti, B. Crispo, and E. Fernandes, "Moses: Supporting and enforcing security profiles on smartphones," *IEEE Trans. Dependable Secure Comput.*, vol. 11, no. 3, pp. 211–223, May/Jun. 2014.

[467] J. Paupore, E. Fernandes, A. Prakash, S. Roy, and X. Ou, "Practical always-on taint tracking on mobile devices," presented at the 15th Workshop Hot Topics Operating Syst., Kartause Ittingen, Switzerland, May 18–20, 2015.

[468] M. Ongtang, K. R. B. Butler, and P. D. McDaniel, "Porscha: Policy oriented secure content handling in Android," presented at the 26th Annu. Comput. Secur. Appl. Conf., Austin, Texas, USA, Dec. 6–10, 2010.

[469] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh, "Taming information-stealing smartphone applications (on Android)," presented at the 4th Conf. Trust Trustworthy Comput., Pittsburgh, PA, USA, Jun. 22–24, 2011.

[470] X. Xiao, N. Tillmann, M. Fähndrich, J. D. Halleux, and M. Moskal, "User-aware privacy control via extended static-information-flow analysis," presented at the IEEE/ACM Int. Conf. Automated Softw. Eng., Essen, Germany, Sep. 3–7, 2012.

[471] Z. Zhao and F. Osono, "TrustDroid: Preventing the use of smartphones for information leaking in corporate networks through the used of static analysis taint tracking," presented at the 7th Int. Conf. Malicious Unwanted Softw., Fajardo, PR, USA, Oct. 16–18, 2012.

[472] V. Rastogi, Z. Qu, J. McClurg, Y. Cao, and Y. Chen, "Uranine: Real-time privacy leakage monitoring without system modification for Android," presented at the 11th Int. Conf. Secur. Privacy Commun. Netw., Dallas, TX, USA, Oct. 26–29, 2015.

[473] J. P. Achara, M. Cunche, V. Roca, and A. Francillon, "WifiLeaks: Underestimated privacy implications of the access_wifi_state Android permission," presented at the 7th ACM Conf. Secur. Privacy Wireless Mobile Netw., Oxford, U. K., Jul. 23–25, 2014.

[474] H. Huang, S. Zhu, K. Chen, and P. Liu, "From system services freezing to system server shutdown in Android: All you need is a loop in an app," presented at the 22nd ACM SIGSAC Conf. Comput. Commun. Secur., Denver, CO, USA, Oct. 12–6, 2015.

[475] K. Yang, J. Zhuge, Y. Wang, L. Zhou, and H. Duan, "IntentFuzzer: Detecting capability leaks of Android applications," presented at the 9th ACM Symp. Inf. Comput. Commun. Secur., Kyoto, Japan, Jun. 3–6, 2014.

[476] R. Johnson, Z. Wang, C. Gagnon, and A. Stavrou, "Analysis of Android applications' permissions," presented at the 6th Int. Conf. Softw. Secur. Rel., Gaithersburg, MD, USA, Jun. 20–22, 2012.

[477] X. Zhang, A. Ahlawat, and W. Du, "Aframe: Isolating advertisements from mobile applications in Android," presented at the Annu. Comput. Secur. Appl. Conf., New Orleans, LA, USA, Dec. 9–13, 2013.

[478] P. Institute, "Big data analytics in cyber defense," Feb. 2013. [Online]. Available: http://www.ponemon.org/library/big-data-analytics-in-cyber-defense

[479] N. Hardy, "The confused deputy: (Or why capabilities might have been invented)," *ACM SIGOPS Operating Syst. Rev.*, vol. 22, no. 4, pp. 36–38, 1988.

[480] S. Rasthofer, S. Arzt, E. Lovat, and E. Bodden, "DroidForce: Enforcing complex, data-centric, system-wide policies in Android," presented at the 9th Int. Conf. Availability Rel. Secur., Fribourg, Switzerland, Sep. 8–12, 2014.

[481] C. Jung, D. Feth, and C. Seise, "Context-aware policy enforcement for Android," presented at the IEEE 7th Int. Conf. Softw. Secur. Rel., Gaithersburg, MD, USA, Jun. 18–20, 2013.

[482] D. Tian, X. Li, J. Hu, G. Xu, and Z. Feng, "API-level multi-policy access control enforcement for Android middleware," presented at the 11th Int. Conf. Secur. Privacy Commun. Netw., Dallas, TX, USA, Oct. 26–29, 2015.

[483] A. Desnos and G. Gueguen, "Android: From reversing to decompilation," presented at the Black Hat, Abu Dhabi, 2011.

[484] D. Barrera, H. G. Kayacik, P. C. V. Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to Android," presented at the 17th ACM Conf. Comput. Commun. Secur., Chicago, Illinois, USA, Oct. 4–8, 2010.

[485] F. D. Cerbo, A. Girardello, F. Michahelles, and S. Voronkova, "Detection of malicious applications on Android OS," presented at the 4th Int. Workshop Comput. Forensics, Tokyo, Japan, Nov. 11–12, 2010.

[486] S. Malek, N. Esfahani, T. Kacem, R. Mahmood, N. Mirzaei, and A. Stavrou, "A framework for automated security testing of Android applications on the cloud," presented at the 6th Int. Conf. Softw. Secur. Rel., Gaithersburg, MD, USA, Jun. 20–22, 2012.

[487] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "ProfileDroid: Multi-layer profiling of Android applications," presented at the 18th Annu. Int. Conf. Mobile Comput. Netw., Istanbul, Turkey, Aug. 22–26, 2012.

[488] Y. Nan, M. Yang, Z. Yang, S. Zhou, G. Gu, and X. Wang, "UIPicker: User-input privacy identification in mobile applications," presented at the 24th USENIX Secur. Symp., Washington, DC, USA, Aug. 12–14, 2015.

[489] B. Davis and H. Chen, "RetroSkeleton: Retrofitting Android apps," presented at the 11th Annu. Int. Conf. Mobile Syst. Appl. Services, Taipei, Taiwan, Jun. 25–28, 2013.

[490] L. Li, et al., "I know what leaked in your pocket: Uncovering privacy leaks on Android apps with static taint analysis," *arXiv:1404.7431 [cs]*, Apr. 2014. [Online]. Available: http://arxiv.org/abs/1404.7431

[491] H. Bagheri, A. Sadeghi, R. Jabbarvand, and S. Malek, "Practical, formal synthesis and automatic enforcement of security policies for Android," in *Proc. 46th IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2016, pp. 514–525.

[492] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proc. IEEE*, vol. 63, no. 9, pp. 1278–1308, Sep. 1975.

[493] F. Shen, et al., "Information flows as a permission mechanism," presented at the ACM/IEEE Int. Conf. Automated Softw. Eng., Vasteras, Sweden, Sep. 15–19, 2014.

[494] D. Titze and J. Schütte, "Apparecium: Revealing data flows in Android applications," presented at the 29th IEEE Int. Conf. Adv. Inf. Netw. Appl., Gwangju, South Korea, Mar. 24–27, 2015.

[495] R. Mahmood, N. Esfahani, T. Kacem, N. Mirzaei, S. Malek, and A. Stavrou, "A whitebox approach for automated security testing of Android applications on the cloud," presented at the 7th Int. Workshop Automation Softw. Test, Zurich, Switzerland, Jun. 2–3, 2012.

[496] W. You, K. Qian, M. Guo, P. Bhattacharya, Y. Qian, and L. Tao, "A hybrid approach for mobile security threat analysis," presented at the 8th ACM Conf. Secur. Privacy Wireless Mobile Netw., New York, NY, USA, Jun. 22–26, 2015.

[497] M. Backes, S. Bugiel, S. Gerling, and P. V. Styp-Rekowsky, "Android security framework: Extensible multi-layered access control on Android," presented at the 30th Annu. Comput. Secur. Appl. Conf., New Orleans, LA, USA, Dec. 8–12, 2014.

[498] S. Dai, T. Wei, and W. Zou, "DroidLogger: Reveal suspicious behavior of Android applications via instrumentation," in *Proc. 7th Int. Conf. Comput. Convergence Technol.*, 2012, pp. 550–555.

[499] L. Lei, Y. Wang, J. Jing, Z. Zhang, and X. Yu, "MeadDroid: Detecting monetary theft attacks in Android by DVM monitoring," presented at the 15th Int. Conf. Inf. Secur. Cryptology, Seoul, Korea, Nov. 28–30, 2012.

[500] I. Bente, et al., "Towards permission-based attestation for the Android platform," presented at the 4th Int. Conf. Trust Trustworthy Comput., Pittsburgh, PA, USA, Jun. 22–24, 2011.

[501] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel, "Semantically rich application-centric security in Android," presented at the 25th Annu. Comput. Secur. Appl. Conf., Honolulu, Hawaii, Dec. 7–11, 2009.

[502] A. Armando, G. Costa, and A. Merlo, "Formal modeling and reasoning about the Android security framework," presented at the 7th Int. Symp. Trustworthy Global Comput., Newcastle upon Tyne, U.K., Sep. 7–8, 2012.

[503] T. Reps, S. Horwitz, and M. Sagiv, "Precise interprocedural dataflow analysis via graph reachability," presented at the Conf. Rec. 22nd ACM SIGPLAN-SIGACT Symp. Principles Program. Languages, San Francisco, CA, USA, Jan. 23–25, 1995.

[504] S. Rasthofer, S. Arzt, and E. Bodden, "A machine-learning approach for classifying and categorizing Android sources and sinks," presented at the 21st Annu. Netw. Distrib. Syst. Secur. Symp., San Diego, CA, USA, Feb. 23–26, 2014.

[505] Y. Zhauniarovich, O. Gadyatskaya, and B. Crispo, "Enabling trusted stores for Android," presented at the ACM SIGSAC Conf. Comput. Commun. Secur., Berlin, Germany, Nov. 4–8, 2013.

[506] F. Maggi, A. Valdi, and S. Zanero, "AndroTotal: A flexible, scalable toolbox and service for testing mobile malware detectors," presented at the ACM Workshop Secur. Privacy Smartphones Mobile Devices, Berlin, Germany, Nov. 8, 2013.

[507] Smali. [Online]. Available: https://code.google.com/p/smali/, accessed 2016.
[508] C. Miller and C. Mulliner, “Fuzzing the phone in your phone,” presented at the Black Hat, Las Vega, USA, Jun. 25, 2009.
[509] A. Chaudhuri, “Language-based security on Android,” presented at the Workshop Program. Languages Anal. Secur., Dublin, Ireland, Jun. 15–21, 2009.
[510] Google play market. [Online]. Available: http://play.google.com/store/apps, accessed 2016.
[511] F-Droid: Free and open source Android app repository. [Online]. Available: https://f-droid.org/, accessed 2016.
[512] Virustotal. [Online]. Available: https://www.virustotal.com/, accessed 2016.
[513] Virusshare. [Online]. Available: http://virusshare.com/, accessed 2016.
[514] Droidbench. [Online]. Available: http://sseblog.ec-spride.de/tools/droidbench, accessed 2016.
[515] ICC-Bench. [Online]. Available: https://github.com/fgwei/ICC-Bench, accessed 2016.
[516] S. S. Response, “2015 internet security threat report,” 2015. [Online]. Available: http://www.symantec.com
[517] W. Ford and M. S. Baum, *Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption*. Englewood Cliffs, NJ, USA: Prentice Hall, 2000.

Alireza Sadeghi received the BSc degree in computer (software) engineering and the MSc degree in information technology from Sharif University of Technology, in 2008 and 2010, respectively. He is working toward the PhD degree in the Department of Informatics of ICS School, University of California, Irvine. His research interests focus on software engineering, specifically, application of program analysis in security assessment and testing of mobile applications. He is a member of the ACM and the ACM SIGSOFT.

Hamid Bagheri received the BSc degree in computer engineering from the University of Tehran, the MSc degree in software engineering from Sharif University of Technology, and the PhD degree in computer science from the University of Virginia. He is an assistant professor in the Computer Science and Engineering Department, University of Nebraska–Lincoln. Previously, he was a postdoctoral researcher in the School of Information and Computer Sciences, University of California, Irvine. He has also visited Massachusetts Institute of Technology as a postdoctoral research fellow. He is broadly interested in software engineering, and particularly in practical software analysis and synthesis using concepts from fields like formal methods, program analysis, model-driven development, and software architecture. He has been a finalist at the ACM student research competition. His publications in several conferences have been recognized as best papers. He has served on the program committee of several major conferences, and reviewed for multiple journals. He is a member of the ACM and the IEEE.

Joshua Garcia received the BS degree in computer engineering and computer science, the MS degree in computer science, and the PhD degree in computer science, all from the University of Southern California. He is an assistant project scientist in the Institute for Software Research, University of California, Irvine. He conducts research in software engineering with a focus on software analysis and testing, software security, and software architecture.

Sam Malek received the BS degree in information and computer Science from the University of California, Irvine (UCI), and the MS and PhD degrees in computer science from the University of Southern California. He is an associate professor in the School of Information and Computer Sciences, University of California, Irvine. He is also the director of Software Engineering and Analysis Laboratory and a faculty member of the Institute for Software Research, UCI. Previously he was an associate professor in the Computer Science Department, George Mason University. His general research interests include the field of software engineering, and to date his focus has spanned the areas of software architecture, autonomic computing, software security, and software analysis and testing. He has received numerous awards for his research contributions, including the US National Science Foundation CAREER award, GMU Emerging Researcher/Scholar/Creator award, and the GMU Computer Science Department Outstanding Faculty Research award. He is a member of the ACM, the ACM SIGSOFT, and the IEEE.