Available online at www.sciencedirect.com

ScienceDirect

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

Computers & Security

Android security assessment: A review, taxonomy and research gap study

Shivi Garg[a,b,*], *Niyati Baliyan*[a]

[a] *Information Technology Department, Indira Gandhi Delhi Technical University for Women, Delhi, India*
[b] *Faculty of Informatics and Computing, J.C. Bose University of Science and Technology YMCA, Faridabad, India*

ARTICLE INFO

Article history:
Received 30 June 2020
Revised 8 September 2020
Accepted 13 October 2020
Available online 28 October 2020

Keywords:
Android
Dynamic analysis
Hybrid analysis
Machine learning
Security
Static analysis
Systematic literature review

ABSTRACT

Security threats are escalating exponentially posing a serious challenge to mobile platforms, specifically Android. In recent years the number of attacks has not only increased but each attack has become more damaging to the platform. Therefore, it is important to develop more stringent counter-measures to defend the mobile systems. Although in the last few years significant research progress is seen in the field of the detection and mitigation of Android security, yet numerous challenges and gaps still exist. This paper presents a comprehensive and sound taxonomy to review the state-of-the-art approaches used in Android security. We have highlighted the trends and patterns of different analysis approaches, identified the key aspects in terms of objectives, analysis techniques, code representations, tools and frameworks used, etc. and enumerated the research areas for future work. To carry out this study, the proper systematic literature review process is followed and the results of nearly 200 research publications have been comprehended based on different security aspects.

1. Introduction

Google's Android is the most prevalent mobile platform among different smartphone platforms. Android's market share is ~73% as of 2020 (Statcounter GlobalStats 2020). According to Google, there are ~2.5 Bn active Android devices in 2019, making Android the most popular mobile platform amongst users (Liam, 2020). Growing mobile app markets have increased security threats and are specifically targeting mobile platforms. A large number of Android app markets contain vulnerable and malicious apps, thereby compromising millions of mobile devices. The malicious apps can cause severe repercussions such as privacy leaks, app crashes, financial losses (caused by malware triggered premium rate SMSs) and arbitrary code installation, etc. Hence, Android security is a major concern amongst researchers as seen in the last few years.

1.1. Motivation

The researchers have investigated Android's security features that overlap with different domains like software engineering, programming language analysis, and mobile computing. The existing knowledge base in this field is present as published literature. However, the current literature is not comprehensive and does not provide a holistic view of state-of-the-art approaches. This paper provides an exhaustive review of all the existing approaches in a structured format. The objectives of this paper are:

1. To identify different purposes of Android security assessment techniques

* Corresponding author at: Information Technology Department, Indira Gandhi Delhi Technical University for Women, Delhi, India.
E-mail addresses: shivi002phd16@igdtuw.ac.in (S. Garg), niyatibaliyan@igdtuw.ac.in (N. Baliyan).
https://doi.org/10.1016/j.cose.2020.102087

2. To propose a taxonomy of Android security analysis approaches
3. To present a Systematic Literature Review (SLR) of the state-of-the-art approaches in Android security domain using the proposed taxonomy
4. To highlight research gaps and challenges among various Android security analysis mechanisms and provide future directions to the research communities

To build a comprehensive taxonomy, rigorous, and intensive SLR process is required. This helps in analyzing the content iteratively, which is collected from the set of papers using reputed literature search engines. To the best of our knowledge, this study is the most comprehensive and elaborative in the area of research.

1.2. *Our contributions*

The unique contributions of this paper are:

1. Better publication coverage from 2013 – 2020
2. Proposing more deep and comprehensive taxonomy of Android security analysis approaches
3. Covering all the program analysis approaches from static to Machine Learning (ML) including semi-supervised, reinforced and Deep Learning (DL)
4. Focusing on all the major security issues such as vulnerabilities, privacy leaks, app cloning, permission misuse, cryptographic issues, malware detection, test case generation, code verification and energy consumption

1.3. *Paper organization*

The structure of the paper is as follows. Section 2 talks about the evolution of computing paradigms and mobile phones over a period of time. Section 3 presents related surveys carried out in the Android security domain. Section 4 mentions the research methodology and fundamental protocol for this SLR. Section 5 presents the taxonomy constructed from the existing literature. Section 6 presents the research gaps and provide future directions for the Android security research community. Potential threats to validity and multiple ways to mitigate those are discussed in Section 7. Finally, the paper is concluded in Section 8.

2. Evolution of computing paradigms and mobile phones

With technological advancements, Internet speed and bandwidth have improved considerably over the years. The computing paradigms have evolved and gone through multiple stages of advancements. Fig. 1 reflects this evolution of computing paradigms and their effect on mobile phones (and Android).

The 1980s marked the era of cluster computing, during which the same type of standalone computer systems were interconnected using a high-speed local area network (LAN) (van Steen and Tanenbaum, 2016). Cluster computing deployed a centralised network topology where a centralised server controlled the scheduling of tasks. Although cluster computing reduced execution time and increased the system performance, yet the standalone computer systems were not able to handle large computations. Hence, this led to the development of grid computing (GC) in the 1990s. GC networks were distributed and had decentralised network topology. It offered three-pronged benefits i.e. resource shareability, system scalability, and computing reliability, but it suffered from data confidentiality and integrity issues (Agarwal and Srivastava, 2017). In 1999 introduction to Salesforce.com provided a breakthrough in cloud computing (CC) that allowed users to avail remotely accessible services as a platform (PaaS), infrastructure (IaaS), and software (SaaS) (Bhatia and Verma, 2017). CC helped in offloading resource-heavy tasks to cloud servers however, there were security and privacy issues. Therefore, the ubiquitous computing (UC) paradigm became popular in the 2000s. UC introduced seamless access to remote information resources with high fault tolerance, availability, and security. However, the major drawbacks of UC were energy consumption and human-machine interference (Barkallah et al., 2017). The year 2006 marked the inception of the concept of the Internet of Things (IoT) in which the computing devices are interconnected via the Internet. IoT provides support for complex event processing (CEP) (García-Valls et al., 2018) and is a unique feature in comparison to other computing paradigms.

The evolution of computing paradigms also revolutionised mobile phones. In the era of cluster computing, there was hardly any concept of mobile phones. Motorola introduced the first portable mobile phone Dyna TAC 8000X in 1983. The launch of GSM in 1991 when GC was prevailing paved the way for consumer handsets with digital displays. The focus of mobile phones was still on calling and sending text messages. With the introduction of Salesforce.com and CC in 1999, the growth of feature phones revolutionised the world. There were technological advancements in the field of mobile hardware and software. Basic mobile phones were transformed to smartphones to access data and services remotely. Google released the Android mobile operating system (OS) in September 2008. The advent of UC, which primarily focused on computing everywhere and anywhere led to significant improvements in smartphones in terms of memory, display, etc. A large number of Android versions were released during this period to offer a lot more features apart from communication such as voice recognition, video calling, etc. The integration of IoT with smartphones led to the development of minicomputers. Mobile phones can now control home appliances (smart home automation), measure health parameters (digital health), and manage home security (CCTVs), etc. (Evolution of the Mobile Phone, 2020).

Since the paper aims at Android, it is important to focus on Android versions and how they have evolved in terms of security over the years (Android Timeline and Versions, 2020). Table 1 summarises different Android versions with their security features.

Despite security improvements over the years, Android still suffers from multiple vulnerabilities, potential threats, and malware attacks. Users can detect malicious applications by analyzing the permissions model of Android. Malware in An-

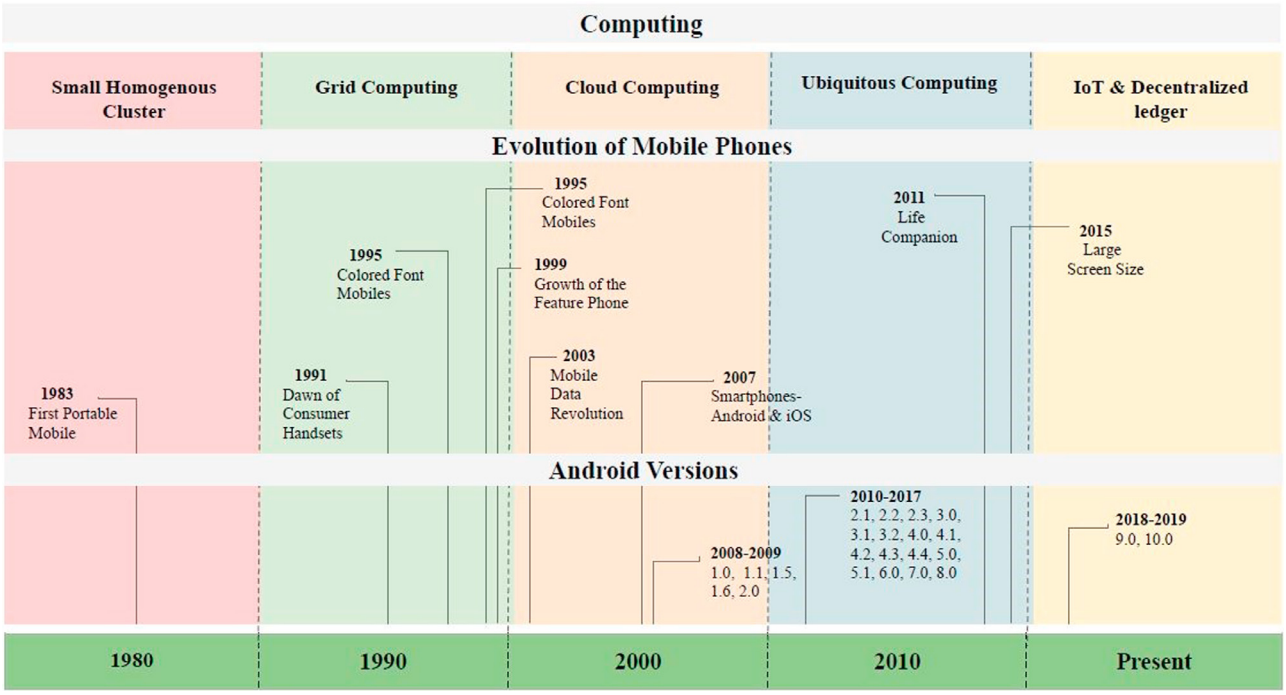

Fig. 1 – Evolution of computing paradigms and mobile phones.

droid can exploit vulnerabilities to obtain root-level access and can do nasty things.

Different layers in Android OS software stack are affected by multiple vulnerabilities (Mazuera-Rozo et al., 2019). Fig. 2 shows the distribution of different vulnerabilities across the Android OS layers. The enhancements in the software stack of Android have reduced the number of vulnerabilities, however the basic framework is same in all the Android versions.

The threat of malware on Android is real and will continue to grow, therefore it is important to study different analysis techniques, which can provide some real benefits and protection to the users.

3. Related surveys

It is important to identify, examine, and understand the contributions in the field of mobile security since the emergence of mobile phones. To the best of our knowledge, this study is first of its kind dealing with Android security analysis. There is no other survey that particularly focuses on all the aspects of Android security. Multiple works and surveys related to Android security have been proposed in the literature. There are very few SLRs existing in the Android security domain however, they lack completeness and comprehensiveness.

Suarez-Tangil et al. (2013) reviewed the malware evolution and analyzed 20 research efforts that detected mobile malware. Haris et al. (2014) presented a survey on mobile computing that addressed issues related to privacy. This study comprised of 16 user studies and 13 privacy leak detection tools in mobile privacy. Shrivastava et al. (2019) surveyed on the privacy issues caused by permissions in Android application.

The survey of Rashidi and Fung (2015) focused on the current security threats in Android and security implementation solutions. They classified security mechanisms in Android into four different dimensions: Denial of Service (DoS) attacks, Information Leaks, App Cloning and Vulnerabilities.

Another survey of Tan et al. (2015) discussed static and dynamic analysis approaches. They proposed taxonomy with five categories of existing security solutions on Android. They concluded that static analysis is the most leveraged technique for addressing Android security issues.

In another survey, Martin et al. (2016) analyzed the works based on the app store in the field of software engineering. The authors reported the technical and non-technical learning behaviors of software repositories. In particular, they analyzed existing works in 7 dimensions such as feature analysis, API analysis, review analysis, store ecosystem, security, size and effort prediction, and others.

In the survey of Faruki et al. (2014), they focused on the malware growth, anti-analysis techniques, and malware detection techniques. They also discussed that stealthy techniques such as encryption and code transformation can generate variants of malware. They analyzed static and dynamic approaches for malware detection.

In a technical report presented by Sadeghi et al. (2016), a qualitative comparison of program analysis techniques was done based on Android security. They proposed the taxonomy after reviewing 336 research papers, including both static analysis and dynamic analysis approaches. However, the very less focus was on the hybrid and ML techniques supplementing the conventional analysis approaches.

Table 2 summarises the contributions of different literature reviews. The focus of these surveys is mainly on mobile malware. They have used limited parameters and criteria for com-

Table 1 – Android versions with security features.

Versions	Release year	Security features
Android 1.5 (Cupcake)	2009	• Very little software vetting • Pro-Police to prevent buffer overflows • Safe integer operations to reduce integer overflows
Android 2.0 (Éclair)	2009	• Handled null pointer dereference privilege escalation • Prevention of code execution on the heap and stack • Format string vulnerability protections
Android 3.0 (Honeycomb)	2011	• Encryption of all user data • Write access to memory cards not allowed to applications • HTTPS stack improvement with Server Name Indication (SNI)
Android 4.0 (Ice Cream Sandwich)	2011	• SELinux support • ASLR to randomize key locations in memory • Read-only relocations / immediate binding • Handled kernel addresses leakage • Verified boot
Android 5.0 (Lollipop)	2014	• Improved full disk encryption • Smart Lock • Android sandbox reinforced with SELinux • Updated cryptography for HTTPS and TLS/SSL • Kill-switch option for full factory reset
Android 6.0 (Marshmallow)	2015	• Automatic update security patches on a device • Runtime Permissions • **Hardware-Isolated Security** • **Fingerprints**
Android 7.0 (Nougat)	2016	• File-based encryption • Verified and Direct boot • Kernel hardening • Updated SELinux • Improved ASLR
Android 8.0 (Oreo)	2017	• Google Play Protect to automatically scan Google play store • Verified boot with Rollback protection • Tight Sandboxing with Webview • User-space and Kernel hardening
Android 9.0 (Pie)	2018	• Support for biometrics • APK signature scheme v3 • Support for kernel control flow integrity (CFI) • Default HTTPS for apps • Passcode for device restoration
Android 10.0 (Quince Tart)	2019	• Support for the WPA3 Wi-Fi security protocol • Device and File-based encryption mandatory for all devices

paring and categorizing malware detection techniques. Several important granular aspects of security are ignored such as for purposes, features, and evaluation of analysis techniques. Indeed, this SLR has better publication coverage in the domain of Android security than the aforementioned surveys.

4. Research methodology

The research methodology used to prepare this SLR is based on the general guidelines proposed by Kitchenham and Brereton (2013). 200 research papers published in reputed journals and conferences with a significant number of citations are studied and analyzed. We have compared different concepts presented in the proposed taxonomy to identify the current trends and research gaps in the presented literature and provided future directions to the researchers that will shape the domain of Android security. Fig. 3 explains the protocol that is used to carry out this SLR.

The key steps in the formation of SLR are detailed below:

- Formulated research questions describing the purpose of the SLR (cf. Section 4.1).
- Enumerated distinct search keywords to find prominent journals and conferences (cf. Section 4.2.1). The searching process considered two scenarios, first one focused on finding reputed publication repositories while the second one focused on publications from top venues including conferences, workshops, and symposia (cf. Section 4.2.2).
- Applied inclusion/ exclusion criteria to filter retrieved papers that fit in the scope (cf. Section 4.3).

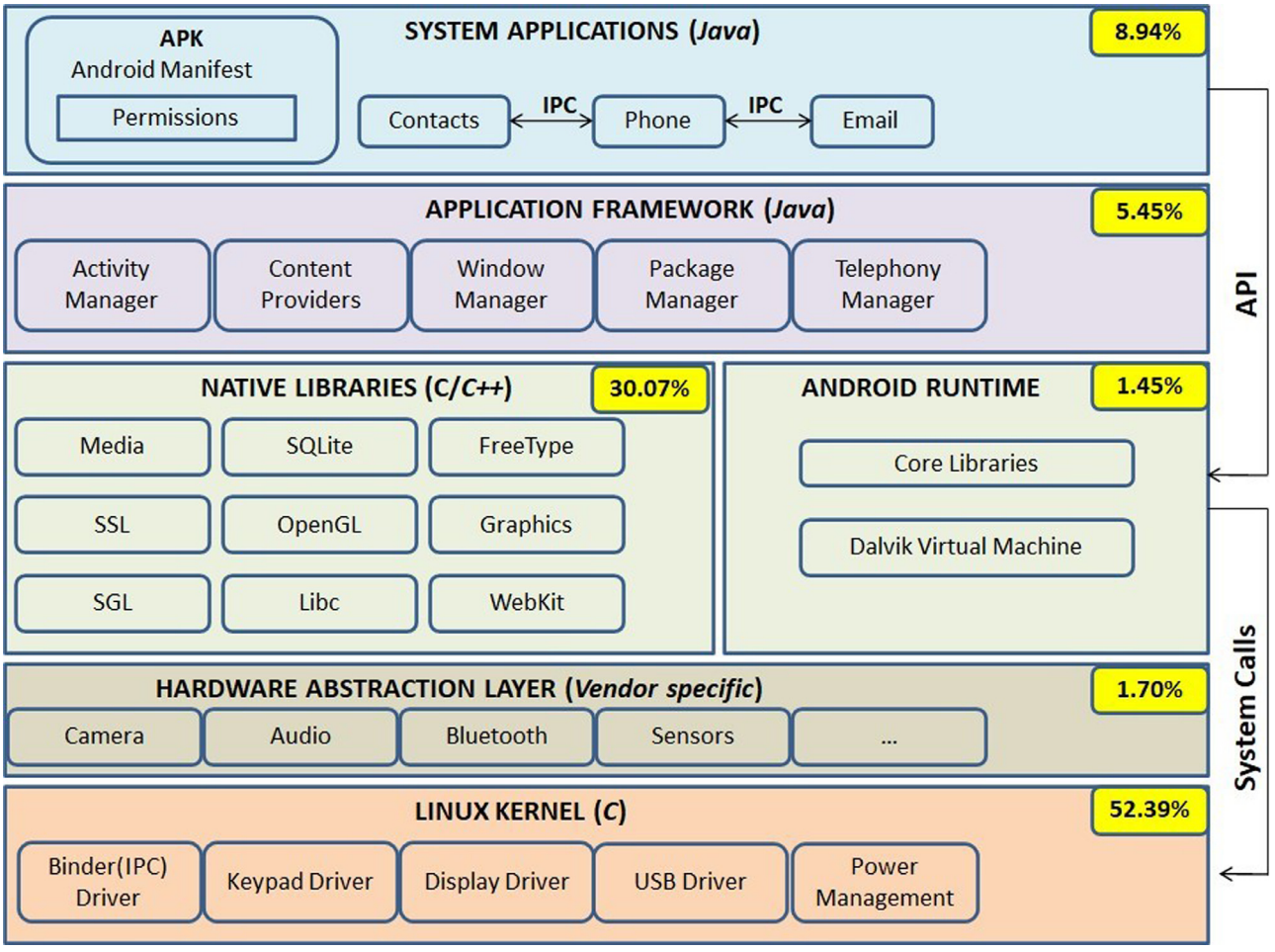

Fig. 2 – Distribution of vulnerabilities in Android framework layers.

- Performed citation chaining to refine the search process with relevant literature (cf. Section 4.4).
- Results from different research findings are combined to carry out future research (cf. Section 4.5).

4.1. *Research questions (RQ) identification*

This SLR aims to answer the following research questions (RQs):

1. RQ1: What are the different approaches to analyze Android security? To answer this survey-of-surveys is built with complete taxonomy for classifying Android security analysis approaches.
2. RQ2: How are these analysis approaches designed and implemented? For this question, research results cataloged during SLR are reviewed to understand concepts and techniques of implementation.
3. RQ3: What is the current state-of-the-art concerning Android security? To answer this, research articles are categorised across multiple dimensions that help perform quantitative/ qualitative analysis.
4. RQ4: What research gaps and challenges need to be inferred and addressed? Finally, with this question, we studied remaining issues that are still open for current research efforts.

4.2. *Search strategy*

The strategy of keyword-based search and repository and dataset search for obtaining the relevant research publications is now discussed in detail.

4.2.1. *Search keywords*

The focus is on specific keywords to confine the scope of the literature review. The keywords are searched on the papers' title, abstracts, and meta-data like tags. Table 3 describes the research domains and the related keywords that appear in these domains.

There are three research domains, denoted by R, which fit in the scope of the SLR. These are as follows:

1. *Android Mobile OS* – this domain focuses on a mobile platform and target programs
2. *Technical Approach* – this domain includes the analysis techniques- static, dynamic, hybrid or ML
3. *Security Aspects* – this domain covers the analysis methods to be applied to the potential security issues.

Table 2 – Summary of literature reviews.

Research paper	Years	Research objective	Mobile platform	Detection approach
Suarez-Tangil et al. (Suarez-Tangil et al., 2013)	2010–2013	Malware & greyware detection	Android, Windows, iOS, Blackberry	Static & dynamic
Harris et al. (Haris et al., 2014)	2010–2014	Privacy leaks	Android, iOS	Static, dynamic & hybrid
Srivastava et al. (Shrivastava et al., 2019)	2007–2019	Privacy issues of android application permissions	Android	Static, dynamic & hybrid
Rashidi et al. (Rashidi and Fung, 2015)	2010–2015	Information leakage, privilege escalation, Repackaging apps, DoS attack, Colluding	Android	Static & dynamic
Tam et al. (Tan et al., 2015)	2009–2014	Information leakage, privilege escalation, Runtime malware detection, Isolation systems	Android	Static & dynamic
Martin et al. (Martin et al., 2016)	2000–2015	App store analysis	Android, iOS, Nokia Widgets, Blackberry and Windows Phone	Static & dynamic
Faruki et al. (Faruki et al., 2014)	2010–2014	Malware penetration	Android	Static Analysis, Dynamic Analysis, Behavioral Analysis
Sadeghi et al. (Sadeghi et al., 2016)	2008–2016	Malware, greyware and vulnerability detection	Android	Static, dynamic, hybrid, ML (supervised & unsupervised only)

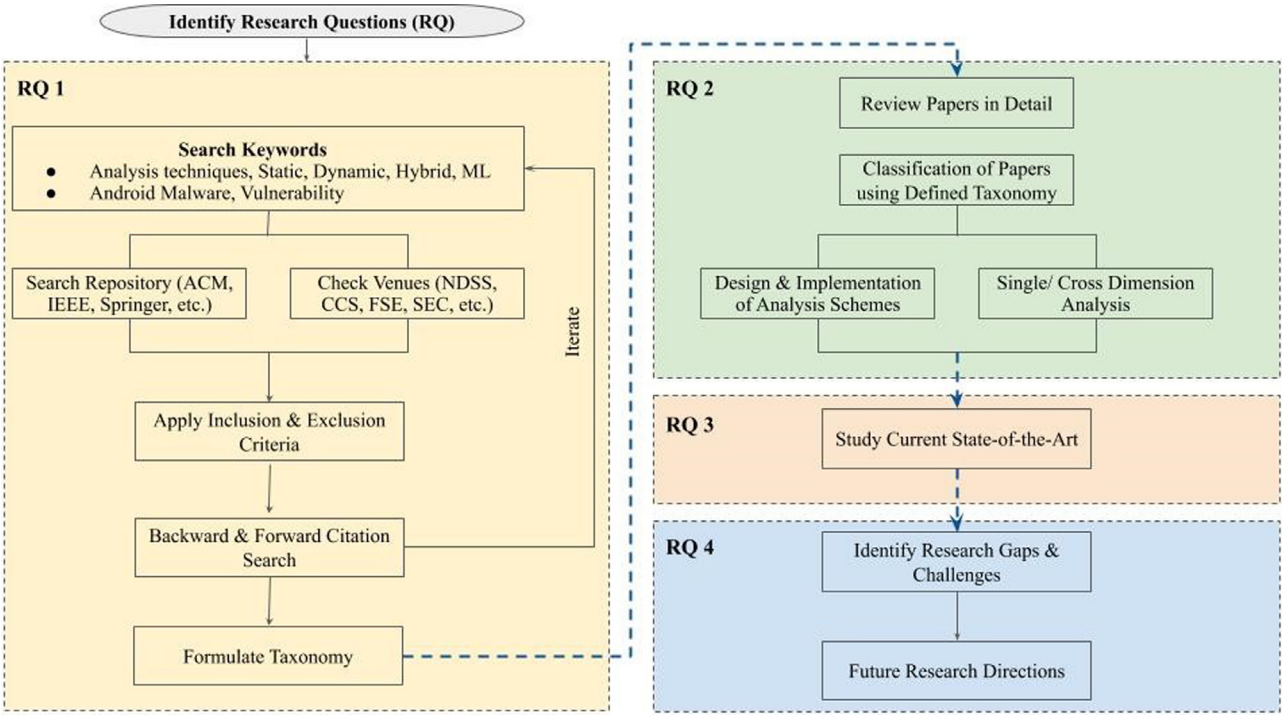

Fig. 3 – Schematic representation of SLR.

Table 3 – Search keywords specific to each research domain.

Research Domain (R)	Keywords (K)
Android Mobile OS	Android, Smartphone, Mobile, Application, App
Technical Approach	Static, Dynamic, Hybrid, Machine Learning, ML (Analysis)*, Detection
Security Aspects	Vulnerability, Malware, Security, Privacy

Table 4 – Top 15 conference proceedings venues in SE/ PL and S&P Fields.

Acronym	Full name	H-Index
Software Engineering and Programming Languages (SE/PL)		
ICSE	International Conference on Software Engineering	57
AINA	Advanced Information Networking and Applications	42
FSE	Foundations of Software Engineering	38
ISSTA	International Symposium on Software Testing and Analysis	32
ISSRE	International Symposium on Software Reliability Engineering	21
ASE	Automated Software Engineering	14
QRS	Conference on Software Quality, Reliability, and Security	6
Security and Privacy (S&P)		
CCS	Computer and communications security	65
NDSS	Network and Distributed System Security Symposium	39
ASIACRYPT	International Conference on The Theory and Application of Cryptology and Information Security	34
ACSAC	Computer Security Applications Conference	30
ESORICS	European Symposium on Research in Computer Security	24
WiSEC	Security and Privacy in Wireless and Mobile Networks	22
RAID	Recent Advances in Intrusion Detection	20
SEC	International Information Security and Privacy Conference	19

The search query (q) is a conjunction of the three research domains, namely, R_1: Android Mobile OS, R_2: Technical Approach, R_3: Security Aspects, where $R_1, R_2, R_3 \in R$. Each domain in the search query string is represented as a disjunction of its corresponding keywords (K_r), as shown in Table 1. The search query is:

$$q = \wedge_{r \in \{R_1,R_2,R_3\}} \left(\wedge_{keywords \in K_r} keyword \right)$$

In simplified form,

$$q = R_1 \; AND \; R_2 AND \; R_3$$

$$R_1 = \{Android \; OR \; Smartphone \; OR \; Mobile \; OR \; App^* OR \; Application\}$$

4.2.2. *Search repositories and datasets*

We have considered reputed electronic repositories such as IEEE Xplore Digital Library, Science Direct, ACM Digital Library, and Springer Link to find relevant datasets of publications. Some repository search engines pose a limit to download to the search result meta-data. In that case, the search string is split and iterated over multiple times to retrieve the relevant results.

Other data sources such as conference proceedings, workshops and symposia that are not listed in the aforementioned repositories are also considered. To make our search exhaustive, we considered top 15 publication venues from security and privacy (S&P) field and software engineering and programming languages (SE/ PL) field ranked according to the H-Index, as summarised in the Table 4. The H-Index of a publication as defined by Google Scholar is the count of h (largest value) published papers by a given author/ journal that each been cited at least h times (Google Scholar Metrics, 2020). It is considered that larger the value of H-Index, the better is the venue. Fig. 4 illustrates the word cloud of the examined publications by conferences symposia and workshop names (Fig. 4a) and titles of the selected papers (Fig. 4b).

4.3. *Selection criteria*

A selection criterion viz. inclusion/ exclusion is used to filter out the research results, since all the retrieved results do not lie in the scope of this SLR.

4.3.1. *Inclusion criteria*

We have limited the scope of the SLR that lie at the intersection of three research domains (discussed in the Section 4.2.1). We have collected the papers from 2013–2020.[1]

4.3.2. *Exclusion criteria*

Exclusion criteria followed in this SLR are:

1. Research publications written in English are only considered since it is the most common language spoken by reviewers and researchers. Publications written in non-English are omitted.

[1] Research papers published after June 2020 are not included in this SLR.

(a). Conferences, symposia and workshops names (b). Title of the selected papers

Fig. 4 – Word cloud of examined publications.

2. Short papers with page length of less than 4 pages in IEEE/ACM-like double-column format or with less than 7 pages in LNCS single-column format are rejected. These papers are preliminary works, which are later published as full-length papers and so should be included in the final set.
3. Duplicate papers that are first published in conference proceedings and later on extend in the journals are excluded. Such papers are identified by comparing the author list, paper title, abstract and meta-data.
4. Search term contains keywords such as "Mobile", "Smartphone", to retrieve good volume of papers. However, the retrieved result set includes papers about "Windows/iOS" platforms, "mobile networking", etc. Therefore, non Android-related publications are excluded from this study.
5. Papers published before 2013 are excluded from the SLR to include the recent trends and researches.

4.4. *Backward and forward citation*

Citation chaining or reference mining is a method to expand the research. It identifies additional relevant articles by reviewing the cited works/ bibliographies/ references list of a specific article from the search list (Citation chaining in Google Scholar and PubMed, 2020). The article can be traced both in backward and forward direction.

1. Backward Chaining – It helps in finding those articles that are cited in the existing resources. It identifies past resources regarding the same topic.
2. Forward Chaining – It helps in finding those articles that cite the existing resources. Research databases such as Web of Science and Scopus are used for forward chaining. They show a list of references for a particular work if that item has been cited by other works in those particular databases.

4.5. *Research publication selection*

Results of final selection of publications are described in the Table 5. In Row 1, keyword-based search is performed on the research databases and initial set of papers are obtained. The search engine of each database treats the search query differently; therefore, initial filtration is performed over the initial set of papers as specified in Row 2. Row 3 shows the merged results from all the databases. Finally, selection criterions' consisting of inclusion and exclusion criteria are applied to get the final set of selected papers. This process yields total of 200 research publications required to carry out this SLR.

Fig. 5 shows the distribution of publications by types. Over 41% of the retrieved papers are published in conferences. Other sources include workshops, symposia, book chapters, Ph.D. dissertations, etc.

Distribution of published papers in journals such as IEEE, ACM, Elsevier, and Springer is shown in Fig. 6. IEEE and ACM digital library form the largest share of published papers followed by Springer and Elsevier. Others include publications from IGI Global, Wiley and non-official proceedings such as arXiv.

5. Taxonomy construction

After retrieving the relevant papers, taxonomy based on Android security analysis is constructed using properties and dimensions from the existing literature. This taxonomy of information will be helpful in order to (1) answer the above enumerated RQs, (2) provide systematic assessment of each paper, and (3) present a basis for categorizing and comparing the different approaches. The aforementioned surveys or SLRs described in Section 3, though relevant and useful, are not sufficient enough to categorise Android security analysis approaches. The related surveys either focus on mobile malware in general, or other sub-areas, such as inter-application vulnerabilities or malware families in Android, but not on the security analysis as a whole.

Thus, we have defined the novel taxonomy to help classify existing work in this domain. Fig. 7 describes the defined tax-

Table 5 – Number of papers retrieved during paper selection.

	Databases				
Selection Phase	IEEE	ACM	Elsevier	Springer	Top 15 Venues
Keyword-based Search	1201	395	2512	7143	135
Initial Filtering	742	305	267	431	135
Merging					1880
Applying Criteria					200

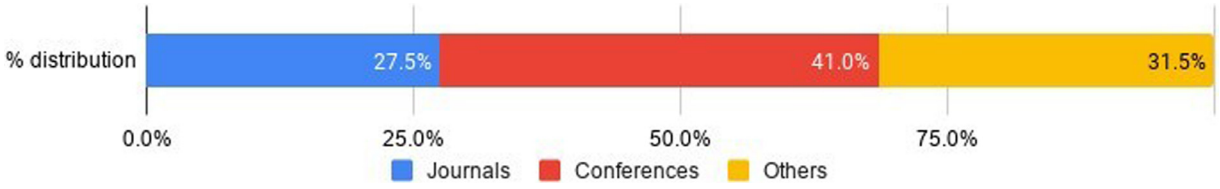

Fig. 5 – Distribution of published papers in different publication domains.

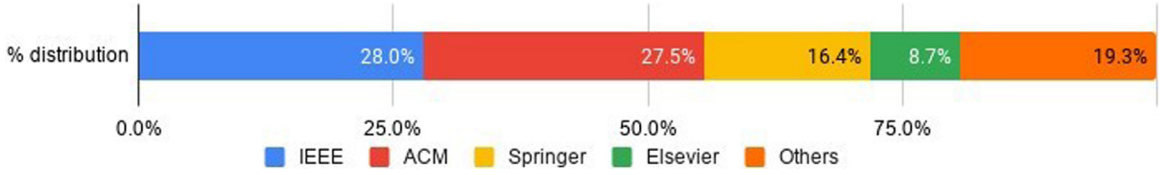

Fig. 6 – Distribution of published papers in research journals.

onomy for Android security analysis. The taxonomy defines three dimensions based on the following three questions:

1. What are the purposes of Android security analysis?
2. What are the different techniques used in analysis?
3. What are the attributes associated with these analysis techniques?

5.1. *Android security analysis objectives*

This dimension identifies different objectives for which analyses are performed. Several security issues are addressed while performing such analyses. Some of these are permission management concerns, code verification problems, automation of test case generation, private data leaks, clone detection, assessing code efficiency in terms of energy consumption, etc. This SLR identifies 9 objectives of Android security analysis approaches. The statistics of the different approaches targeting these objectives are also presented as follows.

1. *Permission exploitation:* Permission-based security model forms the basis of Android architecture. Permissions must be granted to access the system resources. However, some inherent risks are associated with the permissions since apps can leverage extra permissions what they actually need (Bartel et al., 2014). Malicious apps can control permissions and can launch different types of attacks such as data loss attacks, data integrity attacks, DoS and Distributed DoS (DDoS) attacks.
2. *Passive content leaks:* It is also referred to as leaking private data. Privacy of data is the major concern among Android researchers. Potential privacy data leaks include phone information, WiFi data, GPS location, audio recorded with the microphone, etc.
3. *Code verification:* The purpose of the code verification is to ensure the correctness of a given app. There are very few works addressing this purpose. Cassandra (Lortz et al., 2014) checks whether apps in Android are in compliance with their privacy requirements before app installation.
4. *Misusing cryptography techniques:* Implementation issues in cryptography are another cause of concern among the researchers. Cryptography misuse includes validation failure in the SSL/TLS resulting in man in the middle (MITM) attacks that violate the system authentication. CMA (Shuai et al., 2014) analyses cryptography misuse using crypto misuse analyzer.
5. *Vulnerability detection:* Android suffers from large number of security vulnerabilities. Intent injection and content hijacking are most common vulnerabilities in Android. Intent injection occurs due to the execution of arbitrary code by manipulating the user data. Content hijacking occurs when private and protected resources are accessed in an unauthorised way through exported components in vulnerable apps. Epicc (Octeau et al., 2013) uses static analysis for detecting inter-component vulnerabilities.
6. *Energy consumption:* Modern smartphones with large screen sizes have high energy consuming components. Battery stand-by time is a major concern for mobile devices. According to Li et al. (Li et al., 2014) modern smart-

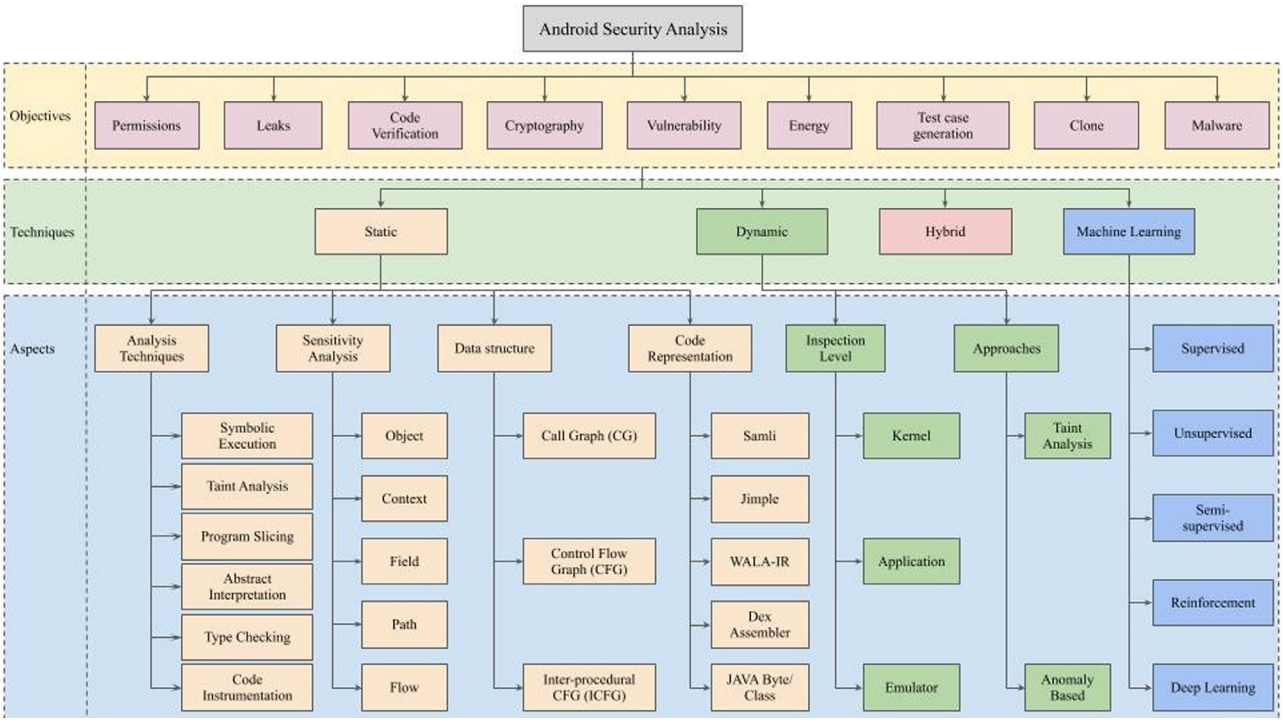

Fig. 7 – Proposed Taxonomy for Android Security Analysis.

phones consume more energy when the light colors are displayed compared to dark colors. They investigated that the energy consumption could be reduced by 40% by building dark background color web pages for the mobile systems to generate efficient web pages.

7. *Test case generation:* Test case generation provides a set of test cases that are executed for automatic and repeatable testing. Symbolic execution is performed on the source code to ensure the reachability of branches. As an example, SIG-Droid (Mirzaei et al., 2015) is a framework for system testing of Android apps. It automatically generates test cases through symbolic execution using interface model and behavior model. Interface model finds the values that a given app can receive, whereas behavior model generates the sequences of events to drive the symbolic execution.
8. *App Cloning:* Researchers have used several analysis approaches to detect app cloning in Android. According to Mojica et al. (2013) app cloning is very common in mobile apps. In another work, AnDarwin (Crussell et al., 2013) has proved to be efficient in detecting the cloned apps.
9. *Malware detection:* Vulnerabilities in Android are increasing leading to malware attacks. Malware can be of different types such as spyware, adware, Trojan, ransomware, etc., which can have a potential impact on the system. Therefore, modern studies are focusing more on malware detection.

Primary research publications serving these 9 objectives are enumerated in Table 6. Due to space constraints, these security objectives are represented in 25 clusters with the count of publications in each cluster as shown Fig. 8. Fig. 9 shows the statistics of the publications focusing on these security objectives. It is seen that majority of the studies focus on vulnerability detection (32%) followed by privacy leaks (28%) and malware detection (18%).

5.2. *Android analysis techniques*

The second dimension of the taxonomy, which tries to answer RQ1, is concerned with classifying the different techniques used in Android security analysis. There are four different program analysis techniques leveraged in the security domain of Android. These are Static, Dynamic, Hybrid and ML.

Static analysis examines the potential behavior of the program structure. These techniques parse the program source code or bytecode by traversing the program paths to check the properties of the program. Dynamic analysis observes the actual behavior of the program at runtime. In addition to pure static or dynamic techniques, there are hybrid techniques that leverage benefits from both static and dynamic techniques. In hybrid techniques, static analysis is first used to detect potential security concerns, and then dynamic analysis is used to eliminate the false warnings, thereby improving their precision. Apart from these techniques, other supplementary techniques such as ML are also used to complement the analysis. The program analysis either provides the input for, or consumes the output of, the supplementary techniques.

Each technique has its own merits and demerits according to their intrinsic properties. Static analysis techniques are sound and conservative however, dynamic analysis techniques are unsound but precise (Ernst, 2003). In case of dynamic analysis, certain events are required to run the appli-

Table 6 – Publications with analysis objectives.

Clusters	Publications	# Papers
Vulnerability (V)	Bao et al. (Bao et al., 2017), Bartsch et al. (Bartsch et al., 2013), BaseSAFE (Maier et al., 2020), Chen et al. (C.M. Chen et al., 2014), ClickRelease (Micinski et al., 2015), ConDroid (Schütte et al., 2015), COVERT (Bagheri et al., 2015), DEvA (Safi et al., 2015), DroidSIFT (Zhang et al., 2014), EASEAndroid (Wang et al., 2015), Epicc (Octeau et al., 2013), Fang et al. (Fang and Yan, 2018), Gallingani et al. (Gallingani, 2014), Georgiev et al. (Georgiev et al., 2014), Graa et al. (Graa et al., 2014), Hopper (Blackshear et al., 2015), Liu at al. (Liu et al., 2018), Luo et al. (Luo et al., 2019), Malik et al. (Malik et al., 2019), MIGDroid (Hu et al., 2014), Min et al. (Min et al., 2019), MobSafe (Xu et al., 2013), PaddyFrog (Wu et al., 2015), Pang et al. (Pang et al., 2017), Pegasus (Chen et al., 2013), Poeplau et al. (Poeplau et al., 2014), SMV-Hunter (Sounthiraraj et al., 2014), Spartan Jester (Sexton et al., 2017), Tang et al. (Tang et al., 2020), Wognsen et al. (Wognsen et al., 2014), Xiong et al. (Xiong et al., 2017), Yang et al. (Yang and Huang, 2018), Zhong et al. (Zhong and Xiao, 2014), Zou et al. (Zuo et al., 2015)	35
Leaks (L)	Apparecium (Titze and Schütte, 2015), AppAudit (Xia et al., 2015), AppCaulk (Schütte et al., 2014), AppIntent (Yang et al., 2013), Apposcopy (Feng et al., 2014), AppSealer (Zhang and Yin, 2014), AsDroid (Huang et al., 2014), Bastani et al. (Bastani et al., 2015), Bonett et al. (Bonett et al., 2018), Brox (Ma et al., 2013), Capper (M. Zhang and Yin, 2014), Cortesi et al. (Cortesi et al., 2015), DescribeMe (Zhang et al., 2015), Dflow + DroidInfer (Huang et al., 2015), DroidJust (Chen and Zhu, 2015), DynaLog (Alzaylaee et al., 2016), FlowDroid (Arzt et al., 2014), Graa et al. (Graa et al., 2017), HelDroid (Andronio et al., 2015), HornDroid (Calzavara et al., 2016), IccTA (Li et al., 2015), IFT (Ernst et al., 2014), Jiang et al. (Jiang et al., 2017), Octeau et al. (Octeau et al., 2016), Relda (Guo et al., 2013), Sufatrio et al. (Chua et al., 2015), TASMAN (Arzt et al., 2015), Tuan et al. (Tuan et al., 2019), Uranine (Al Nidawi et al., 2017), WeChecker (Cui et al., 2015)	30
Malware detection (M)	Han et al. (Han et al., 2020), Han et al. (Han and Olivier, 2020), AdDroid (Mehtab et al., 2020), AMD-EC (Ghaffari et al., 2017), AndroTaint (Shankar et al., 2017), Appice et al. (Appice et al., 2020), DroidDetector (Yuan et al., 2016), Fadadu et al. (Fadadu et al., 2019), Hsien-De et al. (Hsien-De Huang and Kao, 2018), Kabakus et al. (Kabakus, 2019), Mahindru et al. (Mahindru and Sangal, 2020), Mantoo et al. (Mantoo and Khurana, 2020), Martín et al. (Martín et al., 2019), Nguyen-Vu et al. (Nguyen-Vu et al., 2019), RARE (Darki et al., 2018), SecureDroid (Chen et al., 2017), Sharif et al. (Sharif and Nauman, 2019), Sourav et al. (Sourav et al., 2019), StormDroid (Chen et al., 2016), VizMal (De Lorenzo et al., 2020), Wang et al. (Wang et al., 2019), Wu (Wu, 2020), Yang et al. (Yang et al., 2015), Zarni Aung et al. (Zarni Aung, 2013)	24
Leaks, Vulnerability (LV)	ATFuzzer (Karim et al., 2019), Chen et al. (Chen et al., 2017), ContentScope (Jiang and Xuxian, 2013), COVERT (Bagheri et al., 2015), DroidAlarm (Zhongyang et al., 2013), DroidSafe (Gordon et al., 2015), DroidUnPACK (Duan et al., 2018), DroidVulMon (Ham et al., 2013), Schoepe et al. (Schoepe et al., 2016), Tiwari (Tiwari, 2019)	10
Vulnerability, Malware detection (VM)	ATMPA (Liu et al., 2019), Chuang et al. (Chuang and Wang, 2015), Coulter et al. (Coulter et al., 2020), Rasthofer et al. (Rasthofer et al., 2015), Suárez-Tangi et al. (Suárez-Tangil et al., 2018), Tian (Tian, 2018), Yang et al. (Yang et al., 2017)	7
Permissions (P)	Android-app-analysis-tool (Geneiatakis et al., 2015), AppContext (Yang et al., 2015), Bartel et al. (Bartel et al., 2014), DroidDet (Zhu et al., 2018), Geneiatakis et al. (Geneiatakis et al., 2015), Kabakus (Kabakus, 2019), Lin et al. (Lin et al., 2014)	7
Test case generation (T)	ADS-SA (Song et al., 2019), Jensen et al. (Jensen et al., 2013), Rountev et al. (Rountev and Yan, 2014), SIG-Droid (Mirzaei et al., 2015), Vuong et al. (Vuong and Takada, 2019)	5
Permissions, Leaks, Vulnerability (PLV)	Anadroid (Liang et al., 2013), FUSE (Ravitch et al., 2014), Mandal et al. (Mandal et al., 2019), PermissionFlow (Sbîrlea et al., 2013), SEFA (Wu et al., 2013)	5
App Cloning (A)	AnDarwin (Crussell et al., 2013), AndRadar (M. Lindorfer et al., 2014a), Chen et al. (Chen et al., 2014a), DroidSim (Sun et al., 2014)	4
Energy (E)	EcoDroid (Behrouz et al., 2015), eLens (Hao et al., 2013a), Uranine (Al Nidawi et al., 2017), vLens (Li et al., 2013)	4
Leaks, Malware detection (LM)	EspyDroid+ (Gajrani et al., 2020), IntelliDroid (Wong and Lie, 2016), Spreitzenbarth et al. (Spreitzenbarth et al., 2015)	3
Permissions, Vulnerability (PV)	Caputo et al. (Caputo et al., 2019), SAAF (Hoffmann et al., 2013), VettDroid (Zhang et al., 2013)	3
Permissions, Leaks (PL)	Appscalpel (Meng et al., 2019), BlueSeal (Shen et al., 2014), JN-SAF (Wei et al., 2018)	3
Cryptography (C)	CMA (Shuai et al., 2014, August), CryptoLint (Egele et al., 2013)	2

(*continued on next page*)

Table 6 (*continued*)

Clusters	Publications	# Papers
Leaks, Energy (LE)	Degu et al. (Degu, 2019), Jiang et al. (Jiang and Zhuang, 2017)	2
Leaks, Vulnerability, Malware detection (LVM)	DINA (Alhanahnah et al., 2020), DYDROID (Qu et al., 2017)	2
Permissions, Vulnerability, Malware detection (PVM)	DroidAnalyst (Faruki et al., 2016), Sharmeen et al. (Sharmeen et al., 2020)	2
Vulnerability, Energy (VE)	Nyx (Li et al., 2014)	1
Vulnerability, Cryptography (VC)	Wang et al. (Wang et al., 2020)	1
Code verification (Co)	Cassandra (Lortz et al., 2014)	1
Leaks, Test case generation (LT)	W2AIScanner (Hassanshahi et al., 2015)	1
Leaks, Energy, Test case generation (LET)	Yang et al. (Yang et al., 2018)	1
Permissions, Code verification (PCo)	Firdaus et al. (Firdaus et al., 2018)	1
Permissions, Leaks, Malware detection (PLM)	MARVIN (Lindorfer et al., 2015)	1
Permissions, Leaks, Vulnerability, Malware detection (PLVM)	Jitana (Tsutano et al., 2019)	1

cation. Since it is not possible to record all the events at a time and so the provided test cases are likely to be incomplete, therefore, apps behaviors are captured. These results in false negatives, i.e., vulnerabilities or malicious behaviors are missed in the security analysis. Moreover, dynamic analysis approaches are often deceived by advanced malware, such as anti-taint tracking techniques bypass the dynamic taint analyses (Rasthofer et al., 2015). Hybrid techniques have their own limitations that inhibit them from delivering impeccable results. It is seen that hybrid techniques have produced the worst results as those of static and dynamic techniques. ML approaches require large volume of data for training and hence demand for considerable amount of system resources and high computational power however, these problems can be resolved using DL.

Fig. 10 shows distribution of research publications between 2013– 2020. It is observed that more focus is laid on static techniques between 2013–2015. With advancement in technology and bypassing mechanisms, there is a need for dynamic and hybrid techniques that are more stringent with respect to static techniques. Post 2018, there is a shift towards ML techniques, which are robust and scalable enough to perform program analyses.

5.3. *Analysis techniques aspects*

The third dimension talks about the different aspects of the program analyses techniques, which distinguishes them from each other. This dimension tries to answer RQ2 and RQ3, which is based on the design and implementation details of these approaches and current state of Android security. Seven

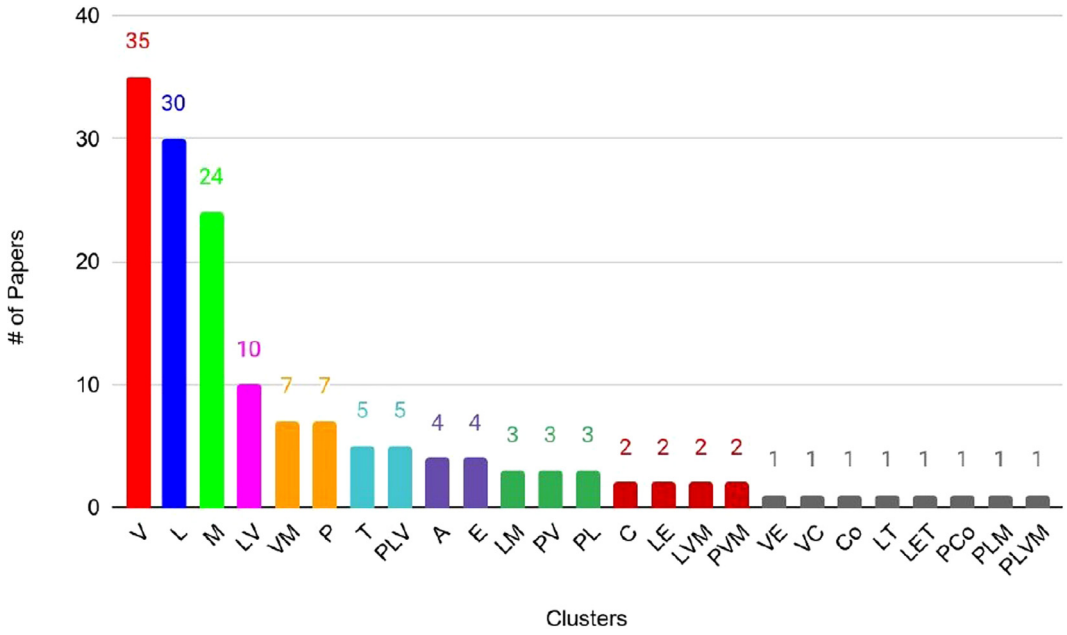

Fig. 8 – Distribution of papers in security objective clusters.

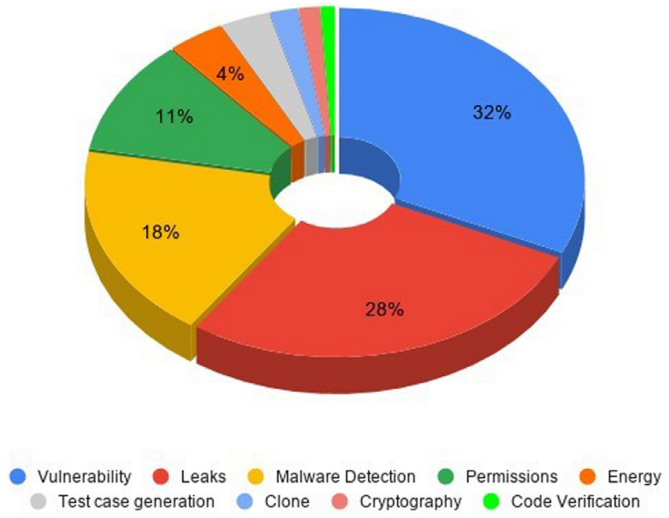

Fig. 9 – Statistics research publications addressing the main objectives.

sub-dimensions are discussed below, where the first four belongs to static analysis techniques, the next two are applied to dynamic analyses and the last one is for ML techniques.

First, static analysis dimensions, which are discussed in this SLR are Analysis techniques, Sensitivity analysis, Data structure used and Code representation.

5.3.1. *Analysis techniques*

In this study, we have investigated six fundamental techniques. Table 7 shows the papers applying these analysis techniques. These are discussed as follows:

1. *Symbolic execution:* Symbolic execution helps analyze the program by determining different inputs executing different parts of program. Inputs for propagating the program execution are symbolic values. These symbolic values are then used to generate variables, expressions and constraints that are used to produce possible outcomes for each conditional branch. These inputs are then used as test cases to explore the given path. The given path is considered to be infeasible when no input is produced. AppIntent (Yang et al., 2013a) generates GUI manipulations sequences using symbolic execution that lead to data transmission. Symbolic execution proves to be time-consuming for Android apps however, AppIntent reduces the search space without sacrificing the code coverage.
2. *Taint analysis:* It is a method of information flow analysis where an object is marked with an identifier, called as taint. The tainted object is then tracked using data-flow

Table 7 – Publications applying the static analysis techniques.

Techniques	Publications	Percentage
Symbolic Execution	AppIntent (Yang et al., 2013a), ClickRelease (Micinski et al., 2015), Gallingani et al. (Gallingani, 2014), Jensen et al. (Jensen et al., 2013), SIG-Droid (Mirzaei et al., 2015), TASMAN (Arzt et al., 2015), W2AIScanner (Hassanshahi et al., 2015)	6.7%
Taint Analysis	Anadroid (Liang et al., 2013), Apparecium (Titze and Schütte, 2015), AppAudit (Xia et al., 2015), AppCaulk (Schütte et al., 2014), AppContext (Yang et al., 2015b), Apposcopy (Feng et al., 2014), AppSealer (Zhang and Yin, 2014a), Bastani et al. (Bastani et al., 2015), Brox (Ma et al., 2013), Capper (Zhang and Yin, 2014b), Cortesi et al. (Cortesi et al., 2015), DescribeMe (Zhang et al., 2015a), Dflow+DroidInfer (Huang et al., 2015a), DroidJust (Chen and Zhu, 2015), DroidSafe (Gordon et al., 2015), FlowDroid (Arzt et al., 2014), FUSE (Ravitch et al., 2014), Gallingani et al. (Gallingani, 2014), Graa et al. (Graa et al., 2014), HelDroid (Andronio et al., 2015), IccTA (Li et al., 2015a), Lotrack (Lillack et al., 2017), MobSafe (Xu et al., 2013), PermissionFlow (Sbîrlea et al., 2013), SEFA (Wu et al., 2013), Sufatrio et al. (Chua et al., 2015), TASMAN (Arzt et al., 2015), Uranine (Al Nidawi et al., 2017), W2AIScanner (Hassanshahi et al., 2015), WeChecker (Cui et al., 2015)	28.6%
Program Slicing	Apparecium (Titze and Schütte, 2015), AppCaulk (Schütte et al., 2014), AppSealer (Zhang and Yin, 2014a), Bartsch et al. (Bartsch et al., 2013), Brox (Ma et al., 2013), Capper (Zhang and Yin, 2014b), CryptoLint (Egele et al., 2013), eLens (Hao et al., 2013a), Hopper (Blackshear et al., 2015), MobSafe (Xu et al., 2013), Poeplau et al. (Poeplau et al., 2014), Rocha et al. (Rocha et al., 2013), SAAF (Hoffmann et al., 2013)	12.4%
Abstract Interpretation	Anadroid (Liang et al., 2013), Bartsch et al. (Bartsch et al., 2013), Cortesi et al. (Cortesi et al., 2015), Hopper (Blackshear et al., 2015), Mandal et al. (Mandal et al., 2019), Rocha et al. (Rocha et al., 2013)	5.7%
Type Checking	Choi et al. (Choi and Chang, 2014), COVERT (Bagheri et al., 2015a), Dflow+DroidInfer (Huang et al., 2015a), DroidAlarm (Zhongyang et al., 2013), IFT (Ernst et al., 2014)	4.8%
Code Instrumentation	Android-app-analysis-tool (Geneiatakis et al., 2015a), AppCaulk (Schütte et al., 2014), AppSealer (Zhang and Yin, 2014a), AsyncDroid (Lin et al., 2015), Bastani et al. (Bastani et al., 2015), Brahmastra (Bhoraskar et al., 2014), Capper (Zhang and Yin, 2014b), Cassandra (Lortz et al., 2014), CMA (Shuai et al., 2014, August), DroidSafe (Gordon et al., 2015), IccTA (Li et al., 2015a), Nyx (Li et al., 2014), ORBIT (Yang et al., 2013b), Rocha et al. (Rocha et al., 2013), SIF (Hao et al., 2013b), Sufatrio et al. (Chua et al., 2015), Uranine (Al Nidawi et al., 2017), vLens (Li et al., 2013)	17.1%

analysis. An exception is raised when a tainted object flows to a sink. FlowDroid (Arzt et al., 2014) detects sensitive data leaks by using static taint analysis. AppSealer (Zhang and Yin, 2014a) generates patches for Android component hijacking attacks by leveraging taint analysis.

3. *Program slicing:* Program slicing is used to test a group of statements in a program for particular test cases or conditions that may affect a value at a particular point of time, while keeping the program behavior unchanged. Static program slices are large and consider all the possible execution program paths. Hoffmann et al. (2013) designed a framework SAAF. Program slices in SAAF track parameter values for a given Android method by performing backward data-flow analysis. CryptoLint (Egele et al., 2013) on the other hand, analyze cryptographic API methods by computing static program slices.
4. *Abstract interpretation:* Abstract interpretation is generally viewed as a partial execution of a program, where the semantics of a program such as data flow, control flow, etc. are taken into consideration.
5. *Type/Model checking:* Type and model checking are two commonly used approaches for program verification. Type checking verifies the type constraints of a program. It can occur either at compile time (static) or at execution time (dynamic). Type checking ensures the type-safety of a given program where there is a possibility of errors such as an integer operator applied to the strings or a float operation performed on an integer. On the other hand, model checking verifies the given specification of a finite-state system. Type and model checking are complimentary to each other where type checking is based on syntactic and modular style and model checking is based on semantic and whole-program style. As an example, COVERT (Bagheri et al., 2015a) uses model checking to verify the security specifications of a given app. In another example, Cassandra (Lortz et al., 2014) uses type checking to verify the compliance of Android

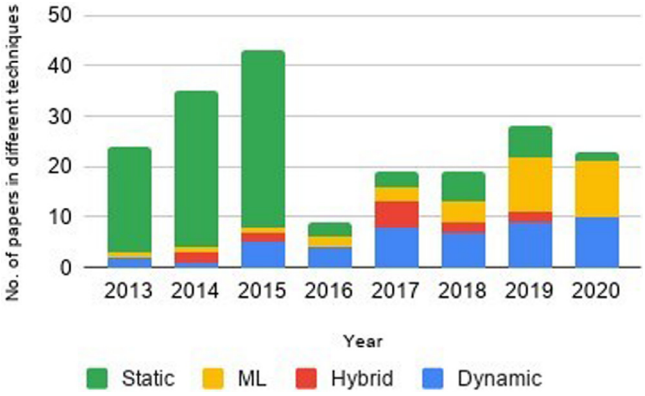

Fig. 10 – Year wise distribution of research publications over different analysis techniques.

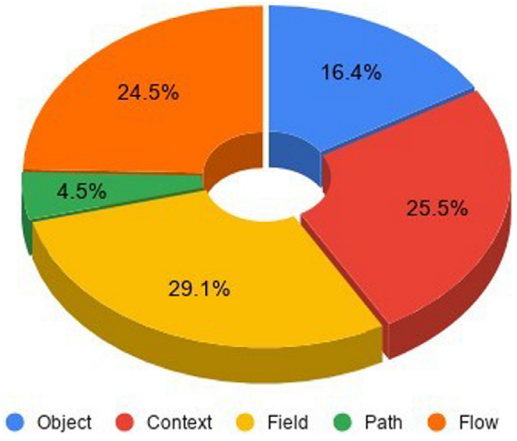

Fig. 11 – Distribution of research publications over sensitivities.

apps with their personal privacy requirements even before installing these apps.

6. *Code instrumentation:* Static analysis can be used to find the spots in the program where the code can be inserted to collect runtime behavior. As an example, IccTa (Li et al., 2015a) uses code instrumentation in Android apps to analyze the problems due to inter-component taint propagation. In another application, Nyx (Li et al., 2014) uses code instrumentation in Android web apps to make them more energy efficient by changing the background color of web pages from light to dark. In Android, various tools/ frameworks such as Soot (Lam et al., 2011) and WALA (Fink and Dolby, 2012) support code instrumentation.

It is observed that taint analysis is the most applied technique (28.6%) followed by code instrumentation (17.1%) and program slicing (12.4%). It should be noted that the percentage sum is less than 100% since some of the papers use basic data-flow analysis techniques and therefore, they are not categorised.

5.3.2. *Sensitivity analysis*

Static analyses techniques are required to be precise and abstract. Analysing sensitivities can fine tune the precision of a static analysis. Thus, different types of sensitivities are applied on the static analyses techniques are discussed. Table 8 classifies the different approaches according to the sensitivities they take into consideration. Fig. 11 shows the distribution of publications according to the sensitivities.

1. *Object Sensitivity:* This approach distinguishes method calls made on different objects. The code in a method can call other methods to create instances of objects or manipulate existing objects.
2. *Context Sensitivity:* This approach keeps track of the calling context of a method call and compute separate information for different calls of the same procedure.
3. *Flow Sensitivity:* This approach considers the order of statements and computes the separate information for each statement.
4. *Path Sensitivity:* This approach analyses execution path taken and distinguishes the information obtained from different paths.
5. *Field sensitivity:* A field-sensitive approach models each field of each object.

Field-sensitivity appears to be the most prominent with ~29% of the publications since Android is based on Java (Object-Oriented language), where object fields hold the data pervasively. Flow-sensitivity and Context-sensitivity are also considered largely (with 24% and 25% publications respectively). Only, 5% of the publications consider Path-sensitivity, primarily due to the scalability issues that it poses. It is considered that approaches that use more sensitivity are more precise in analysis. However, they are less scalable. Hopper (Blackshear et al., 2015) considers all the sensitivities into account and generate accurate results, but raises scalability issues.

5.3.3. *Data structures*

Heavyweight static analysis approaches that tend to give but more accurate results leverage well-known data structures to provide abstraction to the underlying programs. Table 9 shows the publications leveraging data structures.

1. *Call Graph (CG):* It is a directed graph, where each node indicates a method, and an edge represents the return from or call to a method.
2. *Control Flow Graph (CFG):* CFG is a directed graph that represents the basic flow of statements in a program. The nodes represent the program statements and edges denote the control flow of a program.
3. *Inter-procedural Control Flow Graph (ICFG):* It combines CG and CFG of all program procedures by connecting the call from and return to edges in the program procedure.

CGs (with 38%) are used to propagate taint information and to determine the source-to-sink reachablity analysis. As an example, ContentScope (Jiang and Xuxian, 2013) detects database leakage by traversing CG to find paths form pub-

Table 8 – Publications classified according the sensitivities considered.

Sensitivities	Publications	Percentage
Object	Anadroid (Liang et al., 2013), AppContext (Yang et al., 2015b), Apposcopy (Feng et al., 2014), Capper (Zhang and Yin, 2014b), DescribeMe (Zhang et al., 2015a), Dflow + DroidInfer (Huang et al., 2015a), DroidJust (Chen and Zhu, 2015), DroidSafe (Gordon et al., 2015), FlowDroid (Arzt et al., 2014), HelDroid (Andronio et al., 2015), Hopper (Blackshear et al., 2015), HornDroid (Calzavara et al., 2016), IccTA (Li et al., 2015a), Sufatrio et al. (Chua et al., 2015), TASMAN (Arzt et al., 2015), W2AIScanner (Hassanshahi et al., 2015), WeChecker (Cui et al., 2015)	16.4%
Context	Anadroid (Liang et al., 2013), AppCaulk (Schütte et al., 2014), AppContext (Yang et al., 2015b), Apposcopy (Feng et al., 2014), AppSealer (Zhang and Yin, 2014a), Bartsch et al. (Bartsch et al., 2013), Brox (Ma et al., 2013), COVERT (Bagheri et al., 2015a), DescribeMe (Zhang et al., 2015a), Dflow + DroidInfer (Huang et al., 2015a), DroidJust (Chen and Zhu, 2015), DroidSafe (Gordon et al., 2015), DroidSIFT (Zhang et al., 2014), Epicc (Octeau et al., 2013), FlowDroid (Arzt et al., 2014), FUSE (Ravitch et al., 2014), HelDroid (Andronio et al., 2015), Hopper (Blackshear et al., 2015), IccTA (Li et al., 2015a), IFT (Ernst et al., 2014), Jiang et al. (Jiang et al., 2017), Sufatrio et al. (Chua et al., 2015), TASMAN (Arzt et al., 2015), W2AIScanner (Hassanshahi et al., 2015), Yang et al. (Yang et al., 2018)	25.5%
Field	Anadroid (Liang et al., 2013), Apparecium (Titze and Schütte, 2015), AppAudit (Xia et al., 2015), AppCaulk (Schütte et al., 2014), AppContext (Yang et al., 2015b), Apposcopy (Feng et al., 2014), Appscalpel (Meng et al., 2019), AppSealer (Zhang and Yin, 2014a), AsDroid (Huang et al., 2014), Bartel et al. (Bartel et al., 2014), Bastani et al. (Bastani et al., 2015), Capper (Zhang and Yin, 2014b), Cassandra (Lortz et al., 2014), COVERT (Bagheri et al., 2015a), CryptoLint (Egele et al., 2013), DescribeMe (Zhang et al., 2015a), Dflow + DroidInfer (Huang et al., 2015a), DroidJust (Chen and Zhu, 2015), DroidSafe (Gordon et al., 2015), FlowDroid (Arzt et al., 2014), HelDroid (Andronio et al., 2015), Hopper (Blackshear et al., 2015), IccTA (Li et al., 2015a), IFT (Ernst et al., 2014), Pegasus (Chen et al., 2013), PermissionFlow (Sbîrlea et al., 2013), Rountev et al. (Rountev and Yan, 2014), SEFA (Wu et al., 2013), Sufatrio et al. (Chua et al., 2015), TASMAN (Arzt et al., 2015), WeChecker (Cui et al., 2015), Wognsen et al. (Wognsen et al., 2014)	29.1%
Path	Anadroid (Liang et al., 2013), Bartel et al. (Bartel et al., 2014), ContentScope (Jiang and Xuxian, 2013), Hopper (Blackshear et al., 2015), IFT (Ernst et al., 2014)	4.5%
Flow	ADS-SA (Song et al., 2019), Apparecium (Titze and Schütte, 2015), AppContext (Yang et al., 2015a), AppSealer (Zhang and Yin, 2014a), AsDroid (Huang et al., 2014), Bartsch et al. (Bartsch et al., 2013), Brox (Ma et al., 2013), Capper (Zhang and Yin, 2014b), Chen et al. (Chen et al., 2014a), DescribeMe (Zhang et al., 2015a), DroidJust (Chen and Zhu, 2015), DroidSafe (Gordon et al., 2015), DroidSIFT (Zhang et al., 2014), Epicc (Octeau et al., 2013), FlowDroid (Arzt et al., 2014), HelDroid (Andronio et al., 2015), Hopper (Blackshear et al., 2015), HornDroid (Calzavara et al., 2016), IccTA (Li et al., 2015a), JN-SAF (Wei et al., 2018), Pegasus (Chen et al., 2013), PermissionFlow (Sbîrlea et al., 2013), Sufatrio et al. (Chua et al., 2015), TASMAN (Arzt et al., 2015), Tuan et al. (Tuan et al., 2019), WeChecker (Cui et al., 2015), Wognsen et al. (Wognsen et al., 2014)	24.5%

lic content provider interfaces to the database function APIs. PermissionFlow (Sbîrlea et al., 2013) maps Android permissions to the corresponding APIs by traversing CG. AsDroid (Huang et al., 2014) generates CG to track intent messages.

27% of the publications use CFGs in this SLR. As an example, in ContentScope (Jiang and Xuxian, 2013) CFG is used to extract the constraints corresponding to potentially dangerous paths. These constraints are then fed into a constraint solver to generate inputs corresponding to candidate path executions.

Another data structure is based on the combination of CFG and CG is ICFG that links the individual CFGs according to how they call each other. More advanced and comprehensive program analyses rely on ICFG. In FlowDroid (Arzt et al., 2014), tainted variables are tracked by traversing ICFG. Epicc (Octeau et al., 2013) traverses ICFG to perform string analysis. In IccTA (Li et al., 2015a) inter-component data leaks are detected by running data-flow analysis over ICFG. Only few approaches leverage ICFG since the generated ICFGs are complex and potentially not scalable.

5.3.4. *Code representation*

The intermediate representation (IR) of the code is a simplified format that represents the original Dalvik bytecode and processes it, since Dalvik bytecode is considered to complex and difficult to manipulate. Static analysis approaches are implemented as off-the-shelf frameworks that implement analysis on their own intermediate representation (IR) of program code. Various code representations and different tools used in static analysis approaches are enumerated in the Tables 10 and 11 respectively. The following code representations used in static analyses are:

1. *Smali:* This intermediate representation (IR) is Apktool, reverse engineering tool for Android apps.
2. *Jimple:* Jimple is known to be the simplified version of Java bytecode. It is used by Soot (Lam et al., 2011) framework. Soot uses Dexpler (Bartel et al., 2012) plugin that translates Dalvik bytecode to Jimple IR.
3. *WALA:* WALA stands for T. J. Watson Libraries for Analysis. WALA IR is SSA-based representation that is built

Table 9 – Publications using different data structures.

Data structures	Publications	Percentage
Call Graph (CG)	ADS_SA (Song et al., 2019), Apparecium (Titze and Schütte, 2015), AppAudit (Xia et al., 2015), AppCaulk (Schütte et al., 2014), AppContext (Yang et al., 2015b), AppIntent (Yang et al., 2013a), Apposcopy (Feng et al., 2014), AppSealer (Zhang and Yin, 2014a), AsDroid (Huang et al., 2014), BlueSeal (Shen et al., 2014), Brahmastra (Bhoraskar et al., 2014), Brox (Ma et al., 2013), Capper (Zhang and Yin, 2014b), CMA (Shuai et al., 2014, August), ContentScope (Jiang and Xuxian, 2013), COVERT (Bagheri et al., 2015a), CryptoLint (Egele et al., 2013), DroidAlarm (Zhongyang et al., 2013), DroidGuard (Bagheri et al., 2015b), DroidSafe (Gordon et al., 2015), DroidSIFT (Zhang et al., 2014), Ecodroid (Behrouz et al., 2015), Epicc (Octeau et al., 2013), Flowdroid (Arzt et al., 2014), FUSE (Ravitch et al., 2014), Gallingani et al. (Gallingani, 2014), IccTA (Li et al., 2015a), JN-SAF (Wei et al., 2018), PaddyFrog (Wu et al., 2015), PermissionFlow (Sbîrlea et al., 2013), Poeplau et al. (Poeplau et al., 2014), Relda (Guo et al., 2013), SAAF (Hoffmann et al., 2013), SEFA (Wu et al., 2013), StaDynA (Zhauniarovich et al., 2015),TaskDroid (He et al., 2019), WeChecker (Cui et al., 2015), Zuo (Zuo et al., 2015)	38.1%
Control Flow Graph (CFG)	Anadroid (Liang et al., 2013), Apparecium (Titze and Schütte, 2015), AppCaulk (Schütte et al., 2014), AppContext (Yang et al., 2015b), Apposcopy (Feng et al., 2014), AsDroid (Huang et al., 2014), BlueSeal (Shen et al., 2014), Brahmastra (Bhoraskar et al., 2014), Capper (Zhang and Yin, 2014b), CMA (Shuai et al., 2014, August), ContentScope (Jiang and Xuxian, 2013), COVERT (Bagheri et al., 2015a), CryptoLint (Egele et al., 2013), DroidAlarm (Zhongyang et al., 2013), DroidSIFT (Zhang et al., 2014), DroidSim (Sun et al., 2014), Epicc (Octeau et al., 2013), Flowdroid (Arzt et al., 2014), Gallingani et al. (Gallingani, 2014), Graa et al. (Graa et al., 2014), IccTA (Li et al., 2015a), PaddyFrog (Wu et al., 2015), PermissionFlow (Sbîrlea et al., 2013), Poeplau et al. (Poeplau et al., 2014), SAAF (Hoffmann et al., 2013), SEFA (Wu et al., 2013), WeChecker (Cui et al., 2015), Wognsen (Wognsen et al., 2014)	26.7%
Inter-procedural Control Flow Graph (ICFG)	AppContext (Yang et al., 2015b), Apposcopy (Feng et al., 2014), Capper (Zhang and Yin, 2014b), COVERT (Bagheri et al., 2015a), CryptoLint (Egele et al., 2013), DroidAlarm (Zhongyang et al., 2013), Epicc (Octeau et al., 2013), Flowdroid (Arzt et al., 2014), Gallingani et al. (Gallingani, 2014), IccTA (Li et al., 2015a), LoTrack (Lillack et al., 2017), PermissionFlow (Sbîrlea et al., 2013), Poeplau et al. (Poeplau et al., 2014), SEFA (Wu et al., 2013), WeChecker (Cui et al., 2015)	15.2%

on a Java/Javascript static analysis framework WALA (Fink and Dolby, 2012).

4. *Java_bytecode/ Class:* Android has its own Dalvik bytecode format called Dex. It is different from Java and is executable by the Android virtual machine (VM). Tools such as ded (Octeau et al., 2010), Dare (Octeau et al., 2012), and dex2jar (dex2jar 2020) translates Dalvik to Java bytecode prior to the analysis using APK-to-JAR transformers.
5. *Dex_Assembler:* It disassesmbles DEX files using dedexer, dexdump and dx tools.

According to this study, Jimple is the most adopted IR (29.5%) and Soot is the most popular tool (25.7%) for static analysis of Android apps.

To monitor the runtime behavior of the source code of a program, dynamic analysis is performed. There are two main aspects with respect to dynamic analysis.

5.3.5. *Inspection level*

Dynamic analyses are categorised based on the inspection levels. These are Emulator-based/ Virtual Machine (VM) level, Kernel-level and App-level. Table 12 lists the dynamic analysis techniques using different inspection levels.

1. *Emulator-based/ Virtual Machine (VM)-level*: This level inspects the events that occur within the emulators by modifying VMs. There are two types of VMs-Dalvik VM and QEMU VM. Introspection based on Dalvik VM monitor the Android APIs execution through modifications in the Dalvik VM. QEMU based introspection are capable to trace native code. Dalvik VM are more efficient as compared to QEMU based VM. However, emulators are prone to emulator evasion (Arzt et al., 2014).
2. *Kernel-level:* API call execution can be monitored by collecting the system calls such as ltrace and strace using kernel modules. It allows partial tracing of the native code.
3. *App-level:* This level of monitoring is also called method tracing, where Java method invocation can be traced by injecting the bytecode and log statements inside the original Android app code or framework.

It can be seen that app-level monitoring accounts for 36.2% of the studied dynamic analysis approaches. Different libraries can be used to facilitate app-level monitoring such as SIF (Hao et al., 2013b). VM level accounts for 29.8% and about 21% of surveyed dynamic techniques capture app behavior through monitoring system calls using kernel level inspection e.g., Andrubis (Lindorfer et al., 2014b)).

Table 10 – Publications using different code representations .

IR	Publications	Percentage
Smali	ADS-SA (Song et al., 2019), Anadroid (Liang et al., 2013), ApkCombiner (Li et al., 2015b), Apparecium (Titze and Schütte, 2015), AppCaulk (Schütte et al., 2014), Capper (Zhang and Yin, 2014b), Chen (Chen et al., 2014b), ClickRelease (Micinski et al., 2015), CMA (Shuai et al., 2014, August), ContentScope (Jiang and Xuxian, 2013), DroidSim (Sun et al., 2014), HelDroid (Andronio et al., 2015), Jensen et al. (Jensen et al., 2013), MIGDroid (Hu et al., 2014), MobSafe (Xu et al., 2013), PaddyFrog (Wu et al., 2015), SAAF (Hoffmann et al., 2013), SEFA (Wu et al., 2013), SMV-Hunter (Sounthiraraj et al., 2014), Uranine (Al Nidawi et al., 2017), Wognsen (Wognsen et al., 2014)	21.9%
Jimple	android-app-analysis-tool (Geneiatakis et al., 2015a), Androlic (Pan et al., 2019), ApkCombiner (Li et al., 2015b), AppContext (Yang et al., 2015b), AppIntent (Yang et al., 2013a), Apposcopy (Feng et al., 2014), AppSealer (Zhang and Yin, 2014a), Bartel et al. (Bartel et al., 2014), Bartsch et al. (Bartsch et al., 2013), Bastani et al. (Bastani et al., 2015), BlueSeal (Shen et al., 2014), Brahmastra (Bhoraskar et al., 2014), Capper (Zhang and Yin, 2014b), COVERT (Bagheri et al., 2015a), DescribeMe (Zhang et al., 2015a), DEvA (Safi et al., 2015), DroidJust (Chen and Zhu, 2015), DroidSafe (Gordon et al., 2015), EcoDroid (Behrouz et al., 2015), Epicc (Octeau et al., 2013), Flowdroid (Arzt et al., 2014), Gallingani et al. (Gallingani, 2014), HelDroid (Andronio et al., 2015), IccTA (Li et al., 2015a), Lotrack (Lillack et al., 2017), Sufatrio et al. (Chua et al., 2015), TASMAN (Arzt et al., 2015), W2AIScanner (Hassanshahi et al., 2015), WeChecker (Cui et al., 2015)	29.5%
WALA_IR	Anadroid (Liang et al., 2013), AnDarwin (Crussell et al., 2013), AsDroid (Huang et al., 2014), Asynchronizer (Lin et al., 2014b), Brox (Ma et al., 2013), Hopper (Blackshear et al., 2015), ORBIT (Yang et al., 2013b), PermissionFlow (Sbîrlea et al., 2013), Poeplau et al. (Poeplau et al., 2014), Relda (Guo et al., 2013)	12.4%
Java_bytecode/ Class	AnDarwin (Crussell et al., 2013), AppAudit (Xia et al., 2015), AppIntent (Yang et al., 2013a), AppSealer (Zhang and Yin, 2014a), AsDroid (Huang et al., 2014), AsyncDroid (Lin et al., 2015), Bartsch et al. (Bartsch et al., 2013), Capper (Zhang and Yin, 2014b), Cen (Cen et al., 2014), Chen et al. (Chen et al., 2014a), Choi et al. (Choi and Chang, 2014), DroidAlarm (Zhongyang et al., 2013), DroidSIFT (Zhang et al., 2014), EcoDroid (Behrouz et al., 2015), eLens (Hao et al., 2013a), IFT (Ernst et al., 2014), Pegasus (Chen et al., 2013), PermissionFlow (Sbîrlea et al., 2013), Relda (Guo et al., 2013), SIF (Hao et al., 2013b), vLens (Li et al., 2013)	22.9%
Dex_Assembler	AndRadar (Lindorfer et al., 2014a), Brahmastra (Bhoraskar et al., 2014), Brox (Ma et al., 2013), CryptoLint (Egele et al., 2013), Dflow + DroidInfer (Huang et al., 2015a), Lin et al. (Lin et al., 2014a), Relda (Guo et al., 2013), StaDynA (Zhauniarovich et al., 2015)	7.6%

Table 11 – Publications using support tools for static analysis.

Tools	Publications	Percentage
Soot	ADS_SA (Song et al., 2019), Anadroid (Liang et al., 2013), Androlic (Pan et al., 2019), ApkCombiner (Li et al., 2015b), AppCaulk (Schütte et al., 2014), AppIntent (Yang et al., 2013a), Apposcopy (Feng et al., 2014), Bartel et al. (Bartel et al., 2014), Bartsch et al. (Bartsch et al., 2013), Brahmastra (Bhoraskar et al., 2014), Capper (Zhang and Yin, 2014b), Ding et al. (Li et al., 2015c), DroidSIFT (Zhang et al., 2014), Ecodroid (Behrouz et al., 2015), Epicc (Octeau et al., 2013), FlowDroid (Arzt et al., 2014), Lotrack (Lillack et al., 2017), MobSafe (Xu et al., 2013), Octeau et al. (Octeau et al., 2015), Pegasus (Chen et al., 2013), PermissionFlow (Sbîrlea et al., 2013), Rountev et al. (Rountev and Yan, 2014), SAAF (Hoffmann et al., 2013), Taskdroid (He et al., 2019), WeChecker (Cui et al., 2015), Wognsen et al. (Wognsen et al., 2014)	25.7%
APKTool	Anadroid (Liang et al., 2013), Chen et al. (Chen et al., 2014a), CMA (Shuai et al., 2014, August), DroidAlarm (Zhongyang et al., 2013), DroidSim (Sun et al., 2014), Kabakus (Kabakus, 2019), MobSafe (Xu et al., 2013), PermissionFlow (Sbîrlea et al., 2013), SAAF (Hoffmann et al., 2013), SMV-Hunter (Sounthiraraj et al., 2014)	9.5%
dex2jar	AsDroid (Huang et al., 2014), Capper (Zhang and Yin, 2014b), Chen et al. (Chen et al., 2014a), eLens (Hao et al., 2013a), Epicc (Octeau et al., 2013), Kabakus (Kabakus, 2019), PermissionFlow (Sbîrlea et al., 2013), Relda (Guo et al., 2013), SIF (Hao et al., 2013b), vLens (Li et al., 2013)	9.5%
Androguard	DroidAlarm (Zhongyang et al., 2013), MobSafe (Xu et al., 2013) Poeplau et al. (Poeplau et al., 2014), Relda (Guo et al., 2013)	3.8%
ded/ Dare	Epicc (Octeau et al., 2013), MobSafe (Xu et al., 2013), Relda (Guo et al., 2013)	2.9%
Dexpler	Brahmastra (Bhoraskar et al., 2014), FlowDroid (Arzt et al., 2014)	1.9%
BCEL	SIF (Hao et al., 2013b)	-

Table 12 – Dynamic analysis approaches using different inspection levels.

Inspection Levels	Publications	Percentage
VM-level	AppAudit (Xia et al., 2015), AppsPlayground (Rastogi et al., 2013), Aquifer (Nadkarni and Enck, 2013), CopperDroid2 (Tam et al., 2015), DroidUnPACK (Duan et al., 2018), Graa (Graa et al., 2014), Intellidroid (Wong and Lie, 2016), Marvin (Lindorfer et al., 2015), Min et al. (Min et al., 2019) , Neuner et al. (Neuner et al., 2014), StaDynA (Zhauniarovich et al., 2015), Suárez-Tangi et al. (Suárez-Tangil et al., 2018), VettDroid (Zhang et al., 2013), Zuo (Zuo et al., 2015)	29.8%
Kernel-level	Afonso et al. (Afonso et al., 2015), Andrubis (Lindorfer et al., 2014b), ASF (Backes et al., 2014), ASM (Heuser et al., 2014), AVDTester (Huang et al., 2015b), Compac (Wang et al., 2014), DataChest (Zhou et al., 2014), DeepDroid (Wang et al., 2015), FineDroid (Y. Zhang et al., 2015b), FlaskDroid (Bugiel et al., 2013)	21.3%
App-level	AppCaulk (Schütte et al., 2014), AppIntent (Yang et al., 2013a), AppSealer (Zhang and Yin, 2014a), Bartsch et al. (Bartsch et al., 2013), Brahmastra (Bhoraskar et al., 2014), Capper (Zhang and Yin, 2014b), CMA (Shuai et al., 2014, August), ContentScope (Jiang and Xuxian, 2013), Graa (Graa et al., 2014), MobSafe (Xu et al., 2013), PaddyFrog (Wu et al., 2015), Pegasus (Chen et al., 2013), SMV-HUNTER (Sounthiraraj et al., 2014), Uranine (Al Nidawi et al., 2017), Wijesekera (Wijesekera et al., 2015)	36.2%

Table 13 – Publications using dynamic analysis approaches.

Approaches	Publications	Percentage
Taint Analysis	AndroTaint (Shankar et al., 2017), Dai et al. (Dai et al., 2018), DroidAnalyst (Faruki et al., 2016), DYDROID (Qu et al., 2017), Graa et al. (Graa et al., 2017), IntelliDroid (Wong and Lie, 2016), Jiang at al. (Jiang and Zhuang, 2017), Luo et al. (Luo et al., 2019), Min et al. (Min et al., 2019), Neuner et al. (Neuner et al., 2014), VettDroid (Zhang et al., 2013), Xiong et al. (Xiong et al., 2017), Zou et al. (Zuo et al., 2015)	27.7%
Anomaly-based	AMD-EC (Ghaffari et al., 2017), AndroTaint (Shankar et al., 2017), Malik et al. (Malik et al., 2019), Suárez-Tangi et al. (Suárez-Tangil et al., 2018)	8.5%

5.3.6. *Approaches*

There are two main approaches for dynamic analysis: Taint analysis and Anomaly-based. Table 13 shows the list of publications using these approaches.

1. *Taint analysis:* This approach is similar to that of static taint analysis however, the tainted data is tracked during the program execution. TaintDroid (Enck et al., 2014) was the first technique to implement dynamic taint analysis however, it is out of the scope of this SLR.
2. *Anomaly-based:* This approach monitors regular behavioral activities in the device and looks for anomalies that deviate from the normal behavior. This approach tends to be more costly since it invokes large number of system calls (Bagheri et al., 2015c).

Taint analysis approaches are more adoptable (27.7%) as compared to anomaly-based approaches (8.5%) in the surveyed literature because anomaly-based detection approaches prove to be costly in terms of system calls.

Besides conventional program analysis techniques, there are other supplementary techniques such as ML to leverage the analysis. ML is a technique based on artificial intelligence, where the systems have the capability to learn from the available data, recognise patterns and can make decisions with least human intervention.

5.3.7. *Methods*

Various ML methods are: Supervised learning, Unsupervised learning, Semi-supervised learning, Reinforcement learning and Deep learning.

1. *Supervised learning (SL):* This class of algorithms takes known set of input data along with known responses to the output data and trains the model to generate predictions for the new dataset. Classification and regression are some of the well-known SL techniques. Classification is used to predict the discrete outcomes. Examples of classification algorithms are Naïve Bayes (NB), Support Vector Machine (SVM), Logistic Regression (LR), Decision Trees (DT), *k*-Nearest Neighbor (kNN) and Neural Networks (NN). For example, StormDroid (Chen et al., 2016) used SL for malware detection over 8000 apps with wide range of static and dynamic features. It achieved an accuracy of 93.8%. Garg and Baliyan (2019a) used parallel ensemble classifiers using Multi-Layer Perceptron (MLP), SVM, Pruning Rule-Based Classification Tree (PART), Ripple Down Rule Learner (RIDOR) to detect Android malware. They achieved an accuracy of 98.27%. Data processing and feature extraction details are given in Garg and Baliyan (2019b).
2. *Unsupervised learning (UL):* This class of algorithms draws inferences from the unlabeled input dataset. Clustering is commonly used UL technique. Common clustering algorithms are k-means, k-medoids, Hidden

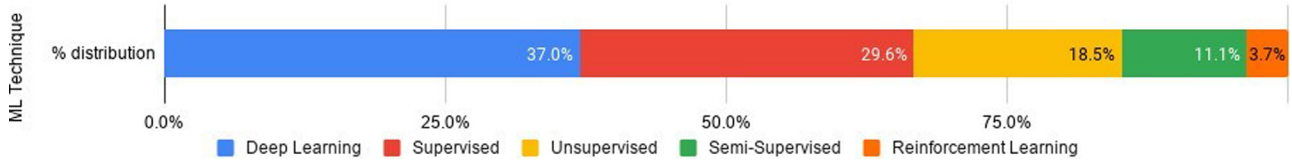

Fig. 12 – Distribution of research publications over ML methods.

Markov Models (HMV), etc. In Zarni Aung (2013) authors used k-means clustering algorithm cascaded with DT and Random Forest (RF) to detect Android malware. They achieved an accuracy of 91.75% for dataset 1 and 91.58% for dataset 2 using RF.

3. *Semi-supervised learning (SSL):* This class of algorithms combines both supervised and unsupervised learning techniques with some labeled and unlabeled data. In Mahindru and Sangal (2020), authors used Learning with Local and Global Consistency (LLGC), SSL method for malware detection with permissions and API calls as features and achieved an accuracy of 93.78%.
4. *Reinforcement learning (RL):* RL algorithms enables the agent to learn to achieve a goal in a potentially complex and uncertain environment by maximizing the cumulative reward. Some examples of RL are Deep Q Network (DQN), Deep Deterministic Policy Gradient (DDPG), Soft Actor-Critic (SAC), etc. Vuong and Takada (2019) proposed an automated GUI testing tool for Android applications using DQN. Authors have identified the semantic meanings of GUI elements and used them as an input to a NN, which through training approximates the behavioral model of the application under test.
5. *Deep learning (DL):* It is a class of ML algorithms that relates to the functioning of brain in the form of Artificial Neural Network (ANN). It has the capability to process large volumes of data. Well known DL algorithms are Deep Neural Networks (DNN), Convolutional Neural Network (CNN), Recurrent Neural Networks (RNN), Deep Belief Networks (DBN), etc. Pang et al. (2017) predicted vulnerable components and Java Android applications suing DNN and achieved an accuracy of 92.87%.

Table 14 summarises publications using ML techniques along with datasets, features, and evaluation parameters in the SLR. Fig. 12 shows the distribution of ML methods employed in the survey.

It is observed that most of the publications focus on DL (37%) since a large amount is data is easily processed by DL techniques. This is followed by supervised (29.6%) and unsupervised (18.5%) learning methods.

According to the "No Free Lunch" (NFL) theorem (Ryan et al., 2019), there is no single ML model that is best suited for every problem. The assumptions of one model for a particular problem may not hold for another problem therefore, multiple ML models can be tried and tested to find the one that works best for a particular problem.

6. Discussions and future research directions

While investigating the SLR, many discussion points hover around. To address the fourth research question (RQ4), various challenges and gaps are identified and potential future research directions are then presented.

6.1. *Cross-analysis*

This study is further extended to the different directions of the proposed taxonomy. The aim is to perform cross-analysis to get a clearer vision of the Android security analysis.

It is important to introspect the recurrent purposes of Android security addressed by program analysis techniques. It is observed that static and dynamic analysis approaches are used for detecting data leaks and vulnerabilities. The static techniques are generally used for detecting privacy data leaks than vulnerabilities (50% vs. 34.52%), while dynamic techniques are frequently leveraged for vulnerability detection than data leaks and malware detection (58.82% vs. 41.18%). Hybrid approaches, though at lower scales are mostly used for the vulnerability detection (19%) whereas ML approaches are widely used in malware detection (56%).

Another element of cross-analysis is the depth of the analysis techniques, i.e., at the application level or the framework level. The application-level analyzes the application software. Apps from unknown third party market stores pose a serious security challenge. Framework-level analyzes the potential system-level design flaws and issues encountered in the Android platform. It is observed that the dynamic approaches are more often leveraged for framework-level analysis (~55%). This is because dynamic approaches can track implicit relations between the Android permissions and system calls by deploying runtime modules, such as monitors in the Android framework. Moreover, the large code size of the Android framework (over ten million lines of code) makes it impossible for the static analysis approaches to monitor the runtime framework-level activity. Thus, dynamic techniques tend to be more scalable and less-expensive for framework-level monitoring.

6.2. *Challenges posed by Android specificities*

The inherent characteristics of Android pose several challenges for analysis techniques. These are listed as app component lifecycle, Inter-component Communications (ICC), XML file, and permissions. Component lifecycle callback methods such as onStart(), onStop(), onPause(), etc. are not connected with the apps or amongst themselves. Therefore, it is difficult for static analysis techniques to construct CFGs. ICC is a

Table 14 – Publications using ML Methods.

Work	ML Technique	Classifier	Dataset	No. of Apps	Features	Evaluation
AdDroid (Mehtab et al., 2020)	SL	Adaboost with DT	Contagio Dump, VirusShare, Google play store	Benign= 510 Malicious= 910	Permissions, API calls, functions, code features	Accuracy= 99.11%
Appice et al. (Appice et al., 2020)	UL	k-means++	GooglePlay Store, alternative Chinese Markets, alternative Russian Markets, Android Malware Genome Project	Benign=1,23,453 Malicious= 5560	Permissions, API calls, Network Addresses	Accuracy= 96.6%
ATMPA (Liu et al., 2019)	DL	RF, SVM, CNNs	Kaggle Microsoft Malware Classification Challenge	Benign=1000, Malware samples=10,867	-	Attack transferability rate=88.7%
DroidDetector (Yuan et al., 2016)	DL	DBN	Google Play Store, Contagio Community, Genome Project	Benign=20,000 Malicious=1760	Permissions, Sensitive API, Dynamic behaviors	Accuracy=96.76%
EASEAndroid (Wang et al., 2015a)	SSL	K-NN	Real-world Samsung devices	1.3 million audit logs		learns 2518 benign & malicious access patterns & generates 331 policy rules
Fadadu et al. (Fadadu et al., 2019)	DL	RF, DT, XGBoost, NN, CNN, RNN	CDAC Mohali, Malshare, VirusShare	Benign= 10,000 Malicious= 10,000	API call sequences	Accuracy=91.79%
Garg et al. (Garg and Baliyan, 2019a)	SL	MLP, SVM, PART, RIDOR, Ensemble Classifiers	Google play store, AMD, Androzoo, Wandiouja	Benign= 60,000 Malicious= 24,000	Permissions, API Calls, Libraries, Broadcast receivers	Accuracy=98.27%
Han et al. (Han et al., 2020)	SL	SVM	Google Play, Amazon AppStore, APKPure	Benign= 28,489 Malicious= 30,113	API calls	Accuracy=99.75%
Han et al. (Han and Olivier, 2020)	UL	K-means	-	Event sequences = 56,800	API system call traces-file events, memory events, network events, registry events, thread events	F1-score=0.8
Hsien-De et al. (Hsien-De Huang and Kao, 2018)	DL	CNN	Cheetah Mobile	Benign = 2 Mn Malicious=2 Mn	API calls	Accuracy=93%
Mahindru et al. (Mahindru and Sangal, 2020)	SSL	LLGC	Google play store, pandaap, gfan, hiapk, ANdrdoid, appchina, mumayi, slideme, Android Malware Genome Project, AndroMalShare	3,00,000 apks	Permissions, API calls	Accuracy=93.78%
Mantoo et al. (Mantoo and Khurana, 2020)	SL	k-NN, LR	Androtrack, Google play store	Benign= 300 Malicious= 300	Static-SMS, Phone, Storage, Contacts, Location, Camera, Microphone Dynamic-Close, Read, Get time of day, Futex, Clock get time, Mprotect, Epoll_pwait, Receive from, Send to, Ioctl, Write, Getuid Intrinsic-Size	Accuracy=97.5%

(*continued on next page*)

Table 14 (*continued*)

Work	ML Technique	Classifier	Dataset	No. of Apps	Features	Evaluation
Martín et al. (Martín et al., 2019)	SL	LR, RF	TACYT	82,866 suspicious apps, 259,608 malware signatures	Antivirus	F1-score= 0.84
Nguyen-Vu et al. (Nguyen-Vu et al., 2019)	DL	DNN	VirusShare, AMD, Google Play Store	Benign= 1901 Malicious= 1600	API calls	Accuracy=91.7%
Pang et al. (Pang et al., 2017)	DL	DNN	Java Android Applications	-	Continuous sequences of tokens in source code files	Accuracy=92.87%
SecureDroid (Chen et al., 2017a)	SL	SVM	Comodo Cloud Security Center	Set 1=8046 apps (4729 benign, 3, 317 malicious) Set 2= 72,891 apps (40,448 benign, 32,443 malicious)	Permissions, Filtered Intents, API calls, New-Instances	Accuracy=96.34%
Sharif et al. (Sharif and Nauman, 2019)	DL	CNN	AMD, Drebin	100 Android instructions	Binaries	F1-score= 0.76
Sharmeen et al. (Sharmeen et al., 2020)	DL, SSL	DNN	Drebin data set, Androzoo, ApkPure, ApkMirror	Benign= 10,000 Malicious= 8560	Permissions, API call	Accuracy=99.02%
Sourav et al. (Sourav et al., 2019)	DL	ANN, LR, NB, RF, GB	Kaggle	300 applications	Permissions, API call, command related	Accuracy=96.75%
Spreitzenbarth et al. (Spreitzenbarth et al., 2015)	SL	SVM	Drebin, Asian third-party mobile markets	Benign= 123,453 Malicious= 5560	Permissions, API calls	21% of them actually use native calls
StormDroid (Chen et al., 2016)	SL	SVM, DT, ANN, NB, k-NN, and Bagging predictor	Google Play Store, Contagio Mobile, MobiSec Lab	Benign= 4350 apps Malicious=3620 apps	Permissions, API calls, Sequences, Dynamic behaviors	Accuracy=93.8%
Tian (Tian, 2018)	SL	KNN, SVM, DT, RF	Malware Genome, VirusShare database, Google Play	Benign=683 Malicious =923	Permissions, API calls	FNR=0.35%, when evaluating malicious apps FPR=2.96%, when evaluating benign apps
Zarni Aung et al. (Zarni Aung, 2013)	UL	k-Means cascaded with DT, RF	Dataset 1 Dataset 2	200 apps, 500 apps	Permissions	Dataset1 =91.75% Dataset2 =91.58% using RF

message-passing mechanism used by the applications insulated in sandbox. ICC introduces several security issues such as interception of intent messages due to a lack of encryption mechanisms. AndroidManifest.xml is one of the most important configuration files that is present in every Android app. It specifies the key components of an Android application such as enforced permissions. During compile time, the values in the manifest file are bound to the Android app, and thus, cannot be altered at runtime. Permissions are building blocks of the Android security model. The permissions present in the manifest file of an app grant access to the various resources and cross-application interactions. Recent versions of Android employ dynamic permission management where users are allowed to revoke permissions at run time. Several research publications have widely discussed the shortcomings posed by permission model like least privilege principle violated by

coarse-grained permissions (Bugiel et al., 2013), delegation attacks caused by enforcing access control policies at the level of individual (Ham et al., 2013), ignorance by end-users due to lack of permission awareness (Zhang et al., 2013), and misusing permissions (Zarni Aung, 2013). Other challenges are related to Java since Android applications are written in Java and hence face issues like reflection, managing of dynamic code loading, multithreading, polymorphism, and integration of native code.

6.3. *Future directions*

The proposed taxonomy uncovers various research gaps and challenges that need more focus on analysis approaches used in Android security. The following research directions are presented to the research community to stay ahead of today's advancing security threats:

1. *Android Security will remain a strong focus of research:* Android's open-source nature generates high interest among the researchers. Malware writers find it easy to exploit the security loopholes and researchers find it easy to gather data, execute experiments, and implement test solutions on Android.
2. *Integrating multiple analytical approaches to obtain more precise and accurate results:* Relying on the single technique will never give the best result due to their inherent limitations. More reliable hybrid techniques along with ML and DL, consolidating the advantages of both static and dynamic approaches could be leveraged for high precision analysis.
3. *Analyzing different code forms for thorough analysis:* Apart from Java, other code forms are also present in Android apps such as native C/C++, Dalvik bytecode, and binaries. Dynamic code loading (DCL) (Chen et al., 2013) and reflection (Andronio et al., 2015) are used to access Dalvik bytecode, whereas the Java Native Interface (JNI) API (Lortz et al., 2014) is used to access binaries. Very few state-of-the-art static analysis approaches have considered DCL and JNI for analysis, thereby obtaining incomplete results.
4. *Shifting from single app analysis to system-level analysis to unfold compositional vulnerabilities:* Malware writers can easily exploit the vulnerabilities of multiple benign apps. However, most of the studies in the existing literature focus on single app or individual system components. As an example, Bagheri et al. (2015c) analyzed the permission protocol of Android to check the security requirements of preventing an unauthorised access. It is challenging to identify such attacks that require complete system analysis rather than analyzing individual parts of the system in isolation. Therefore, it is important to completely analyze vulnerabilities to predict all the possible ways to exploit the vulnerabilities at the system level.
5. *Focusing on repudiation:* The search process employed in this SLR retrieved no results for Android repudiation. The research community should take into account the potential threats and weaknesses that are caused by the vetting process of the Android. Vetting process is a procedure of code signing where developers can verify an authorship of an application. A User ID (UID) is assigned to digitally sign the application based on the digital certificate. If same certificate is used to sign multiple applications then android:sharedUserId key is specified the in the manifest file to share the same UID. This key can cause non-deterministic behavior when assigning the UID (Crăciunescu, 2020).

7. Threats to validity

This SLR outlines different threats to validity to highlight the potential impact and measures for mitigation.

7.1. *Construct validity*

Threats to construct validity can arise because of keywords-based search performed on repositories and conference venues. The overall publication list may not be exhaustive since only the top 15 conference venues are taken into consideration that are related to SE/PL and S&P domains. However, the attempt to search top-ranked venues guarantees that the influential 15 papers are taken into account. To mitigate this, citation chaining is done that helps reap the best-related papers. In addition, the iterative keyword-based search is incorporated to refine the search results timely.

7.2. *Internal validity*

Threats to internal validity can be primarily due to limited and focused searches on the topic concerned. It is primarily related to the soundness of proposed taxonomy, i.e., whether the proposed taxonomy can categorise the different aspects of Android security. To handle this, an iterative-content analysis method is employed, where the taxonomy is evolving constantly to account for new concepts and ideas encountered in the papers.

7.3. *External validity*

The analysis based on this study is performed on the research publications collected from 2013–2020, to cover all possible approaches used in Android security to date. Though, the trends may vary for different time-period.

7.4. *Conclusion validity*

The threat may also occur due to consideration of studies with individual reviewer's bias. This may lead to flaws and biases in our study. To maintain the objectiveness cross-checking mechanism is applied, where a thorough examination is done to ensure that no paper is reviewed by a single reviewer. To further minimise the individual reviewer's bias, inferences made in this SLR are based on cumulative and general observations from different papers rather than relying on a single reviewer's interpretation.

8. Conclusion

With the growing security threats in mobile platforms, particularly Android has led to considerable research efforts in the domain of Android security. This paper presented the state-of-the-art approaches concerning Android security. To achieve this, a comprehensive taxonomy is constructed for categorizing different aspects of Android security. A formal process of SLR is followed by analyzing 200 research papers published in the domains of security and privacy, software engineering, programming languages, and mobile computing from 2013 to 2020. The search strategy and selection criteria employed in this article play a key role in making this study more sound and complete. It is evident from the results of SLR that Android security has gained a lot of focus in the last few years primarily due to the popularity among mobile platforms as well as increasing incidents of malware and vulnerabilities. To summarise, this study points out (1) the purposes of multiple analysis approaches, (2) most of the analysis approaches are built on Soot framework, (3) taint analysis is the most used analysis technique, (4) path sensitivity is ignored by the majority of the research works, (5) integrated analysis techniques prove to be much beneficial than pure static or dynamic approaches.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

REFERENCES

Afonso VM, de Amorim MF, Grégio ARA, Junquera GB, de Geus PL. Identifying Android malware using dynamically obtained features. J . Computer Virol . Hacking Tech. 2015;11(1):9–17.

Agarwal M, Srivastava GMS. Cloud computing: a paradigm shift in the way of computing. Int. J . Mod. Educ . Comput. Sci. 2017;9(12).

Al Nidawi HSA, Wei KT, Dawood KA, Khaleel A. Energy consumption patterns of mobile applications in Android platform: a systematic literature review. J . Theor . Appl. Inf. Technol. 2017;95(24).

Alhanahnah M, Yan Q, Bagheri H, Zhou H, Tsutano Y, Srisa-An W, Luo X. DINA: detecting hidden Android inter-app communication in dynamic loaded code. IEEE Trans. Inf. Forensics Secur. 2020;15:2782–97.

Alzaylaee MK, Yerima SY, Sezer S. DynaLog: An automated dynamic analysis framework for characterizing android applications. In: International Conference on Cyber Security and Protection Of Digital Services (Cyber Security); 2016. p. 1–8.

Android Timeline and Versions, 2020. Available from: http://faqoid.com/.

Andronio N, Zanero S, Maggi F. Heldroid: Dissecting and detecting mobile ransomware. In: International Symposium on Recent Advances in Intrusion Detection; 2015. p. 382–404.

Appice A, Andresini G, Malerba D. Clustering-aided multi-view classification: a case study on android malware detection. J . Intell. Inf. Syst. 2020:1–26.

Arzt S, Rasthofer S, Fritz C, Bodden E, Bartel A, Klein J, Le Traon Y, Octeau D, McDaniel P. Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. ACM Sigplan Notices 2014;49(6):259–69.

Arzt S, Rasthofer S, Hahn R, Bodden E. Using targeted symbolic execution for reducing false-positives in dataflow analysis. In: Proceedings of the 4th ACM SIGPLAN International Workshop on State of the Art in Program Analysis; 2015. p. 1–6.

Backes M, Bugiel S, Gerling S, von Styp-Rekowsky P. Android security framework: extensible multi-layered access control on android. In: Proceedings of the 30th Annual Computer Security Applications Conference; 2014. p. 46–55.

Bagheri H, Sadeghi A, Garcia J, Malek S. Covert: Compositional analysis of android inter-app permission leakage. IEEE Trans. Softw. Eng. 2015a;41(9):866–86.

Bagheri H, Sadeghi A, Jabbarvand R, Malek S. Automated Dynamic Enforcement of Synthesized Security Policies in Android. George Mason University; 2015b. Tech. Rep. GMU-CS-TR-2015-5.

Bagheri H, Kang E, Malek S, Jackson D. Detection of design flaws in the android permission protocol through bounded verification. In: International Symposium on Formal Methods; 2015c. p. 73–89.

Bao W, Yao W, Zong M, Wang D. Cross-site scripting attacks on android hybrid applications. In: Proceedings of the International Conference on Cryptography, Security and Privacy; 2017. p. 56–60.

Barkallah H, Gzara M, Abdallah HB. Evolution of the distributed computing paradigms: a brief road map. Int. J . Comput . Digit. Syst. 2017;6(05):233–49.

Bartel A, Klein J, Le Traon Y, Monperrus M. Dexpler: converting android dalvik bytecode to jimple for static analysis with soot. In: Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program analysis; 2012. p. 27–38.

Bartel A, Klein J, Monperrus M, Le Traon Y. Static analysis for extracting permission checks of a large-scale framework: the challenges and solutions for analyzing android. IEEE Trans. Softw. Eng. 2014;40(6):617–32.

Bartsch S, Berger B, Bunke M, Sohr K. The transitivity-of-trust problem in android application interaction. In: 2013 International Conference on Availability, Reliability and Security; 2013. p. 291–6.

Bastani O, Anand S, Aiken A. Interactively verifying absence of explicit information flows in Android apps. ACM SIGPLAN Notices 2015;50(10):299–315.

Behrouz RJ, Sadeghi A, Garcia J, Malek S, Ammann P. Ecodroid: an approach for energy-based ranking of android apps. In: 2015 4th International Workshop on Green and Sustainable Software; 2015. p. 8–14.

Bhatia T, Verma AK. Data security in mobile cloud computing paradigm: a survey, taxonomy and open research issues. J . Supercomput. 2017;73(6):2558–631.

Bhoraskar R, Han S, Jeon J, Azim T, Chen S, Jung J, Nath S, Wang R, Wetherall D. Brahmastra: driving apps to test the security of third-party components. In: 23rd {USENIX} Security Symposium ({USENIX} Security 14); 2014. p. 1021–36.

Blackshear S, Chang BYE, Sridharan M. Selective control-flow abstraction via jumping. ACM SIGPLAN Notices 2015;50(10):163–82.

Bonett R, Kafle K, Moran K, Nadkarni A, Poshyvanyk D. Discovering flaws in security-focused static analysis tools for android using systematic mutation. In: 27th {USENIX} Security Symposium ({USENIX} Security 18); 2018. p. 1263–80.

Bugiel S, Heuser S, Sadeghi AR. Flexible and fine-grained mandatory access control on android for diverse security and privacy policies. In: 22nd {USENIX} Security Symposium ({USENIX} Security 13); 2013. p. 131–46.

Calzavara S, Grishchenko I, Maffei M. HornDroid: Practical and sound static analysis of Android applications by SMT solving. In: European Symposium on Security and Privacy (EuroS&P); 2016. p. 47–62.

Caputo D, Verderame L, Aonzo S, Merlo A. Droids in disarray: detecting frame confusion in hybrid Android apps. In: IFIP Annual Conference on Data and Applications Security and Privacy; 2019. p. 121–39.
Cen L, Gates CS, Si L, Li N. A probabilistic discriminative model for android malware detection with decompiled source code. IEEE Trans. Dependable Secure Comput. 2014;12(4):400–12.
Chen X, Zhu S. DroidJust: automated functionality-aware privacy leakage analysis for Android applications. In: Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks; 2015. p. 1–12.
Chen KZ, Johnson NM, D'Silva V, Dai S, MacNamara K, Magrino TR, Wu EX, Rinard M, Song DX. Contextual policy enforcement in android applications with permission event graphs. In: NDSS; 2013. p. 234.
Chen CM, Lin JM, Lai GH. Detecting mobile application malicious behaviors based on data flow of source code. In: 2014 International Conference on Trustworthy Systems and their Applications; 2014a. p. 1–6.
Chen K, Liu P, Zhang Y. Achieving accuracy and scalability simultaneously in detecting application clones on android markets. In: Proceedings of the 36th International Conference on Software Engineering; 2014b. p. 175–86.
Chen S, Xue M, Tang Z, Xu L, Zhu H. Stormdroid: a streaminglized machine learning-based system for detecting android malware. In: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security; 2016. p. 377–88.
Chen L, Hou S, Ye Y. Securedroid: enhancing security of machine learning-based detection against adversarial android malware attacks. In: Proceedings of the 33rd Annual Computer Security Applications Conference; 2017a. p. 362–72.
Chen H, Leung HF, Han B, Su J. Automatic privacy leakage detection for massive android apps via a novel hybrid approach. In: International Conference on Communications (ICC); 2017b. p. 1–7.
Choi K, Chang BM. A type and effect system for activation flow of components in Android programs. Inf. Process. Lett. 2014;114(11):620–7.
Chua TW, Tan DJ, Thing VL. Accurate specification for robust detection of malicious behavior in mobile environments. In: European Symposium on Research in Computer Security; 2015. p. 355–75.
Chuang HY, Wang SD. Machine learning based hybrid behavior models for Android malware analysis. In: International Conference on Software Quality, Reliability and Security; 2015. p. 201–6.
Citation chaining in Google Scholar and PubMed, 2020. Available from: https://canvas.seattlecentral.edu/courses/1252744/pages/citation-chaining-in-google-scholar-and-pubmed#/.
Cortesi A, Ferrara P, Pistoia M, Tripp O. Datacentric semantics for verification of privacy policy compliance by mobile applications. In: International Workshop on Verification, Model Checking, and Abstract Interpretation pp.; 2015. p. 61–79.
Coulter R, Han QL, Pan L, Zhang J, Xiang Y. Code analysis for intelligent cyber systems: a data-driven approach. Inf. Sci. 2020.
Denis Crăciunescu, The layers of the Android security model, 2020, Available from: https://proandroiddev.com/the-layers-of-the-android-security-model-90f471015ae6/
Crussell J, Gibler C, Chen H. Andarwin: scalable detection of semantically similar android applications. In: European Symposium on Research in Computer Security; 2013. p. 182–99.
Cui X, Wang J, Hui LC, Xie Z, Zeng T, Yiu SM. Wechecker: efficient and precise detection of privilege escalation vulnerabilities in android apps. In: Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks; 2015. p. 1–12.
Dai P, Pan Z, Li Y. In: 2018 3rd Joint International Information Technology, Mechanical and Electronic Engineering Conference (JIMEC 2018). Review of researching on dynamic taint analysis technique. Atlantis Press; 2018.
Darki A, Chuang CY, Faloutsos M, Qian Z, Yin H. Rare: a systematic augmented router emulation for malware analysis. In: International Conference on Passive and Active Network Measurement; 2018. p. 60–72.
De Lorenzo A, Martinelli F, Medvet E, Mercaldo F, Santone A. Visualizing the outcome of dynamic analysis of Android malware with VizMal. J . Inf. Secur. Appl. 2020;50.
Degu A. Android application memory and energy performance: systematic literature review. IOSR J. Comput. Eng. 2019;21(3):20–32.
dex2jar, 2020. Available: https://code.google.com/p/dex2jar/.
Duan Y, Zhang M, Bhaskar AV, Yin H, Pan X, Li T, Wang X, Wang X. In: NDSS. Things you may not know about Android (un) packers: a systematic study based on whole-system emulation; 2018.
Egele M, Brumley D, Fratantonio Y, Kruegel C. An empirical study of cryptographic misuse in android applications. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & Communications Security; 2013. p. 73–84.
Enck W, Gilbert P, Han S, Tendulkar V, Chun BG, Cox LP, Jung J, McDaniel P, Sheth AN. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Trans . Comput. Syst. (TOCS) 2014;32(2):1–29.
Ernst MD, Just R, Millstein S, Dietl W, Pernsteiner S, Roesner F, Koscher K, Barros PB, Bhoraskar R, Han S, Vines P. Collaborative verification of information flow for a high-assurance app store. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security; 2014. p. 1092–104.
Ernst MD. Static and dynamic analysis: synergy and duality. In: WODA 2003: ICSE Workshop on Dynamic Analysis; 2003. p. 24–7.
Evolution of the Mobile Phone, 2020. Available from: https://www.tigermobiles.com/evolution/#start/.
Fadadu F, Handa A, Kumar N, Shukla SK. Evading API call sequence based malware classifiers. In: International Conference on Information and Communications Security; 2019. p. 18–33.
Fang K, Yan G. Emulation-instrumented fuzz testing of 4G/LTE Android mobile devices guided by reinforcement learning. In: European Symposium on Research in Computer Security; 2018. p. 20–40.
Faruki P, Bharmal A, Laxmi V, Ganmoor V, Gaur MS, Conti M, Rajarajan M. Android security: a survey of issues, malware penetration, and defenses. IEEE Commun. Surv . Tutor. 2014;17(2):998–1022.
Faruki P, Bhandari S, Laxmi V, Gaur M, Conti M. Droidanalyst: synergic app framework for static and dynamic app analysis. In: Recent Advances in Computational Intelligence in Defense and Security; 2016. p. 519–52.
Feng Y, Anand S, Dillig I, Aiken A. Apposcopy: Semantics-based detection of android malware through static analysis. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering; 2014. p. 576–87.
Fink, S. and Dolby, J., 2012. WALA-The TJ Watson libraries for analysis.
Firdaus A, Anuar NB, Karim A, Ab Razak MF. Discovering optimal features using static analysis and a genetic search based method for Android malware detection. Front . Inf. Technol . Electron. Eng. 2018;19(6):712–36.
Gajrani J, Agarwal U, Laxmi V, Bezawada B, Gaur MS, Tripathi M, Zemmari A. EspyDroid+: precise reflection analysis of android apps. Comput . Secur. 2020;90.

Gallingani, D., 2014. Static detection and automatic exploitation of intent message vulnerabilities in Android applications.
García-Valls M, Dubey A, Botti V. Introducing the new paradigm of social dispersed computing: applications, technologies and challenges. J. Syst. Archit. 2018;91:83–102.
Garg S, Baliyan N. A novel parallel classifier scheme for vulnerability detection in android. Comput . Electr. Eng. 2019a;77:12–26.
Garg S, Baliyan N. Data on vulnerability detection in android. Data Brief 2019b;22:1081–7.
Geneiatakis D, Fovino IN, Kounelis I, Stirparo P. A permission verification approach for android mobile applications. Comput . Secur. 2015a;49:192–205.
Geneiatakis D, Fovino IN, Kounelis I, Stirparo P. A Permission verification approach for android mobile applications. Comput . Secur. 2015b;49:192–205.
Georgiev M, Jana S, Shmatikov V. In: NDSS Symposium, (Vol. 2014, p. 1). NIH Public Access. Breaking and fixing origin-based access control in hybrid web/mobile application frameworks; 2014.
Ghaffari F, Abadi M, Tajoddin A. AMD-EC: anomaly-based android malware detection using ensemble classifiers. In: 2017 Iranian Conference on Electrical Engineering (ICEE); 2017. p. 2247–52.
Google Scholar Metrics: available metrics, 0 0 0 0, 2020. Available from: https://scholar.google.com.sg/intl/en/scholar/metrics.html#metrics/.
Gordon MI, Kim D, Perkins JH, Gilham L, Nguyen N, Rinard MC. Information flow analysis of android applications in droidsafe, 15; 2015. p. 110.
Graa M, Boulahia NC, Cuppens F, Cavalliy A. Protection against code obfuscation attacks based on control dependencies in Android Systems. In: Eighth International Conference on Software Security and Reliability-Companion; 2014. p. 149–57.
Graa M, Cuppens-Boulahia N, Cuppens F, Lanet JL, Moussaileb R. Detection of side channel attacks based on data tainting in android systems. In: IFIP International Conference on ICT Systems Security and Privacy Protection; 2017. p. 205–18.
Guo C, Zhang J, Yan J, Zhang Z, Zhang Y. Characterizing and detecting resource leaks in Android applications. In: 28th International Conference on Automated Software Engineering (ASE); 2013. p. 389–98.
Ham YJ, Lee HW, Lim JD, Kim JN. DroidVulMon –Android based mobile device vulnerability analysis and monitoring system. In: Seventh International Conference on Next Generation Mobile Apps, Services and Technologies; 2013. p. 26–31.
Han X, Olivier B. Interpretable and adversarially-resistant behavioral malware signatures. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing; 2020. p. 1668–77.
Han H, Lim S, Suh K, Park S, Cho SJ, Park M. Enhanced Android malware detection: an SVM-based machine learning approach. In: International Conference on Big Data and Smart Computing (BigComp); 2020. p. 75–81.
Hao S, Li D, Halfond WG, Govindan R. Estimating mobile application energy consumption using program analysis. In: 35th International Conference on Software Engineering (ICSE); 2013a. p. 92–101.
Hao S, Li D, Halfond WG, Govindan R. SIF: a selective instrumentation framework for mobile applications. In: Proceeding of the 11th annual international conference on Mobile systems, applications, and services; 2013b. p. 167–80.
Haris, M., Haddadi, H. and Hui, P., 2014. Privacy leakage in mobile computing: tools, methods, and characteristics. arXiv preprint arXiv:1410.4978.
Hassanshahi B, Jia Y, Yap RH, Saxena P, Liang Z. Web-to-application injection attacks on android: characterization and detection. In: European Symposium on Research in Computer Security; 2015. p. 577–98.
He J, Chen T, Wang P, Wu Z, Yan J. Android multitasking mechanism: formal semantics and static analysis of apps. In: Asian Symposium on Programming Languages and Systems; 2019. p. 291–312.
Heuser S, Nadkarni A, Enck W, Sadeghi AR. {ASM}: a programmable interface for extending android security. In: 23rd {USENIX} Security Symposium ({USENIX} Security 14); 2014. p. 1005–19.
Hoffmann J, Ussath M, Holz T, Spreitzenbarth M. Slicing droids: program slicing for smali code. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing; 2013. p. 1844–51.
Hsien-De Huang T, Kao HY. R2-d2: color-inspired convolutional neural network (CNN)-based android malware detections. In: International Conference on Big Data (Big Data); 2018. p. 2633–42.
Hu W, Tao J, Ma X, Zhou W, Zhao S, Han T. Migdroid: detecting app-repackaging android malware via method invocation graph. In: 23rd International Conference on Computer Communication and Networks (ICCCN); 2014. p. 1–7.
Huang J, Zhang X, Tan L, Wang P, Liang B. Asdroid: detecting stealthy behaviors in android applications by user interface and program behavior contradiction. In: Proceedings of the 36th International Conference on Software Engineering; 2014. p. 1036–46.
Huang W, Dong Y, Milanova A, Dolby J. Scalable and precise taint analysis for android. In: Proceedings of the 2015 International Symposium on Software Testing and Analysis; 2015a. p. 106–17.
Huang H, Chen K, Ren C, Liu P, Zhu S, Wu D. Towards discovering and understanding unexpected hazards in tailoring antivirus software for android. In: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security; 2015b. p. 7–18.
Jensen CS, Prasad MR, Møller A. Automated testing with targeted event sequence generation. In: Proceedings of the 2013 International Symposium on Software Testing and Analysis; 2013. p. 67–77.
Jiang YZX, Xuxian Z. Detecting passive content leaks and pollution in android applications. Proceedings of the 20th Network and Distributed System Security Symposium (NDSS), 2013.
Jiang L, Zhuang Y. Detect storage vulnerability of user-input privacy in Android applications with static and dynamic analysis. In: International Conference on Cloud Computing and Security; 2017. p. 280–91.
Jiang H, Yang H, Qin S, Su Z, Zhang J, Yan J. Detecting energy bugs in Android apps using static analysis. In: International Conference on Formal Engineering Methods; 2017. p. 192–208.
Kabakus AT. What static analysis can utmost offer for Android malware detection. Inf. Technol . Control 2019;48(2):235–49.
Karim I, Cicala F, Hussain SR, Chowdhury O, Bertino E. Opening Pandora's box through ATFuzzer: dynamic analysis of AT interface for Android smartphones. In: Proceedings of the 35th Annual Computer Security Applications Conference; 2019. p. 529–43.
Kitchenham B, Brereton P. A systematic review of systematic review process research in software engineering. Inf . Softw. Technol. 2013;55(12):2049–75.
Lam P, Bodden E, Lhoták O, Hendren L. The Soot framework for Java program analysis: a retrospective, 15; 2011. p. 35.
Li D, Hao S, Halfond WG, Govindan R. Calculating source line level energy information for android applications. In: Proceedings of the 2013 International Symposium on Software Testing and Analysis; 2013. p. 78–89.
Li D, Tran AH, Halfond WG. Making web applications more energy efficient for OLED smartphones. In: Proceedings of the 36th International Conference on Software Engineering; 2014. p. 527–38.

Li L, Bartel A, Bissyandé TF, Klein J, Le Traon Y, Arzt S, Rasthofer S, Bodden E, Octeau D, McDaniel P. Iccta: Detecting inter-component privacy leaks in android apps, 1; 2015a. p. 280–91.
Li L, Bartel A, Bissyandé TF, Klein J, Le Traon Y. Apkcombiner: combining multiple android apps to support inter-app analysis. In: IFIP International Information Security and Privacy Conference; 2015b. p. 513–27.
Li D, Lyu Y, Wan M, Halfond WG. String analysis for Java and Android applications. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering; 2015c. p. 661–72.
Liam Tung, Bigger than Windows, bigger than iOS: Google now has 2.5 billion active Android devices, 2020, Available from: https://www.zdnet.com/article/bigger-than-windows-bigger-than-ios-google-now-has-2-5-billion-active-android-devices-after-10-years/
Liang S, Keep AW, Might M, Lyde S, Gilray T, Aldous P, Van Horn D. Sound and precise malware analysis for android via pushdown reachability and entry-point saturation. In: Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices; 2013. p. 21–32.
Lillack M, Kästner C, Bodden E. Tracking load-time configuration options. IEEE Trans. Softw. Eng. 2017;44(12):1269–91.
Lin J, Liu B, Sadeh N, Hong JI. Modeling users' mobile app privacy preferences: restoring usability in a sea of permission settings. In: 10th Symposium on Usable Privacy and Security (SOUPS); 2014a. p. 199–212.
Lin Y, Radoi C, Dig D. Retrofitting concurrency for android applications through refactoring. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering; 2014b. p. 341–52.
Lin Y, Okur S, Dig D. Study and refactoring of android asynchronous programming (t). In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE); 2015. p. 224–35.
Lindorfer M, Volanis S, Sisto A, Neugschwandtner M, Athanasopoulos E, Maggi F, Platzer C, Zanero S, Ioannidis S. AndRadar: fast discovery of android applications in alternative markets. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment; 2014a. p. 51–71.
Lindorfer M, Neugschwandtner M, Weichselbaum L, Fratantonio Y, Van Der Veen V, Platzer C. Andrubis –1,000,000 apps later: a view on current Android malware behaviors. In: 2014 Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS); 2014b. p. 3–17.
Lindorfer M, Neugschwandtner M, Platzer C. Marvin: efficient and comprehensive mobile app classification through static and dynamic analysis, 2; 2015. p. 422–33.
Liu Y, Zuo C, Zhang Z, Guo S, Xu X. An automatically vetting mechanism for SSL error-handling vulnerability in android hybrid Web apps. World Wide Web 2018;21(1):127–50.
Liu X, Zhang J, Lin Y, Li H. ATMPA: attacking machine learning-based malware visualization detection methods via adversarial examples. In: Proceedings of the International Symposium on Quality of Service; 2019. p. 1–10.
Lortz S, Mantel H, Starostin A, Bähr T, Schneider D, Weber A. Cassandra: towards a certifying app store for android. In: Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices; 2014. p. 93–104.
Luo L, Zeng Q, Cao C, Chen K, Liu J, Liu L, Gao N, Yang M, Xing X, Liu P. Tainting-assisted and context-migrated symbolic execution of android framework for vulnerability discovery and exploit generation. IEEE Trans . Mob. Comput . 2019.
Ma S, Tang Z, Xiao Q, Liu J, Duong TT, Lin X, Zhu H. Detecting GPS information leakage in Android applications. In: 2013 IEEE Global Communications Conference (GLOBECOM); 2013. p. 826–31.
Mahindru A, Sangal AL. Feature-based semi-supervised learning to detect malware from Android. In: Automated Software Engineering: a Deep Learning-Based Approach; 2020. p. 93–118.
Maier, D., Seidel, L. and Park, S., 2020. BaseSAFE: baseband SAnitized fuzzing through emulation. arXiv preprint arXiv:2005.07797.
Malik Y, Campos CRS, Jaafar F. Detecting Android security vulnerabilities using machine learning and system calls analysis. In: 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C); 2019. p. 109–13.
Mandal AK, Panarotto F, Cortesi A, Ferrara P, Spoto F. Static analysis of Android auto infotainment and ODB-II apps. Softw. Pract. Exp. 2019.
Mantoo BA, Khurana SS. Static, dynamic and intrinsic features based Android malware detection using machine learning. In: Proceedings of ICRIC 2019. Cham: Springer; 2020. p. 31–45.
Martín I, Hernández JA, de los Santos S. Machine-learning based analysis and classification of android malware signatures. Future Gener. Comput. Syst. 2019;97:295–305.
Martin W, Sarro F, Jia Y, Zhang Y, Harman M. A survey of app store analysis for software engineering. IEEE Trans. Softw. Eng. 2016;43(9):817–47.
Mazuera-Rozo A, Bautista-Mora J, Linares-Vásquez M, Rueda S, Bavota G. The Android OS stack and its vulnerabilities: an empirical study. Empir. Softw. Eng. 2019;24(4):2056–101.
Mehtab A, Shahid WB, Yaqoob T, Amjad MF, Abbas H, Afzal H, Saqib MN. AdDroid: rule-based machine learning framework for android malware analysis. Mob. Netw . Appl. 2020;25(1):180–92.
Meng Z, Xiong Y, Huang W, Qin L, Jin X, Yan H. AppScalpel: combining static analysis and outlier detection to identify and prune undesirable usage of sensitive data in Android applications. Neurocomputing 2019;341:10–25.
Micinski K, Fetter-Degges J, Jeon J, Foster JS, Clarkson MR. Checking interaction-based declassification policies for android using symbolic execution. In: European Symposium on Research in Computer Security; 2015. p. 520–38.
Min Z, Haimin Y, Ping C, Zhengxing Y. Android software vulnerability mining framework based on dynamic taint analysis technology. In: 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC); 2019. p. 2112–15.
Mirzaei N, Bagheri H, Mahmood R, Malek S. Sig-droid: Automated system input generation for android applications. In: 26th International Symposium on Software Reliability Engineering (ISSRE); 2015. p. 461–71.
Mojica IJ, Adams B, Nagappan M, Dienst S, Berger T, Hassan AE. A large-scale empirical study on software reuse in mobile apps. IEEE Softw. 2013;31(2):78–86.
Nadkarni A, Enck W. Preventing accidental data disclosure in modern operating systems. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security; 2013. p. 1029–42.
Neuner, S., Van der Veen, V., Lindorfer, M., Huber, M., Merzdovnik, G., Mulazzani, M. and Weippl, E., 2014. Enter sandbox: Android sandbox comparison. arXiv preprint arXiv:1410.7749.
Nguyen-Vu L, Ahn J, Jung S. Android fragmentation in malware detection. Comput . Secur. 2019;87.
Octeau D, Enck W, McDaniel P. The Ded Decompiler.. University Park, PA, USA: Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University; 2010. Tech. Rep. NAS-TR-0140-2010.
Octeau D, Jha S, McDaniel P. Retargeting Android applications to Java bytecode. In: Proceedings of the ACM SIGSOFT 20th international symposium on the foundations of software engineering; 2012. p. 1–11.

Octeau D, McDaniel P, Jha S, Bartel A, Bodden E, Klein J, Le Traon Y. Effective inter-component communication mapping in android: an essential step towards holistic security analysis. In: 22nd {USENIX} Security Symposium ({USENIX} Security 13); 2013. p. 543–58.
Octeau D, Luchaup D, Dering M, Jha S, McDaniel P. Composite constant propagation: application to android inter-component communication analysis, 1; 2015. p. 77–88.
Octeau D, Jha S, Dering M, McDaniel P, Bartel A, Li L, Klein J, Le Traon Y. Combining static analysis with probabilistic models to enable market-scale android inter-component analysis. In: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages; 2016. p. 469–84.
Pan L, Cui B, Yan J, Ma X, Yan J, Zhang J. Androlic: an extensible flow, context, object, field, and path-sensitive static analysis framework for Android. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis; 2019. p. 394–7.
Pang Y, Xue X, Wang H. Predicting vulnerable software components through deep neural network. In: Proceedings of the 2017 International Conference on Deep Learning Technologies; 2017. p. 6–10.
Poeplau S, Fratantonio Y, Bianchi A, Kruegel C, Vigna G. Execute this! analyzing unsafe and malicious dynamic code loading in android applications, 14; 2014. p. 23–6.
Qu Z, Alam S, Chen Y, Zhou X, Hong W, Riley R. Dydroid: measuring dynamic code loading and its security implications in android applications. In: 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN); 2017. p. 415–26.
Rashidi B, Fung CJ. A survey of Android security threats and defenses. J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl. 2015;6(3):3–35.
Rasthofer S, Asrar I, Huber S, Bodden E. How current android malware seeks to evade automated code analysis. In: IFIP International Conference on Information Security Theory and Practice; 2015. p. 187–202.
Rastogi V, Chen Y, Enck W. AppsPlayground: automatic security analysis of smartphone applications. In: Proceedings of the third ACM conference on Data and application security and privacy; 2013. p. 209–20.
Ravitch T, Creswick ER, Tomb A, Foltzer A, Elliott T, Casburn L. Multi-app security analysis with fuse: Statically detecting android app collusion. In: Proceedings of the 4th Program Protection and Reverse Engineering Workshop; 2014. p. 1–10.
Rocha BP, Conti M, Etalle S, Crispo B. Hybrid static-runtime information flow and declassification enforcement. IEEE Trans. Inf. Forensics Secur. 2013;8(8):1294–305.
Rountev A, Yan D. Static reference analysis for GUI objects in Android software. In: Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization; 2014. p. 143–53.
Ryan S, Corizzo R, Kiringa I, Japkowicz N. Deep learning versus conventional learning in data streams with concept drifts. In: 2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA); 2019. p. 1306–13.
Sadeghi A, Bagheri H, Garcia J, Malek S. A taxonomy and qualitative comparison of program analysis techniques for security assessment of android software. IEEE Trans. Softw. Eng. 2016;43(6):492–530.
Safi G, Shahbazian A, Halfond WG, Medvidovic N. Detecting event anomalies in event-based systems. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering; 2015. p. 25–37.
Sbîrlea D, Burke MG, Guarnieri S, Pistoia M, Sarkar V. Automatic detection of inter-application permission leaks in Android applications. IBM J. Res. Dev. 2013;57(6):10–11.
Schütte J, Titze D, De Fuentes JM. Appcaulk: data leak prevention by injecting targeted taint tracking into android apps. In: 13th International Conference on Trust, Security and Privacy in Computing and Communications; 2014. p. 370–9.
Schütte J, Fedler R, Titze D. Condroid: Targeted dynamic analysis of android applications. In: 29th International Conference on Advanced Information Networking and Applications; 2015. p. 571–8.
Schoepe D, Balliu M, Piessens F, Sabelfeld A. Let's face it: faceted values for taint tracking. In: European Symposium on Research in Computer Security; 2016. p. 561–80.
Sexton J, Chudnov A, Naumann DA. Spartan Jester: end-to-end information flow control for hybrid Android applications. In: IEEE Security and Privacy Workshops (SPW); 2017. p. 157–62.
Shankar VG, Somani G, Gaur MS, Laxmi V, Conti M. AndroTaint: an efficient android malware detection framework using dynamic taint analysis. In: ISEA Asia Security and Privacy (ISEASP); 2017. p. 1–13.
Sharif A, Nauman M. Function identification in Android binaries with deep learning. In: 2019 Seventh International Symposium on Computing and Networking (CANDAR); 2019. p. 92–101.
Sharmeen S, Huda S, Abawajy J, Hassan MM. An adaptive framework against android privilege escalation threats using deep learning and semi-supervised approaches. Appl. Soft Comput. 2020;89.
Shen F, Vishnubhotla N, Todarka C, Arora M, Dhandapani B, Lehner EJ, Ko SY, Ziarek L. Information flows as a permission mechanism. In: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering; 2014. p. 515–26.
Shrivastava G, Kumar P, Gupta D, Rodrigues JJ. Privacy issues of android application permissions: a literature review. Trans . Emerg. Telecommun. Technol. 2019:e3773.
Shuai S, Guowei D, Tao G, Tianchang Y, Chenjie S. Modelling analysis and auto-detection of cryptographic misuse in android applications. In: 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing; 2014. p. 75–80.
Song H, Lin D, Zhu S, Wang W, Zhang S. ADS-SA: system for automatically detecting sensitive path of Android applications based on static analysis. In: International Conference on Smart City and Informatization; 2019. p. 309–22.
Sounthiraraj D, Sahs J, Greenwood G, Lin Z, Khan L. Smv-hunter: large scale, automated detection of SSL/TLS man-in-the-middle vulnerabilities in android apps. Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS'14), 2014.
Sourav, S., Khulbe, D. and Kapoor, N., 2019. Deep learning based android malware detection framework. arXiv preprint arXiv:1912.12122.
Spreitzenbarth M, Schreck T, Echtler F, Arp D, Hoffmann J. Mobile-Sandbox: combining static and dynamic analysis with machine-learning techniques. Int. J. Inf. Secur. 2015;14(2):141–53.
Statcounter GlobalStats, Mobile operating system market share worldwide, 2020, Available from: https://gs.statcounter.com/os-market-share/mobile/worldwide
Suárez-Tangil G, Dash SK, García-Teodoro P, Camacho J, Cavallaro L. Anomaly-based exploratory analysis and detection of exploits in android mediaserver. IET Inf. Secur. 2018;12(5):404–13.
Suarez-Tangil G, Tapiador JE, Peris-Lopez P, Ribagorda A. Evolution, detection and analysis of malware for smart devices. IEEE Commun. Surv . Tutor. 2013;16(2):961–87.
Sun X, Zhongyang Y, Xin Z, Mao B, Xie L. Detecting code reuse in android applications using component-based control flow graph. In: IFIP International Information Security Conference; 2014. p. 142–55.

Tam K, Khan SJ, Fattori A, Cavallaro L. In: NDSS. Copperdroid: automatic reconstruction of android malware behaviors; 2015.
Tan DJ, Chua TW, Thing VL. Securing android: a survey, taxonomy, and challenges. ACM Comput. Surv. (CSUR) 2015;47(4):1–45.
Tang J, Li R, Wang K, Gu X, Xu Z. A novel hybrid method to analyze security vulnerabilities in Android applications. Tsinghua Sci . Technol. 2020;25(5):589–603.
Tian K. Learning-based Cyber Security Analysis and Binary Customization for Security. Virginia Tech; 2018.
Titze D, Schütte J. Apparecium: Revealing data flows in android applications. In: 29th International Conference on Advanced Information Networking and Applications; 2015. p. 579–86.
Tiwari A. Enhancing Users' Privacy: Static Resolution of the Dynamic Properties of Android. University of Potsdam; 2019.
Tsutano Y, Bachala S, Srisa-an W, Rothermel G, Dinh J. JITANA: a modern hybrid program analysis framework for android platforms. J . Comput. Lang. 2019;52:55–71.
Tuan LH, Cam NT, Pham VH. Enhancing the accuracy of static analysis for detecting sensitive data leakage in Android by using dynamic analysis. Clust. Comput. 2019;22(1):1079–85.
van Steen M, Tanenbaum AS. A brief introduction to distributed systems. Computing 2016;98(10):967–1009.
Vuong T, Takada S. Semantic analysis for deep Q-network in android GUI testing. In: 31st International Conference on Software Engineering and Knowledge Engineering, SEKE 2019. Knowledge Systems Institute Graduate School; 2019. p. 123–8.
Wang Y, Hariharan S, Zhao C, Liu J, Du W. Compac: enforce component-level access control in Android. In: Proceedings of the 4th ACM Conference on Data and Application Security and Privacy; 2014. p. 25–36.
Wang R, Enck W, Reeves D, Zhang X, Ning P, Xu D, Zhou W, Azab AM. Easeandroid: automatic policy analysis and refinement for security enhanced android via large-scale semi-supervised learning. In: 24th {USENIX} Security Symposium ({USENIX} Security 15); 2015a. p. 351–66.
Wang X, Sun K, Wang Y, Jing J. In: NDSS. DeepDroid: dynamically enforcing enterprise policy on android devices; 2015b.
Wang W, Zhao M, Gao Z, Xu G, Xian H, Li Y, Zhang X. Constructing features for detecting android malicious applications: issues, taxonomy and directions. IEEE Access 2019;7:67602–31.
Wang Y, Xu G, Liu X, Mao W, Si C, Pedrycz W, Wang W. Identifying vulnerabilities of SSL/TLS certificate verification in Android apps with static and dynamic analysis. J . Syst . Softw. 2020.
Wei F, Lin X, Ou X, Chen T, Zhang X. JN-SAF: Precise and efficient NDK/JNI-aware inter-language static analysis framework for security vetting of android applications with native code. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security; 2018. p. 1137–50.
Wijesekera P, Baokar A, Hosseini A, Egelman S, Wagner D, Beznosov K. Android permissions remystified: a field study on contextual integrity. In: 24th {USENIX} Security Symposium ({USENIX} Security 15); 2015. p. 499–514.
Wognsen ER, Karlsen HS, Olesen MC, Hansen RR. Formalisation and analysis of Dalvik bytecode. Sci . Comput. Program. 2014;92:25–55.
Wong MY, Lie D. IntelliDroid: a targeted input generator for the dynamic analysis of Android malware, 16; 2016. p. 21–4.
Wu L, Grace M, Zhou Y, Wu C, Jiang X. The impact of vendor customizations on android security. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security; 2013. p. 623–34.
Wu J, Cui T, Ban T, Guo S, Cui L. PaddyFrog: systematically detecting confused deputy vulnerability in Android applications. Secur . Commun. Netw. 2015;8(13):2338–49.
Wu H. A systematical study for deep learning based Android malware detection. In: Proceedings of the 2020 9th International Conference on Software and Computer Applications; 2020. p. 177–82.
Xia M, Gong L, Lyu Y, Qi Z, Liu X. Effective real-time android application auditing. In: IEEE Symposium on Security and Privacy; 2015. p. 899–914.
Xiong B, Xiang G, Du T, He JS, Ji S. Static taint analysis method for intent injection vulnerability in android applications. In: International Symposium on Cyberspace Safety and Security; 2017. p. 16–31.
Xu J, Yu Y, Chen Z, Cao B, Dong W, Guo Y, Cao J. MobSafe: cloud computing based forensic analysis for massive mobile applications using data mining. Tsinghua Sci . Technol. 2013;18(4):418–27.
Yang G, Huang J. Automated generation of event-oriented exploits in android hybrid apps. Proc. of the Network and Distributed System Security Symposium (NDSS'18), 2018.
Yang Z, Yang M, Zhang Y, Gu G, Ning P, Wang XS. Appintent: analyzing sensitive data transmission in android for privacy leakage detection. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security; 2013a. p. 1043–54.
Yang W, Prasad MR, Xie T. A grey-box approach for automated GUI-model generation of mobile applications. In: International Conference on Fundamental Approaches to Software Engineering; 2013b. p. 250–65.
Yang T, Yang Y, Qian K, Lo DCT, Qian Y, Tao L. Automated detection and analysis for android ransomware. In: 12th International Conference on Embedded Software and Systems; 2015a. p. 1338–43.
Yang W, Xiao X, Andow B, Li S, Xie T, Enck W. Appcontext: differentiating malicious and benign mobile app behaviors using context, 1; 2015b. p. 303–13.
Yang G, Mendoza A, Zhang J, Gu G. Precisely and scalably vetting Javascript bridge in android hybrid apps. In: International Symposium on Research in Attacks, Intrusions, and Defenses; 2017. p. 143–66.
Yang S, Wu H, Zhang H, Wang Y, Swaminathan C, Yan D, Rountev A. Static window transition graphs for Android. Automated Software Engineering 2018;25(4):833–73.
Yuan Z, Lu Y, Xue Y. Droiddetector: android malware characterization and detection using deep learning. Tsinghua Sci . Technol. 2016;21(1):114–23.
Zarni Aung WZ. Permission-based android malware detection. Int. J . Sci . Technol. Res. 2013;2(3):228–34.
Zhang M, Yin H. In: NDSS. AppSealer: automatic generation of vulnerability-specific patches for preventing component hijacking attacks in Android applications; 2014a.
Zhang M, Yin H. Efficient, context-aware privacy leakage confinement for android applications without firmware modding. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security; 2014b. p. 259–70.
Zhang Y, Yang M, Xu B, Yang Z, Gu G, Ning P, Wang XS, Zang B. Vetting undesirable behaviors in android apps with permission use analysis. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security; 2013. p. 611–22.
Zhang M, Duan Y, Yin H, Zhao Z. Semantics-aware android malware classification using weighted contextual api dependency graphs. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security; 2014. p. 1105–16.
Zhang M, Duan Y, Feng Q, Yin H. Towards automatic generation of security-centric descriptions for android apps. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security; 2015a. p. 518–29.
Zhang Y, Yang M, Gu G, Chen H. Finedroid: enforcing permissions with system-wide application execution context. In: International Conference on Security and Privacy in Communication Systems; 2015b. p. 3–22.

Zhauniarovich Y, Ahmad M, Gadyatskaya O, Crispo B, Massacci F. Stadyna: addressing the problem of dynamic code updates in the security analysis of android applications. In: Proceedings of the 5th ACM Conference on Data and Application Security and Privacy; 2015. p. 37–48.
Zhong H, Xiao J. Design for a cloud-based hybrid Android application security assessment framework. In: International Conference on Reliability, Maintainability and Safety (ICRMS); 2014. p. 539–46.
Zhongyang Y, Xin Z, Mao B, Xie L. DroidAlarm: an all-sided static analysis tool for Android privilege-escalation malware. In: Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security; 2013. p. 353–8.
Zhou Y, Singh K, Jiang X. Owner-centric protection of unstructured data on smartphones. In: International Conference on Trust and Trustworthy Computing; 2014. p. 55–73.
Zhu HJ, You ZH, Zhu ZX, Shi WL, Chen X, Cheng L. DroidDet: effective and robust detection of android malware using static analysis along with rotation forest model. Neurocomputing 2018;272:638–46.
Zuo C, Wu J, Guo S. Automatically detecting SSL error-handling vulnerabilities in hybrid mobile web apps. In: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security; 2015. p. 591–6.

Shivi Garg received her M.Tech in Information Security branch from Delhi Technological University, India in 2014. She has been a Ph.D. scholar at Indira Gandhi Delhi Technical University for Women, India since August 2016. Her research interests include Information Security, Android Malware, Machine Learning.

Niyati Baliyan received her Ph.D. in Computer Science and Engineering from Indian Institute of Technology, Roorkee, India in 2016. She has been an Assistant Professor at Department of IT, IGDTUW, since March, 2018. Her research interests include Semantic Web, Knowledge Engineering and Data analytics.