

Received May 5, 2020, accepted June 11, 2020, date of publication June 16, 2020, date of current version July 2, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3002842

# A Systematic Literature Review of Android Malware Detection Using Static Analysis

YA PAN<sup>1</sup>, XIUTING GE<sup>1,2</sup>, CHUNRONG FANG<sup>2</sup>, AND YONG FAN<sup>1</sup>

<sup>1</sup>Department of Computer Science and Technology, Southwest University of Science and Technology, Mianyang 621000, China

<sup>2</sup>State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China

Corresponding author: Ya Pan (panya@swust.edu.cn)


This work was supported in part by the National Natural Science Foundation of China under Grant 61802171 and Grant 61772014, in part by the Fundamental Research Funds for the Central Universities under Grant 021714380017, and in part by the Open Foundation of State Key Laboratory for Novel Software Technology in Nanjing University under Grant ZZKT2017B09.

**ABSTRACT** Android malware has been in an increasing trend in recent years due to the pervasiveness of Android operating system. Android malware is installed and run on the smartphones without explicitly prompting the users or without the user's permission, and it poses great threats to users such as the leakage of personal information and advanced fraud. To address these threats, various techniques are proposed by researchers and practitioners. Static analysis is one of these techniques, which is widely applied to Android malware detection and can detect malware quickly and prohibit malware before installation. To provide a clarified overview of the latest work in Android malware detection using static analysis, we perform a systematic literature review by identifying 98 studies from January 2014 to March 2020. Based on the features of applications, we first divide static analysis in Android malware detection into four categories, which include Android characteristic-based method, opcode-based method, program graph-based method, and symbolic execution-based method. Then we assess the malware detection capability of static analysis, and we compare the performance of different models in Android malware detection by analyzing the results of empirical evidence. Finally, it is concluded that static analysis is effective to detect Android malware. Moreover, there is a preliminary result that neural network model outperforms the non-neural network model in Android malware detection. However, static analysis still faces many challenges. Thus, it is necessary to derive some novel techniques for improving Android malware detection based on the current research community. Moreover, it is essential to establish a unified platform that is used to evaluate the performance of a series of techniques in Android malware detection fairly.

**INDEX TERMS** Android malware detection, static analysis, systematic literature review.

## I. INTRODUCTION

With the explosive growth of the mobile market in the last decade, Android has become the largest intelligent operating system. So far, the number of Android system has accounted for more than 80% of the whole market of smartphones.<sup>1</sup> When it is popular with plenty of application developers and users, Android operating system also becomes a preferred attack target for malicious offenders. In September 2010, it was the first time to detect malware in the Android system. Shortly later, the number of malicious applications is gradually increasing. Zhou *et al.* [1] discovered thousands of malicious applications in the following years. According to

The associate editor coordinating the review of this manuscript and approving it for publication was Zhaojun Li .

<sup>1</sup><https://www.tenda.com.cn/faq/7983.html>

G data,<sup>2</sup> the number of Android malware reached 3.2 million and increased by 40% year-on-year in the third quarter of 2018.

Android applications have been ubiquitous in daily life. According to Google Play,<sup>3</sup> there were 2.6 million Android applications available for users as of September 2018. However, many malicious applications are hidden in the Android market, which poses great threats to users. Android malware is installed and run on smartphones without explicitly prompting the users or without the users' permission. Generally, it has one or more of the following behaviours: forced installation, browser hijacking, stealing and modifying user data, malicious collection of user information,

<sup>2</sup><http://www.199it.com/archives/793849.html>

<sup>3</sup>[https://zh.wikipedia.org/wiki/Google\\_Play](https://zh.wikipedia.org/wiki/Google_Play)

malicious installation, malicious bundling, and other malicious behaviours. These behaviours would seriously infringe on the legitimate rights of users, and will even bring huge loss of interests to users. Based on these behaviours [2], Android malware can be divided into four categories, which include malware installation (e.g., repacking, update attack, and drive-by download), malware activation, malicious payloads (e.g., privilege escalations, remote control, finance charge, and information collection), and permission abuse.

To address such threats mentioned above, researchers and practitioners propose various techniques, which mainly include dynamic analysis and static analysis. Dynamic analysis instruments and executes the APK for malware detection, while static analysis can detect malware by scanning the entire APK without running the APK. Compared with dynamic analysis, although static analysis has some problems in resisting malicious deformation techniques such as java reflection and dynamic code loading, static analysis can detect malware quickly and prohibit malware before installation. Moreover, static analysis is not only scalable and usable when facing batch unknown APKs detection, but also can traverse all possible execution paths of the APKs.

Currently, static analysis has been already an important technique in Android malware detection. For example, Enck *et al.* [3] proposes a tool, which is mainly based on the combinations of dangerous permissions to identify vulnerable applications. It is an early representative static analysis methods to apply permissions to detect Android malware. ScanDal [4] takes Dalvik bytecode as input to detect private information leaks in Android applications. IccTA [5] extracts components from applications to detect malicious payloads. Alam *et al.* [6] automatically identify sensitive data leak paths through control flow graph and taint data analysis.

In order to learn about Android malware detection using static analysis, there are some surveys and reviews published before. Faruki *et al.* [7] discuss the problems of malware penetration and stealth techniques from the perspective of Android security architecture between 2010 and 2014. In addition, it also enumerates existing methods related to Android malware detection using static analysis. Calleja *et al.* [8] mainly focus on the evolution of malware and estimate of the development costs of malware repair based on the size and code quality of malware. Zachariah *et al.* [9] just present Android malware detection methods from three aspects of static analysis (i.e., signature-based detection, permission-based detection, and Dalvik bytecode detection) and discuss the advantages and limitations of these methods. However, there is not extensive coverage, which only includes 27 studies, and it also does not make the discussions and state the future work. Besides, a recent survey [10] in detail illustrates on the sensitivity of static program analysis (i.e., flow sensitivity, context sensitivity, path sensitivity, field sensitivity, and object sensitivity) and describes the application of static analysis techniques in the field of Android such as code detection, test case generation, and code verification. However, there is a gap in

the Android malware detection investigation in recent years. More importantly, with the rapid evolution of Android malware, some studies related to Android malware detection have obviously increased in past years (see Section III), and some novel solutions have emerged to detect Android malware. For example, analogizing graph classification technology, Canfora *et al.* [11] combine control flow graphs and graph kernels to analyze the difference between malicious and benign applications to identify Android malware. Therefore, it is indispensable to summarize Android malware detection using static analysis in recent years.

To have a clear view of Android malware detection using static analysis in the past few years, we perform this systematic literature review (SLR) after identifying thoroughly related studies. The main contributions of this SLR are listed as follows.

- We perform this SLR based on the key aspects of Android malware detection using static analysis.
- This paper divides static analysis in Android malware into four categories based on the features of applications. Thereafter, by analyzing the empirical evidence, this paper assesses the malware detection capability of static analysis and compares the performance of different models in Android malware detection.
- According to the results of empirical evidence, static analysis is effective and neural network model outperforms non-neural network model in Android malware detection.
- Finally, we make the discussions and provide the future work in Android malware detection using static analysis.

The rest of this SLR is organized as follows. Section II presents the research questions and the overall review protocols. Section III shows the results of the studies. Section IV discusses the research directions. Section V depicts the threats to the validity. Section VI concludes this paper and provides future work.

## II. REVIEW PROTOCOL

This SLR is based on the guidelines of Budgen and Brereton [12]. Figure 1 shows the complete process of this SLR, which is divided into three phases.

- Planning the review. This phase aims to identify the goal and develop the protocols of this SLR.
- Conducting the review. This phase points out the main contents of research in this SLR, which can be divided into six steps.
  - 1) Research questions. Research questions locate the issues that need to be analyzed in this SLR, and the answers to research questions perform the basis of the section of discussions.
  - 2) Search strategy. This step aims to identify search sources and search terms in order to collect primary studies.
  - 3) Study selection criteria. Study selection criteria include inclusion criteria and exclusion criteria. These criteria determine which studies are

TABLE 1. Research questions.

ID	Research Questions	Motivation
RQ1	RQ1: What are the popular static analysis techniques in Android malware detection?	Identify the category of static analysis techniques in Android malware detection
RQ2	How do these primary studies conduct the empirical experiments of Android malware detection using static analysis?	Identify the process of the empirical experiments
RQ2.1	Which datasets are used for Android malware detection?	Identify the appropriate experimental datasets
RQ2.2	Which support tools are used for static analysis in Android malware detection?	Identify the appropriate support tools for static analysis
RQ2.3	Which features are commonly used in Android malware detection?	Identify the commonly used features
RQ2.4	Which techniques are used for feature reduction in Android malware detection?	Identify the appropriate techniques for selecting and extracting relevant features
RQ2.5	Which models are used for Android malware detection?	Identify the commonly applied models
RQ2.6	Which performance measures are used for Android malware detection using static analysis?	Identify the commonly used performance measures
RQ3	What is the overall performance of static analysis techniques in Android malware detection based on empirical evidence?	Assess the effectiveness of static analysis techniques in Android malware detection
RQ4	Whether the performance of Model A is better than Model B in Android malware detection based on empirical evidence?	Compare the performance of different models

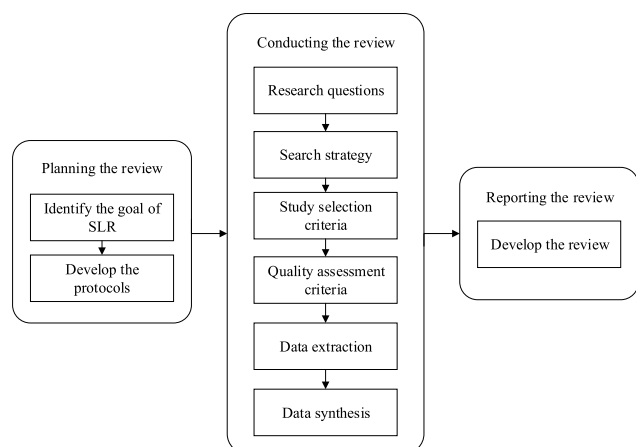


FIGURE 1. The overview of systematic literature review process.

included or excluded from this SLR, and we screen irrelevant studies on the basis of inclusion and exclusion criteria.

- 4) Quality assessment criteria. This step is used for assessing the relevance between selected studies and the objective of this SLR.
  - 5) Data extraction. The objective of this step is to design data extraction form to accurately record the information, which is related to research questions.
  - 6) Data synthesis. This step aims to collate and summarize the results of primary studies.
- Reporting the review. According to these guidelines, this phase aims to fulfill with this SLR.

**A. RESEARCH QUESTIONS**

This section mainly devises research questions, which is used to identify the primary studies and form the main part of this paper. Table 1 presents four research questions related to Android malware detection using static analysis. RQ1 identifies the category of static analysis in Android malware detection. RQ2 identifies the general process of assessing

the empirical evidence in the primary studies. This question includes six dimensions, which focus on the experimental datasets, the support tools for static analysis, the commonly used features, feature reduction techniques, the selected models to detect Android malware, and the performance measures, respectively. On the basis of empirical evidence, we research the RQ3 and RQ4. As for RQ3, the performance of static analysis techniques in Android malware detection is assessed by the commonly used performance measures. RQ4 aims to investigate whether the performance of Model A is better than Model B on the same dataset and performance measure.

**B. SEARCH STRATEGY**

To obtain studies related to Android malware detection using static analysis, we form some search terms that are associated with this paper. The main approach is to apply boolean expressions to combine search terms, and these boolean expressions include 'AND' or 'OR'. The search terms can be mainly summarized (mobile OR Android OR Smartphone) AND (malware OR malicious behaviours OR suspicious behaviours OR vulnerability) and (detection OR detect) OR (static analysis OR data flow OR control flow).

After ensuring these search terms, the relevant digital repositories are selected. We search the seven electronic databases, which are listed as follows.

- IEEE Xplore Digital Library
- ScienceDirect
- ACM Digital Library
- Wiley Online Library
- Google scholar
- SpringerLink
- Web of Science

The searching process is carried on the above seven electronic databases, which include main journals and conferences. These journals and conferences are mainly from Software Engineering and Programming Languages (SE/PL) and Security and Privacy (S&P), which are listed in Table 2.

TABLE 2. The main journals and conferences.

Category	Acronym	Full name
Journal	-	Computers and Security
	SCN	Security and Communication Networks
	-	IEEE Access
	TIFS	IEEE Transactions on Information Forensics and Security
	-	IET Information Security
	-	Neural Computing and Application
	TDSC	IEEE Transactions on Dependable and Secure Computing
	IST	Information and Software Technology
	JSS	Journal of Systems and Software
Conference	JCST	Journal of Computer Science and Technology
	ICSE	International Conference on Software Engineering
	FSE	Foundations of Software Engineering
	ISSTA	Conference on Object-Oriented Programming Systems, Languages, and Applications
	S&P	IEEE Symposium on Security and Privacy
	NDSS	Network and Distributed System Security Symposium
	ACSAC	Annual Computer Security Applications Conference
	CCS	ACM Conference on Computer and Communications Security
	OOPSLA	Conference on Object-Oriented Programming Systems, Languages, and Applications
	ICPC	IEEE International Conference on Program Comprehension
ICST	The IEEE International Conference on Software Testing, Verification and Validation	

Second, we also collect the studies that are related to Android malware detection using static analysis in the reference section, where studies take up a small proportion in the primary studies. All the studies related to search terms are taken into account, and the searching range is from January 2014 to December 2018.

C. STUDY SELECTION CRITERIA

To select the related studies, we build three inclusion criteria in the searching process based on seven electronic databases. First, search terms are included in the title or abstract or keywords. Second, static analysis techniques are introduced in the studies. Third, empirical experiments are conducted in the studies.

To filter out studies which are irrelevant with the objective of this SLR, a series of exclusion criteria are formed as follows.

- We filter out the studies that are not published in English.
- We exclude less extensive studies with corresponding duplicate papers. Generally, some studies are published in conferences and journals at the same time. According to authors, titles, and abstracts, we locate these studies, where less extensive studies are excluded.
- The studies related to window applications are excluded.
- The studies that apply dynamic analysis to detect Android malware are filtered out.
- Due to “Android” in the search terms, some studies about Android operating system are excluded.

D. QUALITY ASSESSMENT CRITERIA

To assess the quality of the selected studies, we follow the guidelines of quality assessment criteria in Table 3 and screen these studies. To ensure the reliability of the results, we use the cross-checking method to identify whether the selected studies meet these criteria. After the step of quality assessment criteria, the final studies are obtained, which include 94 studies and 4 SLRs related to Android malware detection.

TABLE 3. Quality assessment criteria.

ID	Quality assessment criteria
1	Are the aims of the study clear?
2	Is static analysis technique clearly stated?
3	Are the experimental datasets clearly stated?
4	Are the used features clearly stated?
5	Is the model clearly stated?
6	Are empirical experiments clearly stated?
7	Are performance measures clearly stated?
8	Does the study make the contribution to this SLR?

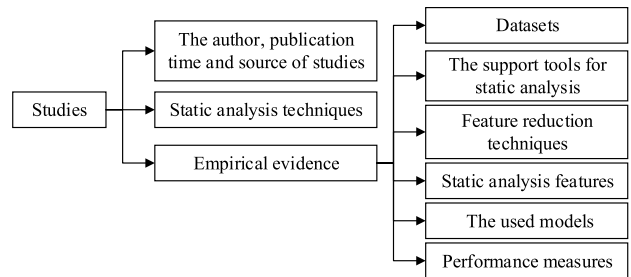


FIGURE 2. The information of the extracted information.

E. DATA EXTRACTION AND DATA SYNTHESIS

The process of data extraction is to design forms for accurately extracting the information from the primary studies. According to the information in the data extraction forms, we can obtain the answer to research questions. The extracted information in the data extraction forms is listed in Figure 2.

- The author, publication time, and publication source of studies. In this part, we can obtain the author, publication time, and publication source, including journals and conferences in the primary studies.
- Static analysis techniques. This part mainly focuses on static analysis techniques adopted by primary studies. According to the features of applications, we classify static analysis techniques.

- Empirical evidence. To answer these research questions, we summarize from six dimensions: experimental datasets, the support tools for static analysis, feature reduction techniques, static analysis features, the used models, and performance measures.

In the step of data synthesis, it mainly summarizes similar and comparable results from data extraction forms, which can provide supporting evidence for conclusive answers to research questions. After the step of data synthesis, the results are saved into an excel file. Finally, we adopt the histogram, pie chart, and table to show the conclusive results clearly.

### III. RESULTS

This section aims to present the results obtained from primary studies. We first describe primary studies. According to facts, we thereafter show the results of this SLR in light to the research questions.

#### A. DESCRIPTION OF STUDIES

This section describes 98 studies from the perspective of publication time and publication source.

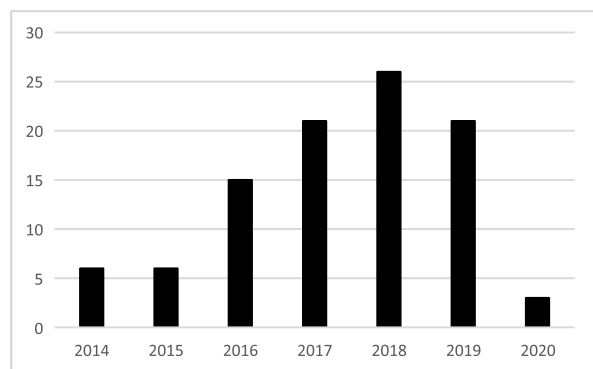


FIGURE 3. The year-wise distribution of studies.

#### 1) PUBLICATION TIME

Figure 3 shows that the number of studies from 2014 to 2020 is 6, 6, 15, 21, 26, 21, 3, respectively. According to this figure, it can be seen that the number of studies in 2018 takes the largest proportion. Except for papers in 2020 (some paper in 2020 are not published, so the period of these papers in 2020 is from January to March), the number of studies related to Android malware detection using static analysis is overall in sharp growth from 2014 to 2019. From this figure, it suggests that Android malware detection has always been a hot topic in the last few years.

#### 2) PUBLICATION SOURCE

Table 4 shows the main type of publication source and the number of studies related to publication source in the primary studies. According to Table 4, it presents that the first five publication sources are Computer and Security, Security and Communication Networks, IEEE Access, Information and

TABLE 4. The results of publication source.

ID	Publication Name	Type	No.
1	Computers and Security	Journal	13
2	Security and Communication Networks	Journal	8
3	IEEE Access	Journal	7
4	Information and Software Technology	Journal	3
5	IEEE Transactions on Information Forensics and Security	Journal	3
6	IET Information Security	Journal	2
7	Neural Computing and Application	Journal	2
8	IEEE Transactions on Dependable and Secure Computing	Journal	2
9	Future Generation Computer System	Journal	2
10	IEEE Transactions on Information Forensics and Security	Journal	2
11	Wireless Personal Communications	Journal	2
12	ACM Conference on Computer and Communications Security	Conference	2
13	Annual Computer Security Applications Conference	Conference	2

Software Technology, and IEEE Transactions on Information Forensics and Security, and the number of them is 13, 8, 7, 3, 3, respectively. It is noted that the remaining of the publication sources are not listed such as IEEE transaction on reliability and the Network and Distributed System Security Symposium. According to this table, it can be inferred that the number of studies from the journal occupies more than 50%.

#### B. RQ1: WHAT ARE THE POPULAR STATIC ANALYSIS TECHNIQUES IN ANDROID MALWARE DETECTION?

Static analysis is a kind of code analysis technique that takes source code or binary code of a program as input. It examines this code without running the program so as to verify the specifications (e.g., security and reliability) of this program. Compared to the dynamic analysis, the major advantages of static analysis do not need to execute the application, so it has high execution efficiency and fast speed. Due to these advantages, static analysis techniques are widely adopted in Android malware detection.

In general, an APK is mainly composed of META-INF, resource files, library directory, resource directory, AndroidManifest.xml, and binary files. According to extracted features from APKs, we divide static analysis into four categories, which include Android characteristic, opcode, program graph, and symbolic execution.

#### 1) ANDROID CHARACTERISTIC-BASED METHOD

This category mainly collects features related to Android characteristic, which can be almost obtained from both configuration files and bytecode programs in the binary files. After that, a series of features can be collected, such as permissions, API calls, intents, and hardware components. These features are considered as feature vectors, which are trained in the machine learning models or used in statistical models (see RQ2.5) for Android malware detection. For example, Feizollah *et al.* [15] extract the intents of applications as

features for the malware classification. Yerima *et al.* [16] attempt to use permissions and sensitive API calls to detect malware and the experimental result indicates these features achieves promising performance in Android malware detection.

## 2) OPCODE-BASED METHOD

The opcode is extracted from binary files of the APK. The binary files consist of many smali files, and each smali file represents a class file of this application, which is made up of a series of opcode sequences. This method focuses on treating opcode sequences as text and then combining natural language processing models or deep learning models to detect malware. For instance, Jerome *et al.* [17] extract all possible n-gram (i.e., the n-gram model is a natural language processing model, which is based on the assumption that the occurrence of the *n*th word is only related to the previous *n-1* words) from the opcode sequences as an initial feature sets, and combine the machine learning models to classify malware. Yan *et al.* [18] learn the contextual semantics of opcode sequences by Long Short-Term Memory to identify malware.

## 3) PROGRAM GRAPH-BASED METHOD

The program graph is extracted from binary files of APK. According to primary studies, the program graph can capture more syntax or semantic information compared to the two methods above. Thus, many studies adopt program graph-based method to analyze and detect malware. For example, Allix *et al.* [38] rely on the abstract representation control flow graph of an application, collect all basic blocks that compose and refer to them as features, combine machine learning models to detect malware. Arzt *et al.* [19] present to build data flow graphs among system calls, and then use graph matching techniques for Android malware detection. Fan *et al.* [20] extract function call graphs (FCGs), and then construct frequent subgraphs (fregraphs) from FCGs to represent common behaviours of malware samples that belong to the same malware family, and finally develop the tool of FalDroid, which automatically classifies malware and selects representative fregraphs in accordance with malware.

## 4) SYMBOLIC EXECUTION-BASED METHOD

The symbolic execution is an accurate procedure variable computing technology, which simulates application execution by replacing abstract symbols with variables. According to the conditional branches of given paths in the application, these abstract symbols are used to generate expressions and constraints, which could be used by constraint solver. Then the expressions and constraints of malicious applications are collected as rule library. If the expressions and constraints of an application are in this rule library, this application will be considered as a malicious application. For example, to reduce the high false-positive rate of Android malware detection using Flowdroid, TASMAL [21] uses symbolic execution to further detect Android malware and improve the performance

of Flowdroid. It points out in [90] symbolic execution has still faced many challenges especially path selection and constraint solving, thus the number of studies that use symbolic execution-based method to detect Android malware just takes up a very small proportion.

**TABLE 5.** The category of static analysis techniques.

Category	Studies	No.	PCT
Android characteristic-based method	[15], [16], [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [57], [58], [59], [60], [61], [62], [63], [64], [94], [95], [98], [99], [100], [101], [102], [105], [109], [111], [113], [114], [115]	52	53%
Opcode-based method	[17], [18], [65], [66], [67], [68], [69], [70], [71], [22], [72], [73], [25], [23], [96], [110], [112], [116]	18	19%
Program graph-based method	[20], [38], [74], [75], [24], [76], [77], [78], [79], [80], [81], [83], [90], [93], [97], [103], [104], [106], [107], [108]	20	20%
Symbolic execution-based method	[21], [26]	2	2%

Table 5 enumerates the category of static analysis techniques and shows the number (No.) and percentage (PCT) of the studies corresponding categories. Excluded 4 SLRs ([7]–[10]) and 2 studies ([24] and [82]) that combine multiple static analysis techniques, this table includes 92 studies. In this table, it shows Android characteristic-based method is the most commonly used static analysis technique, which accounted for about 53% in all the studies. And the number of primary studies related to the opcode-based method and program graph-based method occupies about 19% and 20%, respectively. The number of studies related to the symbolic execution-based method is the smallest. It suggests that Android characteristic-based method has been an important focus.

RQ1: According to extracted features from APKs, we divide static analysis into four categories: Android characteristic-based method, opcode-based method, program graph-based method, and symbolic execution-based method. Of these, Android characteristic-based method is most commonly used in Android malware detection.

## C. RQ2: HOW DO THESE PRIMARY STUDIES CONDUCT THE EMPIRICAL EXPERIMENTS OF ANDROID MALWARE DETECTION USING STATIC ANALYSIS

In response to RQ2, we analyze and summarize empirical evidence in detail. Figure 4 shows the five steps of the process of empirical experiments in Android malware detection. First, data collection aims to collect benign and malicious datasets. In general, the more experimental datasets, the more convincing the results are. Second, feature extraction aims to extract features from APKs by support tools for

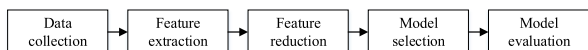


FIGURE 4. The steps of Android malware detection.

static analysis. Third, feature reduction techniques are adopted to determine and select significant features. Fourth, model selection aims to find an appropriate model to distinguish malicious applications from benign applications. These models include statistical models and machine learning models. Fifth, model evaluation aims to assess the generalization of models by performance measures. According to the process of empirical experiments, the following sections determine experimental datasets, support tools for static analysis, commonly used features, approaches to feature reduction, the used models, and performance measures in Android malware detection.

1) RQ2.1: WHICH DATASETS ARE USED FOR ANDROID MALWARE DETECTION?

In the primary studies, various malicious datasets have been adopted by researchers and practitioners. Based on the source of datasets [38], they can be divided into two categories: in-the-lab datasets and in-the-wild datasets. In-the-lab datasets are generally regarded as baselines in Android malware detection and mainly include Drebin,<sup>4</sup> Genome.<sup>5</sup> Genome is the first malware dataset that is collected and published. Based on Genome [2], Drebin is proposed to evaluate Android malware detection method, which is the first efficient and explainable method [35]. In-the-wild datasets are continuously updated and maintained. Due to a large number of samples and diverse categories, the performance of methods evaluated in the in-the-wild datasets is more credible than that of in-the-lab datasets. Concretely, in-the-wild datasets include Virustotal,<sup>6</sup> Virusshare,<sup>7</sup> Contagio,<sup>8</sup> AMD [20], McAfee,<sup>9</sup> and Androzoo.<sup>10</sup> In addition, some studies such as [22]–[24] adopt Kaggle<sup>11</sup> and Anzhi.<sup>12</sup> Kaggle is a competition platform, and Anzhi is a Chinese Android market. The datasets mentioned above are publicly available. However, a few studies still use unpublic datasets such as [25], [26]. In terms of benign datasets, the applications are mainly from Google Play Store,<sup>13</sup> and China Mobile Application Market, and benign datasets have no open source in most studies.

Excluded 4 SLRs and 6 studies that do not disclose malicious datasets, Figure 5 shows the number of datasets adopted

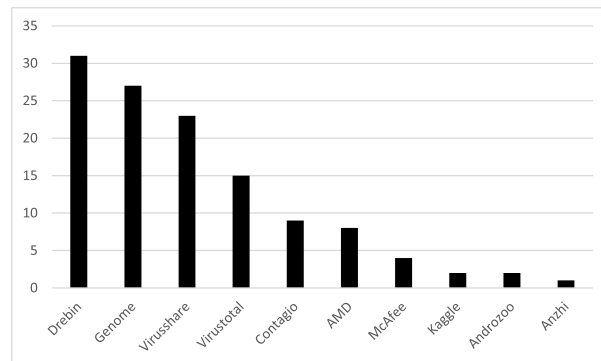


FIGURE 5. The number of malicious datasets in the primary studies.

in primary studies. To verify the usability and effectiveness of proposed methods, some malicious datasets are used multiple times by some studies. According to this figure, it presents Drebin and Genome are the most commonly used datasets in the in-the-lab datasets, and the most commonly used dataset is Virusshare in the in-the-wild datasets.

RQ2.1: In-the-lab and in-the-wild datasets occupy a similar proportion in the studies of Android malware detection. Of these, Drebin and Genome are the most commonly used experimental datasets.

2) RQ2.2: WHICH SUPPORT TOOLS ARE USED FOR STATIC ANALYSIS IN ANDROID MALWARE DETECTION?

For the purpose of supporting static analysis, a series of tools are adopted in primary studies. There are some off-the-shelf support tools to implement the preliminary static analysis. The difference among these tools is that intermediate representations [10] are different during the process of static analysis. The intermediate representations of these support tools include smali, dex\_assembler, jimple, and Java\_class. In order to obtain the terminal representation format of Android applications, it is still needed further analysis and processing.

Figure 6 enumerates current support tools for static analysis and shows the number of these support tools. According to this figure, it can be seen that Apktool takes up the largest proportion and is often used to decompile APKs. Table 6 shows the descriptions and intermediate representations (IR) of support tools and gives a few examples of specific studies with corresponding support tools.

RQ2.2: The difference in support tools for static analysis is intermediate representations. In primary studies, Apktool is the most commonly used support tool for static analysis in Android malware detection.

3) RQ2.3: WHICH FEATURES ARE COMMONLY USED IN ANDROID MALWARE DETECTION?

Different categories of features can represent behaviours of APKs at different levels. Thus, malware detection capabilities of these features are inconsistent. In order to detect

<sup>4</sup><http://www.sec.cs.tu-bs.de/danarp/drebin/>

<sup>5</sup><http://www.malgenomeproject.org/>

<sup>6</sup><https://www.virustotal.com>

<sup>7</sup><https://virusshare.com/>

<sup>8</sup><http://contagiominidump.blogspot.hk/>

<sup>9</sup><https://home.mcafee.com/Default.aspx?rfhs=1>

<sup>10</sup><https://androzoo.uni.lu/>

<sup>11</sup><https://www.kaggle.com/>

<sup>12</sup><http://www.anzhi.com/>

<sup>13</sup><https://play.google.com/store>

TABLE 6. The details of support tools for stat analysis.

Tool	Description	IR	Example
Flowdroid	A tool to extract the source and sink information	jimple	[50], [73]
Aapt	A tool to compile resource files into DEX	smali	[35], [46]
Baksmali	A DEX to smali translator	smali	[17], [30]
Soot	A Java/Android static analysis and optimization framework	jimple	[50], [85]
Apktool	A tool for reverse engineering in Android applications	smali	[36], [40]
Androguard	Reverse engineering, malware/goodware analysis of Android applications	dex_assembler	[62], [76]
Dex2jar	A DEX to Java bytecode translator	Java_class	[38], [39]
Dexdump	A disassembler for DEX files	dex_assembler	[46]
Dedexer	A disassembler for DEX files	dex_assembler	[72]

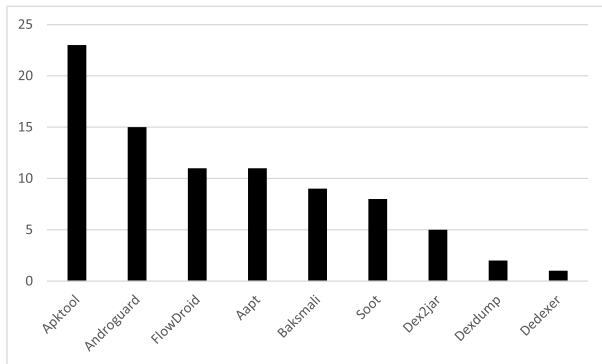
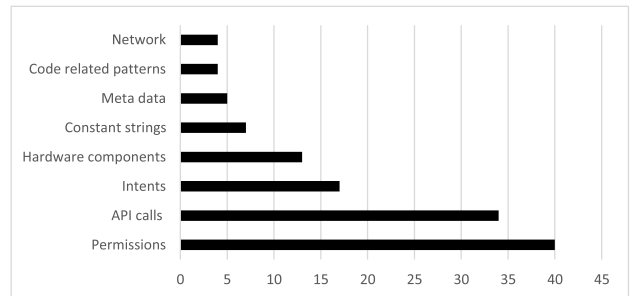


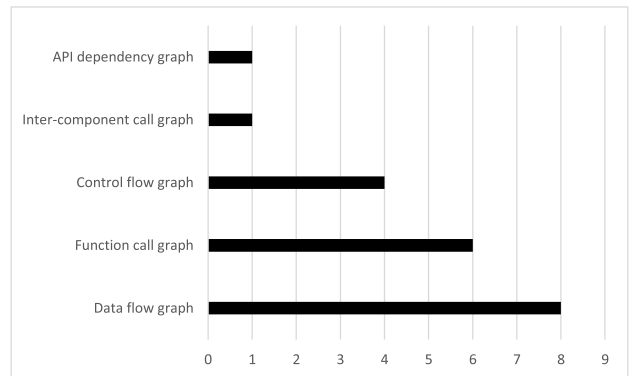
FIGURE 6. The results of support tools for static analysis.

malware, primary studies use a series of features to represent behaviours of APKs. According to four categories of static analysis techniques in RQ1, opcode-based method regards opcode sequences as text and opcode sequences can capture context semantics of APKs. Additionally, symbolic execution is a complex technique and is adopted by 2 studies, so this section focuses on Android characteristic and program graph-based methods to detect malware, the more details of features are listed in Figure 7.

Figure 7(a) presents that features related to Android characteristic-based method are divided into eight categories. In this figure, permissions and API calls are the most commonly used features in the primary studies, and the number of them is far more than that of other features. Permissions are security-related resources and are proactively granted by users during the process of installation. As for API calls, some of them are related to permissions and are restricted to Android permission system. For example, if you want to connect the network, you need to request the permission of ACCESS\_NETWORK\_STATE and call the API of getNetworkOperator(). Besides, some API calls are connected with sensitive data or resources such as getDeviceID(). In general, if an API call is sensitive, the frequencies of this API call in benign applications is less than that in malicious applications. It is noted that all the possible API calls in the static analysis can be extracted from binary files of the APK without running the APK, while API calls in the dynamic analysis are obtained by executing the APK in the protected environment such as Android emulators or sandboxes. Thus, API calls in the dynamic analysis are effective to resist malicious deformation



(a) The category of Android characteristic.



(b) The category of program graph.

FIGURE 7. The overview of features.

techniques, but dynamic analysis often misses some important API calls compared with static analysis. This figure also shows that the least used features are network (e.g., [36]) and code related patterns (e.g., [39]) in primary studies. The feature about the network is used to fetch data and files and may be related to the botnet, and code related patterns are collected by summarizing code patterns of benign and malicious APKs. In addition, the feature about meta data in this figure is the description information of released APKs in the application market such as ratings, downloads, developer reputation, and others.

Figure 7(b) shows five categories of program graph, including API dependency graph, inter-component call graph, control flow graph, function flow graph, and data flow graph. These program graphs can be obtained from support tools for static analysis. In the API dependency graph, the contextual API operations are nodes and data dependency between operations are edges [79]. The inter-component



call graph is formed by the control flow graph and data flow graph [78]. As for the control flow graph, basic blocks represent nodes and the connection between last executed basic block and currently active basic block represents edges. Concerning the function call graph, functions are regarded as nodes and calling relationships of functions are considered as edges, which are identified by invocation statements such as invoke. The data flow graph is different from the control flow graph, and it depicts the program by the procedure of data flow and data processing, and most static taint analysis techniques adopt the data flow graph. According to this figure, the number of data flow graph accounts for the largest proportion in the program graph-based methods.

RQ2.3: Overall, features related to Android characteristic-based method are the most, where permissions and API calls account for the most significant proportion.

4) RQ2.4: WHICH TECHNIQUES ARE USED FOR FEATURE REDUCTION IN ANDROID MALWARE DETECTION?

In order to determine and select significant features in Android malware detection, a variety of feature reduction techniques have been used in the primary studies. Feature reduction techniques are mainly divided into two categories: feature selection and feature extraction. Feature selection can eliminate redundant and noisy features, and select the most relevant features, such as information gain (IG). Feature extraction can reduce the dimensions of these origin features and obtain new combinations of features, such as principal component analysis (PCA). In primary studies, except for 4 SLRs, about 24% studies adopt off-the-shelf feature reduction techniques, which can be directly used in the studies. About 12% studies improve off-the-shelf feature reduction techniques and propose some novel feature reduction techniques such as [52]. Besides, about 17% studies automatically extract features from applications by the neural network, where word embedding technique is the most common [32], [74]. For example, Wang *et al.* [39] extract seven categories of static features, of which the number is up to 34 thousand. To process these features efficiently, deep autoencoders (DAE) are applied to reconstruct the origin features. In addition, about 47% studies do not use feature reduction techniques, and these studies apply origin features directly to detect Android malware such as [31].

Table 7 gives descriptions and distributions of off-the-shelf feature reduction techniques. This table lists seven off-the-shelf feature reduction techniques, including IG,  $\chi^2$ , fisher-score (Fisher), GR, co-relation based feature selection (CFS), evolutionary algorithms (EA), and PCA. According to this table, the most commonly used feature reduction technique is IG, and IG of feature  $f$  is defined as the difference between prior uncertainty and expected posterior uncertainty using feature  $f$ . If IG of feature  $f_1$  is higher than that of feature  $f_2$ , then feature  $f_1$  is considered better than feature  $f_2$ . To find

TABLE 7. The information of feature reduction techniques.

Name	Description	No.	Example
IG	It computes each feature brings information to the label.	12	[17]
$\chi^2$	It calculates the correlation of each feature to the label.	4	[42]
Fisher	It measures the ability of features to distinguish between malicious and benign applications.	2	[28]
GR	It reduces bias towards features with high information gain value on the basis of IG.	2	[45]
CFS	It measures the linear correlation among features.	2	[59]
EA	It assigns suitable weights to the features to ensure the best Android malware detection performance.	2	[58]
PCA	It maps n-dimensional features to k-dimensional features and reconstructs the k-dimensional features based on the original n-dimensional features.	1	[29]

the most suitable feature reduction technique, it is noticed that some studies make the comparison among different techniques. For example, to find a suitable feature reduction techniques, Martín *et al.* [45] compare the performance of IG, Chi-square ( $\chi^2$ ), and gain ratio (GR).

RQ2.4: Only a part of the studies adopt feature reduction techniques, and IG accounts for the largest proportion in the off-the-shelf feature reduction techniques.

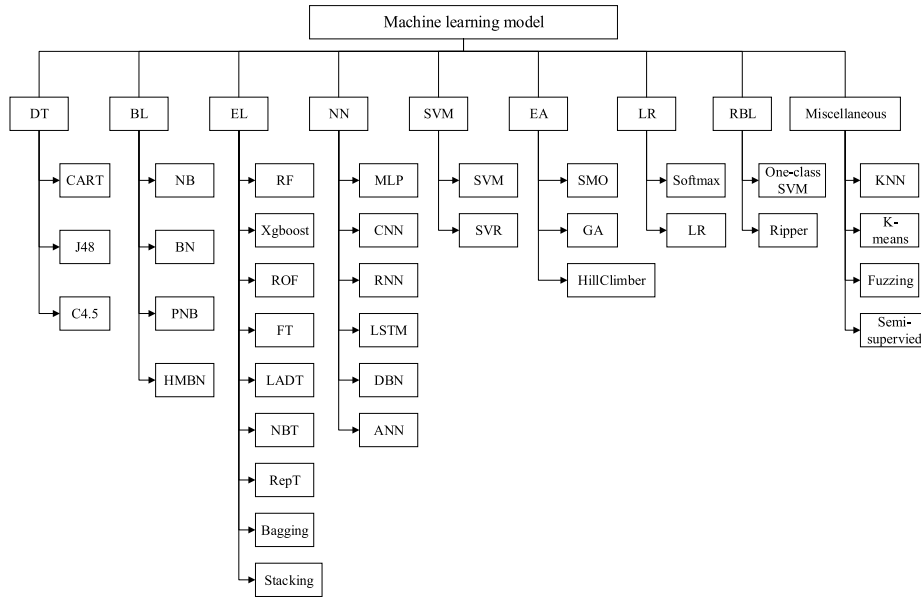
5) RQ2.5: WHICH MODELS ARE USED FOR ANDROID MALWARE DETECTION?

This section presents the used models in Android malware detection. We divided the used models into two categories: machine learning model and statistical model.

*a: MACHINE LEARNING MODEL*

With the rapid development of machine learning techniques in natural language processing, image detection, and other aspects, machine learning techniques have been gradually widely applied to Android malware detection in recent years. In the primary studies, the commonly used machine learning models are Support Vector Machine (SVM), Naive Bayes (NB), Logistic Regression (LR), Ensemble Learning (EL), neural network.

The neural network is a new area of machine learning and widely applied to many research communities such as image recognition and detection. It can also perform well in Android malware detection. Xiao *et al.* [44] propose to associate API calls from static analysis of Android applications and identify malware using Artificial Neural network (ANN). The evaluation result presents a high F-measure. Wang *et al.* [39] use deep autoencoder (DAE) as a pre-training method of Convolutional Neural Networks (CNN). With the combination of DAE and CNN (DAE-CNN), this method can learn more flexible patterns in a short time. Compared with SVM, accuracy with DAE-CNN is improved by 5%.



**FIGURE 8.** The classification of machine learning models. (CART: classification and regression trees, NB: naive bayes, BN: bayesian networks, PNB: naive bayes with informative priors, HMBN: hierarchical mixture of naive bayes, RF: random forest, ROF: rotation forest, FT: functional tree, LADT: multiclass alternating decision tree, NBT: naive bayes tree, RepT: fast decision tree learner, MLP: multilayer perceptron, CNN: convolutional neural network, RNN: recurrent neural network, LSTM: long short-term memory, DBN: deep belief network, ANN: artificial neural network, SVM: support vector machine, SVR: support vector regression, GA: generic algorithm, SMO: sequential minimal optimization, LR: logistic regression, KNN: K-nearest neighbour.)

*b: STATISTICAL MODEL*

The statistical model in Android malware detection is to build a rule library from malicious applications. Then we calculate the degree of similarity between an unknown APK and this rule library. If the degree of similarity is more than a certain threshold, this APK is regarded as a malicious APK. Conversely, if the degree of similarity is less than a certain threshold, this APK is regarded as a benign APK. For example, Ali-Gombe *et al.* [70] predefine a rule library based on extracting permissions and opcode sequences found in sensitive functional modules, and then apply this rule library to detect malware.

Except for 4 SLRs, we find that most studies adopt machine learning model, and the number of studies related to machine learning model is 82, while the number of studies related to statistical model is 12, which only takes up nearly 13%. It indicates that the machine learning model has been an important focus in Android malware detection recently.

Furthermore, we carry out classification in the field of the machine learning model. The classification strategy category is based on Malhotra [87], and machine learning model is divided into nine categories: Decision Trees (DT), Bayesian Learning (BL), Ensemble Learning (EL), Neural Networks (NN), Support Vector Machines (SVM), Evolutionary Algorithms (EA), Logistic Regression (LR), Rule-based Learning (RBL), and Miscellaneous. Figure 8 shows more detailed categories. According to this figure, EL has the most categories. Table 8 presents the number and percentage of the studies with respect to corresponding categories. It is found

**TABLE 8.** The distribution of studies related to machine learning model.

Model	No.	PCT
EL	41	22%
SVM	35	19%
NN	31	17%
Miscellaneous	21	11%
BL	18	10%
DT	18	10%
LR	13	7%
EA	5	3%
RBL	3	2%

that the three most commonly used models are ensemble learning with taking up 22%, SVM with accounting for 19%, and neural network with occupying 17%, respectively. The further detailed distribution in the machine learning model is presented in Figure 9, and the most widely applied models in corresponding categories are shown in the pie charts. For example, RF takes up the largest proportion in the field of EL, CNN occupies the largest proportion in the field of neural network.

RQ2.5: The most widely used model is machine learning model in Android malware detection, where ensemble learning takes up the largest proportion.

6) RQ2.6: WHICH PERFORMANCE MEASURES ARE USED FOR ANDROID MALWARE DETECTION USING STATIC ANALYSIS?

The performance measures are used for evaluating the generalization ability of models in Android malware detection.

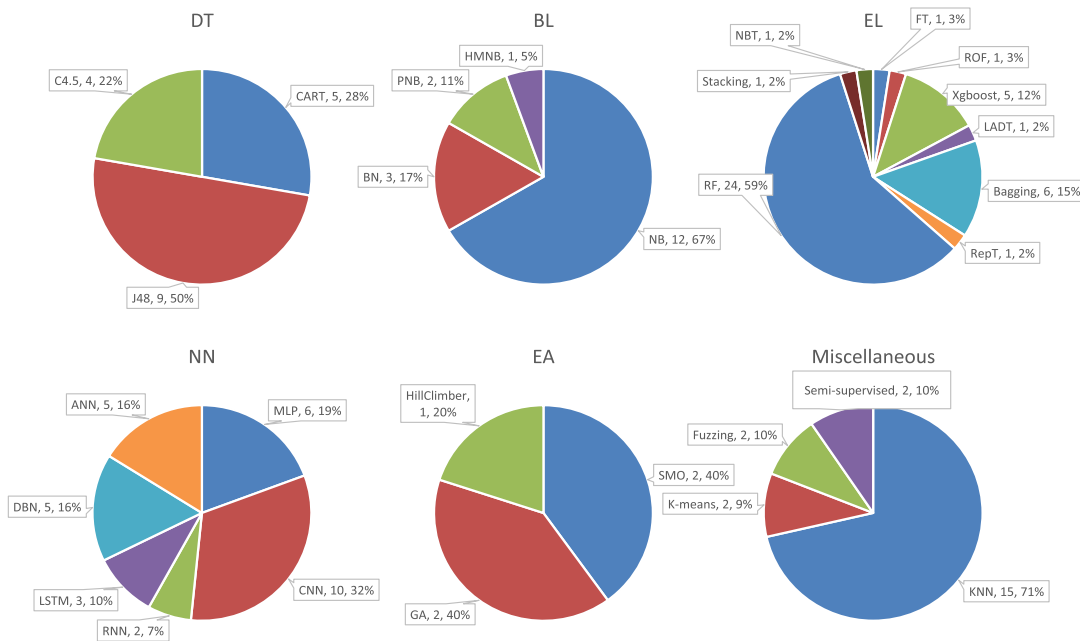


FIGURE 9. The distribution of subcategories with DT, BL, EL, NN, EA, and Miscellaneous.

TABLE 9. The results of performance measures.

Performance measure	Definition	No.
Accuracy	$Accuracy = (TN + TP) / (TN + FN + FP + TP)$	59
Precision	$Precision = TP / (TP + FP)$	39
Recall (TPR/Sensibility)	$Recall = TP / (TP + FN)$	37
False Positive Rate (FPR)	$FPR = FP / (FP + TN)$	34
F-measure (F1)	$F1 = 2 * Recall * Precision / (Recall + Precision)$	33
False Negative Rate (FNR)	$FNR = FN / (TP + FN)$	12
Area Under Curve (AUC)	The area under the ROC curve	11
Receiver Operating Characteristic (ROC)	ROC (receiver operating characteristic curve) is plotted with recall values on the y-axis and the 1-specificity values on the x-axis	9
Correlation coefficient	Jaccard, MCC, Standard deviation, P-value	6
Error rate	$Error\ rate = (FP + FN) / (TN + FN + FP + TP)$	4
Specificity (TNR)	$Specificity = 1 - FP$	4

Table 9 presents the definition of performance measures and the number of studies related to performance measures. In this table, true positive (TP), true negative (TN), false positive (FP), and false negative (FN) is applied to describe the definition of performance measures, where TP is the number of benign applications accurately predicted as benign applications, TN is the number of benign applications falsely predicted as malware, FP is the number of malware falsely predicted as benign applications, and FN is the number of malware accurately predicted as malware, respectively. As for the FPR, FNR, and error rate, the lower the value of the performance measures, the better the generalization ability of models. Correlation coefficient mainly includes Jaccard, MCC (Matthews correlation coefficient), standard deviation, and p-value in the primary studied, such as [19] and [68]. Apart from the aforementioned performance measures, the higher values of performance measures mean the better generalization ability of models, such as accuracy

and F-measure. As shown in this table, the most commonly used performance measure is accuracy, and the number of studies using precision and recall has a similar proportion.

RQ2.6: Accuracy is the most commonly used performance measure in Android malware detection.

**D. RQ3: WHAT IS THE OVERALL PERFORMANCE OF STATIC ANALYSIS TECHNIQUES IN ANDROID MALWARE DETECTION BASED ON EMPIRICAL EVIDENCE?**

This section aims to assess the effectiveness of static analysis techniques in Android malware detection. Due to some studies combining multiple datasets into a whole dataset, we only record values of performance measures on the separate dataset, and these values are presented in an excel.<sup>14</sup> Then we count the number of studies using the same static

<sup>14</sup><http://mrw.so/6y27U7>

analysis technique and choose accuracy, precision, and recall as evaluation criteria, which are the three most commonly used performance measures in RQ2.6. Finally, we calculate the minimum, maximum, mean, and standard deviation of three selected performance measures in the studies, which use the same static analysis technique.

TABLE 10. The performance of static analysis techniques.

Category	Performance measure	No.	Min	Max	Mean	Std
Android characteristic	Accuracy	25	88.26%	99.82%	96.47%	3.00
	Precision	13	89.16%	99.85%	95.68%	3.99
	Recall	18	88.40%	100%	95.54%	3.90
Opcode	Accuracy	5	93.60%	99.68%	97.58%	2.36
	Precision	7	92.30%	99.00%	96.24%	2.64
	Recall	7	91.00%	99.80%	96.20%	3.43
Program graph	Accuracy	7	86.90%	99.16%	95.83%	4.59
	Precision	5	88.20%	99.30%	95.47%	4.52
	Recall	7	89.00%	100%	95.65%	4.50
Symbolic execution	Accuracy	2	90.71%	97.20%	93.96%	3.17
	Precision	2	91.67%	95.23%	93.45%	2.52
	Recall	2	90.41%	96.37%	93.39%	4.21

Table 10 presents the number of studies (No.) and minimum (Min.), maximum (Max.), mean (Mean.), and standard deviation (Std.) of accuracy, precision, and recall. As for mean values, the performance of the opcode-based method is the best with 97.58% in the accuracy, 96.24% in the precision, and 96.20% in the recall. The Android characteristic and program graph-based methods have a similar performance. As shown in Table 10, the overall performance is more than 88.16%, and the standard deviation ranges from 2.36 to 4.59 in four static analysis techniques. Overall, it suggests that static analysis techniques are effective to detect Android malware.

RQ3: Overall, the results show that it is effective for static analysis techniques to detect Android malware.

**E. RQ4: WHETHER THE PERFORMANCE OF MODEL A IS BETTER THAN MODEL B IN ANDROID MALWARE DETECTION BASED ON EMPIRICAL EVIDENCE?**

We explore this research question based on the information from RQ3. By analyzing and comparing, it is found that in the case of ignoring the sample size of the dataset, it exists that Model A is better than Model B. Furthermore, we divide these models into two categories, which include neural network model and non-neural network model. Then we calculate the average accuracy, precision, and recall of these two models on the same dataset. Finally, after comparing the performance between two models, we reached a preliminary result that neural network model outperforms non-neural network model.

Figure 10 shows the performance of Android characteristic-based method on Drebin, Genome, Virusshare, and VirusTotal (Other results of static analysis techniques are shown in the link of RQ3). Concretely, except for the accuracy on Genome, the performance of neural network model is better than that of non-neural network model. Especially on

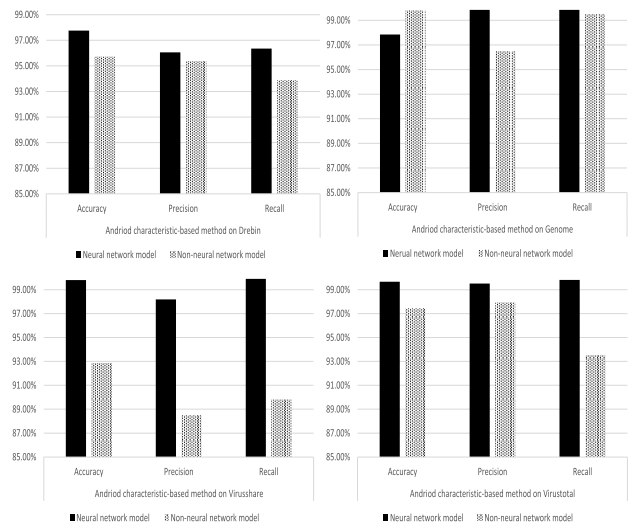


FIGURE 10. The overall performance of Android characteristic-based method on Drebin, Genome, Virusshare, and VirusTotal.

Virusshare, the precision of neural network model obviously exceeds that of non-neural network model. Although the accuracy of neural network model is better than that of non-neural network model on Genome, the accuracy between neural network model and non-neural network model is close. According to this figure, the overall performance of neural network model is better than that of non-neural network model.

RQ4: The results show that it exists that Model A is better than Model B, where neural network model outperforms non-neural network model.

**IV. DISCUSSIONS**

Based on the investigation above, static analysis is a reliable technique for Android malware detection. However, according to RQ3, it is found that static analysis techniques can detect malware effectively, but some studies have low performance such as [11] and [29]. It suggests that static analysis is still not sound and complete for malware detection. To improve Android malware detection using static analysis, this section constitutes several discussions around the results of research questions.

**A. COMBINING MULTIPLE STATIC ANALYSIS TECHNIQUES COULD BE ESSENTIAL TO HIGHLY PRECISE ANDROID MALWARE DETECTION.**

Static analysis techniques in this paper extract different features of applications, which can capture the information of APKs at different levels. Android characteristic-based method extracts the information of configuration as features such as permissions and intents, and these features are manually designed and are generally coarse-grained. The opcode-based method can capture contextual semantics of applications, but it missed structural semantics of APKs.

The program graph-based method can extract structural semantics. Moreover, the program graphs, especially control flow graph, are capable of resisting code obfuscation and encryption [115]. The symbolic execution-based method can obtain the abstract representations of executed paths in the applications. In general, these static analysis techniques can detect malware to some extent. For example, Yuan *et al.* [32] apply permissions and API calls as features, and the evaluation result shows this method is suitable for identifying Android malware. McLaughlin *et al.* [66] learn contextual semantics from opcode sequences through CNN, and the evaluation result shows high performance in Android malware detection. Allix *et al.* [38] present the effectiveness of the control flow graph in classifying malware.

To some extent, these static analysis techniques are complementary. In practice, combining multiple static analysis techniques can have more promising results in comparison with applying one static analysis technique. As an example, DroidEnsemble [82] combines Android characteristic-based method, including permissions, hardware features, intents, API calls, code patterns, as well as program graph-based method like function call graph, then applies SVM, KNN and RF to evaluate the performance of these two types of features. The experimental result shows that using two types of features has more potential results than using one type of features. However, according to RQ1, we find most studies apply one static analysis technique in Android malware detection. Thus, for further improvements in Android malware detection, it is essential to explore the complementarity of static analysis techniques and reasonably combine multiple static analysis techniques.

### **B. APPLYING SUITABLE FEATURE REDUCTION TECHNIQUES TO DIFFERENT FEATURES IS NECESSARY TO IMPROVE THE PERFORMANCE OF ANDROID MALWARE DETECTION.**

Feature reduction techniques can select significant features and improve the performance of Android malware detection. However, as shown in RQ2.3, there is still a part of studies which don't use feature reduction techniques in the primary studies. These studies can be divided into two categories. First, some studies such as [55] mainly adopt statistical models to detect Android malware, it is not necessary for these studies to use feature reduction techniques. In fact, these studies take up a small proportion in the whole primary studies. Second, to improve the performance of Android malware detection, some studies which don't use feature reduction techniques should adopt feature reduction techniques. For example, Wang *et al.* [40] extract a series of features (e.g., permissions and API calls) and apply deep belief network (DBN) to these features. DBN is generally composed of Restricted Boltzmann Machine (RBM), which can pre-train these features. Compared with the method that directly uses origin features for Android malware detection in [35], the experimental result shows that the method of applying DBN to these features can achieve better performance.

Therefore, there is still a need to apply feature reduction techniques to form some more precise methods of Android malware detection.

According to RQ2.3, we find that there are some studies using feature reduction techniques. Yet, these techniques have differences in primary studies. Recently, Morales-Ortega *et al.* [34] apply a series of feature reduction techniques (e.g.,  $\chi^2$ , relief, and IG) on permissions and hardware components. Based on the classification results, RF with relief achieves better performance than RF with  $\chi^2$  and IG. Bhattacharya and Goswami [42] propose a novel feature reduction technique based on rough set quick reduct algorithm with community detection scheme and carry on the experiments on permissions. Compared with the off-the-shelf feature reduction techniques (e.g., IG and GR), the proposed method shows more promising classification accuracy.

According to these points, we find that it is important to adopt feature reduction techniques in Android malware detection. Furthermore, to improve the performance of Android malware detection methods, it is necessary to apply suitable feature reduction techniques to different features.

### **C. USING NEURAL NETWORK TO DETECT ANDROID MALWARE USING STATIC ANALYSIS IS RECOMMENDED**

In recent years, Compared to the statistical model, we find that it has been a hot topic to apply machine learning model to detect Android malware from RQ2.5. Particularly, we also observe that most studies adopt non-neural network model to identify malware in the field of the machine learning model. However, it is not still satisfying to apply non-neural network model to Android malware detection. For example, Morales-Ortega *et al.* [34] encode permissions and hardware components by one-hot encoding, but the evaluation result on ensemble learning still exists a high FPR. Canfora *et al.* [11] convert opcode sequences into n-grams and take the frequencies of n-grams as features. Thereafter, machine learning models are trained to detect malware. Yet, the best result on RF presents that the value of AUC is only 72.7%.

According to the results of RQ4, we find that neural network model achieves better performance than non-neural network model in the field of the machine learning model. A study in [39] shows CNN is better than some machine learning models such as RF on Android characteristic-based method. And Yousefi-Azar *et al.* [67] also present DNN outperforms SVM and KNN on the opcode-based method.

Moreover, to bridge the gap between semantic representations of APKs and Android malware detection, neural network is applied to fill this gap in some studies to some extent. For example, Yan *et al.* [18] convert opcode sequences into continuous vectors by opcode embedding and apply LSTM to learn distribution representations of contextual semantics from these opcode sequences for malware detection. The evaluation result presents this method has better performance than n-gram-based malware detection. Narayanan *et al.* [89] extract control flow graphs (CFGs) from APKs, and get rooted subgraphs from these CFGs by graph kernels.

Then neural network is leveraged to learn the distribution representations of structural semantics from these rooted sub-graphs. The experimental result shows this method outperforms Android characteristic-based method [35].

In summary, neural network is a potential model in the field of Android malware detection. In the case of continuously promoting the robustness of neural network, it is recommended to apply neural network to detect Android malware using static analysis.

#### **D. ESTABLISHING A UNIFIED PLATFORM IS NECESSARY TO ASSESS THE PERFORMANCE OF DIFFERENT TECHNIQUES IN ANDROID MALWARE DETECTION FAIRLY**

In the process of analyzing the primary studies, we find that there are some biases in the results of RQ3 and RQ4. The main reason is the inconsistency of datasets. First, different studies adopt different datasets. Some studies use in-the-lab datasets, some studies use in-the-wild datasets, and a few studies use unpublic datasets, such as [25], [26]. Second, some studies which use the same dataset are still some inconsistencies. (1) The performance measures used in these studies are not unified. For example, Arp *et al.* [35] adopt the accuracy and ROC to evaluate the performance of Android malware detection, and Grosse *et al.* [47] use FNR as performance measure; (2) The size of the datasets in these studies is inconsistent. Chen *et al.* [56] adopt more than 10000 samples, while the size of the dataset is about 1000 in [60]; (3) The benign applications in the primary studies are rarely public. To some extent, these situations above can reduce the reliability of these results in RQ3 and RQ4.

In addition, the experimental results of some studies are inconsistent. For example, to train an effective classifier to distinguish between malicious and benign applications, some studies adopt n-gram features which collect from opcode sequences. The best accuracy of classification can be obtained by 2-gram in [11]. However, Jerome *et al.* [17] present that 5-gram can achieve the best result in terms of malware classification. Hence, it indicates that the selection of parameter n for n-gram can have an impact on the accuracy of malware classifiers.

According to these points above, it is necessary to establish a unified platform and assess the performance of techniques proposed by different researchers and practitioners fairly.

#### **V. LIMITATIONS**

The main threats to validity in this paper include construct validity, internal validity, and external validity.

Construct validity is about the collection of studies. In this paper, we try our best to collect the relevant studies from journals and conferences of seven electronic databases as much as possible, but some of the relevant publications may be still missing in our collected studies. The additional aspect of the construct validity is that we are likely to have a few errors in the process filtering out the studies by inclusion or exclusion criteria. To further avoid these errors, we analyze the list

of publications by cross-checking method in the primary studies.

Internal validity is related to data extraction and data analysis. There is a heavy workload in the data extraction and data analysis, so our data are also collected by cross-checking method, and the terminal data are obtained after we make an agreement on the results of the comparison. However, we are still likely to make a few mistakes in the process of extracting and analyzing data. Furthermore, in order to reduce mistakes, these data should be verified by original authors in the primary studies.

External validity is about the summary of the results obtained from the primary studies. There is a threat to the validity of the results in RQ3 and RQ4. Due to the inconsistencies of studies used for comparison, it may not lead to definitely conclusive results. Thus, we propose to establish a unified platform to reduce the inconsistencies in the primary studies. Moreover, more studies related to Android malware detection using static analysis should be collected in order to obtain definite and generalized results.

#### **VI. CONCLUSION AND FUTURE WORK**

This paper summarizes the state-of-the-art techniques and provides a comprehensive overview of Android malware detection using static analysis. Concretely, this SLR is performed by 98 studies from 2014 to 2020. And this SLR investigates the categories of static analysis techniques, the process of conducting empirical experiments, the malware detection capability of static analysis techniques, and the performance of different models in Android malware detection. Based on the results in the primary studies, we make the discussions and implications about Android malware detection using static analysis.

According to this SLR, we find that (1) The most commonly used static analysis technique is Android characteristic in Android malware detection; (2) In the empirical experiments, in-the-lab datasets such as Drebin and Genome occupy the largest proportion. Most studies adopt Apktool as a support tool for static analysis. IG is the most commonly used feature reduction technique. Permissions and sensitive API calls are the most used features. Machine learning model takes up the largest proportion in all the used models. And accuracy is most applied performance measure; (3) The results of empirical evidence show static analysis techniques are effective to detect malware; (4) By analyzing and comparing primary studies, we draw a preliminary result that neural network model outperforms the non-neural network model.

Based on results from research questions, it is concluded from this SLR that Android malware detection using static analysis still faces some challenges. To alleviate this challenge, we propose some guidelines that developing some novel techniques improves the performance of Android malware detection and establishing a unified platform for assessing the performance of various techniques fairly. In the future, we will follow these guidelines to promote Android malware detection methods.

## REFERENCES

- [1] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party Android marketplaces," in *Proc. 2nd ACM Conf. Data Appl. Secur. Privacy*, 2012, pp. 317–326.
- [2] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 95–109.
- [3] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proc. 16th ACM Conf. Comput. Commun. Secur. (CCS)*, Chicago, IL, USA, 2009, pp. 235–245.
- [4] J. Kim, Y. Yoon, K. Yi, and J. Shin, "ScanDal: Static analyzer for detecting privacy leaks in Android applications," *MoST*, vol. 12, no. 110, p. 1, 2012.
- [5] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, and P. Mcdaniel, "IccTA: Detecting inter-component privacy leaks in Android apps," in *Proc. 37th Int. Conf. Softw. Eng. (ICSE)*, 2015, pp. 280–291.
- [6] S. Alam, I. Traore, and I. Sogukpinar, "Annotated control flow graph for metamorphic malware detection," *Comput. J.*, vol. 58, no. 10, pp. 2608–2621, Oct. 2015.
- [7] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan, "Android security: A survey of issues, malware penetration, and defenses," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 2, pp. 998–1022, 2nd Quart., 2015.
- [8] A. Calleja, T. F. Bissyandé, and J. Caballero, "A look into 30 years of malware development from a software metrics perspective," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defense*, 2016, pp. 325–345.
- [9] R. Zachariah, K. Akash, M. S. Yousef, and A. Chacko, "Android malware detection a survey," in *Proc. IEEE Int. Conf. Circuits Syst. (ICCS)*, Dec. 2017, pp. 238–244.
- [10] L. Li, T. F. Bissyandé, M. Papadakis, S. Rasthofer, A. Bartel, D. Oceau, J. Klein, and L. Traon, "Static analysis of Android apps: A systematic literature review," *Inf. Softw. Technol.*, vol. 88, pp. 67–95, Aug. 2017.
- [11] G. Canfora, A. D. Lorenzo, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Effectiveness of Opcode ngrams for detection of multi family Android malware," in *Proc. 10th Int. Conf. Availability, Rel. Secur.*, Aug. 2015, pp. 333–340.
- [12] D. Budgen and P. Brereton, "Performing systematic literature reviews in software engineering," in *Proc. 28th Int. Conf. Softw. Eng. (ICSE)*, 2006, pp. 1051–1052.
- [13] C. K. Behera, G. Sanjog, and D. L. Bhaskari, "Control flow graph matching for detecting obfuscated programs," in *Proc. Softw. Eng.*, 2019, pp. 267–275.
- [14] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Softw. Eng.*, vol. 10, no. 4, pp. 405–435, Oct. 2005.
- [15] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, "AndroDialysis: Analysis of Android intent effectiveness in malware detection," *Comput. Secur.*, vol. 65, pp. 121–134, Mar. 2017.
- [16] S. Y. Yerima, I. Muttik, and S. Sezer, "High accuracy Android malware detection using ensemble learning," *IET Inf. Secur.*, vol. 9, no. 6, pp. 313–320, Nov. 2015.
- [17] Q. Jerome, K. Allix, R. State, and T. Engel, "Using opcode-sequences to detect malicious Android applications," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2014, pp. 313–320.
- [18] J. Yan, Y. Qi, and Q. Rao, "LSTM-based hierarchical denoising network for Android malware detection," *Secur. Commun. Netw.*, vol. 2018, pp. 1–18, Jan. 2018.
- [19] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Oceau, and P. McDaniel, "FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," *ACM SIGPLAN Notices*, vol. 49, no. 6, pp. 259–269, Jun. 2014.
- [20] M. Fan, J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zheng, and T. Liu, "Android malware familial classification and representative sample selection via frequent subgraph analysis," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 8, pp. 1890–1905, Aug. 2018.
- [21] S. Arzt, S. Rasthofer, R. Hahn, and E. Bodden, "Using targeted symbolic execution for reducing false-positives in dataflow analysis," in *Proc. 4th ACM SIGPLAN Int. Workshop State Art Program Anal. (SOAP)*, 2015, pp. 1–6.
- [22] S. Ni, Q. Qian, and R. Zhang, "Malware identification using visualization images and deep learning," *Comput. Secur.*, vol. 77, pp. 871–885, Aug. 2018.
- [23] J.-Y. Kim, S.-J. Bu, and S.-B. Cho, "Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders," *Inf. Sci.*, vols. 460–461, pp. 83–102, Sep. 2018.
- [24] W. Wang, Y. Li, X. Wang, J. Liu, and X. Zhang, "Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers," *Future Gener. Comput. Syst.*, vol. 78, pp. 987–994, Jan. 2018.
- [25] F. Dong, Y. Li, X. Wang, J. Liu, and X. Zhang, "Defect prediction in Android binary executables using deep neural network," *Wireless Pers. Commun.*, vol. 102, no. 3, pp. 987–994, 2018.
- [26] D. Galligani and R. Gjomemo, "Static detection and automatic exploitation of intent message vulnerabilities in Android applications," Polytechnic Univ. Milan, Milan, Italy, 2015, pp. 1–11.
- [27] Z.-U. Rehman, S. N. Khan, K. Muhammad, J. W. Lee, Z. Lv, S. W. Baik, P. A. Shah, K. Awan, and I. Mehmood, "Machine learning-assisted signature and heuristic-based detection of malwares in Android devices," *Comput. Electr. Eng.*, vol. 69, pp. 828–841, Jul. 2018.
- [28] J. Yu, Q. Huang, and C. Yian, "DroidScreening: A practical framework for real-world Android malware analysis," *Secur. Commun. Netw.*, vol. 9, no. 11, pp. 1435–1449, 2016.
- [29] H.-J. Zhu, Z.-H. You, Z.-X. Zhu, W.-L. Shi, X. Chen, and L. Cheng, "DroidDet: Effective and robust detection of Android malware using static analysis along with rotation forest model," *Neurocomputing*, vol. 272, pp. 638–646, Jan. 2018.
- [30] S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song, and H. Yu, "SAMADroid: A novel 3-level hybrid malware detection model for Android operating system," *IEEE Access*, vol. 6, pp. 4321–4339, 2018.
- [31] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and efficient behavior-based Android malware detection and prevention," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 1, pp. 83–97, Jan. 2018.
- [32] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: Android malware characterization and detection using deep learning," *Tsinghua Sci. Technol.*, vol. 21, no. 1, pp. 114–123, Feb. 2016.
- [33] T. Ban, T. Takahashi, S. Guo, D. Inoue, and K. Nakao, "Integration of multi-modal features for Android malware detection using linear SVM," in *Proc. 11th Asia Joint Conf. Inf. Secur. (AsiaJIS)*, Aug. 2016, pp. 141–146.
- [34] S. Morales-Ortega, P. J. Escamilla-Ambrosio, A. Rodriguez-Mota, and L. D. Coronado-De-Alba, "Native malware detection in smartphones with Android OS using static analysis, feature selection and ensemble classifiers," in *Proc. 11th Int. Conf. Malicious Unwanted Softw. (MALWARE)*, Oct. 2016, pp. 1–8.
- [35] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of Android malware in your pocket," in *Proc. NDSS*, vol. 14, 2014, pp. 23–26.
- [36] X. Wang, D. Zhang, X. Su, and W. Li, "Mlifdect: Android malware detection based on parallel machine learning and information fusion," *Secur. Commun. Netw.*, vol. 2017, pp. 1–14, Aug. 2017.
- [37] J. Garcia, M. Hammad, and S. Malek, "Lightweight, obfuscation-resilient detection and family identification of Android malware," *ACM Trans. Softw. Eng. Methodol.*, vol. 26, no. 3, p. 11, 2016.
- [38] K. Allix, T. F. Bissyandé, Q. Jérôme, J. Klein, R. State, and Y. Le Traon, "Empirical assessment of machine learning-based malware detectors for Android," *Empirical Softw. Eng.*, vol. 21, no. 1, pp. 183–211, Feb. 2016.
- [39] W. Wang, M. Zhao, and J. Wang, "Effective Android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," *J. Ambient Intell. Hum. Comput.*, vol. 10, pp. 3035–3043, Apr. 2018.
- [40] Z. Wang, J. Cai, S. Cheng, and W. Li, "DroidDeepLearner: Identifying Android malware using deep learning," in *Proc. IEEE 37th Sarnoff Symp.*, Sep. 2016, pp. 160–165.
- [41] A. Firdaus, N. B. Anuar, A. Karim, and M. F. A. Razak, "Discovering optimal features using static analysis and a genetic search based method for Android malware detection," *Frontiers Inf. Technol. Electron. Eng.*, vol. 19, no. 6, pp. 712–736, Jun. 2018.
- [42] A. Bhattacharya and R. T. Goswami, "A hybrid community based rough set feature selection technique in Android malware detection," in *Proc. Smart Trends Syst., Secur. Sustainability*, 2018, pp. 249–258.
- [43] X. Yang, D. Lo, L. Li, X. Xia, T. F. Bissyandé, and J. Klein, "Characterizing malicious Android apps by mining topic-specific data flow signatures," *Inf. Softw. Technol.*, vol. 90, pp. 27–39, Oct. 2017.

- [44] X. Xiao, Z. Wang, Q. Li, S. Xia, and Y. Jiang, "Back-propagation neural network on Markov chains from system call sequences: A new approach for detecting Android malware with system call sequences," *IET Inf. Secur.*, vol. 11, no. 1, pp. 8–15, Jan. 2017.
- [45] I. Martín, J. A. Hernández, A. Muñoz, and A. Guzmán, "Android malware characterization using metadata and machine learning techniques," *Secur. Commun. Netw.*, vol. 2018, pp. 1–11, Jul. 2018.
- [46] W. Chen, D. Aspinall, A. D. Gordon, C. Sutton, and I. Muttik, "More semantics more robust: Improving Android malware classifiers," in *Proc. 9th ACM Conf. Secur. Privacy Wireless Mobile Netw. (WiSec)*, 2016, pp. 147–158.
- [47] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial examples for malware detection," in *Proc. Eur. Symp. Res. Comput. Secur.*, Sep. 2017, pp. 62–79.
- [48] S. Sen, E. Aydogan, and A. I. Aysan, "Coevolution of mobile malware and anti-malware," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 10, pp. 2563–2574, Oct. 2018.
- [49] K. Riad and L. Ke, "RoughDroid: Operative scheme for functional Android malware detection," *Secur. Commun. Netw.*, vol. 2018, pp. 1–10, Sep. 2018.
- [50] W. Yang, D. Kong, T. Xie, and C. A. Gunter, "Malware detection in adversarial settings: Exploiting feature evolutions and confusions in Android apps," in *Proc. 33rd Annu. Comput. Secur. Appl. Conf.*, Dec. 2017, pp. 288–302.
- [51] L. Sun, Z. Li, Q. Yan, W. Srisa-An, and Y. Pan, "SigPID: Significant permission identification for Android malware detection," in *Proc. 11th Int. Conf. Malicious Unwanted Softw. (MALWARE)*, Oct. 2016, pp. 1–8.
- [52] L. Chen, S. Hou, and Y. Ye, "SecureDroid: Enhancing security of machine learning-based detection against adversarial Android malware attacks," in *Proc. 33rd Annu. Comput. Secur. Appl. Conf.*, Dec. 2017, pp. 362–372.
- [53] F. Idrees, M. Rajarajan, M. Conti, T. M. Chen, and Y. Rahulamathavan, "PIndroid: A novel Android malware detection system using ensemble learning methods," *Comput. Secur.*, vol. 68, pp. 36–46, Jul. 2017.
- [54] P. Palumbo, L. Sayfullina, D. Komashinskiy, E. Eirola, and J. Karhunen, "A pragmatic Android malware detection procedure," *Comput. Secur.*, vol. 70, pp. 689–701, Sep. 2017.
- [55] Q. Li, B. Sun, M. Chen, and H. Dong, "Detection malicious Android application based on simple-Dalvik intermediate language," *Neural Comput. Appl.*, vol. 31, no. S1, pp. 185–194, Jan. 2019.
- [56] S. Chen, M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu, and B. Li, "Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach," *Comput. Secur.*, vol. 73, no. 1, pp. 326–344, Mar. 2018.
- [57] L. Cen, C. S. Gates, L. Si, and N. Li, "A probabilistic discriminative model for Android malware detection with decompiled source code," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 4, pp. 400–412, Jul. 2015.
- [58] Y. Xu, C. Wu, K. Zheng, X. Wang, X. Niu, and T. Lu, "Computing adaptive feature weights with PSO to improve Android malware detection," *Secur. Commun. Netw.*, vol. 2017, pp. 1–14, May 2017.
- [59] K. Xu, Y. Li, and R. H. Deng, "ICCDetector: ICC-based malware detection on Android," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 6, pp. 1252–1264, Jun. 2016.
- [60] D. Hu, Z. Ma, X. Zhang, P. Li, D. Ye, and B. Ling, "The concept drift problem in Android malware detection and its solution," *Secur. Commun. Netw.*, vol. 2017, pp. 1–13, Sep. 2017.
- [61] H.-J. Zhu, T.-H. Jiang, B. Ma, Z.-H. You, W.-L. Shi, and L. Cheng, "HEMD: A highly efficient random forest-based malware detection framework for Android," *Neural Comput. Appl.*, vol. 30, no. 11, pp. 3353–3361, Dec. 2018.
- [62] N. Xie, X. Wang, W. Wang, and J. Liu, "Fingerprinting Android malware families," *Frontiers Comput. Sci.*, vol. 13, no. 3, pp. 637–646, Jun. 2019.
- [63] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for Android malware detection using deep learning," *Digital Invest.*, vol. 24, pp. 48–59, Mar. 2018.
- [64] G. Tao, Z. Zheng, Z. Guo, and M. R. Lyu, "MalPat: Mining patterns of malicious and benign Android apps via permission-related APIs," *IEEE Trans. Rel.*, vol. 67, no. 1, pp. 355–369, Mar. 2018.
- [65] A. Azmoodeh, A. Dehghantanha, and K.-K.-R. Choo, "Robust malware detection for Internet of (battlefield) things devices using deep eigenspace learning," *IEEE Trans. Sustain. Comput.*, vol. 4, no. 1, pp. 88–95, Jan. 2019.
- [66] N. McLaughlin, J. M. del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, and G. J. Ahn, "Deep Android malware detection," in *Proc. 7th ACM Conf. Data Appl. Secur. Privacy*, Mar. 2017, vol. 4, no. 1, pp. 301–308.
- [67] M. Yousefi-Azar, L. G. C. Hamey, V. Varadharajan, and S. Chen, "Malytics: A malware detection scheme," *IEEE Access*, vol. 6, pp. 49418–49431, 2018.
- [68] M. White, C. Vendome, M. Linares-Vasquez, and D. Poshyvanyk, "Toward deep learning software repositories," in *Proc. IEEE/ACM 12th Work. Conf. Mining Softw. Repositories*, May 2015, pp. 334–345.
- [69] G. Canfora, F. Mercaldo, and C. A. Visaggio, "An HMM and structural entropy based detector for Android malware: An empirical study," *Comput. Secur.*, vol. 61, pp. 1–18, Aug. 2016.
- [70] A. Ali-Gombe, I. Ahmed, G. G. Richard, III, and V. Roussev, "Opseq: Android malware fingerprinting," in *Proc. 5th Program Protection Reverse Eng. Workshop*, Dec. 2016, pp. 1–12.
- [71] J. Bai and J. Wang, "Improving malware detection using multi-view ensemble learning," *Secur. Commun. Netw.*, vol. 9, no. 17, pp. 4227–4241, Nov. 2016.
- [72] J. Zhang, Z. Qin, K. Zhang, H. Yin, and J. Zou, "Dalvik opcode graph based Android malware variants detection using global topology features," *IEEE Access*, vol. 6, pp. 51964–51974, 2018.
- [73] J. Brown, M. Anwar, and J. Dozier, "An artificial immunity approach to malware detection in a mobile platform," *EURASIP J. Inf. Secur.*, vol. 2017, no. 1, 2017, Art. no. 7.
- [74] D. Zhu, H. Jin, Y. Yang, D. Wu, and W. Chen, "DeepFlow: Deep learning-based malware detection by mining Android application for abnormal usage of sensitive data," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2017, pp. 438–443.
- [75] M. H. Nguyen, D. L. Nguyen, X. M. Nguyen, and T. T. Quan, "Auto-detection of sophisticated malware using lazy-binding control flow graph and deep learning," *Comput. Secur.*, vol. 76, pp. 128–155, Jul. 2018.
- [76] Y. Liu, L. Zhang, and X. Huang, "Using g features to improve the efficiency of function call graph based Android malware detection," *Wireless Pers. Commun.*, vol. 103, no. 4, pp. 2947–2955, Dec. 2018.
- [77] M. Junaid, D. Liu, and D. Kung, "Dexteroid: Detecting malicious behaviors in Android apps using reverse-engineered life cycle models," *Comput. Secur.*, vol. 59, pp. 92–117, Jun. 2016.
- [78] Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: Semantics-based detection of Android malware through static analysis," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*, Nov. 2014, pp. 576–587.
- [79] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware Android malware classification using weighted contextual API dependency graphs," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Nov. 2014, pp. 1105–1116.
- [80] C.-M. Chen, J.-M. Lin, and G.-H. Lai, "Detecting mobile application malicious behaviors based on data flow of source code," in *Proc. Int. Conf. Trustworthy Syst. Appl.*, Jun. 2014, pp. 1–6.
- [81] F. Wei, S. Roy, X. Ou, and Robby, "Amandroid: A precise and general inter-component data flow analysis framework for security vetting of Android apps," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 1329–1341.
- [82] W. Wang, Z. Gao, M. Zhao, Y. Li, J. Liu, and X. Zhang, "DroidEnsemble: Detecting Android malicious applications with ensemble of string and structural static features," *IEEE Access*, vol. 6, pp. 31798–31807, 2018.
- [83] Y. Du, J. Wang, and Q. Li, "An Android malware detection approach using community structures of weighted function call graphs," *IEEE Access*, vol. 5, pp. 17478–17486, 2017.
- [84] L. Patrick, B. Eric, L. Ondrej, and H. Laurie, "The Soot framework for Java program analysis: A retrospective," in *Proc. Cetus Compiler Infrastructure Workshop*, 2011, pp. 15–35.
- [85] H. Jin, Z. Rongcai, S. Zhen, L. Fudong, Z. Bingling, M. Xi, and W. Hongyan, "Analyzing and recognizing Android malware via semantic-based malware gene," in *Proc. Int. Conf. Cyber-Enabled Distrib. Comput. Knowl. Discovery (CyberC)*, Oct. 2017, pp. 17–20.
- [86] A. Desnos and G. Gueguen, "Android: From reversing to decompilation," in *Proc. Black Hat Abu Dhabi*, 2011, pp. 77–101.
- [87] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Appl. Soft Comput.*, vol. 27, pp. 504–518, Feb. 2015.
- [88] A. Soury and R. Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques," *Hum.-Centric Comput. Inf. Sci.*, vol. 8, no. 1, pp. 1–22, Dec. 2018.



[89] A. Narayanan, G. Meng, L. Yang, J. Liu, and L. Chen, "Contextual Weisfeiler–Lehman graph kernel for malware detection," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2016, pp. 4701–4708.

[90] S. Yan, R. Wang, C. Salls, S. Nick, P. Mario, D. Audrey, G. John, F. Siji, H. Christophe, and V. Giovanmi, "The art of war: Offensive techniques in binary analysis," in *Proc. Secur. Privacy*, 2016, pp. 138–157.

[91] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A combination method for Android malware detection based on control flow graphs and machine learning algorithms," *IEEE Access*, vol. 7, pp. 21235–21245, 2019.

[92] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for Android malware detection using various features," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 3, pp. 773–788, Mar. 2019.

[93] B. Sun, T. Ban, S.-C. Chang, Y. S. Sun, T. Takahashi, and D. Inoue, "A scalable and accurate feature representation method for identifying malicious mobile applications," in *Proc. 34th ACM/SIGAPP Symp. Appl. Comput.*, Apr. 2019, pp. 1182–1189.

[94] L. Zhang, V. L. L. Thing, and Y. Cheng, "A scalable and extensible framework for Android malware detection and family attribution," *Comput. Secur.*, vol. 80, pp. 120–133, Jan. 2019.

[95] X. Ge, Y. Pan, Y. Fan, and C. Fang, "AMDroid: Android malware detection using function call graphs," in *Proc. IEEE 19th Int. Conf. Softw. Qual., Rel. Secur. Companion (QRS-C)*, Jul. 2019, pp. 71–77.

[96] H. Zhang, S. Luo, Y. Zhang, and L. Pan, "An efficient Android malware detection system based on method-level behavioral semantic analysis," *IEEE Access*, vol. 7, pp. 69246–69256, 2019.

[97] S. Turker and A. B. Can, "AndMFC: Android malware family classification framework," in *Proc. IEEE 30th Int. Symp. Pers., Indoor Mobile Radio Commun. (PIMRC Workshops)*, Sep. 2019, pp. 1–6.

[98] G. Shrivastava and P. Kumar, "Android application behavioural analysis for data leakage," *Expert Syst.*, vol. 36, Aug. 2019.

[99] L. Nguyen-Vu, J. Ahn, and S. Jung, "Android fragmentation in malware detection," *Comput. Secur.*, vol. 87, Nov. 2019, Art. no. 101573.

[100] Z. Wang, G. Li, Y. Chi, J. Zhang, T. Yang, and Q. Liu, "Android malware detection based on convolutional neural networks," in *Proc. 3rd Int. Conf. Comput. Sci. Appl. Eng. (CSAE)*, 2019, pp. 1–6.

[101] F. Shen, J. D. Vecchio, A. Mohaisen, S. Y. Ko, and L. Ziarek, "Android malware detection using complex-flows," *IEEE Trans. Mobile Comput.*, vol. 18, no. 6, pp. 1231–1245, Jun. 2019.

[102] Z. Xu, K. Ren, and F. Song, "Android malware family classification and characterization using CFG and DFG," in *Proc. Int. Symp. Theor. Aspects Softw. Eng. (TASE)*, Jul. 2019, pp. 49–56.

[103] S. Badhani and S. K. Muttoo, "CENDroid—A cluster-ensemble classifier for detecting malicious Android applications," *Comput. Secur.*, vol. 85, pp. 25–40, Aug. 2019.

[104] Z. Shan, R. Samuel, and I. Neamtiu, "Device administrator use and abuse in Android: Detection and characterization," in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, Oct. 2019, pp. 1–16.

[105] Z. Shan, R. Samuel, and I. Neamtiu, "Device administrator use and abuse in Android: Detection and characterization," in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, Oct. 2019, pp. 1–16.

[106] M. Fan, X. Luo, J. Liu, M. Wang, C. Nong, Q. Zheng, and T. Liu, "Graph embedding based familial analysis of Android malware using unsupervised learning," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. (ICSE)*, May 2019, pp. 771–782.

[107] A. Alotaibi, "Identifying malicious software using deep residual long-short term memory," *IEEE Access*, vol. 7, pp. 163128–163137, 2019.

[108] G. Canfora, F. Martinelli, F. Mercaldo, V. Nardone, A. Santone, and C. A. Visaggio, "LEILA: Formal tool for identifying mobile malicious behaviour," *IEEE Trans. Softw. Eng.*, vol. 45, no. 12, pp. 1230–1252, Dec. 2019.

[109] S. Y. Yerima and S. Khan, "Longitudinal performance analysis of machine learning based Android malware detectors," in *Proc. Int. Conf. Cyber Secur. Protection Digit. Services (Cyber Secur.)*, Jun. 2019, pp. 1–8.

[110] D. Li, L. Zhao, Q. Cheng, N. Lu, and W. Shi, "Opcode sequence analysis of Android malware by a convolutional neural network," *Concurrency Comput., Pract. Exper.*, May 2019, Art. no. e5308.

[111] J. McGiff, W. G. Hatcher, J. Nguyen, W. Yu, E. Blasch, and C. Lu, "Towards multimodal learning for Android malware detection," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Feb. 2019, pp. 432–436.

[112] S. Alam, S. A. Alharbi, and S. Yildirim, "Mining nested flow of dominant APIs for detecting Android malware," *Comput. Netw.*, vol. 167, pp. 1–10, Feb. 2020.

[113] R. Taheri, M. Ghahramani, R. Javidan, M. Shojafar, Z. Pooranian, and M. Conti, "Similarity-based Android malware detection using Hamming distance of static binary features," *Future Gener. Comput. Syst.*, vol. 105, pp. 230–247, Apr. 2020.

[114] M. Amin, T. A. Tanveer, M. Tehseen, M. Khan, F. A. Khan, and S. Anwar, "Static malware detection and attribution in Android byte-code through an end-to-end deep system," *Future Gener. Comput. Syst.*, vol. 102, pp. 112–126, Jan. 2020.

[115] M. Irshad, H. M. A. Khateeb, A. Mansour, M. Ashawa, and M. Hamisu, "Effective methods to detect metamorphic malware: A systematic review," *Int. J. Electron. Secur. Digit. Forensics*, vol. 10, no. 2, p. 138, 2018.



**YA PAN** received the master's degree from Chongqing University. From 2015 to 2016, she continued to study at Northeastern State University as a Visiting Scholar. She is currently an Associate Professor with the Department of Computer Science and Technology, Southwest University of Science and Technology. Her research interests include software testing and quality assurance technology.



**XIUTING GE** is currently pursuing the master's degree with the Southwest University of Science and Technology. Since 2018, she has been studying at Nanjing University as a Joint Student. She is a Major with the Software Institute. Her research interests include software testing and Android malware detection.



**CHUNRONG FANG** is currently a Research Assistant with the Software Institute, Nanjing University. He has served as the Program Co-Chair of AIST. He has published more than 20 papers in conferences and journals such as ICSE, FSE, ISSTA, ICST, TR, and *Software Quality Journal (SQJ)*. He has applied for more than ten patents and some of his achievements have been transformed by well-known software companies such as Baidu and Huawei. His research interest includes intelligent software engineering.



**YONG FAN** received the Ph.D. degree from Sichuan University, in 2001. From 1995 to 1998, he was a Research Assistant with the China Academy of Engineering Physics. He is currently a Professor with the Department of Computer Science and Technology, Southwest University of Science and Technology. His research interests include visual measurement technology, image analysis and understanding, and algorithm testing.

...