IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 17 February 2024, accepted 8 March 2024, date of publication 18 March 2024, date of current version 22 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3377658

SURVEY

Hybrid Android Malware Detection: A Review of Heuristic-Based Approach

RAJIF AGUNG YUNMAR[1,2], (Graduate Student Member, IEEE), SRI SUNING KUSUMAWARDANI[1], WIDYAWAN[1], AND FADI MOHSEN[3]

[1]Department of Electrical and Information Engineering, Universitas Gadjah Mada, Yogyakarta, Special Region of Yogyakarta 55223, Indonesia
[2]Department of Informatics Engineering, Institut Teknologi Sumatera, Jati Agung, South Lampung Regency 35365, Indonesia
[3]Department of Computer Science, University of Groningen, 9712 CP Groningen, The Netherlands

Corresponding author: Rajif Agung Yunmar (rajif@if.itera.ac.id)

This work was sponsored by Lembaga Pengelola Dana Pendidikan (LPDP), the Republic of Indonesia, and RTA Program Universitas Gadjah Mada with the Grant Number 5075/UN1.P.II/Dit-Lit/PT.01.01/2023.

ABSTRACT Over the last decade, numerous research efforts have been dedicated to countering malicious mobile applications. Given its market share, Android OS has been the primary target for most of these apps. Researchers have devised numerous solutions to protect Android devices and their users, categorizing them into static and dynamic approaches. Each of these approaches has its own advantages and disadvantages. The hybrid approach aims to combine the benefits of both. This study closely examines the hybrid solutions proposed between 2012 and 2023, highlighting their strengths and limitations. The objective of this study is to provide a comprehensive review of existing research on Android malware detection using a hybrid approach. Our review identifies several issues related to hybrid detection approaches, including datasets, feature utilization and selection, working environments, detection order mechanisms, integrity of the detection step, detection algorithms, and the use of automated input generation. Key findings of this study include: (i) the majority of studies have not adequately addressed on-device detection and have overlooked the importance of system usability, (ii) many studies rely on outdated datasets that do not accurately represent the current threat landscape, (iii) there is a need for a methodology to detect zero-day attacks, and (iv) most research has not paid attention to the impact of automated input generation on malware behavior and code coverage. We also discuss some open issues and future directions that will help substantiate the hybrid approach study.

INDEX TERMS Android malware, heuristic-based detection, hybrid approach.

I. INTRODUCTION

Android is the leading operating system for smartphones, powering a staggering 3.3 billion devices [1]. With a dominant market share of 70.79%, Android surpasses other mobile operating systems like iOS, KaiOS, and Windows [2], [3]. However, the widespread adoption of Android has made it an attractive target for cybercriminals. In Q3 2022, over five million instances of mobile malware were reported by Kaspersky Security Network [4], with new malware emerging daily [5]. Around 98% of mobile malware targets smartphones with the Android operating system [6]. These malicious applications engage in activities such as unauthorized access, information theft, spying, downloading additional code, and sending unauthorized messages.

To mitigate the risks posed by malware attacks, two primary detection methods are commonly employed: signature-based and heuristic-based detection. Signature-based detection identifies known malware by comparing its unique fingerprint. This approach offers efficiency, a low false alarm rate, and ease of implementation. However, it has limitations, including the inability to identify new or undiscovered malware and the need for regular updates to the signature database [7]. Heuristic-based detection overcomes these limitations by using sophisticated techniques such as rule-based systems or machine learning. This approach involves code analysis or monitoring the behavior of running applications [8].

The associate editor coordinating the review of this manuscript and approving it for publication was Vicente Alarcon-Aquino.

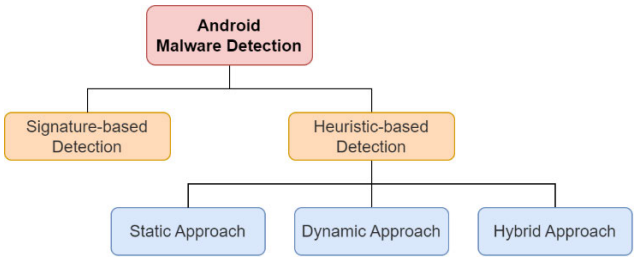

FIGURE 1. Malware detection techniques on Android environment.

Heuristic approaches need to rely on several features. These features can either come from static analysis or dynamic analysis. The set of features that result from statically analyzing the application is called static features, while the set of features that result from the dynamic analysis of the application is called dynamic features. The static analysis is usually done through the unpacking and disassembling of Android applications. The source code and configuration files (manifest) are then examined without the need to execute them. Static analysis is fast but has weaknesses, especially if the code is obfuscated [9], [10]. The dynamic analysis is done by running an Android app in the preconfigured environment while monitoring its behavior. This method is relatively robust to obfuscation; however, it is time-consuming and requires more computational resources. The two approaches have their own pros and cons. However, research by Martinelli et al. and Arora&Peddoju shows that the use of a hybrid approach to the detection process can significantly improve detection accuracy [11], [12]. Fig. 1 depicts a block diagram of malware detection techniques in the Android environment.

Since the first study on hybrid approaches was published in 2012 [13], [14], numerous research papers have focused on malware detection using this combined method. Table 1 provides a summary of some papers that review hybrid approaches. However, despite this surge in research activity, most of the existing review papers do not primarily focus on examining hybrid approaches. For example, Shu et al. [20], Koushki et al. [21], and Meijin et al. [22] review papers include discussions on malware detection that involve static, dynamic, and hybrid approaches. It is crucial to have a focused review that considers the distinct characteristics of hybrid approaches compared to other approaches, such as static and dynamic approaches. Hybrid approaches often cover specific aspects such as working environment, detection order, and detection integrity, which may not be addressed in other approaches.

Conversely, Shabir and Sabahat [15] reviewed articles that utilize a hybrid approach. However, the review remains incomplete due to several factors: it encompasses only five articles, resulting in a narrow scope and lack of depth in the discussion and analysis. This limitation in the breadth and depth of coverage within review papers can hinder the attainment of a comprehensive understanding of the current landscape of hybrid approaches. Therefore, it is important to review a sufficient number of papers to effectively observe the current state of the art in hybrid approaches, identify any existing gaps, discuss various proposed methodologies, and address the challenges still faced by the research community.

The systematic literature review (SLR) is a type of literature review that identifies, evaluates, and interprets all study findings in order to answer predetermined research questions [16]. In this work, we will perform SLR to provide a clear and comprehensive view of the state of the art of Android malware detection using hybrid approaches. The main contribution of this study are the following.

- We present a taxonomy of Android malware detection using hybrid approaches.
- We provide a summary of existing studies on hybrid approaches.
- We investigate the state of the art, identify the gaps, and guide discussion by exploring the methodologies that have been used in the field.
- We offer our evaluation of the current state of malware detection and highlight prospective challenges and directions for future research.

The remainder of this paper is organized as follows: Section II provides a brief description of the Android architecture, application architecture, and security mechanisms. Section III offers an overview of Android malware analysis. In Section IV, we describe the review protocol. Section V presents the findings of the reviewed studies. Section VI discusses research directions and future work. Finally, Section VII concludes the research.

II. ANDROID OVERVIEW

Malicious software, or malware, is just like any other app on Android, except that it is intended to cause harm. Therefore, it is essential to understand the architecture and components of Android, which can subsequently be used as a knowledge base to detect and avoid damage caused. Google launched Android in 2007 as an open-source operating system and officially released the first Android device in 2008. This section offers an encompassing overview of the Android environment, covering aspects such as system architecture, application architecture, and the security mechanisms in place to safeguard against malicious activities.

A. ANDROID OPERATING SYSTEM ARCHITECTURE

Android is a stack-based operating system for mobile devices, which consists of a Linux kernel, hardware abstraction layer, library and runtime, application framework, and application. Each layer has its distinct behavior and provides services to the layers above it ([17]. Fig. 2 depicts the architecture of the Android operating system.

Understanding the role of each component in the Android architecture in crucial to understand how malware analysis works. For instance, the Application System requires service from the Linux Kernel component via system calls to establish an internet connection. In this scenario, system calls serve as intermediaries between components. Within the context of

TABLE 1. Comparison of reviews that overlap with this article.

No.	Reference	Year	Type of Analysis	Review Scopes								Limitation
				Dataset	Feature Utilization	Feature Selection	Working Environment	Detection Order	Integrity of Detection	Detection Algorithm	Automated Input Generation	
1	Shabir&Sabahat [15]	2020	Hybrid	*	*	-	-	-	-	*	-	Limited papers to review, only five papers.
2	Gopinath&Sethuraman [19]	2023	Static, Dynamic, Hybrid	✓	✓	*	-	-	-	✓	-	✧
3	Shu et al. [20]	2023	Static, Dynamic, Hybrid	✓	✓	*	*	-	-	✓	-	✧
4	Koushki et al. [21]	2022	Static, Dynamic, Hybrid	✓	✓	✓	-	-	-	✓	-	✧
5	Meijin et al. [22]	2022	Static, Dynamic, Hybrid	✓	✓	*	-	-	-	✓	-	✧
6	Muzaffar et al. [23]	2022	Static, Dynamic, Hybrid	✓	✓	-	✓	-	-	✓	*	✧
7	Sharma&Rattan [24]	2021	Static, Dynamic, Hybrid	✓	✓	-	*	-	-	✓	-	
8	Garg&Baliyan [25]	2021	Static, Dynamic, Hybrid	✓	✓	-	-	-	-	✓	-	✧
9	Roseline&Geetha [26]	2021	Static, Dynamic, Hybrid	✓	✓	*	-	-	-	✓	-	✧
10	Kouliaridis&Kambourakis [27]	2021	Static, Dynamic, Hybrid	✓	✓	*	-	-	-	✓	-	✧
11	Senanayake et al. [28]	2021	Static, Dynamic, Hybrid	✓	✓	*	*	*	-	✓	*	✧
12	Razgallah et al. [9]	2021	Static, Dynamic, Hybrid	✓	✓	-	*	-	-	✓	-	✧
13	Liu et al. [29]	2020	Static, Dynamic, Hybrid	✓	✓	✓	-	-	-	✓	-	✧
14	Wang et al. [30]	2020	Static, Dynamic, Hybrid	✓	✓	✓	-	-	-	✓	-	✧
15	Alswaina&Elleithy [31]	2020	Static, Dynamic, Hybrid	✓	✓	-	-	-	-	✓	-	✧
16	Qamar et al. [32]	2019	Static, Dynamic, Hybrid	✓	✓	-	✓	-	-	✓	-	✧
17	Ashawa&Morris [33]	2019	Static, Dynamic, Hybrid	-	✓	-	-	-	-	✓	-	✧
18	Odusami et al. [34]	2018	Static, Dynamic, Hybrid	✓	*	-	*	-	-	✓	-	✧
19	Yan&Yan [35]	2018	Dynamic, Hybrid	-	✓	-	-	-	-	✓	-	✧
20	Yu et al. [36]	2018	Static, Dynamic, Hybrid	-	✓	✓	✓		-	✓	-	✧
21	Naway&Li [37]	2018	Static, Dynamic, Hybrid	✓	✓	-	✓	*	-	✓	*	✧
22	This paper	2023	Hybrid	✓	✓	✓	✓	✓	✓	✓	✓	

*Symbol description: ✓ : having content, – : no content, * : little or minimum content, ✧ : not focusing on the hybrid approach and having few reviews in hybrid approach publications.*

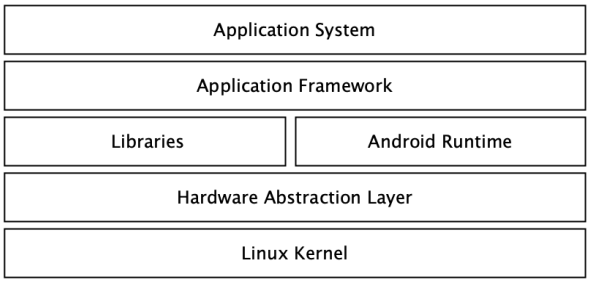

FIGURE 2. Android architecture stack [18].

malware analysis, the frequent occurrence of sensitive system call may indicate potential suspicious behavior within the application. The technical description of Android architecture that is depicted in Fig. 2 is explained in Table 2.

B. ANDROID APPLICATION ARCHITECTURE

In the previous section, we discussed the architecture of the Android operating system. In this section, we will discuss the architecture of an Android application and how it relates to malware analysis. We will focus the discussion on two main aspects of application architecture, namely application configuration and application components.

1) APPLICATION CONFIGURATION

Android applications are distributed as APK files containing crucial components, including the configuration file and source code. The AndroidManifest.xml file within an app's APK determines its behavior and runtime characteristics. In malware analysis, sensitive information and indicators of malicious activities can be extracted from this file. For instance, scrutinizing permission requests helps identify potentially over-privileged applications, which may indicate malicious intent. Table 3 provides a detailed breakdown of the content found within an APK file, shedding light on its composition and relevance in the context of malware analysis.

2) APPLICATION COMPONENT

An Android application consists of multiple loosely connected modules, each serving a specific purpose. Table 4 provides an overview of various components that an Android app can contain. Unlike traditional applications, an Android app can have multiple entry points. Each component within the app can offer several entry points that can be accessed or triggered in various ways. For instance, user interactions or system events such as low battery or loss of internet connection can activate specific entry points. However, the app must register to listen for such events.

C. ANDROID SECURITY

Android incorporates a robust security mechanism to ensure system integrity and user data privacy. Any endeavor to bypass the imposed policies is regarded as an attack. The Android security model comprises three key mechanisms: the sandbox mechanism, permission framework, and application signing. Table 5 provides a detailed description of these components.

TABLE 2. The android architectural stack and the role it plays.

Component	Description
Application System	This is the highest layer of the Android architecture stack, providing the program directly interacting with the user. There are two types of applications, system apps and user-installed apps. System apps are pre-installed on the Android OS image and cannot be uninstalled or modified by the user. On the other hand, user-installed apps can be obtained from sources such as Google Play or other app markets, allowing users to install and customize their devices.
Application Framework	This layer provides the libraries that are not part of standard Java runtime, and this layer can be accessed through the Java APIs app. This layer includes components to build Android apps such as Activity Manager, Bluetooth Manager, Resource Manager, etc. Those managers can access the Linux kernel's functionality by implementing system services.
Android Runtime	The Android runtime component consists of Dalvik VM and core libraries. The Android app that writes on Java will compiled in Java Class. However, Android doesn't run the app on Java class. The Java class will transform into a dex format. Dalvik VM will execute the dex format and relies on a Linux kernel that provides lower-level functionality. As part of Android architecture, the core library has the functionality of a Java application programming interface (API). The Android API version is trimmed from the version we can see on the J2SE version. Android developers do not use the full J2ME version to avoid license and security problems. Each Android app is designed to run on its own VM, as opposed to J2ME, where all of the apps will run in one VM.
Libraries	Android provides libraries that are written in C or C++, which are called native libraries. Those libraries are trimmed from typical libc that are optimized for embedded systems. Several services and components that need hardware optimization can be accessed through Java Native Interface (JNI).
Hardware Abstraction Layer (HAL)	HAL provides standards that handle interface. HAL provides strong drivers for a variety of devices. This is because Linux has an easily understood architecture, hence a vast driver built into the kernel that is available freely. The HAL provides an interface that exposes the functionality of the device hardware to the high-end Java API platform.
Linux Kernel	On the bottom of the stack is a modified Linux kernel that is designed to run on an embedded system. Therefore, not all Linux functionality is found on Android OS, such as the X Windows system. Some specific functionality is added only on Android, and this modification is called Androidisms [38]. Some important Androidisms including Binder, paranoid networking, low memory killer, etc.

TABLE 3. The files that are included within an APK package.

Component	Description
AndroidManifest.xml	Contain the definition of an application component, i.e., activities, services, and broadcast receiver. It also contains information about the package and the required permission.
Classess.dex	This is the bytecode that is optimized for the Android apps that contain construction, i.e., header, string table, list of local variables, the definition of the class table, method, etc.
Resource.arsc	Contains app resources in the form of binary.
Asset folder	A folder that contains assets like images, files, etc., that are accessed using AssetManager.
Lib folder	Contain native code library.
META-INF folder	The folder contains the app certificate and another essential file.

TABLE 4. Component of an android application and the functionality it provides.

Component	Description
Activities	This component is basically a screen that visually interacts with the user. An Android app may have multiple activities that are displayed in a particular order and started independently.
Services	This component does not have a user interface and is running in background mode. Sometimes, this component performs long-running tasks, such as downloading.
Intents	An object within an activity can communicate with other activities, by calling activities within the same package or across different packages within the same project.
Content Providers	Offer a secure and standardized mechanism for accessing and managing shared data between various components within applications, particularly those that exchange sensitive and persistent information.
Broadcast Receivers	An Android app and system component can communicate by subscribing to these Intents through Broadcast Receivers.

III. MALWARE ANALYSIS OVERVIEW

Heuristic-based malware detection demonstrates greater resilience against attack variations compared to signature-based detection. Furthermore, incorporating machine learning techniques enhances the effectiveness of malware detection. In the detection process, the heuristic approach requires detection materials, also called features. There are two sorts of features: static and dynamic, which are provided via static and dynamic analysis, respectively.

In the following section, we will delve into the subject of malware analysis. We will commence by providing an overview of static analysis and the associated features. Then, a discussion of dynamic analysis and the features it produces will follow. In the last section, we look at the hybrid approach, which combines two different features we discussed before.

TABLE 5. Android security mechanism protects the system, user data, and privacy.

Component	Description
Sandboxing	In Android, each running application is assigned a unique User ID (UID), isolating it from other apps and the system. Unlike UNIX, where a single UID can be shared, Android's sandboxing ensures each app operates in its own environment, preventing access to data or information from other apps or the system, improving security.
Permission	Android operates in sandbox mode, allowing apps to only access their own files and resources. To access functionalities or resources like camera, storage, or internet connection, an app must obtain permission from the appropriate authority beforehand.
Application Signing	An Android app must be signed by its developer, which is a process that establishes trust and ensures that the app updates originate from the same author, also known as same-origin privacy.

TABLE 6. List of the static features that are obtained through static analysis.

Feature	Description
Permission	These features describe what functionality is required by the app. This feature is obtained by decoding AndroidManifest.xml into a human-readable format [39].
API Calls	This feature depicts how the app interacts with the operating system [40]. It is acquired by processing the classes.dex file and decompiling it into Java format.
Filter intent	This feature is specified in the AndroidManifest.xml file to define the type of intent that the component will receive.
Broadcast Receiver	This feature is related to intent, where app and system components communicate with subsections to a certain intent.
Activity	This information, written in the manifest file, represents a part of the implementation of the app's visual interface.
Other	In addition to the elements mentioned earlier, other details within the manifest file such as uses-sdk, providers, and services also serve as crucial sources for static features that can be analyzed to identify suspicious patterns.

A. STATIC ANALYSIS

The static analysis provides static features that are obtained by analyzing the app without executing it. To perform static analysis, the app is initially unpacked to retrieve key items such as the AndroidManifest.xml, classess.dex, res, asset, and lib components. These components are then subjected to advanced processing techniques to extract relevant features. Table 6 presents a selection of features that can be obtained from processing these items.

B. DYNAMIC ANALYSIS

Dynamic analysis involves executing the application in a pre-configured environment to observe its behavior and extract dynamic features. Researchers use various tools to monitor and log the application's activities to facilitate this analysis. For example, tools like Strace can be used to record system calls [41], Wireshark can capture network traffic [42], and the Android Debug Bridge (ADB) enables monitoring of memory and CPU consumption [43]. Table 7 provides an overview of features obtained from dynamic analysis [40].

TABLE 7. List of the dynamic features that are produced by dynamic analysis.

Feature	Description
System Calls	The features are derived from the interplay between the app and the OS. The system calls provide functions to access the network, file, etc.
Network Activity	This feature is obtained from app activity that has a connection to a network, sends and receives data, etc.
API Calls	This feature is just like API Calls in static features.
SMS	This feature is obtained when the app sends or receives an SMS message. The SMS may contain suspicious links that are harmful to the user. Some malware sends premium SMS without user permission.
Phone Call	This feature is obtained when the app has phone call functionality that intrudes on user privacy and security.
Battery Status	Some malware conducted suspicious processes that may consume more CPU usage and drain the battery. Malware may be detected by monitoring the battery change.
CPU Usage	Idem.
File Operation	Some malware aims to steal personal user information, such as photos, videos, and notes. Anti-malware programs should monitor suspicious file-related activities.
Other	Besides the above, other behaviors may be used as dynamic features, i.e., phone events, phone status, memory info, etc.

C. HYBRID ANALYSIS

Hybrid analysis involves the integration of both static and dynamic analysis techniques. In this approach, the app is subjected to both static and dynamic analysis using appropriate tools and methodologies. A hybrid detection model would then use features from both types. Studies have shown that hybrid approaches achieve better accuracy than static and dynamic ones [11], [12].

Hybrid detection can take advantage of the complementary strengths of the static and dynamic approaches. As an illustration, malware can use obfuscation techniques to evade static-based detection, yet harmful activity can still be found using a dynamic approach. Likewise, malware that hides malicious behavior during dynamic analysis can still be identified using a static approach by extracting and analyzing application code.

IV. REVIEW PROTOCOL

The systematic literature review (SLR) is a review of the literature that seeks to discover, assess, and interpret all relevant study findings in order to answer specific research

TABLE 8. The keywords and the domains that were used in the search process.

Domain	Keyword
Platform	Android, mobile, smartphone
Program analysis	Hybrid analysis, combination analysis
Security assessment	malware detection, malware analysis
Detection system	Classical approach, machine learning, deep learning, etc.

questions [16]. In this study, we conduct an SLR following the guidelines outlined by Budgen&Brereton [44]. The whole process consists of three main phases as follows:

- Planning the SLR: The purpose of this phase is to describe the goal and define the review protocols.
- Conducting the SLR process: This phase carries out the key research material in this SLR, which consists of several stages, i.e., define the research questions, search strategy, and selection criteria.
- Reporting the result of SLR: The goal of this phase is to fulfill the SLR.

A. RESEARCH QUESTION

In this study, we formulate the research questions, which represent the foundation of the SLR. The research questions of this study are as follows:

- RQ1 : What datasets have been used in this line of research?
- RQ2 : What are the features that have been used for Android malware detection?
- RQ3 : What feature selection algorithms have been used in this line of research?
- RQ4 : What kinds of working environments have been used in the hybrid analysis?
- RQ5 : What are the configurations for the step of detection order?
- RQ6 : What are the configurations for the integrity of the detection system?
- RQ7 : What kind of algorithm is used for Android malware detection?
- RQ8 : What is automated input generation used for triggering malicious activity?

B. SEARCH STRATEGY

The search strategy employed in this study is designed to identify high-quality and highly relevant refereed articles from various sources such as journals, conferences, and other scholarly materials including lecture notes, book chapters, and posters. In Table 8, we display all of the keywords that we used in our search and corresponding domains.

We use different search engines to find relevant articles using our list of search keywords. Our search was further restricted to articles discussing Android malware detection using hybrid analysis. Furthermore, the publication date needs to occur between 2012 and September 2023. Additionally, we expanded our search by examining the reference

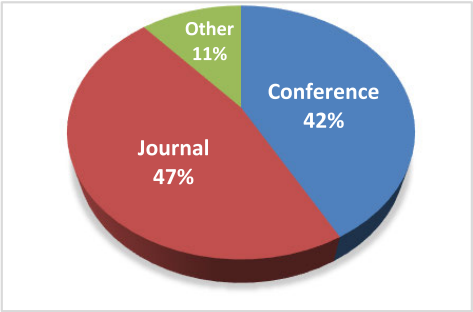

FIGURE 3. Origins of the studied papers.

sections of the obtained articles. Below are some of the search engines and web publishers that were employed in our search:

- IEEE Xplore,
- ScienceDirect Elsevier,
- ACM Digital Library,
- Springer Link,
- DBLP, and
- Google Scholar.

C. SELECTION CRITERIA

The papers collected using a search engine with given keywords will be evaluated according to inclusion and exclusion criteria. As described in Table 8, four domains construct the search keywords. The paper candidate must be at the intersection of four domains, as indicated in Table 8, and fulfill the following inclusion standard.

- The article must cover the hybrid analysis.
- The article's analysis must focus on the Android platform.
- The article is supported by a series of empirical experiments.

Moreover, to filter search results from irrelevant articles that don't alight well with the objective of SLR, the candidate article must be excluded in the following cases:

- If the article focuses on another operating system.
- If the article is not supported by proper analysis, detection technique, or empirical experiment.
- If the article performs analysis only on static analysis or only dynamic analysis,
- If the article claims that they use a hybrid approach to detection, but in reality, they only use either static or dynamic features.

D. SELECTED PAPERS

Conducting the aforementioned search strategy produced 659 candidate articles. We then went over each one of them, considering our selection criteria, which resulted in obtaining 88 of the most appropriate articles that can be reviewed and included in this SLR. Fig. 3 depicts the distribution of the collected articles based on their origin, indicating a balanced representation between conferences and journals. In Fig. 4, we show the distribution based on their publication year.

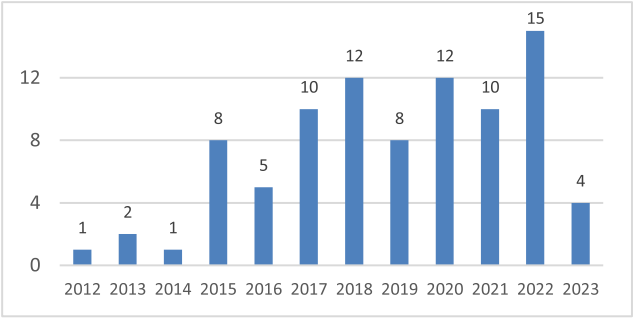

FIGURE 4. The distribution of the studied papers over the years.

V. REVIEW RESULT

The purpose of this section is to present our findings from looking at the primary studies. We will discuss the results of this systematic literature review (SLR) in relation to the research questions. We divide the discussion of the review studies into five sections. To facilitate comprehension, we will refer to Fig. 5. It gives a taxonomy of all the studies we will discuss in this section.

A. DATASET

The availability of representative datasets plays a crucial role in developing and testing malware detection models. These datasets usually contain benign and malicious apps. The dataset can be used to perform thorough and reliable analysis, develop robust models, confirm hypotheses, identify emerging trends, and generate meaningful predictions or recommendations based on empirical evidence. Access to comprehensive datasets empowers researchers to explore and uncover valuable insights, contributing to advancements in their respective fields and supporting evidence-based decision-making processes.

In this section, we will address RQ1 by examining the datasets employed to train and test hybrid detection approaches. Table 9 presents a compilation of the most commonly utilized Android datasets identified in the reviewed articles. The datasets used in the hybrid approaches could be categorized into private datasets, APK-based files, text-based datasets, and image-based datasets.

1) PRIVATE DATASET

We discovered that certain researchers opted to utilize their own datasets, which were not publicly available on the internet. However, they provided insights into their data collection methodologies. Generally, benign applications were obtained from sources such as the Google Play Store, 9App Store, APKpure, or other application stores. Malicious applications, on the other hand, were collected from various groups, blogs, GitHub, and other similar sources. In some cases, these researchers combined their own private datasets with public datasets. For example, Chen et al. merged their dataset with a public dataset from MobiSec Lab and Contagio [64]. Jiao et al. constructed their datasets using apps sourced from the Google Play Store, the Android Malware Genome Project, and third-party markets [65].

The utilization of private datasets has some obstacles due to their lack of public accessibility, which restricts the ability of other researchers to verify and corroborate the findings. Moreover, several studies that employ private datasets fail to verify the legitimacy of the applications encompassed inside their datasets. In order to tackle this matter, it is possible to utilize supplementary technologies like VirusTotal or alternative antivirus software to authenticate the condition of the applications and guarantee their integrity.

2) APK-BASED DATASET

Most datasets that are publicly available on the Internet are in the form of APK files. To generate a dataset, researchers collect applications from various sources during a certain time range and evaluate them utilizing specialized methods. The subsequent step involves identifying and labeling Android apps as either malicious or benign.

The Android Malware Genome is a popular dataset that contains 1,200 malicious applications representing hundreds of malware families. The apps were collected in the range of August 2010 to October 2011 [110]. The Drebin dataset contains samples collected from the Google Play store, Chinese App stores, Russian App markets, and other sources, i.e., forums, blogs, and websites from 2010 to 2012. Additionally, Drebin also includes a dataset from the Malware Genome Project [45].

While many datasets are publicly available, some cannot be downloaded directly. Email authorization is required to access datasets like Drebin [111] and Androzoo [112]. Access controls are often used to protect data privacy, monitor dataset consumption, or ensure that users understand and follow usage rules. Nevertheless, it is important to consider some obstacles that may arise, such as the incorporation of supplementary protocols that might impact the duration of the research project, as well as the potential risk of rejection due to the regulations imposed by data providers.

3) TEXT-BASED DATASET

The datasets are available in text formats such as CSV and JSON. These datasets are typically derived from the analysis of APK files, both statically and dynamically. Studies conducted by researchers like Imtiaz et al. [99] and Jannat et al. [66], which focus on malware classification or feature selection, may find these datasets suitable for their purposes.

However, text-based datasets may have inherent limitations in terms of feature representation. A prime example can be seen in datasets like MalDroid-2020 [113] and Kronodroid [114], where system calls are represented using frequency calculations. This approach may inadvertently discard crucial information regarding the sequence of system call operations. The chronological order of system calls can serve as a significant indicator of malicious behavior. In such

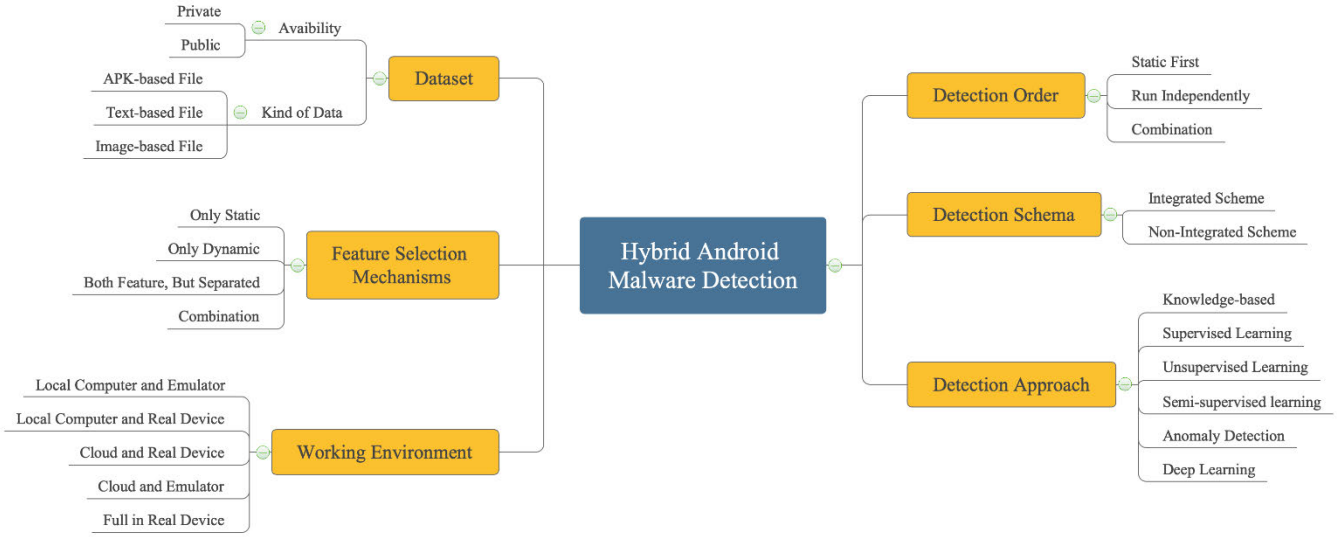

FIGURE 5. Taxonomy of hybrid malware detection.

TABLE 9. A summary of the datasets that were used by the studied works.

Dataset	Created	Last Update	Num of Benign	Num of Malware	Accessibility	Dataset Type	Related Studies
Drebin	2014	-	123,453	5,560	Public	APK File	[11] [45] [43] [46] [47] [48] [49] [50] [51] [52] [53] [54] [55] [56] [57] [58] [59] [60] [61] [62] [63]
Malgenome / Genome Project	2011	-	-	1,200	Public, Discontinue	APK File	[11] [12] [64] [65] [45] [66] [46] [47] [50] [55] [58] [67] [68] [69] [70] [71] [72] [73] [41] [74] [75] [76] [77] [78]
Contagio	2011	2018	-	> 500	Public	APK File	[11] [64] [67] [72] [74] [75] [76] [79] [80] [81] [82] [77] [78] [83]
Androzoo*	2016	-	-	> 20,000	Public	APK File	[51] [54] [84] [85] [86] [87]
AMD (Android Malware Dataset)	2017	-	25,553	-	Public, Discontinue	APK File	[54] [84] [85] [88]
VirusShare	2012	2022	-	479,811	Public	APK File	[67] [84] [89] [90] [72] [41] [74] [91] [75] [92] [93] [94] [83]
RansomProber	2019	-	2,280	5,500	Public, Discontinue	APK File	[43] [95] [62]
CIC-AndMal2017	2017	-	6,500	4,354	Public	CSV & APK File	[42] [96] [97] [98]
CIC-InvesAndMal2019	2019	-	5,065	426	Public	CSV & APK File	[99] [97]
CIC-MalDroid2020	2020	-	1,795	9,803	Public	CSV & APK File	[47] [100] [101] [83] [102]
OmniDroid	2018	-	11,000	11,000	Public, Discontinue	JSON File	[86] [103] [104] [105] [106]
Kronodroid	2021	-	72,011	70,127	Public	CSV	[107] [108]
MalMem2022	2022	-	29,298	29,298	Public	CSV	[102]
Goorax	2018	-	-	4000	Public	JSON File	[66]
AndroMnist	2022	-	1,500	400	Public	Image	[109]

** No available data indicates the number of malicious and benign entries within the dataset.*

cases, relying solely on the frequency of system calls may not provide an accurate depiction of malicious activity.

4) IMAGE-BASED DATASET

This kind of dataset is distinct from the others as it utilizes visual representations as the foundation for its classifications. Xu et al. [109] created their dataset by converting the network flow into a grayscale image. The network flow data was captured using Wireshark, a network dump software, and represented in PCAP files.

However, converting network dataflow directly into images may potentially incorporate extraneous components.

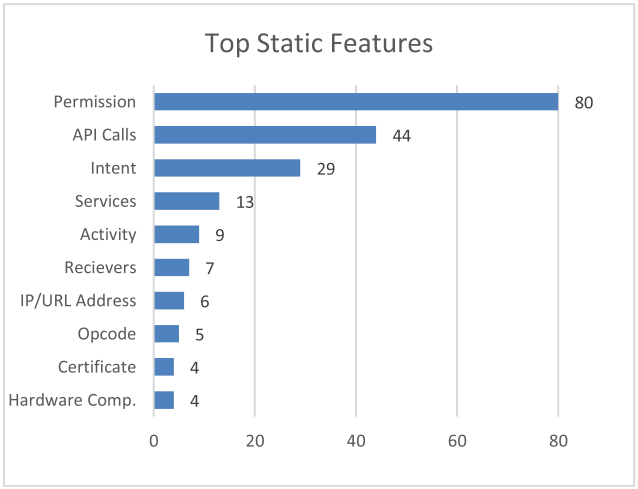

FIGURE 6. The top ten most often used static features based on 88 different studies.

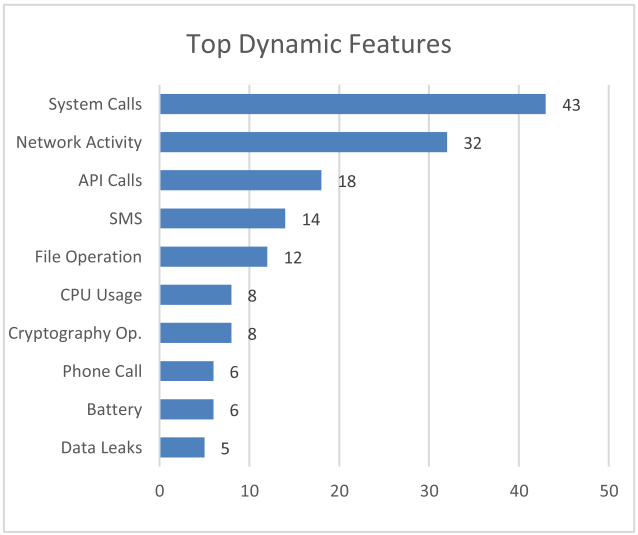

FIGURE 7. The utilization of dynamic features in the top ten of 88 studies.

Before transforming network dataflow into picture format, a feature selection process is vital to identify only essential characteristics and eliminate extraneous information. The obtained feature set exhibits enhanced quality and accurate calibration, augmenting the precision and efficiency of the detection techniques.

B. FEATURE UTILIZATION

Hybrid approaches use features generated from the static and dynamic analysis of the applications. Each type of analysis produces features that can be used as detection material. Researchers combine these detection materials to construct their models. In this sub-section, we address RQ2 by exploring the features utilized for Android malware detection. In Fig. 6 and Fig. 7, we show the usage stats for these features. In Table 10, we provide a summary of the usage of static and dynamic features that were employed as detection materials in 88 studies.

The work conducted by Faghihi et al. [61] presents a notable departure from traditional detection methods. In contrast to alternative techniques, this research solely focuses on static traits for the purpose of detection. The application of dynamic analysis is carefully utilized to examine suspected instances of malware that apply static evasion tactics. Within this particular scenario, the dynamic analysis method thoroughly examines call logs in relation to 400 permission-protected Android APIs. This analysis encompasses both Java and native code components. The only employment of static characteristics has the benefit of accelerating the process of detection. Nevertheless, it is crucial to recognize that the aforementioned detection method may face difficulties when dealing with malware-hardening strategies like obfuscation and dynamic loading.

Algorithm 1 Custom Feature Selection Proposed by Gera et al. [88]

$S_c \leftarrow$ Number of benign samples
$S_m \leftarrow$ Number of malware samples
4: # calculate the sum of the frequency of each feature within
# the benign sample and then standardize the values
for x in (*benign_samples_list*) do
    $\text{Normalized\_Frequency}_{benign}(f_x) \leftarrow \sum_{i=0}^{n} frequency(f_i)/S_c$
9: # calculate the sum of the frequency of each feature within
# the malware sample and then standardize the values
for x in (*malware_samples_list*) do
    $\text{Normalized\_Frequency}_{malware}(f_x) \leftarrow \sum_{i=0} frequency(f_i)/S_m$
14: # calculate the absolute difference
for x in (*unique_feature_list*) do
    $\leftarrow \text{Diff}_{Normalize\_frequency}(f_x)\ \text{Normalized\_Frequency}_{benign}(f_x) - \text{Normalized\_Frequency}_{malware}(f_x)$
18: # sort the difference of normalized frequencies in ascending order
Sort($\text{Diff}_{Normalize\_frequency}(f)$)
21: # select the top 'k' features from the sorted results of
# $\text{Diff}_{Normalize\_frequency}$ as the dominant features
for k in (20, 40, 60, 80)) do
    Select_Dominant_Features(k, $\text{Diff}_{Normalize\_frequency}(f)$)

C. FEATURE SELECTION

Hybrid malware detection techniques offer better results than either static or dynamic approaches. However, hybrid approaches have relatively high computational overhead and require more time. This increase in computation and time is due to the extraction and processing of a larger number of features during and after development. By carefully choosing the appropriate features, the computational burden and resource requirements can be minimized, allowing for efficient and effective detection of malware in on-device scenarios. The selection techniques play a significant role in optimizing the performance of the hybrid approach, ensuring that the chosen features contribute the most to the detection process while maintaining the system's effectiveness.

1) FEATURE SELECTION ALGORITHM

Researchers have employed various feature selection algorithms in hybrid malware detection. The information gain

TABLE 10. A summary of feature utilization by all studied solutions.

Studies	Static Features						Dynamic Features						
	Permission	API Calls	Intent	Services	Activity	Other	System Calls	Network	API Calls	File Operation	SMS	Phone Call	Other
Ahmed et al. [43]	☑	☐	☐	☐	☐	Specific string from DEX	☑	☐	☐	☐	☐	☐	CPU usage, memory usage
Ahmed et al. [101]	☑	☑	☑	☑	☐	Access count info, obfuscation instances	☑	☐	☐	☐	☐	☐	Binder calls, composite behaviors
Ahmed&Lin [62]	☑	☐	☐	☐	☐	Network related info	☑	☐	☐	☐	☐	☐	CPU usage, memory usage
Alzaylaee et al. [115]	☑	☐	☑	☐	☐		☐	☐	☑	☐	☐	☐	
Amer&El-Sappagh [116]	☑	☐	☐	☐	☐		☑	☐	☑	☐	☐	☐	
Anupama et al. [63]	☑	☐	☐	☐	☐		☑	☐	☐	☐	☐	☐	
Arora et al. [117]	☑	☐	☐	☐	☐		☐	☑	☐	☐	☐	☐	
Arora&Peddoju [12]	☑	☐	☐	☐	☐		☐	☑	☐	☐	☐	☐	
Arshad et al. [45]	☑	☑	☑	☑	☐	Broadcast, hardware component, content providers, IP address	☑	☐	☐	☐	☐	☐	
Aurangzeb&Aleem [107]	☑	☐	☑	☐	☐	hardware software requirement, metadata	☑	☐	☐	☐	☐	☐	
Bhandari et al. [67]	☑	☑	☑	☐	☐	Hardware component, IP address, URL	☐	☐	☐	☑	☐	☐	CPU usage, memory usage
Cavli&Sen [46]	☑	☐	☐	☑	☑	Receiver	☐	☐	☑	☐	☐	☐	Activity bigram
Chaulagain et al. [84]	☐	☑	☐	☐	☐		☑	☐	☐	☐	☐	☐	
Chen et al. [64]	☑	☑	☐	☐	☐		☐	☑	☐	☑	☑	☐	OS interaction
Dhalaria&Gandotra [118]	☑	☑	☑	☐	☐	Command string	☑	☐	☐	☐	☐	☐	Cryptography operation, dynamic permission, information leak
Eder et al. [68]	☑	☐	☐	☑	☐	Receivers, package name, URL	☑	☑	☑	☐	☐	☐	
Faghihi et al. [61]	☑	☑	☐	☐	☐	Application classes info	☐	☐	☐	☐	☐	☐	Call logs to set permission APIs
Fang et al. [89]	☑	☑	☐	☐	☐		☑	☐	☐	☐	☐	☐	
Feng et al. [42]	☑	☐	☑	☐	☐		☐	☑	☐	☐	☐	☐	
Garg & Baliyan [85]	☑	☐	☐	☐	☐		☑	☑	☐	☐	☑	☐	CPU usage, battery, memory usage
Gera et al. [88]	☑	☐	☐	☐	☐		☑	☑	☑	☑	☐	☐	Dynamic permission
Hadiprakoso et al. [90]	☑	☐	☑	☐	☐	Command signature	☐	☐	☑	☐	☐	☐	
Hadiprakoso et al. [47]	☑	☑	☑	☐	☐	Command signature, binaries	☑	☐	☐	☐	☐	☐	Binder call, composite behavior
Husainiamer et al. [49]	☑	☑	☐	☐	☐		☑	☐	☐	☐	☐	☐	
Hussain et al. [48]	☑	☐	☑	☐	☐		☐	☐	☑	☐	☐	☐	
Imtiaz et al. [99]	☑	☐	☑	☐	☐		☐	☑	☑	☐	☐	☐	Battery, log states, package, process logs
Iqbal et al. [95]	☑	☑	☐	☐	☐	Text resource asset	☐	☐	☐	☐	☐	☐	Text from screenshot
Jadhav et al. [119]	☑	☑	☐	☐	☐	Broadcast, IP address, string from DEX, cryptographic function	☑	☑	☐	☑	☐	☐	Fingerprint activity, dynamic broadcast
Jang et al. [120]	☑	☑	☑	☐	☐	OS command, forged files, certificate	☐	☐	☐	☐	☐	☐	Memory dump
Jannat et al. [66]	☑	☑	☑	☐	☐	Min SDK info	☐	☑	☑	☑	☐	☑	Battery, DEX class information
Jiao et al. [65]	☑	☐	☐	☐	☐		☑	☐	☐	☐	☐	☐	
Joshi&Kumar [102]	☑	☑	☑	☑	☑	Method tags, package, receivers, number of files	☑	☐	☐	☐	☐	☐	Binder call, memory dump
Kabakus [69]	☑	☐	☐	☐	☑	Features, receiver, line of codes	☐	☑	☐	☐	☐	☐	CPU usage, memory info, fault info

TABLE 10. *(Continued.)* **A summary of feature utilization by all studied solutions.**

Studies	Static Features						Dynamic Features						
	Permission	API Calls	Intent	Services	Activity	Other	System Calls	Network	API Calls	File Operation	SMS	Phone Call	Other
Kandukuru&Sharma [70]	☑	☐	☐	☐	☐		☐	☑	☐	☐	☐	☐	
Kandukuru&Sharma [71]	☑	☐	☐	☐	☐		☑	☑	☐	☐	☐	☐	
Kapratwar et al. [50]	☑	☐	☐	☐	☐		☑	☐	☐	☐	☐	☐	
Khoda et al. [51]	☑	☑	☐	☐	☐	Inter component communication	☑	☐	☐	☐	☐	☐	
Konstantinos [86]	☑	☑	☐	☑	☑	Opcodes, receivers, API packages, system command, flowdroid	☑	☑	☐	☑	☑	☑	Circumvented permission, broadcast receivers, loaded DEX, information leaks, cryptography operation
Kuo et al. [121]	☑	☐	☐	☐	☐		☐	☐	☐	☐	☐	☐	Log file
Lemos et al. [87]	☑	☑	☐	☐	☐		☑	☐	☐	☐	☐	☐	Binder call
Li et al. [122]	☐	☑	☐	☐	☐	APK features	☐	☑	☐	☐	☑	☐	CPU usage, memory usage
Li et al. [123]	☐	☐	☑	☑	☐	IP address, ads modules, system security setting	☐	☐	☑	☐	☐	☐	
Lin et al. [124]	☑	☑	☐	☐	☐	Certificate	☐	☑	☐	☑	☐	☐	
Lindorfer et al. [72]	☑	☑	☐	☐	☐	Certificate	☐	☑	☐	☑	☐	☑	Data leaks, dynamic DEX, dynamic broadcast
Liu et al. [73]	☑	☑	☐	☐	☐		☑	☐	☐	☐	☐	☐	
Liu et al. [41]	☑	☑	☐	☐	☐		☑	☐	☐	☐	☐	☐	Privacy usage
Lu et al. [74]	☑	☑	☑	☑	☑	Broadcast, certificate, package info, content providers	☐	☐	☑	☐	☐	☐	
Mahdavifar et al. [100]	☑	☑	☐	☑	☑	Method tags, package, number of files	☑	☐	☐	☐	☐	☐	Binder call
Martín et al. [103]	☐	☑	☐	☐	☐		☐	☐	☐	☐	☐	☐	Transition probabilities, state frequencies
Martinelli et al. [11]	☐	☐	☐	☐	☐	Opcode	☑	☐	☐	☐	☑	☐	Authorization abused
Maryam et al. [52]	☑	☐	☑	☐	☐		☑	☐	☐	☐	☐	☐	External DES class, data leaks
Oliveira et al. [104]	☑	☐	☑	☐	☐	Raw DEX image	☑	☐	☐	☐	☐	☐	
Patel&Buddadev [79]	☑	☐	☑	☑	☐	Broadcast	☐	☑	☐	☐	☐	☐	Native call
Pektas&Acarman [91]	☑	☑	☐	☐	☐	Installed services	☐	☑	☐	☐	☐	☐	
Qammar et al. [78]	☑	☐	☑	☐	☐	Metadata, reflection, cryptographic functions	☑	☑	☑	☑	☐	☐	Cryptographic operations
Rafiq et al. [108]	☑	☑	☑	☐	☐		☑	☐	☐	☐	☐	☐	
Rahmawati et al. [125]	☑	☐	☐	☐	☐		☑	☐	☐	☐	☐	☐	
Rodrigo et al. [105]	☑	☑	☐	☑	☑	Receivers, API packages, opcode, system commands	☐	☑	☐	☐	☐	☐	Data leaks, loaded DEX class, cryptography operation
Roy et al. [106]	☑	☑	☐	☐	☐	File access	☑	☑	☐	☐	☐	☐	Binder calls
Saif et al. [126]	☑	☑	☐	☑	☑	Uses-features, providers, receivers	☑	☐	☐	☐	☐	☐	
Saracino et al. [75]	☑	☐	☐	☐	☐	Metadata from the app store	☑	☐	☐	☐	☑	☐	User screen activities
Shijo&Salim [92]	☐	☑	☐	☐	☐		☐	☐	☑	☐	☐	☐	
Spreitzenbarth et al. [127]	☑	☑	☑	☐	☐		☐	☑	☐	☐	☑	☐	Native call
Su et al. [81]	☑	☐	☐	☐	☐	Function calls, priority APK settings	☐	☐	☐	☐	☐	☐	Log file
Su&Fung et al. [80]	☑	☐	☑	☐	☐	Java sensitive function	☐	☐	☐	☐	☑	☐	Log file
Subhan Ullah et al. [83]	☑	☑	☐	☑	☐	App ratings	☑	☑	☐	☐	☑	☐	CPU usage, memory usage, battery, dynamic permission, email

TABLE 10. *(Continued.)* **A summary of feature utilization by all studied solutions.**

Studies	Static Features						Dynamic Features						
	Permission	API Calls	Intent	Services	Activity	Other	System Calls	Network	API Calls	File Operation	SMS	Phone Call	Other
Sugunan et al. [53]	☑	☐	☐	☐	☐		☐	☐	☑	☐	☐	☐	
Sun et al. [76]	☑	☐	☑	☐	☐	App component	☑	☐	☐	☐	☐	☐	System services, binder call
Sun et al. [128]	☑	☑	☐	☐	☐		☑	☐	☐	☐	☐	☐	Binder call
Surendran et al. [54]	☑	☑	☐	☐	☐		☑	☐	☐	☐	☐	☐	
Taher et al. [98]	☑	☑	☑	☐	☐	Command strings	☑	☐	☐	☐	☐	☐	Dynamic permission, cryptographic operation, information leak
Taheri et al. [96]	☑	☐	☑	☐	☐		☐	☑	☑	☐	☐	☐	
Upadhayay et al. [55]	☑	☐	☐	☐	☐		☐	☑	☐	☐	☐	☐	
Wang et al. [57]	☑	☐	☐	☐	☐	Hardware component	☐	☐	☐	☑	☐	☐	Register receivers, execute commands, content resolver query, find resource, data leak, cryptography operation
Wang et al. [129]	☑	☐	☐	☐	☐		☐	☐	☑	☐	☐	☐	
Wang et al. [56]	☑	☐	☑	☐	☑	Hardware component, static signature	☐	☑	☐	☑	☑	☑	Register receivers, execute commands, content resolver query, find resource, data leak
Wei et al. [130]	☑	☐	☐	☐	☐		☐	☐	☑	☐	☐	☐	System function
Wen&Yu [58]	☑	☑	☑	☐	☐	Uses-feature, package component name	☐	☐	☐	☐	☑	☐	CPU usage, battery
Xu et al. [109]	☐	☑	☐	☐	☐	Opcode	☐	☑	☐	☐	☐	☐	
Xu et al. [93]	☑	☑	☑	☐	☐	Providers, advertising networks	☑	☐	☐	☐	☐	☐	
Yam et al. [97]	☑	☐	☐	☐	☐		☐	☑	☐	☐	☐	☐	
Yang et al. [59]	☑	☐	☐	☐	☐	Hardware, app component	☐	☑	☐	☐	☑	☐	Start services, cryptography operation
Yuan et al. [82]	☑	☑	☐	☐	☐		☐	☐	☑	☐	☐	☐	
Yuan et al. [77]	☑	☑	☐	☐	☐		☐	☑	☐	☑	☑	☑	Information leaks, cryptography operation
Zhang et al. [60]	☐	☐	☐	☐	☐	Opcode	☑	☐	☐	☐	☐	☐	
Zhao et al. [94]	☑	☑	☐	☐	☐		☐	☑	☐	☐	☑	☑	Battery, binder call
Zheng et al. [131]	☑	☐	☐	☐	☐		☐	☐	☐	☐	☐	☐	Forward symbolic execution
Zhou et al. [14]	☑	☑	☐	☐	☐		☑	☐	☐	☐	☐	☐	
	Permission	**API Calls**	**Intent**	**Services**	**Activity**	**Other**	**System Calls**	**Network**	**API Calls**	**File Operation**	**SMS**	**Phone Call**	**Other**
Total	**80**	**44**	**29**	**13**	**9**		**43**	**32**	**18**	**12**	**14**	**6**	

algorithm has been used as a selection characteristic in studies by Alzaylaee et al. [115], Dhalaria and Gandotra [118], and Li et al. [122]. Meanwhile, Hadiprakoso et al. [47], Hussain et al. [48], and Ahmed [43] opted for principal component analysis (PCA). Additionally, manual methods were also employed in some studies, such as removing specific strings [45], eliminating duplicate apps [118], removing empty features [105], and threshold variation [56], [57]. It is worth noting that a single research work may employ multiple feature selection techniques.

Some researchers have chosen to create their own custom feature selection techniques in addition to using established algorithms. For example, Gera et al. introduced a feature selection approach tailored to identifying the most dominant features [88]. The details of this method are explained in Algorithm 1, which provides a comprehensive overview of the researchers' approach. The algorithm takes input in the form of feature vector data derived from both benign and malware samples and aims to identify and extract the list of k-dominant features that are essential for further analysis and processing.

Additional studies conducted by researchers such as Faghihi et al. [61], Lemos et al. [87], and Arshad et al. [45] have proposed custom or manual feature selection meth-

TABLE 11. The feature selection technique was utilized in 48 different studies.

Feature Selection	Related Studies
Manual or Custom	[57] [56] [87] [64] [45] [92] [41] [105] [88] [89] [106] [61]
Chi-Square Test	[42] [57] [56] [49] [12] [97] [107] [98]
Information Gain	[115] [50] [122] [52] [79] [49] [43] [48] [118] [12] [78] [83] [97] [98]
PCA	[123] [43] [48] [47]
RELIEF	[126] [94]
OTHER	[123] [42] [72] [58] [105] [63] [102] [98]

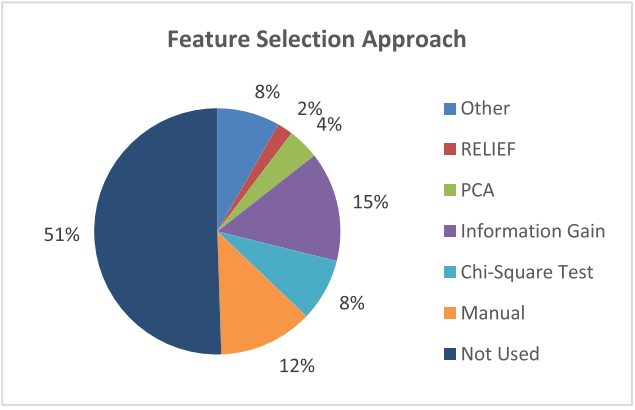

FIGURE 8. Distribution of feature selection algorithm employed by 48 studies.

ods. The selection process in these techniques is heavily influenced by the researcher's understanding of the contextual relevance and domain-specific significance of the traits. Nonetheless, these manual techniques are susceptible to subjectivity and the potential inclusion of bias arising from the researcher's own interpretations, preferences, or preconceived notions about the importance of certain characteristics. In contrast, established feature selection algorithms operate based on predetermined mathematical or statistical criteria. These criteria provide a more objective and data-driven foundation for feature selection, reducing the potential for bias introduction.

In Fig. 8 and Table 11, we show the distribution of the feature selection algorithms used in hybrid detection studies. Our analysis of these studies reveals that most of them either did not use or did not explicitly mention any specific feature selection technique in their works. Out of the 88 studies reviewed, only 48 studies utilized feature selection techniques.

2) FEATURE SELECTION MECHANISM

The feature selection mechanism in hybrid malware detection can be applied in two ways: separately to static and dynamic features or to their combined representation. The accuracy and effectiveness of the resulting detection model can vary based on the chosen approach. In the following discussion, we will explore several feature selection techniques commonly used in hybrid malware detection.

TABLE 12. Summary of feature selection mechanisms.

Feature Selection Mechanism	Related Studies
Both features, but separated	[123] [64] [115] [52] [43] [48] [118] [105] [94] [56] [97] [98]
Only static features	[92] [50] [122] [79] [42] [45] [106] [61]
Only dynamic features	[12] [41] [63]
On the combination	[57] [72] [91] [47] [58] [88] [126] [56] [101] [78] [83] [107]

1. Both features, but separated: in this approach, feature selection is performed on static and dynamic features independently, and the selected features are later merged. Several studies, including those by Hussain et al. [48], Li et al. [123], and Maryam et al. [52], are just a few examples that utilized this feature selection mechanism in their research.
2. Only static features: in this approach, the feature selection technique is only applied to the static feature. Arshad et al. [45] opted for this approach as their proposed scheme generated highly diverse features through static analysis compared to dynamic analysis. Similarly, other researchers, including Feng et al. [42], Shijo and Salim [92], and Patel and Buddadev [79] chose this approach to address specific challenges encountered in their respective works.
3. Only dynamic features: in this approach, the feature selection technique is only applied to dynamic features. Liu et al. [41] utilized feature selection by clustering the function call graph, which contains multiple dimensions. The resulting clusters are then used as feature dimensions to facilitate the subsequent classification process.
4. On the combination: in this method, the feature selection technique is applied after the fusion of static and dynamic features. This methodology has been used by a number of researchers, such as Gera et al. [88], Lindorfer et al. [72], Saif et al. [126], and others. Combining static and dynamic features and applying feature selection enhances malware detection accuracy by leveraging their complementary strengths and selecting the most informative and discriminative features.

Out of the 88 studies examined, 48 of them utilized a feature selection mechanism. However, it is noteworthy that 13 of the 48 studies did not provide a clear description of the feature selection techniques used. In Table 12 and Fig. 9, we show the distribution of the selection techniques that were used by the 35 studies.

In several studies, the feature selection process often narrows down to a subset of attributes, often exclusively focusing on static or dynamic features. For instance, studies by Roy et al. [106], Li et al. [122], and Feng et al. [42] tend to emphasize the selection of static features, while studies by Anupama et al. [63], Liu et al. [41], and Arora and Peddoju [12] direct their attention toward dynamic ones. The primary concern arises from the fact that when the feature selection algorithm is exclusively applied to one type of fea-

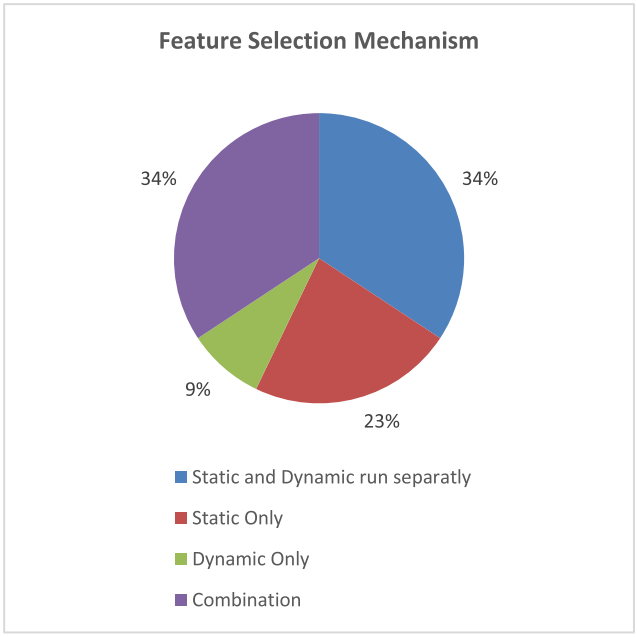

FIGURE 9. The distribution of feature selection mechanism.

ture, it may overlook the intricate interplay and synergies that exist between static and dynamic attributes. Consequently, the algorithm may fall short of identifying the most optimal features within the context of the combined feature set. As a result, this approach may lead to the development of a detection system that is either less effective or less accurate.

D. WORKING ENVIRONMENT

The development of hybrid malware detection techniques requires particular working environments and conditions. The working environment entails the whole process, from feature detection to extraction and processing. The working environment can be any or a combination of the followings: an emulator, a real device, a local computer, or the cloud. There are several approaches proposed by researchers related to choosing a suitable working environment for the research objective. This section explains the types and configurations of different working environments to answer RQ4.

In several studies, the specific context or working environment for the malware analysis and detection process is not clearly described, marked as ''Not Clear''. It is worth mentioning that some studies, such as those by Imtiaz et al. [99] and Oliveira et al. [104], utilized text-based datasets, which do not require a specific working environment for analysis.

In Fig. 10 and Table 13, we present a summary of the working environments utilized in the reviewed studies. The findings indicate that local computers and emulators remain the predominant choices for conducting research in this field. However, it is noteworthy that six studies did not specify the specific working environment used in their research.

1) LOCAL COMPUTER AND EMULATOR

In this approach, the static features are extracted from the APK file on a local computer. On the other hand, the dynamic features are obtained by running the app on an emulator. Various tools and techniques are employed to extract the dynamic features from the emulator. For instance, Ahmed et al. [43] utilized APK Tool software to decompile APK files, facilitating the extraction of resources like manifest files and Java Archive (JAR) files. From these resources, static features were derived, encompassing permissions from the manifest file, network information, and suspicious text features from the JAR file. Additionally, dynamic features were captured to comprehend the application's behavior in an emulator, with system call features acquired through the Strace tool, while CPU and memory usage features were collected via the ADB monitor. The working environment scheme employed by Ahmed et al. is illustrated in Fig. 11.

TABLE 13. Summary of working environment.

Working Environment	Related Studies
Local Computer and Emulator	[46] [84] [68] [85] [95] [119] [66] [69] [51] [86] [121] [74] [100] [103] [127] [81] [80] [54] [93] [59] [82] [77] [60] [131] [57] [72] [91] [88] [126] [56] [41] [87] [49] [53] [128] [123] [64] [52] [43] [48] [118] [94] [89] [92] [50] [79] [101] [62] [63] [78] [125] [83] [61] [102] [98]
Local Computer and Real Device	[117] [124] [75] [96] [55] [129] [130] [12] [115] [42] [45]
Cloud and Emulator	[90] [120] [71] [58] [105]
Cloud and Real Device	[67] [76] [122]
Not Clear	[65] [70] [73] [11] [109] [14]
Using text-based Dataset	[116] [99] [47] [104]

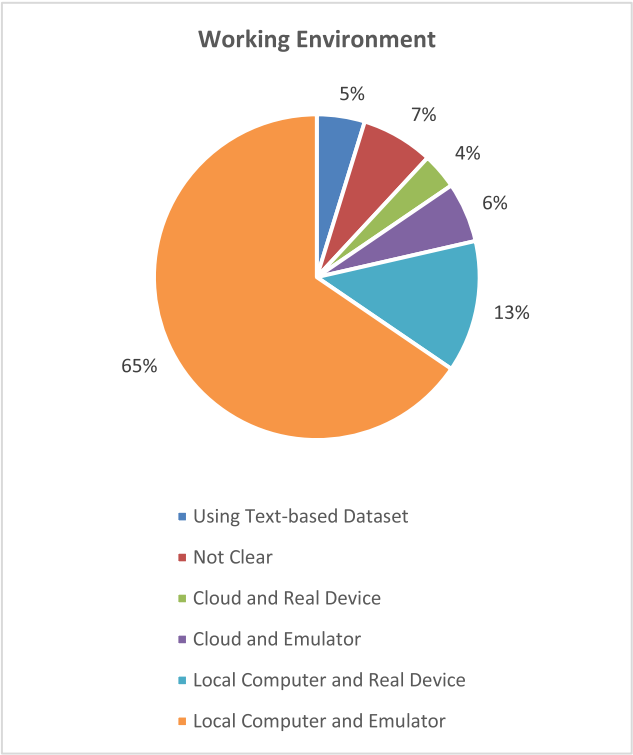

FIGURE 10. Working environment statistic.

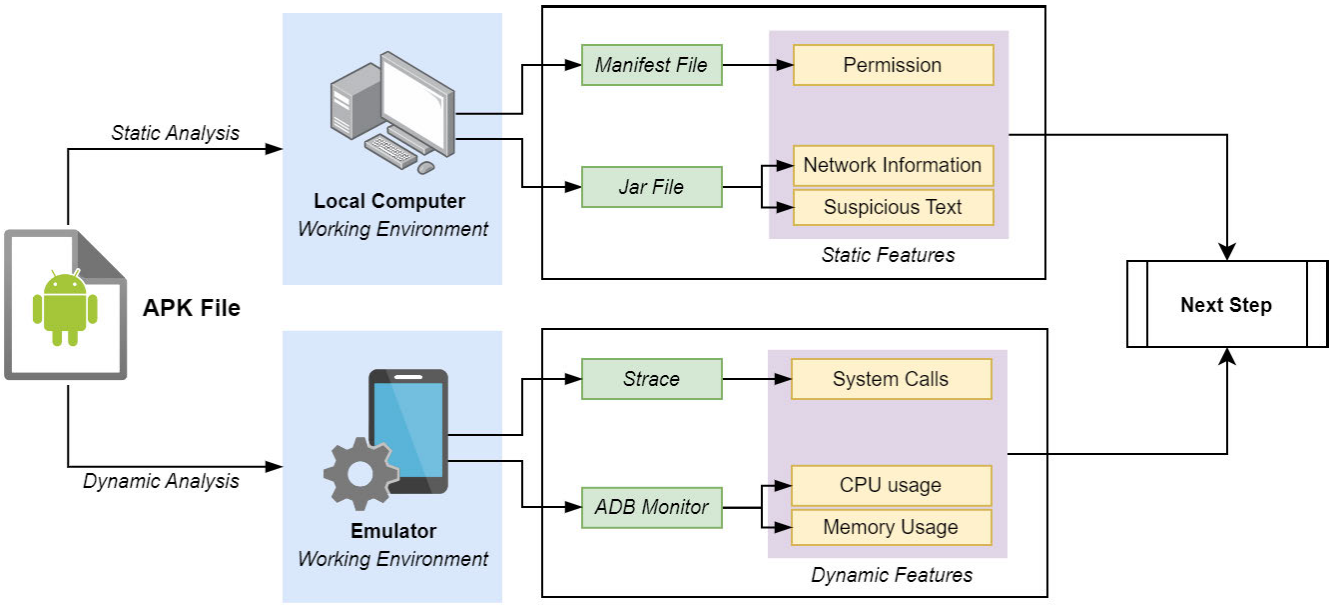

FIGURE 11. The schema of the working environment employs a local computer and emulator used by Ahmed et al. [43].

In another study, Kapratwar et al. utilized Android Debug Bridge (ADB) to execute app samples, MonkeyRunner to trigger random events, and Strace to record monitored system calls [50]. Jadhav et al. utilized AndroSandX to execute the app, capturing system calls, file operations, network activity, and other suspicious activities during dynamic analysis [119].

The predominant approach employed in hybrid analysis across numerous studies involves utilizing a local computer and emulator as the working environment scheme. One of the main motivations for adopting this scheme is its ease of system configuration. However, findings from studies conducted by Sinha et al. [132] and Alzaylaee et al. [133] have raised an important caveat. These studies suggest that executing malware within an emulator does not yield results that surpass those obtained on a real device. This phenomenon has the potential to diminish the overall accuracy of the detection system.

2) LOCAL COMPUTER AND REAL DEVICE

Monitoring executed apps on an emulator can pose challenges, as malware can detect and evade suspicious activity in emulator mode. To overcome this, some researchers try to execute the apps on real devices. Liu et al. utilized a real device to generate logs and saved the feature vector to a file. They used Android Debug Bridge (ADB), which was connected to a local computer to monitor the app using the Strace tool, and dynamic features were extracted using the Monkey tool [73]. Alzaylaee et al. utilized DynaLog on a real device to execute the app and employed various testing generation tools to trigger malicious activities. The subsequent processes, including feature selection with WEKA tools, were performed outside the real device [115]. Arora&Peddoju used a real device to run an app for three days to extract network activity as dynamic features, and the Apktool was used to extract static features such as permissions [12].

While dynamic feature extraction on real devices may offer certain advantages compared to emulators [132], it is important to note that hybrid working environment schemes of this nature do not provide on-device protection. This limitation arises because the extraction of static features and the detection process continue to take place on the local computer rather than being conducted directly on the device.

3) CLOUD AND EMULATOR

Hybrid malware detection can impose a significant computational burden, particularly when executed on the smartphone side. To alleviate this workload for users or clients, the analysis and detection processes are predominantly carried out on the server or cloud side. Wen&Yu [58] conducted the static analysis using Androguard, followed by dynamic analysis using DroidBox. The results of both analyses were combined and classified using the SVM algorithm. Conversely, Rodrigo et al. [105] performed malware analysis and detection on Amazon servers, with clients responsible for submitting samples and receiving analysis results.

The implementation of cloud computing into malware detection schemes has the potential to alleviate the computational burden. Nevertheless, it is crucial to acknowledge that extracting features from malware by running it on an emulator is not preferable to executing it on a real device.

4) CLOUD AND REAL DEVICE

To overcome the computational limitations of mobile devices, conducting a detection process on the cloud or remote servers is a viable solution. However, dynamic analysis is still performed on real devices to ensure effective detection and prevent malware from evading analysis [76], [122].

Bhandari et al. [67] developed an anti-malware solution that extracted dynamic features and some static features on

TABLE 14. A summary of the configurations of detection order.

Detection Order	Related Studies
Combination	[52] [81] [118] [116] [90] [84] [74] [58] [105] [67] [47] [104] [46] [85] [119] [69] [51] [86] [121] [100] [103] [127] [54] [93] [59] [82] [77] [60] [57] [72] [91] [88] [126] [56] [87] [49] [53] [128] [123] [64] [43] [48] [94] [92] [50] [79] [117] [124] [76] [55] [129] [130] [12] [115] [65] [70] [73] [109] [101] [62] [63] [107] [61] [102] [78] [108] [125] [106] [83] [98] [97]
Independently	[68] [80] [75] [74] [120] [99] [95] [66] [41] [45] [84] [90] [116] [118]
Static First	[71] [122] [131] [89] [96] [42] [11] [14] [81] [52]

real devices. On the server, static analysis of DEX files was performed to extract API calls, IP addresses, and URLs. The server was also responsible for conducting the detection process using various machine-learning algorithms. However, researchers have a way to perform the entire on-device detection process. Nonetheless, DEX file analysis requires extensive computational resources, necessitating its execution on the server. Nevertheless, researchers have the option to implement multi-level detection, encompassing on-device analysis for real-time protection and cloud-based analysis if on-device detection fails to identify malware. Fig. 12 illustrates the working schema employed in the study.

E. DETECTION ORDER

In this section, we will explore the different detection configurations used in the hybrid approach that combines static and dynamic features. Specifically, we will explain how these two types of features are incorporated into the detection mechanism. The section will focus on the arrangement of static and dynamic features during the detection phase.

As explained in the previous sections, static features and dynamic features can be utilized separately or in combination, depending on the specific situation. The decision to use either approach or a combination of both depends on factors such as the deployment environment and resource considerations. The resulting detection models may vary in terms of accuracy and performance. In this section, we describe the different orders of detection used in the hybrid analysis, addressing RQ5. The various approaches to detection order are presented and summarized in Table 14 and Fig. 13. The hybrid analysis employs several different detection orders, including combination, static-first, and running independently.

1) COMBINATION OR FUSION

Several studies have demonstrated that combining static features and dynamic features in a single detection cycle leads to better accuracy compared to using these features separately [11], [12]. In this approach, the working environment for feature extraction may vary. For example, Hussain et al. employ an emulator to extract dynamic API calls and analyze manifests for permissions and intents on a local computer [48], while Bhandari et al. utilize a real device and cloud infrastructure. Despite the variations in the working environments, both types of features are combined throughout the detection process [67]. In general, the illustration of the combination of static and dynamic features in the detection schema could be represented by the study of Hussain et al. in Fig. 14.

The combination of all feature types within the malware detection process may pose significant computational overhead. Researchers often contend with the intricate challenge of discerning the precise feature types to integrate into the detection framework, a choice often swayed by subjective factors. Remarkably, a notable paucity of academic literature is dedicated to the systematic exploration of feature-type combinations tailored for optimal computational efficiency.

2) STATIC FIRST

This approach employs static features for detection in the initial stage. If the malware is not detected, it then uses dynamic features for detection in the second stage. The purpose of this two-stage detection is to reduce computational costs. Zhou et al. proposed a two-step Android malware detection approach. In the first step, they relied on the permission list to filter and classify apps as either benign or malware. However, they found that some unknown malware can bypass this filter. In the second step, they monitored the app's behavior through system calls to capture dynamic app behavior [14]. Taheri et al. used permissions and intents as static features, while dynamic features were obtained from network flows and API calls. They employed a layered detection approach, where the first layer used static features to detect malware, and the second layer used dynamic features to identify malware families [96]. Kandukuru&Sharma implemented a two-step detection schema where they first extracted the list of permissions from the manifest file and used Jaccard bitwise similarity to identify malware. If the similarity score falls below a certain threshold, dynamic analysis is conducted as the next step [70]. Fig. 15 illustrates the detection order schema used by Kandukuru&Sharma.

Although static feature analysis requires fewer computational resources than dynamic feature analysis, it is noteworthy that most research still employs the traditional method of conducting static feature extraction on a particular computer. This observation underscores the prevailing focus in many studies on enhancing detection performance rather than delving into the practical implications of such research for the protection of Android end users.

3) RUN INDEPENDENTLY

This approach involves running hybrid Android malware detection using static and dynamic features separately. Kapratwar et al. proposed an anti-malware method that performs detection with static and dynamic analysis as separate processes. At the end of each detection process, the app is classified as benign if both static and dynamic analysis indicate it is benign. If both analyses indicate the app

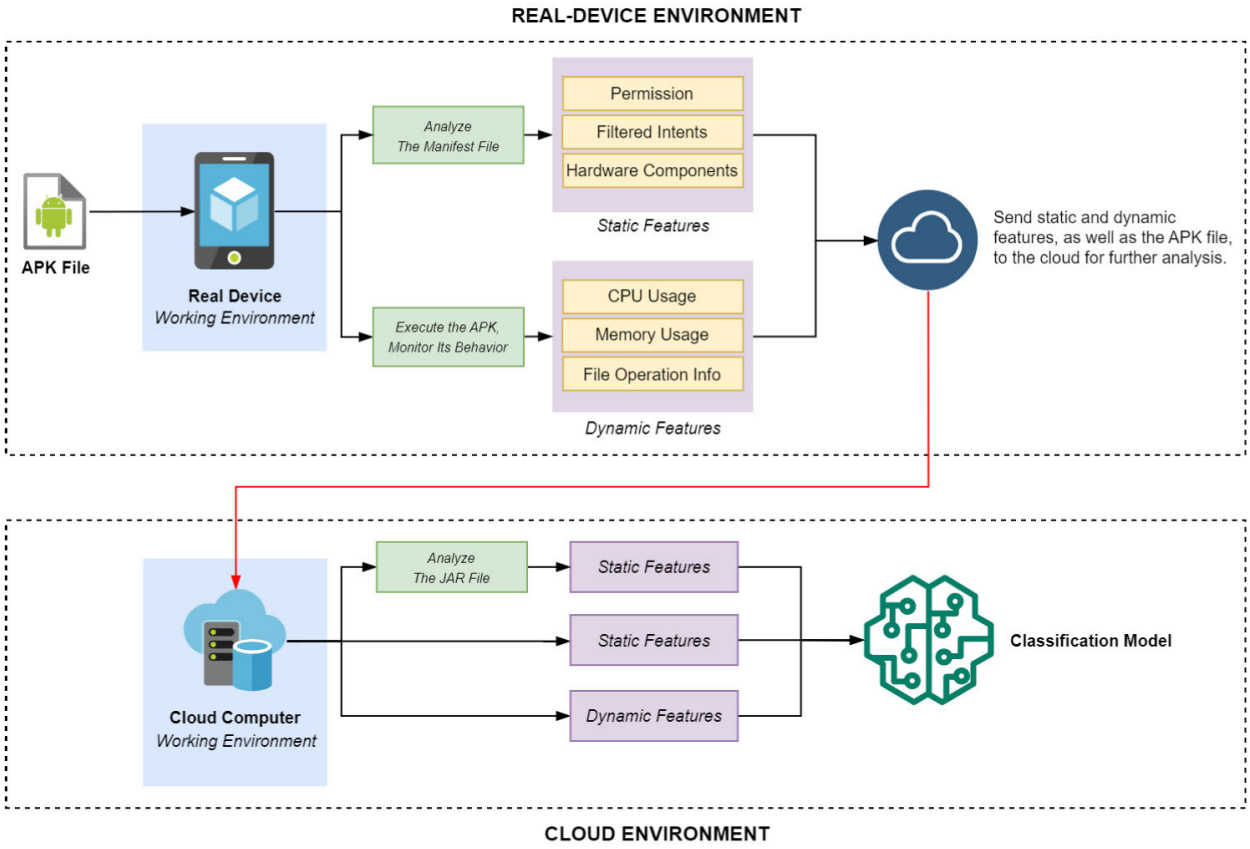

FIGURE 12. Schema of working environment used by Bhandari et al. that utilize cloud and real-device [67].

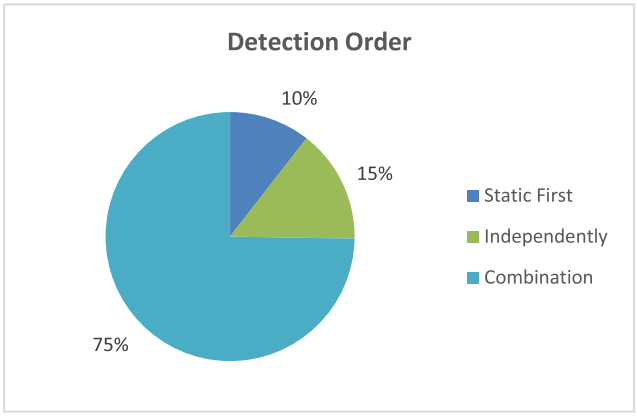

FIGURE 13. A summary of the detection order approaches.

is malware, it is classified as malware. The app is classified as risky if the static and dynamic analysis results are contradictory [50]. Jannat et al. developed a malware detection system with static and dynamic features that also run as separate processes. Different machine-learning algorithms were used to classify the features. The experimental results showed that dynamic analysis achieved higher accuracy than static analysis [66]. Arshad et al. employed SVM to analyze Android malware and then implemented a rule-based system. If both the static and dynamic analyses classify the app as legitimate, it is considered benign. If both analyses classify the app as malicious, it is categorized as malware. The app is classified as risky if there is a discrepancy between the static and dynamic analyses [45].

In situations where combining static and dynamic features is not feasible, malware detection can be performed by separating them. For example, Chaulagain et al. [84] use sequences of API calls as static features and sequences of system calls as dynamic features. The detection process is conducted separately. Combining these two types of features is challenging due to their distinct characteristics.

Within this schema, various studies conducted by researchers such as Arshad et al. [45], Dhalaria and Gandotra [118], Imtiaz et al. [99], and others should consider combining static and dynamic features in their detection processes, particularly when these features involve attributes related to frequency or appearance. For example, attributes like occurrences of some permissions, activity counts, and system call frequencies. Combining static and dynamic features in these scenarios has the potential to enhance detection outcomes.

F. DETECTION SCHEMA

This section describes the integrity of the detection system in order to answer RQ6. The detection scheme outlines the

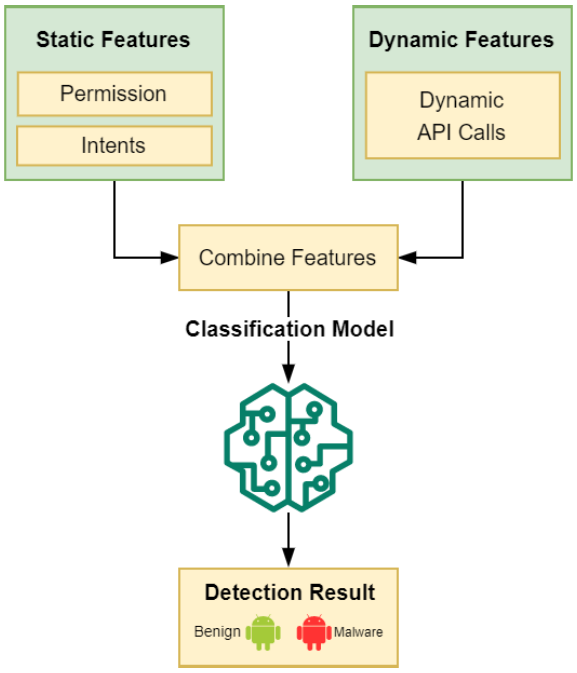

FIGURE 14. Static and dynamic feature combination in Hussain et al.'s detection schema [48].

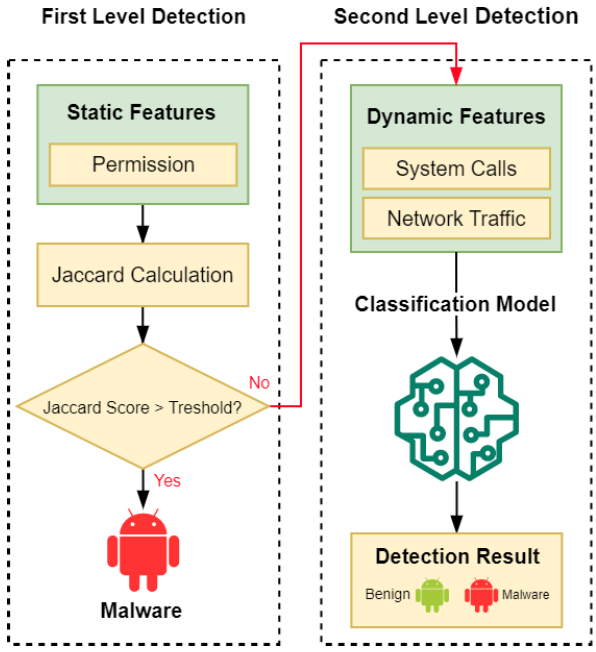

FIGURE 15. Utilization of static-first features in detection order schema by Kandukuru&Sharma [70].

relationship between different processes involved in acquiring the APK file, extracting features, processing them, and making detection decisions. There are two types of detection schemes: integrated and non-integrated.

1) INTEGRATED SCHEME

In an integrated scheme, all processes are tightly integrated and interconnected, facilitating a seamless flow of information and decision-making. This approach ensures efficient communication and coordination between different stages of the detection process. The integrated approach often prioritizes system usability by presenting a simplified user interface that hides the underlying complexity. For instance, Rodrigo et al. employed a client-server architecture for their detection system [105], enabling users to interact with it through a mobile app. The client submits an APK file to the server for analysis and verification, and the server processes the APK, returning the results to the client.

In their study, Sun et al. [76] performed static and dynamic feature extraction on real devices. Nevertheless, the practical implementation of this approach poses challenges for end-users, primarily due to the requirement for specialized or root-level access to facilitate system call extraction. The detection process is performed on the server due to computational cost considerations. However, there is a chance to do on-device detection by removing features that consume a significant amount of resources. An alternative approach that merits consideration involves the adoption of a multi-tiered detection strategy, commencing with on-device detection utilizing static features, followed by dynamic feature detection within a cloud-based infrastructure.

2) NON-INTEGRATED SCHEME

Malware detection involves multiple processes, such as feature extraction, feature processing, classification, and related tasks. In non-integrated detection schemes, these processes often require human intervention to establish connections between them. As a result, the processes do not operate automatically and seamlessly. This leads to fragmented communication and coordination among different components, which is in contrast to integrated schemes wherein all operations are seamlessly interconnected to optimize efficiency. Researchers who do not clarify the interconnections between processes often resort to non-integrated detection methods.

Sugunan et al. [53] extract both static and dynamic features from different locations, with static feature extraction carried out on a local computer using APK Tool software and dynamic features derived from a separate device using an emulator and Droidbox software. In order to proceed further in the classification process, human intervention is required to extract and transform both static and dynamic features into text or CSV formats and integrate them into the Weka software environment. As shown in Fig. 16, this non-integrated method requires human intervention to detect malware.

The primary objective of the research that uses a non-integrated scheme is to conduct experimental experiments with the goal of improving the outcomes of detection processes. Although numerous studies have achieved a commendable level of accuracy, surpassing 90% [45], [85], [126], demonstrating significant effectiveness in reducing malware risks, the exclusive focus on detection performance and accuracy can result in overly complex solutions that are difficult to implement in real-world scenarios. Despite its relative effectiveness, this approach underscores the continued importance

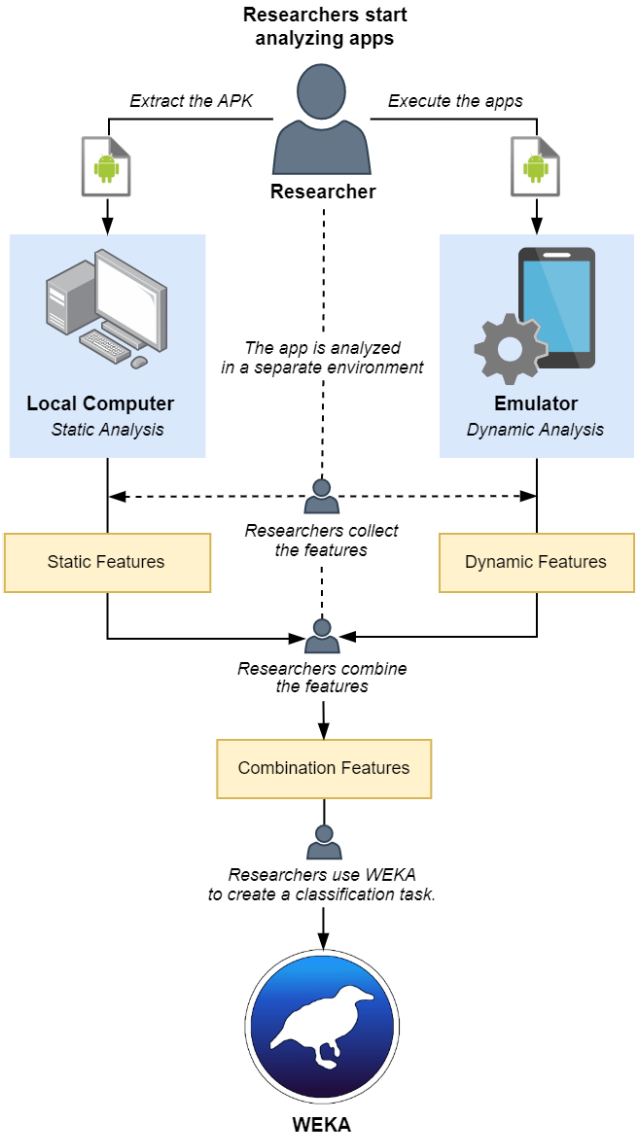

FIGURE 16. Sugunan et al. [53] employ a non-integrated scheme that involves human intervention at each step.

of human involvement in non-integrated detection methods. As a result, the practical relevance of the non-integrated approach in real-world protection scenarios becomes less significant.

The distribution of the detection schema is shown in Table 15 and Fig. 17. Our analysis reveals that the majority of the studied papers, 95.45% (84 papers), focused on the non-integrated scheme for malware detection. In contrast, a smaller portion of the papers employed an integrated detection scheme.

G. DETECTION ALGORITHM

This section provides an overview of the detection approaches employed by various studies addressing RQ7. The choice of detection approach is influenced by several factors, including the availability of the dataset, accuracy, and the ability to detect new malware. The distribution of the detection approaches is summarized in Table 16 and depicted in Fig. 18.

TABLE 15. Summary of detection schema.

Detection Schema	Related Studies
Non-Integrated	[67] [47] [104] [46] [85] [119] [69] [51] [86] [121] [100] [103] [127] [54] [93] [59] [82] [77] [60] [57] [72] [91] [88] [126] [56] [87] [49] [53] [128] [123] [64] [43] [48] [94] [92] [50] [79] [117] [124] [55] [129] [130] [12] [115] [65] [70] [73] [109] [68] [80] [75] [74] [99] [95] [66] [41] [45] [84] [90] [116] [118] [71] [122] [131] [89] [96] [42] [11] [14] [81] [52] [101] [62] [63] [107] [61] [102] [78] [108] [125] [106] [83] [98] [97]
Integrated	[58] [105] [76] [120]

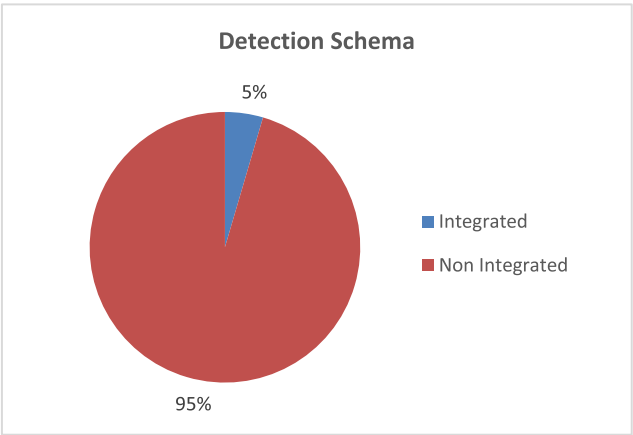

FIGURE 17. Detection schema summarization.

TABLE 16. The distribution of the machine learning algorithms that are used as part of the hybrid detection solutions.

Detection Algorithm	Related Studies
Knowledge-based	[76] [68] [79] [65] [120] [12] [131] [14]
Supervised Learning	[129] [47] [99] [116] [124] [105] [69] [72] [49] [94] [73] [41] [45] [122] [89] [85] [86] [100] [103] [127] [59] [50] [81] [58] [67] [46] [119] [121] [54] [82] [91] [88] [87] [53] [128] [123] [64] [43] [48] [92] [55] [130] [70] [109] [80] [75] [95] [66] [118] [71] [96] [11] [52] [101] [62] [63] [107] [61] [102] [78] [108] [125] [106] [83] [98] [97]
Unsupervised Learning	[117]
Semi-supervised Learning	-
Anomaly Detection	[56] [57]
Deep Learning	[115] [104] [51] [93] [60] [126] [74] [77] [84] [90] [42]

1) KNOWLEDGE-BASED

In this approach, experts create preconfigured and predetermined attack patterns to determine the maliciousness of a

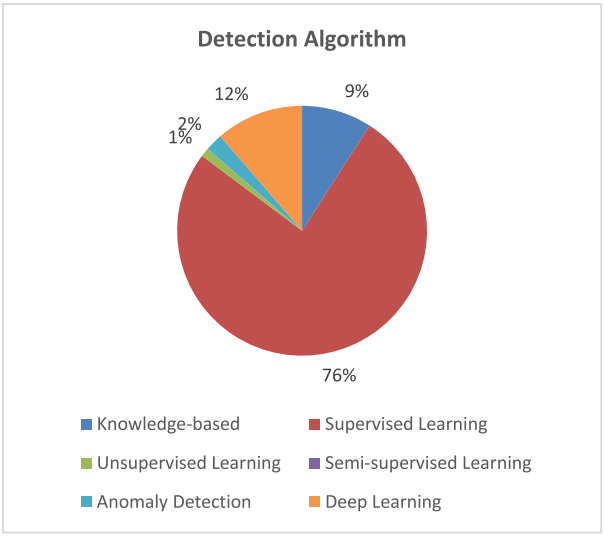

FIGURE 18. The distribution of the detection algorithms used in the hybrid approaches.

program. Several studies have employed this method. For instance, Arora&Peddoju developed NTPDroid, a system that uses both permission and network activity features as combined attributes for malware detection [12]. First, they generated frequent patterns by combining traffic features and permissions for malware and benign datasets. They then employed malware scores and benign scores to quantify the similarity of an app being tested to the malware and benign databases. Next, they analyzed whether any patterns derived from the app being tested corresponded to the frequent patterns in the malware and benign databases. The malware and benign scores were adjusted based on these comparisons. Finally, the application was classified as malicious if the malware score exceeded the benign score; otherwise, it was classified as benign. Algorithm 2 provides a detailed explanation of Arora&Peddoju's knowledge-based detection approach.

Eder et al. introduced ANANAS, an Android malware analysis framework [68]. The framework utilized various plugins to generate logs for malware analysis. In the static analysis phase, the researchers extracted crucial information from the manifest file, including permissions, services, receivers, and package names. Additionally, they monitored and recorded file system changes by comparing the file system before and after running the app on an emulator. The researchers captured system calls, API calls, and network activity for dynamic analysis while the app was executed on the emulator. The framework also included a plugin for searching the APK file's hash in VirusTotal for additional analysis and detection. In this study, an app is classified as malware if its behavior matches the patterns identified by Eder et al. and exhibits characteristics typical of malicious activity, such as unauthorized access to sensitive data, sending unauthorized text messages, or suspicious network activity.

The implementation of a knowledge-based detection system employing a rule-based approach, as exemplified by the works of Jang [120], Patel&Buddadev [79], and Arora&Peddoju [12], offers the advantage of enabling experts to precisely and transparently define the attributes of applications categorized as malware. However, these systems face a series of challenges, particularly in the realm of handling unstructured data, and they exhibit a degree of inflexibility when addressing scenarios that do not conform to pre-established rule sets.

Algorithm 2 Knowledge-Based Detection Method Proposed by Arora&Peddoju [12]

```
# Load benign frequent patterns
benign_freq_pattern ← Load_Frequent_Patterns(benign_db)
 # Load malware frequent patterns
malware_freq_pattern ← Load_Frequent_Patterns(malware_db)
 # Measure the relative frequency of occurrence
# of a pattern in the dataset
normalized_sprt ← FP_Growth(dataset)
 # Extract static & dynamic features from the app
apk_file ← "com.example"
permission_ft ← Extract_Permission(apk_file)
network_ft ← Extract_Network_Act(apk_file)
 # Combine static and dynamic features
combination_pattern ← Combine(permission_ft,network_ft)
 # Set initial score values for malware and benign
benign_score ← 0
malware_score ← 0
 # Calculate the score for benign and malware
for pattern in(combination_pattern) do
     # Check if the pattern matches frequent patterns
    # in the benign database
    if pattern == benign_freq_pattern do
        benign_score ← increment(benign_score) +normalized_sprt
     # Check if the pattern matches frequent patterns
    # in the malware database
    if pattern ==malware_freq_pattern do
        malware_score ← increment(malware_score)+normalized_sprt
 # Classify the app as malware or benign
if  malware_score> benign_score then
    print "Malware"
else
    print "Benign"
```

2) SUPERVISED LEARNING

In this approach, a detection model is built using a known and labeled dataset. The learning process involves providing pairs of inputs and targets to train the model. The goal is to generalize the training data and develop an effective detection model. Several studies have utilized supervised learning algorithms in the detection process. Taheri et al. [96] have developed a two-tier detection process that utilizes the Random Forest algorithm as a classifier. In the first phase, they employ static features, which include permissions and intents, comprising a total of 8,115 distinct features. These static features are then used as input to the Random Forest classifier to identify potentially suspicious software. Suppose the first tier determines that the tested application exhibits suspicious characteristics. In that case, it proceeds to the second tier, where a more comprehensive analysis takes place to classify the malware category and family. This second tier involves the use of dynamic features, such as network flow and API

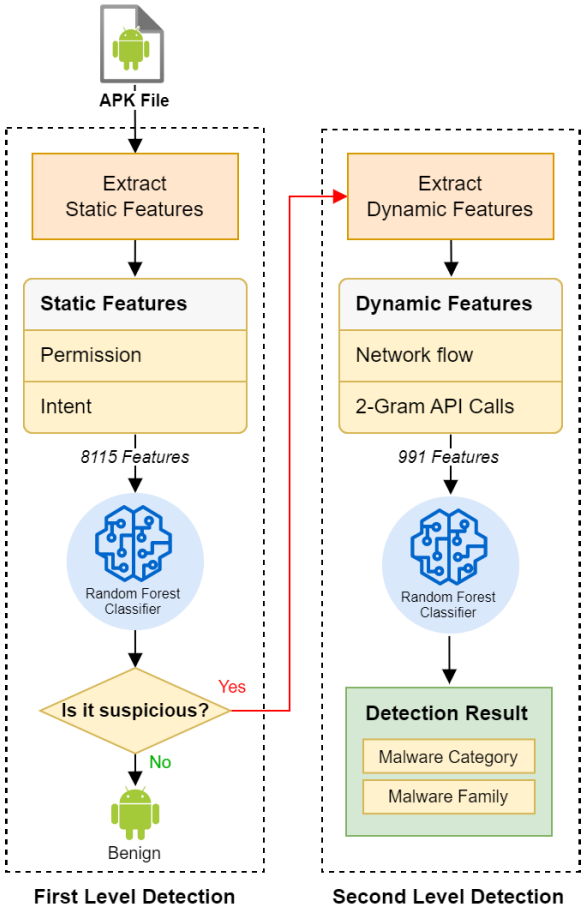

FIGURE 19. The detection schema, as proposed by Taheri et al. in reference [96], incorporates the use of the Random Forest classifier.

calls, which consist of a set of 991 features. Fig. 19 illustrates the detection schema used by Taheri et al., which incorporates the Random Forest classifier.

Alazab et al. employed a combination of permissions for static features and API Calls for dynamic features in their malware detection approach. They tested five machine learning algorithms, namely random forest, J48, random tree, kNN, and Naïve Bayes, using 10-fold cross-validation. The F-Measure results were 94.3%, 90.9%, 89.1%, 87.9%, and 91.8%, respectively [134]. Sugunan et al. utilized a combination of permissions and API Calls and employed Naive Bayes, Random Forest, SVM, and J48 algorithms. The detection performance improved when feature selection was applied to the feature combination [53].

Surendran et al. employed a combination of permissions and API Calls as static features and System Calls as dynamic features for classifying Android malware. Their study investigated the interdependency between the static and dynamic features used in machine learning to address multicollinearity issues. The machine learning algorithm utilized in their work was the Tree Augmented Naive Bayes [54].

Most research utilizing supervised learning techniques for threat detection has consistently produced positive results. However, it is important to highlight that these studies often neglect to consider the potential impact of adversarial attacks. Adversarial attacks have the ability to deceive pre-trained models, introducing the possibility of undetected malicious threats. The vulnerability of models employing supervised learning to adversarial attacks was thoroughly examined in a study conducted by Rafiq et al. [108]. Through a comprehensive series of tests encompassing mimicry attacks, FRA, and MFRA, it became evident that these models were susceptible to such adversarial tactics. Rafiq et al. have proposed several potential solutions to mitigate this vulnerability. These include the implementation of adversarial training techniques, the adoption of multiple classifier systems, and the incorporation of input randomization strategies.

3) UNSUPERVISED LEARNING

In this method, the unsupervised learning algorithm operates independently, discovering the most suitable model that fits the given dataset on its own. This technique is commonly used in clustering problems. Arora et al. [117] employed the K-Medoids algorithm to construct a model of data clusters. The feature vector was created using permissions and network activities. Due to the presence of feature vectors containing strings, the researchers chose to use K-Medoids instead of K-Means. In their work, the algorithm divided the feature vector into two distinct clusters. The researchers utilized the Manhattan distance and Levenshtein distance formulas to assess the similarity between vectors.

However, Arora et al. do not provide explicit information regarding the dataset used, which raises concerns about potential bias in their research. Inadequate dataset size may hinder the clustering model's ability to detect significant patterns within the data. Furthermore, manually selecting the value of K may introduce bias. For instance, choosing a K value that is disproportionately large compared to the available data may result in uninformative clusters. Conversely, opting for an excessively small K value may lead to the inclusion of clusters with substantial data variations. To mitigate bias, determining the optimal K value can be aided by machine learning techniques.

4) ANOMALY DETECTION

This method involves detecting anomalies, which refer to variations from normal behavior [135]. These anomalies are identified based on data and hypotheses regarding the underlying mechanism. In their study, Wang et al. [57] integrated anomaly detection into their detection system. They trained the One-Class SVM exclusively with benign applications and utilized a combination of features. The One-Class SVM, designed to separate normal data from anomalous data, maximizes the margin between normal data points and the decision boundary. The underlying algorithm identifies a hyperplane that includes normal data points while excluding anomalous data points.

The One-Class SVM algorithm was implemented using the sklearn.svm.OneClassSVM module within the scikit-learn

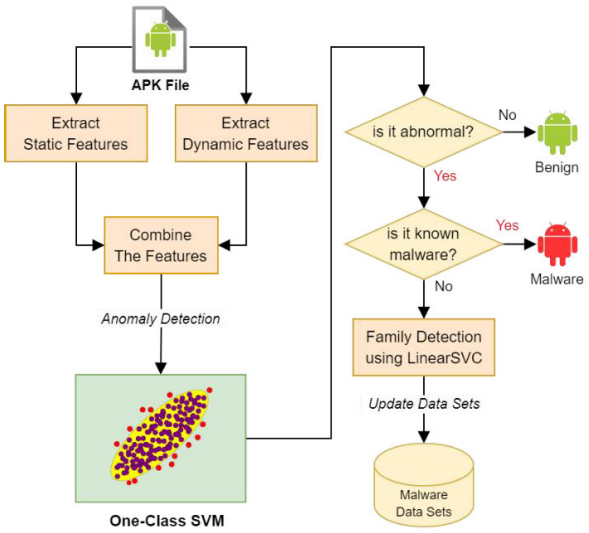

FIGURE 20. Anomaly Detection Schema by Wang et al. [57].

library. When an unknown app is submitted, its feature vector is provided as input to the classifier, which determines whether it is malware or non-malware. If the unknown app is classified as anomalous and not recognized as known malware, a LinearSVC classifier is activated to classify it and determine its malware family. Fig. 20 illustrates the utilization of the anomaly detection schema employed by Wang et al.

In another research, Wang et al. utilized a combination of misuse detection (Linear SVC) and anomaly detection (One-Class SVM) in their work. They used a combination of static and dynamic features for detection [56]. The static features were obtained from the manifest file and disassembled code, including hardware components, activities, intents-filter, permissions, and the application's native code. The dynamic features included file access, register receiver, executed commands, content resolver queries, telephony manager listener, ''find resource'', dynamic suspicious calls, SMS, phone event, data leak, and network operations.

The efficacy of anomaly detection, particularly when employing the One-Class SVM algorithm, could be influenced by the composition or distribution of the dataset. In their investigation, Wang et al. [57] employed a balanced dataset wherein the number of samples across different classes is roughly equivalent. Within such a balanced dataset, the One-Class SVM tends to yield favorable results. However, a study conducted by Dreiseitl et al. [136] indicates that in imbalanced datasets, where instances of anomalies constitute a minority, the One-Class SVM may outperform the two-class SVM. Consequently, the careful selection of algorithms and their alignment with the dataset's specific characteristics emerges as crucial considerations in the development of an effective anomaly detection system.

5) DEEP LEARNING

Deep learning, which is based on artificial neural networks with multiple layers, has gained significant attention in recent years for its potential in various fields, including Android malware detection. Researchers have proposed deep learning-based approaches to improve detection accuracy. For instance, Zhang et al. [60] created the CoDroid system, which uses a CNN-BiLSTM model to analyze both opcode sequences as static features and system call sequences as dynamic features. To accomplish this, word embedding is employed to transform the sequences into low-dimensional vector representations, effectively capturing both semantic and syntactic similarity. Specifically, the opcode sequence is fed into the CNN layer, which employs convolutional filters to extract essential patterns and structures from the opcode data. Meanwhile, the system call sequence is directed to the BiLSTM layer, which captures contextual information and dependencies between individual system calls.

The outputs of the CNN and BiLSTM layers are combined through an attention mechanism, which emphasizes the significant features in the sequences. This mechanism prioritizes the elements that have a greater impact on the classification of benign or malware and subsequently directs the output to a fully connected output layer for classification, making predictions about whether the application is benign or malicious. Fig. 21 provides a visual representation of the CNN-BiLSTM model employed by Zhang et al. in their detection system.

Xu et al. developed a hybrid-based malware analysis using deep learning techniques [93]. They utilized static features derived from permissions, API calls, intent filters, and providers, while dynamic features were obtained from system calls. The static features were transformed into vectors and trained using deep neural networks (DNN), while the dynamic features were represented as a graph and processed using a graph kernel. The outputs from the DNN and graph kernel were combined into a kernel matrix and input into a support vector machine (SVM) for classification. The resulting model achieved an impressive accuracy of 94.7%. This demonstrates the effectiveness of deep learning in improving Android malware detection.

Xu et al. represent system calls as n-gram graphs, with n values ranging from 1 to 4. The findings indicate that an n value of 4 achieved the highest level of accuracy, reaching 87.3%. However, it's imperative to acknowledge that inaccuracies in selecting the appropriate n value can result in the truncation of information with varying degrees of precision. The challenge arises when certain system calls exhibit a higher frequency of occurrence compared to others, potentially occurring within sequences of sensitive system calls. This situation raises concerns about information distortion and potential inaccuracies during the analysis process. In light of these challenges, implementing a strategic approach involving the categorization of system calls based on their specific functionalities [137] emerges as a viable solution to address this issue.

Yuan et al. conducted Android malware detection using a combination of static and dynamic features [82]. The static features included permissions extracted from the Android manifest file, while sensitive API calls were obtained from the

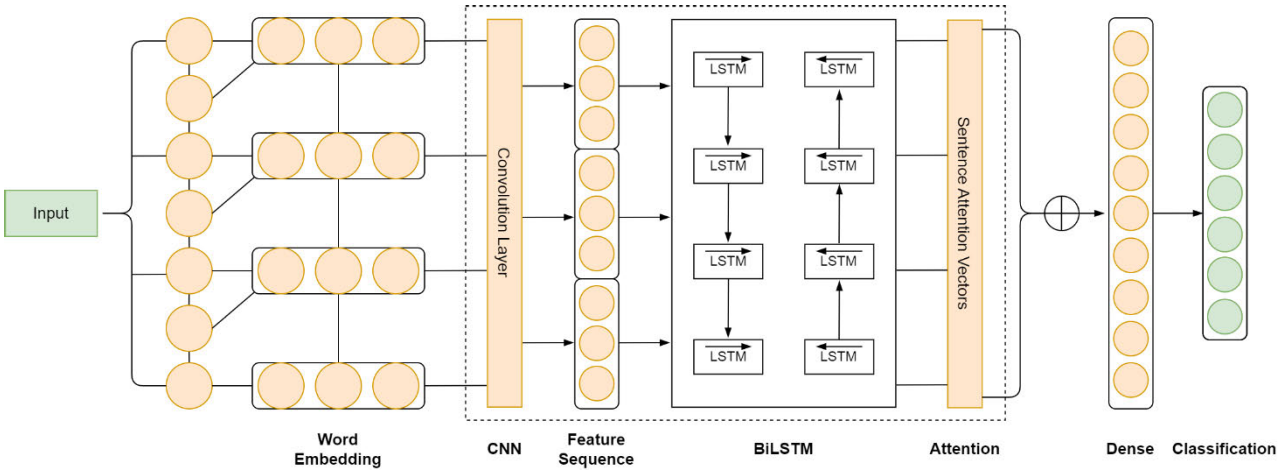

FIGURE 21. Utilizing the CNN-BiLSTM model for Android malware detection in the CoDroid system proposed by Zhang et al. [60].

classes.dex file. On the other hand, the dynamic features were derived from system calls, which provide insights into the application's behavior. The researchers generated a total of 202 feature vectors by analyzing the app's static and dynamic characteristics. These feature vectors comprised 120 vectors from the manifest file, 64 vectors from classes.dex, and 18 vectors related to behavioral aspects. Yuan et al. employed a two-phase approach to leverage the power of deep learning. The first phase involved unsupervised pre-training using Deep Boltzmann Machines (DBM), which aimed to learn meaningful representations of the data. In the second phase, supervised learning was performed using a back-propagation algorithm to fine-tune the model and optimize the classification performance. The experiments demonstrated that their proposed deep learning algorithm achieved an impressive accuracy of 96%, surpassing other popular machine learning algorithms such as SVM, C4.5, Naïve Bayes, Linear Regression, and MLP.

However, the researcher should employ cross-validation techniques as an effective means to mitigate the risk of overfitting. This measure holds particular relevance given the dataset's modest size, comprising only 500 samples across two distinct classes. Moreover, the inclusion of supplementary performance metrics, in addition to accuracy, permits a more thorough evaluation of the constructed models. The utilization of this comprehensive assessment methodology aids in uncovering any potential shortcomings in the model and provides a deeper understanding of its effectiveness within the context of the specific classification task.

H. AUTOMATED INPUT GENERATION

In heuristic-based malware detection, malware is executed either on an emulator or a real device, allowing researchers to observe and record its behavior. Specific configurations are used to simulate user interaction with the malware. This involves providing various inputs such as text input, swiping gestures, touch interactions, double taps, and more. Automated input generation tools play a crucial role in triggering and capturing the malicious activity exhibited by the malware.

These tools are designed to automatically generate and inject inputs into the running malware to stimulate its behavior and uncover any malicious actions. By utilizing these automated input generation tools, researchers can effectively analyze the malware's response to different scenarios and uncover its hidden functionalities. The use of such tools enables researchers to explore the capabilities and actions of malware in a controlled environment, facilitating the detection and analysis of malicious behavior. By generating diverse and targeted inputs, these tools aid in identifying potential threats and contribute to developing effective countermeasures against malware. This section describes the automated input generation tool used to trigger malicious activity to answer RQ8.

Researchers employ various approaches to generate input for heuristic-based malware detection. One approach is manual interaction, as demonstrated by Lin et al. [124]. This method involves a human intervention to interact with the running application, allowing for more natural input generation. Although this method can produce realistic and diverse inputs, it may pose challenges when dealing with large datasets due to the manual effort required. Fig. 22 illustrates the process of researchers providing manual input to the application under execution.

To address the challenges of working with large datasets, automation tools are necessary to provide input to the application under execution. Several studies, such as Wang et al. [57], Lu et al. [74], and Saif et al. [126], have utilized MonkeyRunner as an automated input generation tool. MonkeyRunner allows researchers to create scripts that generate specific inputs for testing purposes. By composing scripts, researchers can simulate various user interactions and systematically generate input patterns. However, it is essential to consider that each application possesses a distinct user interface. Consequently, the input necessary for testing one application may not be universally applicable to others. This discrepancy implies that employing MonkeyRunner for extensive testing

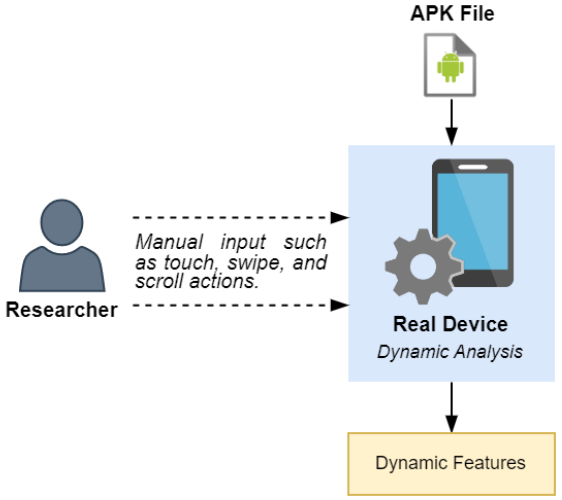

FIGURE 22. Researchers providing manual input to simulate the interaction between the user and the app under execution in Lin et al. [124] study.

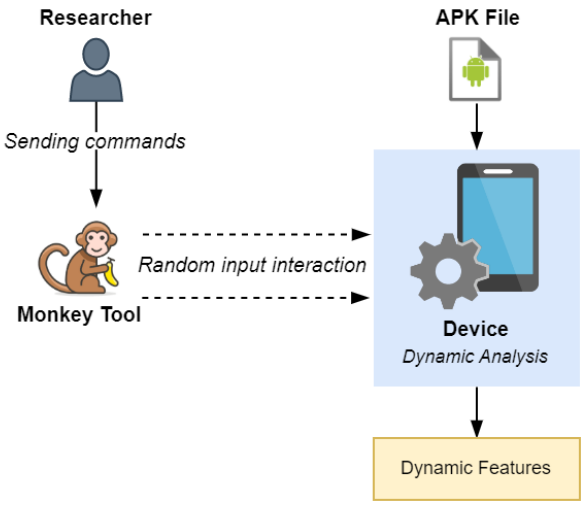

FIGURE 23. Monkey generates random inputs to simulate user interaction with the app under execution.

endeavors may encounter challenges in terms of generating tailored and application-specific input for each case.

Understanding when and how these inputs are generated is crucial for effective malware analysis. Wang et al. [56] employed the Robotium tool, which follows a similar strategy. This approach allows researchers to write test scripts that interact with the application under analysis, generating different types of input, such as text input, touch gestures, and button clicks. However, it is important to note that Robotium, as a gray-box testing tool, necessitates the instrumentation of APK files. This specific feature renders it more susceptible to malware defense techniques like anti-tampering, wherein malware can inspect the integrity of APK files [138] and evade detection if any unauthorized modifications are detected.

Another method involves using Monkey software to generate random inputs. Unlike MonkeyRunner, which requires careful planning of input timing and delivery, Monkey randomly generates inputs without specific scripting. This approach has been employed by researchers such as Khoda et al. [51], Sun et al. [76], Zhao et al. [94], and others, allowing them to simulate a variety of user interactions without the need for precise scripting.

Researchers can easily simulate user interaction using Monkey by sending commands through ADB. For instance, if researchers need a thousand user interactions, Monkey will generate a thousand random inputs to the application under execution. These inputs may include various actions, such as touch, swipe, scroll, and more. Fig. 23 illustrates how studies employ Monkey to simulate user interaction by providing random inputs to the app under execution.

Eder et al. [68] took a combined approach by using Monkey tools along with scripts managed over Telnet. This combination allowed for greater flexibility in generating inputs and controlling the behavior of the application during testing. By leveraging Monkey tools and additional scripting capabilities, researchers can generate a wide range of random inputs and customize the testing process according to their specific requirements.

The Monkey tool provides a convenient, lightweight, and highly scalable testing solution. Nevertheless, it is imperative to acknowledge that malware possesses a nuanced understanding of user interface interactions. This understanding can activate self-defense mechanisms like anti-analysis [139], typically responsive to specific inputs or events rather than arbitrary or random inputs [140], [141].

Other researchers, such as Alzaylaee et al. [115], employed a different approach to generate inputs. They used the Droidbot tool, which can provide input automatically according to the context of the application. Meanwhile, Wang et al. [129] used a different tool called APE.

Several studies indirectly utilize input generation methods by employing frameworks that already incorporate such methods. For instance, the Andrubis framework utilizes the Monkey tool as its input generation method, enabling researchers to automatically generate random inputs for testing purposes [72]. Similarly, the CopperDroid framework utilizes MonkeyRunner as its input generation method, allowing for the generation of specific inputs through scripting [100].

The utilization of automated input generation in malware detection is summarized in Table 17 and depicted in Fig. 24. However, it is noteworthy that many researchers did not explicitly mention the use of an input generation method in their works. Additionally, studies that employed text-based datasets, such as CSV and JSON, were not included in the summary. Out of the total 88 research papers reviewed, only 37 (42.04%) were found to have employed automated input generation in their work.

VI. DISCUSSION AND FUTURE WORK

Malware poses a significant challenge to cybersecurity, and researchers are actively working to address these issues, particularly in the field of heuristic-based malware detection using hybrid approaches. Numerous studies have proposed

TABLE 17. A summary of the automated input generation methods.

Input Generation	Related Studies
Not Mentioned	[88] [42] [118] [64] [84] [46] [67] [12] [117] [43] [90] [48] [95] [119] [120] [66] [70] [71] [121] [87] [123] [11] [52] [91] [75] [92] [80] [53] [128] [54] [96] [55] [130] [58] [109] [82] [77] [131] [14] [62] [125] [83] [98]
Manual	[124]
MonkeyRunner	[85] [86] [100] [103] [127] [59] [57] [126] [50] [79] [65] [74] [81] [101]
Monkey	[105] [76] [69] [51] [93] [60] [72] [49] [94] [73] [41] [45] [122] [89] [115] [68] [63] [78] [61]
Droidbot	[115]
Robotium	[56]
APE	[129]

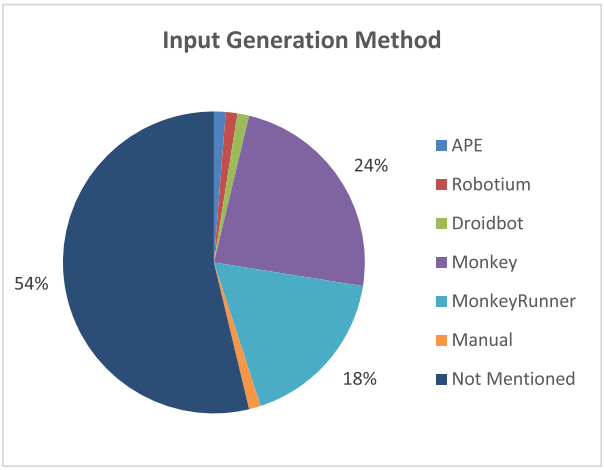

FIGURE 24. A summary of the automated input generation methods that were used in the works that we studied in this paper.

a wide range of hybrid techniques that combine static and dynamic approaches to enhance detection capabilities. In this section, we will delve into various aspects and issues related to hybrid approaches in malware detection and also provide insights into future research directions in this field.

A. DATASETS

The utilization of a high-value dataset is essential for obtaining valuable insights and ensuring that the findings accurately reflect the real-world phenomenon under study. However, many studies rely on datasets that are over five years old, which may fail to accurately represent the current landscape of malware attacks. According to the study conducted by Allix et al., datasets with more recent samples tend to enhance the performance of malware detection. Nevertheless, it remains imperative to strike a balance between older and newer datasets to ensure a more comprehensive representation of malware threats [142]. Therefore, developing and regularly updating malware datasets is crucial to enable researchers to contribute to uncovering new opportunities, advancing knowledge, developing new theories or models, and staying abreast of the dynamic nature of malware threats.

In addition, the study by Allix et al. reveals that a diverse dataset has demonstrated a positive impact on malware detection. Dataset diversity assists detection tools in maintaining consistent and high-performance levels over time, especially concerning precision, and enhances the tool's ability to identify common characteristics that emerge in various types of malware. This finding is of significant importance in addressing the dynamic range of Android malware categories, hence enhancing the effectiveness of security measures against diverse forms of attacks [142].

Furthermore, certain public datasets seem to be inaccessible for direct download. These obstacles may impact the project's timeline and outcome, underscoring the importance of thorough planning and effective data management.

B. FEATURE UTILIZATION AND SELECTION

Malware detection using a combination of static and dynamic features has the potential to enhance detection accuracy [11], [12]. However, combining all of these features introduces additional challenges, such as increased complexity, higher computational resource requirements, and longer processing times. To address these challenges, it is essential to conduct a comprehensive feature selection study to identify the most relevant and informative combination of static and dynamic features.

Furthermore, it is important to investigate the relationship between feature selection and on-device detection performance. Balancing accuracy and computational resources is crucial, and more research is required to understand how feature selection impacts on-device detection performance. By addressing these challenges, researchers can enhance the effectiveness and efficiency of hybrid malware detection systems.

In essence, while researchers frequently select feature types based on their expertise, the lack of comprehensive studies that systematically evaluate the computational efficiency and effectiveness of different feature combinations is a significant limitation in the field of malware detection. There is an opportunity for future research to bridge this gap by providing insights into the optimal feature combinations that balance detection accuracy and computational resource utilization. Such studies would contribute significantly to advancing the state of the art in malware detection.

C. UTILIZATION OF EMULATOR VS REAL DEVICE

This aspect relates to the machine choice used in developing detection models. It is observed that the majority of research studies rely on emulators for feature extraction. However, it is important to note that using an emulator device may not be as effective as using a real device in the detection process [133]. This highlights the need for researchers to pay more attention to the choice of machine and consider the potential limitations and implications it may have on the accuracy and effectiveness of the detection models.

D. ARCHITECTURE SYSTEMS COULD REDUCE COMPUTATIONAL COST

In the working environment, one potential approach is to adopt a tiered layer system architecture to address

computational resource challenges. This involves performing detection based on static features on the real device, while dynamic feature detection is carried out in the cloud. However, it is crucial to conduct further investigations to determine if the reduced computational cost aligns with other important factors. It is necessary to assess whether this cost advantage is balanced with considerations such as detection accuracy and end-user usability. Future research endeavors should aim to address these questions and provide insights into the optimal trade-offs in designing hybrid detection systems.

E. PRACTICAL ASPECTS FOR END USERS

Our findings reveal that only a limited number of studies have addressed these issues. Firstly, the prevalent use of local computers and emulators indicates that many researchers have not sufficiently considered the practicality of detection tools for end-user purposes. Secondly, we found that no research has conducted the entire detection process in a real device environment, commencing from feature extraction and feature processing to the detection process itself. On-device detection technology is pivotal, as it enables the continuous monitoring of suspicious activities and behaviors. This functionality, in turn, enables the immediate identification of cyber threats and the timely implementation of preventive actions. Therefore, this foundational technology is essential for providing real-time protection against potential cyberattacks and minimizing the risk of further damage to the device. A study conducted by the Ponemon Institute found that the ability to prevent attacks in real-time is considered one of the most important aspects of technological instruments [143].

To ensure real-time protection, it is crucial to monitor sensitive data simultaneously. This is typically done using dynamic taint analysis tools such as TaintDroid [144] or its derivatives, like DroidBox [145]. Dynamic taint analysis involves tracing the flow of sensitive data within an application during real-time execution in Android application security testing. Some studies within the hybrid approach scope have adopted this approach, including the works of Liu et al. [41], Yang et al. [59], and Yuan et al. [82]. However, it is worth noting that implementing dynamic taint analysis for regular users is currently not feasible as it requires customization of the user's device. Additionally, the need for code instrumentation in the process of tracking sensitive data gives rise to malware defense methods such as anti-tempering. It should also be noted that the studies referenced were not conducted on the actual devices used by everyday users but rather in controlled laboratory settings.

While various studies have demonstrated impressive results in controlled laboratory settings, the practical implementation of these solutions on real mobile devices presents several challenges. The potential trade-off between prioritizing detection performance or accuracy might lead to the development of excessively complex processes that pose computing challenges when implemented in real-world situations. This complexity often deters researchers from utilizing real devices as their primary detection environment. Additionally, the use of certain features, such as system call features, commonly employed by researchers may require device rooting or customization, which can be a daunting task for the average end-user.

However, these limitations can be overcome by carefully selecting features that do not require significant computational resources and can be obtained without customizing the end-user's mobile device. Bhandari et al.'s study [67] extracted relatively simple features that can be used on low-resource real devices. The static features extracted are related to the AndroidManifest.xml file, while dynamic features are related to system process information that does not require device rooting or customization, such as CPU info, memory info, and file operations. However, it is worth noting that this study chose to perform the malware classification process in a cloud environment rather than on the device itself. While this approach may offer certain advantages, it is important to recognize that cloud-based detection may not always be a reliable option, as not all users have consistent internet connectivity [146], potentially leaving security vulnerabilities.

The study conducted by Krzysztoń et al. [147] demonstrated the feasibility of on-device detection through the extraction of static features from the AndroidManifest.xml file and the design of neural network classification models tailored to the mobile platform and available resources on real devices. Similarly, Arp et al. [148] not only analyzed the manifest file but also utilized a lightweight dex class disassembler for on-device detection. Renjith&Aji [149] developed a system for on-device detection by adapting the optimized liblinear incremental classification algorithm to the Android environment [150], and their study further highlights the effectiveness of on-device detection with careful computational consideration. These studies provide strong evidence that detection systems can be successfully implemented with judicious use of features and classification models on the device.

Another significant challenge in on-device detection is the establishment of a dynamic analysis environment for executing and examining mobile applications. The direct execution of suspicious applications on the user's device poses a risk of infecting the device itself. To address this critical issue and ensure the security of the user's device, the implementation of container technology is essential. This technology serves as a crucial solution to address the challenge and safeguard the user's device by creating a segregated environment dedicated to malware analysis, isolated from the user's regular system environment. Within this controlled space, the analysis of malware takes place, minimizing the likelihood of any detrimental effects on the user's device resulting from the execution of potentially malicious or questionable applications.

Several container technologies are available to facilitate this segregation, including C-Android, a development by Chau&Jung [151], and Condroid, as devised by Xu et al. [152]. While these technologies offer promising

solutions to the challenge at hand, dynamic malware analysis within container systems remains largely understudied. Further research is needed to better understand this area and improve on-device malware detection. This research gap necessitates a deeper exploration of dynamic malware analysis within container systems, which promises to enhance the efficacy of on-device malware detection techniques and contribute to the security of mobile devices.

In order to create more practical and effective solutions for end-users, future research should focus on developing on-device malware detection methodologies. This research should consider various factors, including efficiency, resource allocation, user experience enhancement, and security considerations. By doing so, the balance between technical advancements and practical usability can be achieved, ensuring that the solutions are both effective and feasible.

F. USE OF DEEP LEARNING

The utilization of deep learning in hybrid approaches for solving detection problems has gained significant popularity. However, non-deep learning techniques continue to dominate the field. Therefore, there is a need to promote and encourage the adoption of deep learning in hybrid analysis, as it has the potential to greatly enhance detection accuracy and performance. Additionally, attention should be given to the availability of high-quality and diverse datasets for training and evaluating deep learning models. It is important to note that, as indicated in Table 9, only a small portion of the existing datasets consists of more than 100,000 APK samples. This highlights the importance of expanding the volume and diversity of datasets to effectively train deep learning models and ensure their robustness in addressing the evolving landscape of malware attacks.

G. DEALING WITH ZERO-DAY ATTACK

The staggering number of new Android malware threats, with thousands of unique variants discovered daily [5], necessitates the development of robust and adaptable security measures. Failure to address this growing problem can leave users vulnerable to data breaches and other security risks. A report by mobile security vendor Zimperium [153] uncovered a concerning trend: 31% of all zero-day attacks in 2021 targeted mobile devices. This represents a massive 466% increase compared to the previous year. Remarkably, the aforementioned attacks remain undetected for an average duration of 312 days [154], contributing to a substantial 80% of security breaches, resulting in an average cost of 1.2 million dollars for each attack [143].

However, the information presented in Table 16 highlights that only a limited number of studies have explored approaches to address zero-day attacks. Future research should focus on creating resilient algorithms and techniques to effectively identify and counter the risks presented by these dynamic and formidable threats.

In light of these alarming statistics, it is crucial to prioritize allocating resources toward formulating and implementing effective strategies to mitigate zero-day attacks. The incorporation of an algorithm specifically designed to mitigate these risks is one of the key components of this undertaking. Anomaly detection emerges as a prospective approach to address this issue [155].

One of the major challenges encountered in the field of zero-day detection is the limited availability of datasets. Because zero-day malware remains undisclosed until an attack is discovered, it poses difficulties in providing a fair evaluation and benchmarking process [156].

H. TRIGGERING MALWARE BEHAVIOR

Future research in the area of malware detection should focus on improving input generation techniques to effectively trigger and expose malicious behavior. While most studies rely on tools like Monkey, which uses a random approach to generate input, it is important to address the limitations of such methods. Malware equipped with anti-analysis defense mechanisms can remain concealed without specific input [139]. Hence, there is a need for advanced input generation methods that provide specific and natural input, enabling the identification of hidden malicious behavior in malware.

Another challenge in real-world implementation, particularly for on-device detection, pertains to the utilization of automated input generation. In dynamic malware analysis, suspected apps are executed on specific devices and receive input from automated input generation tools. However, a practical challenge arises when attempting to control automated input generation because it typically requires instructions to be provided from outside the device used for execution. This may not be feasible for real-world scenarios on user devices.

One promising solution is the utilization of container technology, exemplified by C-Android [151] and Condroid [152]. Containerization enables anti-malware software installed on a user's device to effectively provide the necessary input instructions to applications being executed within the container. However, it is noteworthy that there is a scarcity of studies that have integrated container technology with automated input generation tools for in-depth analysis of malware applications. Future research endeavors should focus on investigating the unexploited potential of container technology when combined with automated input generation tools to further refine and improve the capabilities of on-device malware detection.

I. CODE COVERAGE

It is crucial to prioritize code coverage over execution speed. While fast input generation methods based on randomness, such as random-based testing, offer efficiency, they may fall short in covering all possible execution paths of an application. This limitation hinders the activation of potentially malicious code that is triggered only under specific paths or conditions. To overcome this challenge, future research should explore contextual input generation techniques. Contextual input generation aims to provide inputs that are

specific to the application's context, thereby increasing the likelihood of activating targeted code and exposing potential malicious behavior. By emphasizing contextual input generation, researchers can augment the effectiveness of hybrid approaches by uncovering concealed malware and improving the overall code coverage.

VII. CONCLUSION

The rapid evolution of malware threats targeting Android-based operating systems necessitates the development of effective detection methods. Numerous works have been done on heuristic-based detection methods with the hybrid approach to overcome these threats. This study presented a state-of-the-art analysis of the hybrid detecting techniques. We provided an in-depth evaluation and discussed the aspects related to the hybrid detection approach between the years 2012 and 2023. The discussion covered various aspects of hybrid analysis, which include the used datasets, the selection of features, the working environment, the detection order and schema, the detection methods, and the employment of input generation tools.

In addition to the retrospective analysis, this study has discovered several key findings that have significant implications for the field of hybrid Android malware detection.

Firstly, there is a need for on-device detection systems that provide on-device protection against malware threats and offer real-time protection to minimize the risk of further damage to the device. This requires further research into resource-efficient computation for mobile devices, including selecting efficient features and algorithms, along with focusing on enhancing the user experience.

Secondly, many studies rely on outdated datasets that do not accurately represent the current threat landscape. It is important to regularly update malware datasets so that researchers can discover emerging threats, develop new theories, formulate novel models, and create more effective countermeasures against malware threats. An iterative process of dataset updating maintains synchronization between the research community and the ever-changing and evolving nature of malware threats.

Thirdly, there is a need to develop an effective methodology for detecting zero-day attacks to protect users from emerging malware threats. This involves the utilization of specialized algorithms, including anomaly detection algorithms, and the creation of a dedicated dataset for zero-day attacks in the Android malware domain.

Fourthly, it is crucial for future research to study the impact of contextual automated input generation on the triggering of malware behavior and the enhancement of code coverage. Furthermore, additional studies should focus on the practical implementation of containers in dynamic analysis, especially when integrated with automated input generation techniques. This research is particularly relevant in addressing the complexities of on-device malware detection.

In conclusion, this study has provided a comprehensive analysis of hybrid detection techniques in the context of Android malware. We have identified research gaps and highlighted open issues that need to be addressed to improve the accuracy, efficiency, robustness, and scalability of hybrid detection systems. By addressing these challenges and advancing research in this field, we can enhance the security of Android-based systems and mitigate the evolving threats posed by malware.

ACKNOWLEDGMENT

We are primarily grateful to Lembaga Pengelola Dana Pendidikan (LPDP), the Republic of Indonesia, and RTA Program Universitas Gadjah Mada with the Grant Number 5075/UN1.P.II/Dit-Lit/PT.01.01/2023, who sponsored this report. Thanks to the Journal of IEEE Access reviews, their comments and recommendations substantially strengthened the paper.

REFERENCES

[1] D. Ruby. (2023). *20 Android Statistics In 2023 (Market Share & Users)*. Accessed: Jul. 27, 2023. [Online]. Available: https://www.demandsage.com/android-statistics/

[2] Statista. (2009). *Mobile Operating Systems' Market Share Worldwide From 1st Quarter 2009 to 2nd Quarter 2023*. Accessed: Jul. 27, 2023. [Online]. Available: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

[3] Statcounter. (2023). *Mobile Operating System Market Share Worldwide*. Accessed: Jul. 27, 2023. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide

[4] T. Shishkova and A. Kivva. (2022). *IT Threat Evolution in Q3*. Mobile Statistics. [Online]. Available: https://securelist.com/it-threat-evolution-in-q3-2022-mobile-statistics/107978/

[5] G DATA. (2019). *Mobile Malware Report, no Let ip With Android Malware*. Accessed: Mar. 15, 2020. [Online]. Available: https://www.gdatasoftware.co.uk/news/2019/07/35228-mobile-malware-report-no-let-up-with-android-malware

[6] Kaspersky. (2023). *Mobile Security: Android vs IOS—Which One is Safer?* Accessed: Jul. 26, 2023. [Online]. Available: https://www.kaspersky.com/resource-center/threats/android-vs-iphone-mobile-security

[7] A. Firdaus, "Mobile malware anomaly-based detection systems using static analysis features," Dept. Syst. Comput. Technol., Faculty Comput. Sci. Inf. Technol., Univ. Malaya, Kuala Lumpur, Malaysia, Tech. Rep., 2017.

[8] Kaspersky. (2019). *What is Heuristic Analysis?* Accessed: Jan. 29, 2021. [Online]. Available: https://usa.kaspersky.com/resource-center/definitions/heuristic-analysis

[9] A. Razgallah, R. Khoury, S. Hallé, and K. Khanmohammadi, "A survey of malware detection in Android apps: Recommendations and perspectives for future research," *Comput. Sci. Rev.*, vol. 39, Feb. 2021, Art. no. 100358, doi: 10.1016/j.cosrev.2020.100358.

[10] P. Feng, J. Ma, C. Sun, X. Xu, and Y. Ma, "A novel dynamic Android malware detection system with ensemble learning," *IEEE Access*, vol. 6, pp. 30996–31011, 2018, doi: 10.1109/ACCESS.2018.2844349.

[11] F. Martinelli, F. Mercaldo, and A. Saracino, "BRIDEMAID: An hybrid tool for accurate detection of Android malware," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, Apr. 2017, pp. 899–901, doi: 10.1145/3052973.3055156.

[12] A. Arora and S. K. Peddoju, "NTPDroid: A hybrid Android malware detector using network traffic and system permissions," in *Proc. 17th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./12th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, Aug. 2018, pp. 808–813, doi: 10.1109/TrustCom/BIGDATASE.2018.00115.

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

[14] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets," in *Proc. NDSS*, vol. 2, 2012, pp. 1–24.

[15] A. G. Shabir and N. Sabahat, "A review of hybrid malware detection techniques in Android," in *Proc. IEEE 23rd Int. Multitopic Conf. (INMIC)*, Nov. 2020, pp. 1–6, doi: 10.1109/INMIC50486.2020.9318117.

[16] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Newcastle-under-Lyme Durham, 2007. [Online]. Available: https://legacyfileshare.elsevier.com/promis_misc/525444systematicreviewsguide.pdf

[17] A. Dubey and A. Misra, *Android Security, Attacks and Defenses*. Boca Raton, FL, USA: CRC Press, 2013.

[18] AndroidDev. (2021). *Android Platform Architecture*. Accessed: Sep. 1, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Android_(operating_system)

[19] G. M. and S. C. Sethuraman, "A comprehensive survey on deep learning based malware detection techniques," *Comput. Sci. Rev.*, vol. 47, Feb. 2023, Art. no. 100529, doi: 10.1016/j.cosrev.2022.100529.

[20] L. Shu, S. Dong, H. Su, and J. Huang, "Android malware detection methods based on convolutional neural network: A survey," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 7, no. 5, pp. 1330–1350, Oct. 2023, doi: 10.1109/TETCI.2023.3281833.

[21] M. Mehrabi Koushki, I. Abualhaol, A. D. Raju, Y. Zhou, R. S. Giagone, and H. Shengqiang, "On building machine learning pipelines for Android malware detection: A procedural survey of practices, challenges and opportunities," *Cybersecurity*, vol. 5, no. 1, pp. 1–37, Dec. 2022, doi: 10.1186/s42400-022-00119-8.

[22] L. Meijin, F. Zhiyang, W. Junfeng, C. Luyu, Z. Qi, Y. Tao, W. Yinwei, and G. Jiaxuan, "A systematic overview of Android malware detection," *Appl. Artif. Intell.*, vol. 36, no. 1, pp. 497–524, Dec. 2022, doi: 10.1080/08839514.2021.2007327.

[23] A. Muzaffar, H. Ragab Hassen, M. A. Lones, and H. Zantout, "An in-depth review of machine learning based Android malware detection," *Comput. Secur.*, vol. 121, Oct. 2022, Art. no. 102833, doi: 10.1016/j.cose.2022.102833.

[24] T. Sharma and D. Rattan, "Malicious application detection in Android—A systematic literature review," *Comput. Sci. Rev.*, vol. 40, May 2021, Art. no. 100373, doi: 10.1016/j.cosrev.2021.100373.

[25] S. Garg and N. Baliyan, "Android security assessment: A review, taxonomy and research gap study," *Comput. Secur.*, vol. 100, Jan. 2021, Art. no. 102087, doi: 10.1016/j.cose.2020.102087.

[26] S. Abijah Roseline and S. Geetha, "A comprehensive survey of tools and techniques mitigating computer and mobile malware attacks," *Comput. Electr. Eng.*, vol. 92, Jun. 2021, Art. no. 107143, doi: 10.1016/j.compeleceng.2021.107143.

[27] V. Kouliaridis and G. Kambourakis, "A comprehensive survey on machine learning techniques for Android malware detection," *Information*, vol. 12, no. 5, p. 185, Apr. 2021, doi: 10.3390/info12050185.

[28] J. Senanayake, H. Kalutarage, and M. O. Al-Kadri, "Android mobile malware detection using machine learning: A systematic review," *Electronics*, vol. 10, no. 13, p. 1606, Jul. 2021, doi: 10.3390/electronics10131606.

[29] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, "A review of Android malware detection approaches based on machine learning," *IEEE Access*, vol. 8, pp. 124579–124607, 2020, doi: 10.1109/ACCESS.2020.3006143.

[30] Z. Wang, Q. Liu, and Y. Chi, "Review of Android malware detection based on deep learning," *IEEE Access*, vol. 8, pp. 181102–181126, 2020, doi: 10.1109/ACCESS.2020.3028370.

[31] F. Alswaina and K. Elleithy, "Android malware family classification and analysis: Current status and future directions," *Electronics*, vol. 9, no. 6, p. 942, Jun. 2020, doi: 10.3390/electronics9060942.

[32] A. Qamar, A. Karim, and V. Chang, "Mobile malware attacks: Review, taxonomy & future directions," *Future Gener. Comput. Syst.*, vol. 97, pp. 887–909, Aug. 2019, doi: 10.1016/j.future.2019.03.007.

[33] M. Ashawa and S. Morris, "Analysis of Android malware detection techniques: A systematic review," *Int. J. Cyber-Security Digit. Forensics*, vol. 8, no. 3, pp. 177–187, 2019, doi: 10.17781/p002605.

[34] M. Odusami, O. Abayomi-Alli, S. Misra, O. Shobayo, R. Damasevicius, and R. Maskeliunas, "Android malware detection: A survey," *Scientia Sinica Informationis*, vol. 50, no. 8, pp. 255–266, 2018.

[35] P. Yan and Z. Yan, "A survey on dynamic mobile malware detection," *Softw. Quality J.*, vol. 26, no. 3, pp. 891–919, Sep. 2018, doi: 10.1007/s11219-017-9368-4.

[36] B. Yu, Y. Fang, Q. Yang, Y. Tang, and L. Liu, "A survey of malware behavior description and analysis," *Frontiers Inf. Technol. Electron. Eng.*, vol. 19, no. 5, pp. 583–603, May 2018, doi: 10.1631/fitee.1601745.

[37] A. Naway and Y. LI, "A review on the use of deep learning in Android malware detection," 2018, *arXiv:1812.10360*.

[38] K. Yaghmour, *Embedded Android*. California, CA, USA: O'Reilly Media, 2013.

[39] K. Dunham, S. Hartman, M. Quintans, J. A. Morales, and T. Strazzere, *Android Malware and Analysis*. New York, NY, USA: Auerbach, 2014.

[40] W. Wang, M. Zhao, Z. Gao, G. Xu, H. Xian, Y. Li, and X. Zhang, "Constructing features for detecting Android malicious applications: Issues, taxonomy and directions," *IEEE Access*, vol. 7, pp. 67602–67631, 2019, doi: 10.1109/ACCESS.2019.2918139.

[41] Y. Liu, K. Guo, X. Huang, Z. Zhou, and Y. Zhang, "Detecting Android malwares with high-efficient hybrid analyzing methods," *Mobile Inf. Syst.*, vol. 2018, pp. 1–12, Mar. 2018, doi: 10.1155/2018/1649703.

[42] J. Feng, L. Shen, Z. Chen, Y. Wang, and H. Li, "A two-layer deep learning method for Android malware detection using network traffic," *IEEE Access*, vol. 8, pp. 125786–125796, 2020, doi: 10.1109/ACCESS.2020.3008081.

[43] U. Ahmed, J. C.-W. Lin, and G. Srivastava, "Mitigating adversarial evasion attacks of ransomware using ensemble learning," *Comput. Electr. Eng.*, vol. 100, May 2022, Art. no. 107903, doi: 10.1016/j.compeleceng.2022.107903.

[44] D. Budgen and P. Brereton, "Performing systematic literature reviews in software engineering," in *Proc. 28th Int. Conf. Softw. Eng.*, May 2006, vol. 82, no. 2, p. 1051, doi: 10.1145/1134285.1134500.

[45] 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, doi: 10.1109/ACCESS.2018.2792941.

[46] O. Faruk Turan Cavli and S. Sen, "Familial classification of Android malware using hybrid analysis," in *Proc. Int. Conf. Inf. Secur. Cryptol. (ISCTURKEY)*, Dec. 2020, pp. 62–67, doi: 10.1109/ISCTURKEY51113.2020.9308003.

[47] R. B. Hadiprakoso, H. Kabetta, and I. K. S. Buana, "Hybrid-based malware analysis for effective and efficiency Android malware detection," in *Proc. Int. Conf. Informat., Multimedia, Cyber Inf. Syst. (ICIMCIS)*, Nov. 2020, pp. 8–12, doi: 10.1109/ICIMCIS51567.2020.9354315.

[48] S. J. Hussain, U. Ahmed, H. Liaquat, S. Mir, N. Z. Jhanjhi, and M. Humayun, "IMIAD: Intelligent malware identification for Android platform," in *Proc. Int. Conf. Comput. Inf. Sci. (ICCIS)*, Apr. 2019, pp. 1–6, doi: 10.1109/ICCISci.2019.8716471.

[49] M. A. Husainiamer, M. Mohd Saudi, and M. Yusof, "Securing mobile applications against mobile malware attacks: A case study," in *Proc. IEEE 19th student Conf. Res. Develop. (SCOReD)*, Nov. 2021, pp. 433–438, doi: 10.1109/SCOReD53546.2021.9652685.

[50] A. Kapratwar, F. Di Troia, and M. Stamp, "Static and dynamic analysis of Android malware," in *Proc. 3rd Int. Conf. Inf. Syst. Secur. Privacy*, Jan. 2017, pp. 653–662, doi: 10.5220/0006256706530662.

[51] M. E. Khoda, J. Kamruzzaman, I. Gondal, T. Imam, and A. Rahman, "Malware detection in edge devices with fuzzy oversampling and dynamic class weighting," *Appl. Soft Comput.*, vol. 112, Nov. 2021, Art. no. 107783, doi: 10.1016/j.asoc.2021.107783.

[52] A. Maryam, U. Ahmed, M. Aleem, J. C.-W. Lin, M. A. Islam, and M. A. Iqbal, "CHybriDroid: A machine learning-based hybrid technique for securing the edge computing," *Secur. Commun. Netw.*, vol. 2020, pp. 1–14, Nov. 2020, doi: 10.1155/2020/8861639.

[53] K. Sugunan, T. G. Kumar, and K. A. Dhanya, "Static and dynamic analysis for Android malware detection," in *Advances in Big Data and Cloud Computing* (Advances in Intelligent Systems and Computing), vol. 645. Singapore: Springer, 2018, pp. 147–155, doi: 10.1007/978-981-10-7200-0_13.

[54] R. Surendran, T. Thomas, and S. Emmanuel, "A TAN based hybrid model for Android malware detection," *J. Inf. Secur. Appl.*, vol. 54, Oct. 2020, Art. no. 102483, doi: 10.1016/j.jisa.2020.102483.

[55] M. Upadhayay, A. Sharma, G. Garg, and A. Arora, "RPNDroid: Android malware detection using ranked permissions and network traffic," in *Proc. 5th World Conf. Smart Trends Syst. Secur. Sustainability (WorldS4)*, Jul. 2021, pp. 19–24, doi: 10.1109/WorldS451998.2021.9513992.

[56] X. Wang, J. Shi, Y. Yang, K. Xu, Y. Zeng, and C. Tang, "A novel hybrid mobile malware detection system integrating anomaly detection with misuse detection," in *Proc. 6th Int. Workshop Mobile Cloud Comput. Services (MCS)*, 2015, pp. 15–22, doi: 10.1145/2802130.2802132.

[57] X. Wang, Y. Yang, and Y. Zeng, "Accurate mobile malware detection and classification in the cloud," *SpringerPlus*, vol. 4, no. 1, pp. 1–23, Dec. 2015, doi: 10.1186/s40064-015-1356-1.

[58] L. Wen and H. Yu, "An Android malware detection system based on machine learning," *AIP Conf. Proc.*, vol. 1864, no. Aug, Aug. 2017, Art. no. 020136, doi: 10.1063/1.4992953.
[59] F. Yang, Y. Zhuang, and J. Wang, "Android malware detection using hybrid analysis and machine learning technique," in *Cloud Computing and Security* (Lecture Notes in Computer Science, Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 10603. Cham, Switzerland: Springer, 2017, pp. 565–575, doi: 10.1007/978-3-319-68542-7_48.
[60] N. Zhang, J. Xue, Y. Ma, R. Zhang, T. Liang, and Y. Tan, "Hybrid sequence-based Android malware detection using natural language processing," *Int. J. Intell. Syst.*, vol. 36, no. 10, pp. 5770–5784, Oct. 2021, doi: 10.1002/int.22529.
[61] F. Faghihi, M. Zulkernine, and S. Ding, "AIM: An Android interpretable malware detector based on application class modeling," *J. Inf. Secur. Appl.*, vol. 75, Jun. 2023, Art. no. 103486, doi: 10.1016/j.jisa.2023.103486.
[62] U. Ahmed and J. C. W. Lin, "Adversarial ensemble modeling for evasion attack detection," in *Proc. Joint 12th Int. Conf. Soft Comput. Intell. Syst., 23rd Int. Symp. Adv. Intell. Syst. (SCIS ISIS)*, 2022, pp. 1–4, doi: 10.1109/SCISISIS55246.2022.10002083.
[63] M. L. Anupama, P. Vinod, C. A. Visaggio, M. A. Arya, J. Philomina, R. Raphael, A. Pinhero, K. S. Ajith, and P. Mathiyalagan, "Detection and robustness evaluation of Android malware classifiers," *J. Comput. Virol. Hacking Techn.*, vol. 18, no. 3, pp. 147–170, Sep. 2022, doi: 10.1007/s11416-021-00390-2.
[64] S. Chen, M. Xue, Z. Tang, L. Xu, and H. Zhu, "StormDroid: A streaminglized machine learning-based system for detecting Android malware," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, May 2016, pp. 377–388, doi: 10.1145/2897845.2897860.
[65] H. Jiao, X. Li, L. Zhang, G. Xu, and Z. Feng, "Hybrid detection using permission analysis for Android malware," in *International Conference on Security and Privacy in Communication Networks*. Cham, Switzerland: Springer, 2015, doi: 10.1007/978-3-319-23829-6_40.
[66] U. S. Jannat, S. M. Hasnayeen, M. K. B. Shuhan, and M. S. Ferdous, "Analysis and detection of malware in Android applications using machine learning," in *Proc. Int. Conf. Electr., Comput. Commun. Eng. (ECCE)*, Feb. 2019, pp. 1–7, doi: 10.1109/ECACE.2019.8679493.
[67] S. Bhandari, R. Gupta, V. Laxmi, M. S. Gaur, A. Zemmari, and M. Anikeev, "DRACO: Droid analyst combo an Android malware analysis framework categories and subject descriptors," in *Proc. 8th Int. Conf. Secur. Inf. Netw.*, 2015, pp. 283–289.
[68] T. Eder, M. Rodler, D. Vymazal, and M. Zeilinger, "ANANAS—A framework for analyzing Android applications," in *Proc. Int. Conf. Availability, Rel. Secur.*, Sep. 2013, pp. 711–719, doi: 10.1109/ARES.2013.93.
[69] A. T. Kabakuş, "Hybroid: A novel hybrid Android malware detection framework," *Erzincan Üniversitesi Fen Bilimleri Enstitüsü Dergisi*, vol. 14, no. 1, pp. 331–356, Mar. 2021, doi: 10.18185/erzifbed.806683.
[70] S. Kandukuru and R. M. Sharma, "Android malicious application detection using permission vector and network traffic analysis," in *Proc. 2nd Int. Conf. Converg. Technol. (I2CT)*, Apr. 2017, pp. 1126–1132, doi: 10.1109/I2CT.2017.8226303.
[71] S. Kandukuru and R. M. Sharma, "PNSDroid: A hybrid approach for detection of Android malware," in *Recent Findings in Intelligent Computing Techniques* (Advances in Intelligent Systems and Computing). Cham, Switzerland: Springer, 2018, pp. 361–367, doi: 10.1007/978-981-10-8633-5_36.
[72] M. Lindorfer, M. Neugschwandtner, and C. Platzer, "MARVIN: Efficient and comprehensive mobile app classification through static and dynamic analysis," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, vol. 2, Jul. 2015, pp. 422–433, doi: 10.1109/COMPSAC.2015.103.
[73] Y. Liu, Y. Zhang, H. Li, and X. Chen, "A hybrid malware detecting scheme for mobile Android applications," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2016, pp. 155–156, doi: 10.1109/ICCE.2016.7430561.
[74] T. Lu, Y. Du, L. Ouyang, Q. Chen, and X. Wang, "Android malware detection based on a hybrid deep learning model," *Secur. Commun. Netw.*, vol. 2020, pp. 1–11, Aug. 2020, doi: 10.1155/2020/8863617.
[75] 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, doi: 10.1109/TDSC.2016.2536605.
[76] M. Sun, X. Li, J. C. S. Lui, R. T. B. Ma, and Z. Liang, "Monet: A user-oriented behavior-based malware variants detection system for Android," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 5, pp. 1103–1112, May 2017, doi: 10.1109/TIFS.2016.2646641.
[77] 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, doi: 10.1109/TST.2016.7399288.
[78] A. Qammar, A. Karim, Y. Alharbi, M. Alsaffar, and A. Alharbi, "A learning model to detect Android C&C applications using hybrid analysis," *Comput. Syst. Sci. Eng.*, vol. 43, no. 3, pp. 915–930, 2022, doi: 10.32604/csse.2022.023652.
[79] K. Patel and B. Buddadev, "Detection and mitigation of Android malware through hybrid approach," in *Security in Computing and Communications* (Communications in Computer and Information Science), vol. 536, J. H. Abawajy, S. Mukherjea, S. M. Thampi, and A. Ruiz-Martínez, Eds. Cham, Switzerland: Springer, 2015, pp. 455–463.
[80] M.-Y. Su and K.-T. Fung, "Detection of Android malware by static analysis on permissions and sensitive functions," in *Proc. 8th Int. Conf. Ubiquitous Future Netw. (ICUFN)*, Jul. 2016, pp. 873–875, doi: 10.1109/ICUFN.2016.7537161.
[81] M.-Y. Su, J.-Y. Chang, and K.-T. Fung, "Android malware detection approaches in combination with static and dynamic features," *Int. J. Netw. Secur.*, vol. 21, no. 6, pp. 1031–1041, 2019.
[82] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-Sec: Deep learning in Android malware detection," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 371–372, 2015, doi: 10.1145/2619239.2631434.
[83] S. Ullah, T. Ahmad, A. Buriro, N. Zara, and S. Saha, "TrojanDetector: A multi-layer hybrid approach for trojan detection in Android applications," *Appl. Sci.*, vol. 12, no. 21, p. 10755, Oct. 2022, doi: 10.3390/app122110755.
[84] D. Chaulagain, P. Poudel, P. Pathak, S. Roy, D. Caragea, G. Liu, and X. Ou, "Hybrid analysis of Android apps for security vetting using deep learning," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Jun. 2020, pp. 1–9, doi: 10.1109/CNS48642.2020.9162341.
[85] S. Garg and N. Baliyan, "A novel parallel classifier scheme for vulnerability detection in Android," *Comput. Electr. Eng.*, vol. 77, pp. 12–26, Jul. 2019, doi: 10.1016/j.compeleceng.2019.04.019.
[86] A. Konstantinos, "Multiple layer hybrid classification for Android malware detection," M.S. thesis, Dept. Digit. Syst. Secur., Faculty Inf. Commun. Technol., Univ. Piraeus, Piraeus, Greece, 2021.
[87] R. Lemos, T. Heinrich, C. A. Maziero, and N. C. Will, "Is it safe? Identifying malicious apps through the use of metadata and inter-process communication," in *Proc. 16th IEEE Int. Syst. Conf. (SysCon)*, Apr. 2022, pp. 1–8, doi: 10.1109/SysCon53536.2022.9773881.
[88] T. Gera, J. Singh, A. Mehbodniya, J. L. Webber, M. Shabaz, and D. Thakur, "Dominant feature selection and machine learning-based hybrid approach to analyze Android ransomware," *Secur. Commun. Netw.*, vol. 2021, pp. 1–22, Nov. 2021, doi: 10.1155/2021/7035233.
[89] Q. Fang, X. Yang, and C. Ji, "A hybrid detection method for Android malware," in *Proc. IEEE 3rd Inf. Technol., Netw., Electron. Autom. Control Conf. (ITNEC)*, Mar. 2019, pp. 2127–2132, doi: 10.1109/ITNEC.2019.8729017.
[90] R. B. Hadiprakoso, I. K. S. Buana, and Y. R. Pramadi, "Android malware detection using hybrid-based analysis & deep neural network," in *Proc. 3rd Int. Conf. Inf. Commun. Technol. (ICOIACT)*, Nov. 2020, pp. 252–256, doi: 10.1109/ICOIACT50329.2020.9332066.
[91] A. Pektas and T. Acarman, "Ensemble machine learning approach for Android malware classification using hybrid features," in *Proc. Int. Conf. Comput. Recognit. Syst. (CORES)*, vol. 578, 2018, pp. 191–200, doi: 10.1007/978-3-319-59162-9.
[92] P. V. Shijo and A. Salim, "Integrated static and dynamic analysis for malware detection," *Proc. Comput. Sci.*, vol. 46, pp. 804–811, Jan. 2015, doi: 10.1016/j.procs.2015.02.149.
[93] L. Xu, D. Zhang, N. Jayasena, and J. Cavazos, "HADM: Hybrid analysis for detection of malware," in *Proc. SAI Intell. Syst. Conf.*, 2018, pp. 702–724, doi: 10.1007/978-3-319-56991-8_51.
[94] Y. Zhao, G. Xu, and Y. Zhang, "HFA-MD: An efficient hybrid features analysis based Android malware detection method," in *Quality, Reliability, Security and Robustness in Heterogeneous Systems*. Dalian, China: Springer, 2018, doi: 10.1007/978-3-319-78078-8_25.
[95] M. J. Iqbal, S. Aurangzeb, M. Aleem, G. Srivastava, and J. C. Lin, "RThreatDroid: A ransomware detection approach to secure IoT based healthcare systems," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 5, pp. 2574–2583, Sep./Oct. 2023, doi: 10.1109/TNSE.2022.3188597.
[96] L. Taheri, A. F. A. Kadir, and A. H. Lashkari, "Extensible Android malware detection and family classification using network-flows and API-calls," in *Proc. Int. Carnahan Conf. Secur. Technol. (ICCST)*, Oct. 2019, pp. 1–8, doi: 10.1109/CCST.2019.8888430.

[97] A. K. T. Lee Yam, J. M. R. Ballesta, J. A. H. Lanceta, M. K. T. Mogol, and R. Labanan, "Hybrid Android malware detection model using machine learning algorithms," in *Proc. 2nd Int. Conf. Inf. Comput. Res. (iCORE)*, Dec. 2022, pp. 66–71, doi: 10.1109/iCORE58172.2022.00032.

[98] F. Taher, O. AlFandi, M. Al-kfairy, H. Al Hamadi, and S. Alrabaee, "DroidDetectMW: A hybrid intelligent model for Android malware detection," *Appl. Sci.*, vol. 13, no. 13, p. 7720, Jun. 2023, doi: 10.3390/app13137720.

[99] S. I. Imtiaz, S. U. Rehman, A. R. Javed, Z. Jalil, X. Liu, and W. S. Alnumay, "DeepAMD: Detection and identification of Android malware using high-efficient deep artificial neural network," *Future Gener. Comput. Syst.*, vol. 115, pp. 844–856, Feb. 2021, doi: 10.1016/j.future.2020.10.008.

[100] S. Mahdavifar, D. Alhadidi, and A. A. Ghorbani, "Effective and efficient hybrid Android malware classification using pseudo-label stacked auto-encoder," *J. Netw. Syst. Manag.*, vol. 30, no. 1, p. 22, Nov. 2022.

[101] M. F. Ahmed, Z. T. Biash, A. R. Shakil, A. A. N. Ryen, A. Hossain, F. B. Ashraf, and M. I. Hossain, "ShielDroid: A hybrid approach integrating machine and deep learning for Android malware detection," in *Proc. Int. Conf. Decis. Aid Sci. Appl. (DASA)*, Mar. 2022, pp. 911–916, doi: 10.1109/DASA54658.2022.9764984.

[102] A. Joshi and S. Kumar, "Stacking-based ensemble model for malware detection in Android devices," *Int. J. Inf. Technol.*, vol. 15, no. 6, pp. 2907–2915, Aug. 2023, doi: 10.1007/s41870-023-01392-7.

[103] A. Martín, R. Lara-Cabrera, and D. Camacho, "Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset," *Inf. Fusion*, vol. 52, pp. 128–142, Dec. 2019, doi: 10.1016/j.inffus.2018.12.006.

[104] A. S. Oliveira and R. J. Sassi, "Hunting Android malware using multimodal deep learning and hybrid analysis data," *Sociedade Brasileira de Inteligência Computacional*, pp. 1–10, 2021, doi: 10.21528/cbic2021-32.

[105] C. Rodrigo, S. Pierre, R. Beaubrun, and F. El Khoury, "BrainShield: A hybrid machine learning-based malware detection model for Android devices," *Electronics*, vol. 10, no. 23, p. 2948, Nov. 2021, doi: 10.3390/electronics10232948.

[106] S. Roy, S. Bhanja, and A. Das, "AndyWar: An intelligent Android malware detection using machine learning," *Innov. Syst. Softw. Eng.*, pp. 1–9, Jul. 2023, doi: 10.1007/s11334-023-00530-5.

[107] S. Aurangzeb and M. Aleem, "Evaluation and classification of obfuscated Android malware through deep learning using ensemble voting mechanism," *Sci. Rep.*, vol. 13, no. 1, p. 3093, Feb. 2023, doi: 10.1038/s41598-023-30028-w.

[108] H. Rafiq, N. Aslam, B. Issac, and R. H. Randhawa, "On impact of adversarial evasion attacks on ML-based Android malware classifier trained on hybrid features," in *Proc. 14th Int. Conf. Softw., Knowl., Inf. Manage. Appl. (SKIMA)*, Dec. 2022, pp. 216–221, doi: 10.1109/SKIMA57145.2022.10029504.

[109] P. Xu, C. Eckert, and A. Zarras, "Hybrid-falcon: Hybrid pattern malware detection and categorization with network traffic and program code," 2021, *arXiv:2112.10035*.

[110] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 95–109, doi: 10.1109/SP.2012.16.

[111] (2014). *The Drebin Dataset*. Accessed: Sep. 23, 2023. [Online]. Available: https://www.sec.cs.tu-bs.de/~danarp/drebin/

[112] (2016). *AndroZoo Dataset*. Accessed: Sep. 28, 2023. [Online]. Available: https://androzoo.uni.lu/access

[113] (2020). *MalDroid-2020*. Accessed: Sep. 28, 2023. [Online]. Available: https://www.unb.ca/cic/datasets/maldroid-2020.html

[114] A. Guerra-Manzanares, H. Bahsi, and S. Nõmm. (2023). *Kronodroid Dataset*. Accessed: Sep. 28, 2023. [Online]. Available: https://github.com/aleguma/kronodroid

[115] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "DL-Droid: Deep learning based Android malware detection using real devices," *Comput. Secur.*, vol. 89, Feb. 2020, Art. no. 101663, doi: 10.1016/j.cose.2019.101663.

[116] E. Amer and S. El-Sappagh, "Robust deep learning early alarm prediction model based on the behavioural smell for Android malware," *Comput. Secur.*, vol. 116, May 2022, Art. no. 102670, doi: 10.1016/j.cose.2022.102670.

[117] A. Arora, S. K. Peddoju, V. Chouhan, and A. Chaudhary, "Hybrid Android malware detection by combining supervised and unsupervised learning," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw.*, Oct. 2018, pp. 798–800, doi: 10.1145/3241539.3267768.

[118] M. Dhalaria and E. Gandotra, "A hybrid approach for Android malware detection and family classification," *Int. J. Interact. Multimedia Artif. Intell.*, vol. 6, no. 6, p. 174, 2021, doi: 10.9781/ijimai.2020.09.001.

[119] S. Jadhav, T. Oh, J. Jeong, Y. H. Kim, and J. N. Kim, "An assistive system for Android malware analysis to increase malware analysis efficiency," in *Proc. 31st IEEE Int. Conf. Adv. Inf. Netw. Appl. Workshops (WAINA)*, Mar. 2017, pp. 370–374, doi: 10.1109/WAINA.2017.26.

[120] J.-W. Jang, H. Kang, J. Woo, A. Mohaisen, and H. K. Kim, "Andro-dumpsys: Anti-malware system based on the similarity of malware creator and malware centric information," *Comput. Secur.*, vol. 58, pp. 125–138, May 2016, doi: 10.1016/j.cose.2015.12.005.

[121] W.-C. Kuo, T.-P. Liu, and C.-C. Wang, "Study on Android hybrid malware detection based on machine learning," in *Proc. IEEE 4th Int. Conf. Comput. Commun. Syst. (ICCCS)*, Feb. 2019, pp. 31–35, doi: 10.1109/CCOMS.2019.8821665.

[122] Z. Li, W. Li, F. Lin, Y. Sun, M. Yang, Y. Zhang, and Z. Wang, "Hybrid malware detection approach with feedback-directed machine learning," *Sci. China Inf. Sci.*, vol. 63, no. 3, pp. 1–3, Mar. 2020, doi: 10.1007/s11432-018-9615-8.

[123] J. Li, Z. Wang, T. Wang, J. Tang, Y. Yang, and Y. Zhou, "An Android malware detection system based on feature fusion," *Chin. J. Electron.*, vol. 27, no. 6, pp. 1206–1213, Nov. 2018, doi: 10.1049/cje.2018.09.008.

[124] J. Lin, X. Zhao, and H. Li, "Target: Category-based Android malware detection revisited," in *Proc. ACM Int. Conf.*, 2017, pp. 1–9, doi: 10.1145/3014812.3014888.

[125] F. D. Rahmawati, R. B. Hadiprakoso, and R. N. Yasa, "Comparison of single-view and multi-view deep learning for Android malware detection," in *Proc. Int. Conf. Inf. Technol. Res. Innov. (ICITRI)*, Nov. 2022, pp. 53–58, doi: 10.1109/ICITRI56423.2022.9970205.

[126] D. Saif, S. M. El-Gokhy, and E. Sallam, "Deep belief networks-based framework for malware detection in Android systems," *Alexandria Eng. J.*, vol. 57, no. 4, pp. 4049–4057, Dec. 2018, doi: 10.1016/j.aej.2018.10.008.

[127] M. Spreitzenbarth, T. Schreck, F. Echtler, D. Arp, and J. Hoffmann, "Mobile-sandbox: Combining static and dynamic analysis with machine-learning techniques," *Int. J. Inf. Secur.*, vol. 14, no. 2, pp. 141–153, Apr. 2015, doi: 10.1007/s10207-014-0250-0.

[128] S. Sun, X. Fu, H. Ruan, X. Du, B. Luo, and M. Guizani, "Real-time behavior analysis and identification for Android application," *IEEE Access*, vol. 6, pp. 38041–38051, 2018, doi: 10.1109/ACCESS.2018.2853121.

[129] W. Wang, C. Ren, H. Song, S. Zhang, and P. Liu, "FGL_Droid: An efficient Android malware detection method based on hybrid analysis," *Secur. Commun. Netw.*, vol. 2022, pp. 1–11, Apr. 2022, doi: 10.1155/2022/8398591.

[130] L. Wei, W. Luo, J. Weng, Y. Zhong, X. Zhang, and Z. Yan, "Machine learning-based malicious application detection of Android," *IEEE Access*, vol. 5, pp. 25591–25601, 2017, doi: 10.1109/ACCESS.2017.2771470.

[131] M. Zheng, M. Sun, and J. C. S. Lui, "Droid analytics: A signature based analytic system to collect, extract, analyze and associate Android malware," in *Proc. 12th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. Trust*, Dec. 2013, pp. 163–171, doi: 10.1109/TrustCom.2013.25.

[132] A. Sinha, F. Di Troia, P. Heller, and M. Stamp, "Emulation versus instrumentation for Android malware detection," in *Digital Forensic Investigation of Internet of Things (IoT) Devices*. Cham, Switzerland: Springer, 2021, pp. 1–20.

[133] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "EMULATOR vs REAL PHONE: Android malware detection using machine learning," in *Proc. 3rd ACM Int. Workshop Secur. Privacy Anal.*, Mar. 2017, pp. 65–72, doi: 10.1145/3041008.3041010.

[134] M. Alazab, M. Alazab, A. Shalaginov, A. Mesleh, and A. Awajan, "Intelligent mobile malware detection using permission requests and API calls," *Future Gener. Comput. Syst.*, vol. 107, pp. 509–521, Jun. 2020, doi: 10.1016/j.future.2020.02.002.

[135] K. G. Mehrotra, C. K. Mohan, and H. Huang, *Anomaly Detection Principles and Algorithms*. Cham, Switzerland: Springer, 2017.

[136] S. Dreiseitl, M. Osl, C. Scheibböck, and M. Binder, "Outlier detection with one-class SVMs: An application to melanoma prognosis," in *Proc. AMIA Annu. Symp.*, 2010, pp. 172–176.

[137] (2023). *Classification and Grouping of Linux System Calls*. Accessed: Oct. 12, 2023. [Online]. Available: http://seclab.cs.sunysb.edu/sekar/papers/syscallclassif.htm

[138] J. Qiu, B. Yadegari, B. Johannesmeyer, S. Debray, and X. Su, "Identifying and understanding self-checksumming defenses in software," in *Proc. 5th ACM Conf. Data Appl. Secur. Privacy*, Mar. 2015, pp. 207–218, doi: 10.1145/2699026.2699109.
[139] Y. Shen, R. Chien, and S. Hung, "Toward efficient dynamic analysis and testing for Android malware," *INPRA*, vol. 2, no. 3, pp. 14–23, 2014.
[140] S. S. Chakkaravarthy, V. Vaidehi, and P. Rajesh, "Hybrid analysis technique to detect advanced persistent threats," *Int. J. Intell. Inf. Technol.*, vol. 14, no. 2, pp. 59–76, Apr. 2018, doi: 10.4018/ijiit.2018040104.
[141] A. Sadeghi, H. Bagheri, J. Garcia, and S. Malek, "A taxonomy and qualitative comparison of program analysis techniques for security assessment of Android software," *IEEE Trans. Softw. Eng.*, vol. 43, no. 6, pp. 492–530, Jun. 2017, doi: 10.1109/TSE.2016.2615307.
[142] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Are your training datasets yet relevant?" in *Engineering Secure Software and Systems*. Cham, Switzerland: Springer, 2015, pp. 51–67, doi: 10.1007/978-3-319-15618-7_5.
[143] *The Economic Value of Prevention in the Cybersecurity Lifecycle*, Ponemon Inst. LLC DeepInstinct, Michigan, MI, USA, Apr. 2020.
[144] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Trans. Comput. Syst.*, vol. 32, no. 2, pp. 1–29, Jun. 2014, doi: 10.1145/2619091.
[145] P. Lantz. (2012). *Droidbox*. Accessed: Oct. 30, 2023. [Online]. Available: https://github.com/pjlantz/droidbox
[146] A. Delaporte and K. Bahia, "The state of mobile Internet connectivity 2021 connected society," GSMA Intell., London, U.K., Tech. Rep., 2021, pp. 1–66. [Online]. Available: https://www.gsma.com/r/wp-content/uploads/2021/09/The-State-of-Mobile-Internet-Connectivity-Report-2021.pdf
[147] M. Krzysztoń, B. Bok, M. Lew, and A. Sikora, "Lightweight on-device detection of Android malware based on the koodous platform and machine learning," *Sensors*, vol. 22, no. 17, p. 6562, Aug. 2022, doi: 10.3390/s22176562.
[148] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of Android malware in your pocket," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 1–12, doi: 10.14722/ndss.2014.23247.
[149] G. Renjith and S. Aji, "On-device resilient Android malware detection using incremental learning," *Proc. Comput. Sci.*, vol. 215, pp. 929–936, Jan. 2022, doi: 10.1016/j.procs.2022.12.095.
[150] C.-H. Tsai, C.-Y. Lin, and C.-J. Lin, "Incremental and decremental training for linear classification," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2014, pp. 343–352, doi: 10.1145/2623330.2623661.
[151] N.-T. Chau and S. Jung, "Dynamic analysis with Android container: Challenges and opportunities," *Digit. Invest.*, vol. 27, pp. 38–46, Dec. 2018, doi: 10.1016/j.diin.2018.09.007.
[152] L. Xu, G. Li, C. Li, W. Sun, W. Chen, and Z. Wang, "Condroid: A container-based virtualization solution adapted for Android devices," in *Proc. 3rd IEEE Int. Conf. Mobile Cloud Comput., Services, Eng.*, Mar. 2015, pp. 81–88, doi: 10.1109/MOBILECLOUD.2015.9.
[153] Zimperium. (2022). *Global Mobile Threat Report*. [Online]. Available: https://get.zimperium.com/2022-global-mobile-threat-report/
[154] L. Bilge and T. Dumitraǧ, "Before we knew it: An empirical study of zero-day attacks in the real world," in *Proc. ACM Conf. Comput. Commun. Secur.*, Oct. 2012, pp. 833–844, doi: 10.1145/2382196.2382284.
[155] F. Deldar and M. Abadi, "Deep learning for zero-day malware detection and classification: A survey," *ACM Comput. Surv.*, vol. 56, no. 2, pp. 1–37, Feb. 2024, doi: 10.1145/3605775.
[156] Y. Guo, "A review of machine learning-based zero-day attack detection: Challenges and future directions," *Comput. Commun.*, vol. 198, pp. 175–185, Jan. 2023, doi: 10.1016/j.comcom.2022.11.001.

RAJIF AGUNG YUNMAR (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree with the Department of Electrical and Information Engineering, Universitas Gadjah Mada (UGM), Yogyakarta, Indonesia. He is also a Lecturer with the Department of Informatics Engineering, Institut Teknologi Sumatera (ITERA). His research interests include cybersecurity and mobile security, with an emphasis on identifying and mitigating potential security threats to mobile devices and the networks they connect to. He hold several industry-recognized cybersecurity certifications, including Certified Ethical Hacker (CEH), HCIA-Security, Certified Incident Handler (ECIH), and Cisco CCST Cybersecurity.

SRI SUNING KUSUMAWARDANI received the bachelor's, master's, and Ph.D. degrees in electrical and information engineering from Universitas Gadjah Mada (UGM), Yogyakarta, Indonesia. In 2009, she was a Doctoral Research Fellow with the Faculty of Engineering Education, Utah State University, USA. Additionally, she was a Doctoral Research Fellow with the Erasmus Mundus Research Program, Universitat Polytecnica de Catalunya, Barcelona, Catalonia, Spain, from 2012 to 2013. She is currently a Professor with the Department of Electrical and Information Engineering, Faculty of Engineering, UGM. Her research interests include using AI and machine learning techniques to enhance the security of computer systems.

WIDYAWAN received the bachelor's degree in electrical and information technology from Gadjah Mada University, in 1999, the master's degree from Erasmus University, in 2003, and the Ph.D. degree from Cork Institute of Technology, in 2009. He is currently an Assistant Professor and a Lecturer with Universitas Gadjah Mada. His research interests include the seamless integration of computing into everyday life through the use of machine learning, location technology, and ubiquitous computing.

FADI MOHSEN received the bachelor's degree from the University of Jordan, in 2006, the master's degree from the University of Colorado at Colorado Springs, in 2010, and the Ph.D. degree from the University of North Carolina at Charlotte, in 2016. He is currently an Assistant Professor and a Lecturer with the University of Groningen, The Netherlands. His research interests include cyber security, particularly in the domains of web, computer, and mobile phone security. His research endeavors involve scrutinizing access control mechanisms, detecting potential vulnerabilities, implementing countermeasures, and exploring user awareness and comprehension of such vulnerabilities and countermeasures, especially in connection with third-party applications in computing systems.