IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received February 6, 2018, accepted March 11, 2018, date of publication March 13, 2018, date of current version April 18, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2815660

Malware Threats and Detection for Industrial Mobile-IoT Networks

SHAILA SHARMEEN[1], SHAMSUL HUDA[1], (Member, IEEE), JEMAL H. ABAWAJY[1], (Senior Member, IEEE), WALAA NAGY ISMAIL[2], AND MOHAMMAD MEHEDI HASSAN[2], (Member, IEEE)

[1]School of Information Technology, Deakin University, Burwood, Melbourne, Australia
[2]Information Systems Department, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia

Corresponding author: Mohammad Mehedi Hassan (mmhassan@ksu.edu.sa)

This work was supported by the Deanship of Scientific Research, King Saud University, through Research Group under Grant RGP-281.

ABSTRACT Industrial IoT networks deploy heterogeneous IoT devices to meet a wide range of user requirements. These devices are usually pooled from private or public IoT cloud providers. A significant number of IoT cloud providers integrate smartphones to overcome the latency of IoT devices and low computational power problems. However, the integration of mobile devices with industrial IoT networks exposes the IoT devices to significant malware threats. Mobile malware is the highest threat to the security of IoT data, user's personal information, identity, and corporate/financial information. This paper analyzes the efforts regarding malware threats aimed at the devices deployed in industrial mobile-IoT networks and related detection techniques. We considered static, dynamic, and hybrid detection analysis. In this performance analysis, we compared static, dynamic, and hybrid analyses on the basis of data set, feature extraction techniques, feature selection techniques, detection methods, and the accuracy achieved by these methods. Therefore, we identify suspicious API calls, system calls, and the permissions that are extracted and selected as features to detect mobile malware. This will assist application developers in the safe use of APIs when developing applications for industrial IoT networks.

INDEX TERMS Industrial mobile IoT, threats, malware, detection method, machine learning.

I. INTRODUCTION

In the smart industry arena, smartphones are extensively integrated with industrial Internet of Things (IoT) networks. IoT cloud providers are integrating cooperative mobile devices to reduce the transmission latency with remote data centers, and to increase the computation and storage capabilities of IoT networks, resulting in significant reduction in service execution time.

Despite the sensitivity of mobile platforms and their potential for abuse, several security issues not too dissimilar to those already affecting traditional IT network counterparts are beginning to surface. The most significant issue is the emergence of traditional malware such as viruses, worms, Trojan horses, and rootkits. Malicious software in this context behaves similarly to the same threats on traditional IT networks. In this case malware may be targeted at ex-filtrating sensitive data from the mobile platform or further leveraging the compromised asset to access sensitive industrial/critical infrastructures in IoT networks in which the mobile device has legitimate access to the corresponding IoT devices. Second, developers are beginning to offer software for free and instead generate revenue with data analytics and marketing which can invade a user's privacy. An example might be a free address book application that while managing a user's contact list may also provide a copy of this contact list to the developer for marketing purposes. Although not strictly malicious, this behavior is also not always apparent to the user and may not be in their best interest. Researchers have already developed malware which can abuse microphone permissions to subvert the microphone to record sensitive user information input from a keypad as detailed in [64]. Researchers in [65] also find that Cloaker, a stealthy rootkit, exploits features of the ARM processor in order to hide itself. There are two specific ARM hardware features utilized by Cloaker, the ability to change the location of the interrupt vector and the ability to lock addresses in the translation lookaside buffer. This second technique allows memory to be stealthily mapped into processes without modifying the OS level (detectable) page table entries. This demonstrates several security weaknesses in mobile architectures which are significant threats for industrial control systems specific to mission critical systems.

The safety and security of data, corporate information, identity, and the resources of the mobile phones used in IoT networks can be threatened if the malware is not detected as early as possible. The industry may face financial loss, loss of data, loss of corporate information, and identity theft because of the undetected malware present in mobile devices [56]. Users may be unaware of what is happening in the background and may lose personal information that they do not wish to share or expose. Malware detection is immensely important as our data, identity, and resources are at risk. We need to identify malicious applications as early as possible, deactivate them, and take security measures to protect our information on the mobile platform.

Detection of malware is a crucial task, as malware developers hide their malicious activities and introduce new methods to avoid detection [18]. Anti-malware software must cope with new technologies such as code obfuscation, mimicry attacks etc. For Windows and other operating systems, use of resources is not a critical concern, whereas for the operating system of mobile devices resource usage is always a concern. Limited use of resources means that the process of malware detection is not a straightforward problem for the mobile platform. Detecting the malicious activities of mobile applications using limited resources within a limited time period is a challenge to researchers.

Significant research has been carried out to detect malware on the Android mobile platform. However, the detection process usually suffers from limitations. Researchers have used static, dynamic, and hybrid processes to detect mobile malware and malicious activities. Researchers' primary concerns involve accuracy levels, and most the research papers describe the performance of their detection process using accuracy metrics. For the operating system of mobile devices, performance overhead should be considered, as higher accuracy may cause higher overhead. Accuracy and performance overhead need to be well balanced to make the detection process efficient. Moreover, malware must be detected as early as possible before doing any harm to the system. Research indicates that 260,000 devices have been infected by the DroidDream malware within 48 hours before the malware was detected [50]. Therefore, malware should be detected as soon as it enters the apps market place. Researchers need to focus on this major concern involving early detection to ensure Zero-day malware detection.

We analyze the ongoing research efforts covering the three basic categories: static, dynamic, and hybrid analysis. These analyses represent the data set, features, feature selection method, detection method, and the accuracy. We also have mentioned the literature gap and the limitations of current research efforts. Thereby we have identified the suspicious feature lists which are commonly used by malware developers. The main contributions of this research effort are divided into the following main areas:

- Defining the mobile malware detection process for IoT networks.
- Determining the security limitations for mobile platforms in industrial IoT networks.
- A comparative analysis of static, dynamic, and hybrid detection processes and their limitations and scopes.
- Identifying the suspicious permission, API call, and system call lists to enable IoT application developers in the safe use of APIs.

The rest of the paper is organized such that the problem definition is discussed in Section II. The motivation is discussed in Section III followed by the security loop holes of Android in Section IV. The detection process and performance matrices are discussed in Sections V and VI, respectively. Comparison of detection methods and the outcomes of the comparative analysis are discussed in Sections VII and VIII, followed by the Conclusion.

II. PROBLEM DEFINITION

This study involves the malware detection process for the Android platform. There are known and unknown malware and benign apps in the market. Known malware is removed from the market place. Unknown malware evades the detection engine by hiding its malicious activities. We identify these unknown malwares and the benign apps based on binary classification. If there are a finite set of classes, then the classifier will determine the class of a given object. Binary classification involves only two sets of possible classes. In our stated problem, we must classify apps into two finite classes: malware or benign ware. Thus, we can represent this problem as a binary classification problem [63]. We can consider all apps as a set of applications, A. There are different types of applications available in A.

$$A = \{A_1, A_2, A_3, \ldots.., A_n\} \tag{1}$$

In this A set there are n applications. These applications may be malware, benign ware, or new unknown applications. Each application A_i has been defined with some feature vector F_i and class label CL_i.

$$A_i = \{F_i, CL_i\} \tag{2}$$

Here, F_i is a feature vector with a number of selected features, k.

$$F_i = \{F_1, F_2, F_3, \ldots\ldots, F_k\} \tag{3}$$

Class label $[\!(\mathrm{CL})\!]_i$ defines the label of the class where there is a label for benign ware, malware, and Unknown apps.

$$CL_i = \left\{CL_b, CL_m, CL_{uap}\right\} \tag{4}$$

Here, CL_b is the class label of benign ware, CL_m is the class label of malware, and CL_{uap} is the class label of unknown apps.

In the problem definition, a detection engine DE classifies CL_{uap} into either CL_b or CL_m. The detection engine will have a malware detection function (μ) and an achieved accuracy level (Acc_l). The malware detection function will use the feature set and determine the class label. The detection engine will use the detection mechanism to classify the set of apps A by providing each with label CL_b or CL_m.

The definition of the malware detection function is given below:

$$\mu(F): F \rightarrow CL \tag{5}$$

It is desired that the detection engine has maximum accuracy.

III. MOTIVATION

The use of smart phones is increasing daily. Among all smart phones, most use Android as the operating system, with Android users increasing at a significant rate. In the United States, Android users numbered 107.7 million in 2016, whereas the number was 98.5 million in 2015, and 87.7 million in 2014 [35]. The growth rate was 9.34% in 2016 and it is obviously a high growth rate. Moreover, the global market share as of 2016 for the Android operating system was 86.2%, whereas Apple's iOS was only 12.9%, Microsoft 0.6%, RIM 0.1%, and for others was 0.2%. In comparison with the fourth quarter of the year 2015, market shares were 80.7%, 17.7%, 1.1%, 0.2%, and 0.2%, respectively [36]. From these figures, it is obvious that the global market for Android shows rapid growth whereas iOS and Microsoft show negative growth. The popularity of Android is clearly obvious from these given data for 2016 and 2015. This motivated us to choose the Android platform to work with, as security improvements will benefit a large number of mobile users.

Mobile apps developers generally choose to develop apps for the Android platform because of its greater global market. Seemingly every day new mobile applications for Android enter the market. Android allows third party applications, which opened a new horizon in the development of apps, but includes some disadvantages; this type of opportunity, and the immense popularity of Android, has attracted malware developers. The volume of Android mobile malware showed an increase of 230% in 2015, which is alarming [37]. Thus, there is a need for more secure methodologies to detect malware efficiently and promptly. Malware detection becomes a necessity because of the availability of applications from different sources. These malwares can obtain access to personal information, bank details, user name, photos, and other sensitive information. Some can send Short Mail Services (SMS) messages, email, or even install other applications without notifying the user. Some of these need root privileges, which users may allow without knowing the security threats. Obviously, users should be aware of the threats to their identity, leakage of information, or even financial forgery because of these applications.

Malware is a program or a set of programs that can cause harm to identity, data or sensitive information, financial forgery, and resources. According to a survey [37], the top ten Android mobile malwares are given in Table 1 below:

In the year 2016, approximately 87% of smart phone users used Android. Table 2 indicates that the Android market shows positive growth and others show negative growth.

TABLE 1. Top android malware.

Android.Lotoor 36.8%	Android.AdMob 3.3%
Android.RevMob 10%	Android.lconosis 3.1%
Android.Malapp 6.1%	Android.Opfake 2.7%
Android.Fakebank. B 5.4%	Android.Premiumtext 2.0%
Android.Generisk 5.2%	Android.Basebridge 1.7%

TABLE 2. OS comparison [38].

Period	Android	iOS	Windows Phone	Others
2015Q3	84.3%	13.4%	1.8%	0.5%
2015Q4	79.6%	18.6%	1.2%	0.5%
2016Q1	83.4%	15.4%	0.8%	0.4%
2016Q2	87.6%	11.7%	0.4%	0.3%

Security measures are designed to protect mobile platforms. However, existing measures are insufficient considering the causes for potential security threats in the Android OS as given below.

- Exponential growth in the number of malicious applications [49], [56]. Most of the market share for global smartphones belongs to Android. This attracts malware developers. Malware developers choose to target the Android platform for this reason [54].
- The Android Market Place is the public market for apps. Without prior review, developers can place their apps on the Android Market Place [40]. The openness of the Android Market Place allows malware developers to place malicious apps. It is a challenging task to detect the new malware and remove it from the public market place before infection [50]. In 2012, malwares were downloaded 700,000 times before being detected by Google and automatic uninstallation was not possible [56]. In the market place, it is difficult to verify each app and lack of a proper verification mechanism is a reason for the growth of malware [50].
- As a further challenge, variants of malware are created to avoid detection [50]. Because of the diversity of the malware families, all cannot be detected using a fixed set of features [50]. It is a challenging task to investigate each app in the Android Market Place to identify the hidden malware.

Users can and should make some decisions regarding security issues, as unaware users may cause security issues. Users may allow apps which need internet permission, even allowing SMS permission, even though there is no reason to allow sending SMS messages [56]. The lack of knowledge and awareness by users regarding security concerns make the problem more difficult to handle. Moreover, the need for anti-malware software for the Android platform is not considered essential by most users [56]. The lack of knowledge is not only characteristic of the Android user; it is also present in unskilled app developers. Inappropriate use of the security measures of Android by unskilled developers create unintentional loop holes that lead to security threats to users and their data [39].

A wide range of Android operating systems are used. Old versions of Android may suffer from different security issues [55]. Additionally, many smartphone and tablet vendors run third party apps markets which may act as a source of malicious applications [53]. People from China and Asia choose to download apps in their local languages which are available at third party apps store and are a potential source of malware [56].

Crowd sourcing is used for the prevention of malware in Android. However, fake reviews from users can be a security threat. Joining a site such as Admob is comparatively easier than joining iAd as no identity proof is required. This encourages the ad-based malware developers [55].

These security threats can cause the following issues.

Personal data leakage: People are not concerned with the security of data or personal information in mobile devices while they are normally very concerned for the same in PC environments [54]. Some apps steal personal information and at the same time demand payments. Such Trojan apps have been downloaded 9,252 times and 211 affected users paid a total of $250,000 to the malware developers [56]. Malware developers successfully stole personal data such as contacts, emails, SMS, and device information which can be used in identity theft and spamming [56].

Social: GPS location, call log, and contact lists can be captured by malware [56]. The contact list and location are user-sensitive information. This information can be captured by malware and can do harm by leaking social identity that can be used in various ways to threaten the security of a user's social image.

Business: Business organizations have their own apps to run their business. Malware can capture user information or business data which will put the business organization at a risk. The business owner will be at a risk of financial loss as well as reputation.

Financial loss: The motive of malware development has changed and now focuses on financial gain [53]. Capital expenses related to malware average $6–7bn dollars in a fiscal year [53]. "Zeus in the Mobile" is a Trojan that captures the authentication code of the user in a banking application, which may cause financial losses to the user. It is also expensive to remove, where a security firm charged $21/s for the first detection in 2010 [54]. This type of malware can cause user financial losses as well as large financial losses to a business owner in detection fees. In some cases, a user may have to pay large phone bills for premium rate services because of the malicious activity of an app [56].

Considering all the security threats, we can state that a proper detection engine that will detect new and unknown malware to ensure Zero-day detection is a crucial need in the current era. This survey will help researchers to fulfill this crucial need. It will show the direct impact between the security of the mobile application and the user. A proper malware detection system will provide a healthy environment for nursing the mobile-based economy. It will identify many elements of security threats involved in using mobile phones and applications, and the user will feel confident in using these applications

IV. SECURITY LOOP HOLES

Android is an open source operating system for mobile devices and is very popular among smart phone. Developers have designed and introduced many applications for the Android Platform [39]. Instead of having a main function, developers design applications in terms of components.

Android is based on the Linux operating system. There are three layers, named the kernel layer, Android middleware, and Android application layer. The lower layer is the kernel. The kernel is used for device drivers, memory management, process management, and networking [59]. The middleware connects the upper application layer and the lower kernel layer [53]. The application layer has applications with different types of components. Components can communicate with each other, which is called Inter Component communication. The components are namely Activity, Service, Content Provider, and Broadcast receiver. Activity defines the user interface. Service defines the designed actions. Content providers help to store and access data from the database. Broadcast receiver acts as a media to distribute messages to a destination.

The Android security architecture and the apps architecture were designed to provide flexibility in the development of apps. Improper usage and lack of awareness in developing secure application create loop holes which may threaten the security of the data, identity, and may cause financial loss. Security loop holes of the Android Operating system are given below.

Components may be private or public. Private component will reduce the security measures, so there is a need for care when defining private components [39]. If the permission of a public component is not defined in a manifest file, any other application can access it. If we use the default setting, it may lead to security threats [39].

If the permission for broadcasting is not defined, any other application can access it. Unprotected broadcasts may be the cause of leaked information involving personal, sensitive, and/or financial information [39].

Content provider controls the access to data. Separate read/write permissions and URLs are needed to ensure that separate read and write permissions are used to avoid the risk of compromising data integrity. Content sharing should be specific, and not all or none [39].

Pending Intent objects allow another application to "finish" an operation via Remote Procedure Call (RPC), which may allow filling of unspecified values by any remote application. It introduces delegation, which may have negative impact on security [39].

checkPermission() can be invoked by a component. The developer can add a reference hook monitor. Service hooks allow moving a security policy to the application code, which may also be a security threat cause [39].

As Android does not have a native Windows system and also does not support all GNU libraries, it is difficult to port all applications or Linux libraries to all smartphone/tablet platforms [53].

Multitasking allows running an application in the background while another application is running in the foreground and the user may be unware of the background application. Malware usually runs in the background with highest priority and thus increases its lifetime to carry out malicious activities while the user is unaware of the suspicious activities [53].

Killed activities can reboot and return to the same state because the kernel keeps stores the killed activity and its state. This allows the malware to re-activate even if its activity was killed [53].

In microSD, the data storage is not password locked at the operating system level, which allows access to it [53]. Data flow can be observed easily by installing a debugging app [53]. GPS data can be observed, as it is in plain text format [53]. There is no mechanism to check an email header to detect spoofing or other attacks, making the email system insecure in Android [53].

Any application can read the information of the system, such as installed apps, device information, IMEI, IMSI, phone numbers, and others [55]. The content of the SD card is readable, as automatic readable permission is provided to it. Developers may use the SD card as storage, being unware of the threat [55].

In the manifest file, some developers forget to set the "debuggable = false" switch, which causes a major security hole and may allow malicious activities [55]. If the developers do not want their components to be accessed by other components, then components must be designed with care and the export flag should be set to "false" in the manifest file. Most developers are not aware of this, and those who know may forget the requirement [55]. SQL injection can be accomplished by obtaining access to the single database shared by multiple content providers [55].

The diversity in languages and platforms in Android apps development makes it difficult to assure standardized security procedures [55]. Unskilled developers and their poor development practices represent one of the main causes of security loop holes. For example, storing passwords in plain text, debugging features enabled, unnecessary permissions, etc. lead to vulnerable apps [55].

Poor understanding of security issues, protocols, and lack of awareness of best programming practices by new and even professional developers lead to security threats [56]. Excessive requested permissions are one indicator of suspicious apps [56].

Key security concerns involve cryptographic issues, CRLF injection, and information leakage when using Android apps [55]. In the same manner as a Wi-Fi based network, an attacker can capture the communication between a mobile device and mobile network and can redirect the communication to a server and know and record all movements of a user [56].

Execution of Unstructured Supplementary Service Data (USSD) control codes should be handled carefully. Otherwise, malware can wipe all data from the device [56]. iOS uses code-signing and then verifies an app, whereas Android does not use code signing. Android's Bouncer app suffers from a weakness involving fake developer accounts as it does not require proof of identity [56].

Bouncer, an automated tool to detect malware in Google Play runs in only five minutes. To avoid detection, malware can introduce malicious activity after the known five minutes have elapsed [56]. As apps are checked by Bouncer after a certain period, the malware developer determines Bouncer's IP address and sets execution of malicious activities on devices with different IP addresses [56].

Encryption of apps as introduced by Google does not prove the validity of the apps and also suffers from failing to run in many devices [56]. Malware developers can avoid the certificate check by disabling it or introducing a new certificate [56].

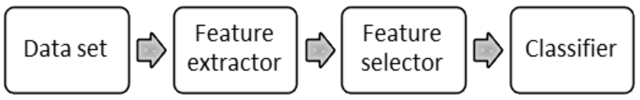

FIGURE 1. **Components of malware detection system.**

V. MALWARE DETECTION PROCESS

Malware detection is a process that contains different components to evaluate whether an application is malware or not. Generally, there are four components of the malware detection system namely, data set, feature extractor, feature selector, and classifier. Figure 1 illustrates a malware detection system,

A. DATA SET

The first component of the malware detection system is the data set. The data set represents a representative collection of malware and benign ware. A proper data set is essential for analysis of the behavior of the malware. We need examples of both the malware and the benign apps for proper detection. The most common sources of Android Mobile Malware are the Genome project, Contagio, DREBIN data set, Virus Share, etc. The most widely used sources for known benign apps are Google Play, App China, Amazon, Android Market, etc. Malwares can hide their malicious activities and be available in Android market places. Android apps are available in both official and third-party apps market places. They are also good sources of Android applications which can be used as the data set in the detection engine.

B. FEATURE EXTRACTOR

The feature Extractor is the component which extracts the desired features from the malware and benign apps.

The feature extractor can extract features from the manifest file, dex file, byte code, and log file. Based on the type of extracted feature set, we can categorize the feature extractor as a static feature extractor, dynamic feature extractor, or hybrid feature extractor. Some feature extractors extract features from a single category and some extract features from multiple categories. We can categorize the feature extractors as shown in Figure 2.

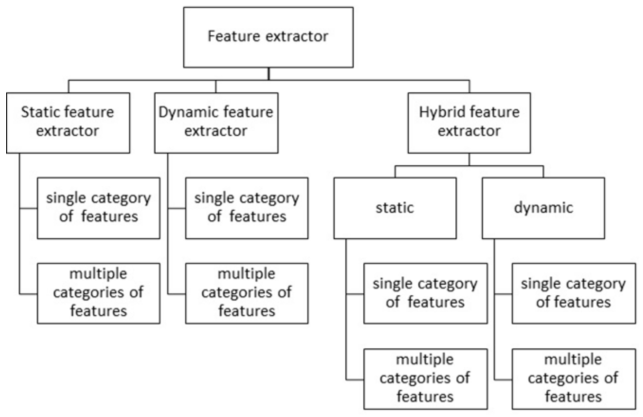
Feature extractor
Static feature extractor
Dynamic feature extractor
Hybrid feature extractor
single category of features
multiple categories of features
single category of features
multiple categories of features
static
dynamic
single category of features
multiple categories of features
single category of features
multiple categories of features

FIGURE 2. Types of feature extractor.

The features that can be extracted by not executing the application are called static features. The mechanism that extracts the static features is called a Static Feature Extractor. Researchers may consider only a single category of static features and others consider a set of multiple static features. Common static features are: Permission, API call, String extracted, Native commands, XML elements, Meta data, Intents, Broadcast receivers, Hardware components, etc.

Static features can be extracted from the manifest file, dex file, and byte code. The most widely used tool is the APK tool. From the APK tool, we obtain the APK file, Manifest file, classes.dex, and Smali file. From these files, we can extract features. The feature extraction process is illustrated in Figure 3.

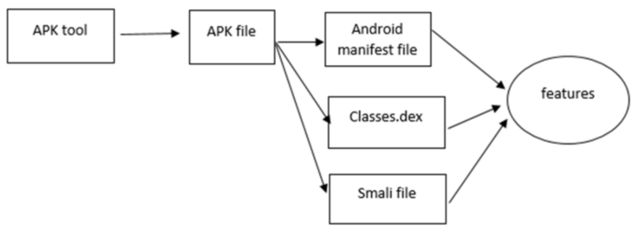

FIGURE 3. Feature extraction using APK tool.

The features that can be extracted from an application by executing it are called dynamic features. The mechanism that extracts the dynamic features is called the Dynamic Feature Extractor. As the static features are not able to represent the full characteristic of an app, researchers use dynamic features, some researchers consider only a single category of dynamic features and others consider a set of multiple dynamic features. Common dynamic features are: System call, Network traffic, SMS, Process id, Process information, Memory usage, IP address, Log events Power consumption, System component, User interaction, etc.

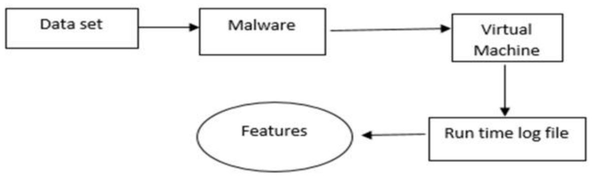

FIGURE 4. Feature extraction using virtual machine.

The log file is generated by executing the application in a virtual machine. From the data set, we collect the malware and benign ware and execute these in the virtual machine (Figure 4), which generates the run time log file. From the log file, we can extract the dynamic features.

To obtain better performance in the detection of malware, researchers use both the static features and the dynamic features. One or multiple static and one or multiple dynamic features are combined to obtain higher accuracy and proper detection of malware. These are called hybrid features. The hybrid feature extractor combines the static feature extractor and the dynamic feature extractor. Common hybrid features are: system call and permission, system call and API call, permission and network traffic, API call, intents and system call, and many more.

C. FEATURE SELECTION

Feature selection is an important task in malware detection. There are many features, but we need to select those among them which will provide better accuracy in the classification process. Widely used feature selection methods for Mobile Malware detection are:

- Information gain algorithm
- Select mostly features used in previous research
- Select a subset of all features (top 20, top 50, etc.)

D. DETECTION METHOD

The detection method or the classifier is used to determine whether an app is malware or not. Based on the features, the classifier classifies apps as malware or benign ware. Most classifiers use machine learning. Classifiers based on machine learning use one or multiple classifiers. Layered classifiers may also be used in the detection process. In this case there are two or three layers and each layer contains a classifier to improve the accuracy of the detection system. In parallel classifiers, there are various individual classifiers. The outputs of these classifier are combined to obtain higher accuracy. Other classifiers such as AHP and penalty calculation do not use the machine learning approach. Common machine learning classifiers used in Android mobile malware detection are: SVM, KNN, J48, RF, Naïve Bayes, C4.5, PART, RIDOR, and many more. Figure 5 categorizes the classifiers.

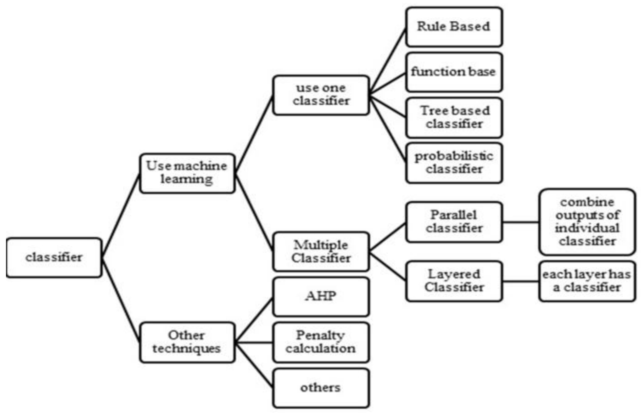

FIGURE 5. Classification of detection method.

VI. PERFORMANCE MATRIX

The True Positive Rate (TPR) defines the percentage of benign apps identified accurately, where TPR=TP/(TP+FN). TP is the number of accurately identified benign apps and FN is the number of incorrectly identified benign apps.

The False Positive Rate (FPR) defines the percentage of incorrectly identified malware apps, where FPR=FP/(TN+FP). FP is the number of incorrectly identified malware and TN is the number of correctly identified malware.

Accuracy is a metric used to describe the overall performance. Accuracy is the percentage of correctly identified apps, where ACC=(TP+TN)/(TP+TN+FP+FN).

VII. COMPARISON OF DETECTION METHODS

Mobile Malware detection commonly involves three techniques, namely static, dynamic, and hybrid detection techniques. The static analysis extracts features without execution of the application. The dynamic analysis involves the execution of the application, and hybrid analysis uses both static and dynamic features for the classification. Table 3 compares the advantages and drawbacks of these methods.

A. STATIC ANALYSIS ON ANDROID MOBILE MALWARE DETECTION

In the following section, we will discuss the static analysis of mobile malware detection. We mention the name of the research work, year, type, a short description, data set, detection method, major outcome, and limitations/future work.

1) DATA SET

The data sets that are used in the static analysis involve a wide range of malware and benign ware. The difficulty lies in obtaining a proper data set of malware because each day different types of apps are introduced in the official and third-party market places of Android. There is a corresponding quick development in the number of malwares and their types. Malware authors hide malicious activities and introduce newer techniques such as code obfuscation, update attacks, etc. If we compare the number of malware and benign applications used in previous research, the result is Table 4.

TABLE 3. Comparing static, dynamic and hybrid analysis.

Type		Benefits	Limitations
Static Analysis	Single category of features	Easy to extract. Low computation.	Code obfuscation. Mimicry attack. Low accuracy.
	Multiple categories of features	Easy to extract. High Accuracy.	Code obfuscation. Mimicry attack. Difficult to handle multiple features. High computation.
Dynamic Analysis	Single category of features	Accuracy better than static. Recover code obfuscation.	Difficult feature extraction process. High computation. Still not a better choice.
	Multiple categories of features	Better Accuracy than static and dynamic with only single category of features. Recover code obfuscation. Relatively high Accuracy	Difficult feature extraction process. High computation. More resource needed. Time consuming.
Hybrid Analysis		Highest accuracy achieved. Better than static and dynamic analysis.	Highest complexity. Need a framework to combine static and dynamic features. More resource needed. Time consuming.

TABLE 4. Comparison based on the number of applications used in experiments.

Ref.	#Malware	#benign apps
[1]	Data set1 200 samples Data set2 500 samples	
[2]	480	124769
[19]	45	300
[3]	238	1500
[4]	5264	12026
[5]	378	324658
[6]	3987	500
[20]	175	621
[21]	1446	2338
[7]	5560	123453
[25]	1536	28548
[26]	2925	3938
[8]	238	1500
[9]	11000 samples	
[24]	3368	6005
[22]	6405	35657

From this comparison table, we can see that for static analysis, the highest number of malware was used in [22]. The Drebin data set [7] involves the malware and benign ware where the number of malware is the second highest and the number of benign apps is the highest.

2) FEATURE EXTRACTION

Static features are extracted mainly from the Manifest file. Permissions are extracted from the Manifest file. The applications are applied to the APK tool which generates the APK file. The AndroidManifest.xml file contains the permission set. This xml file can be decompiled by the AXMLPrinter2 tool to obtain the permission [20] (Figure 6).

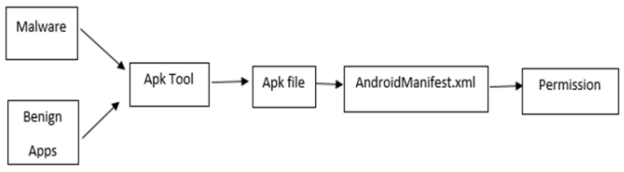

FIGURE 6. Permission extraction.

API calls are extracted from the APK file [20]. The APK file contains the classes.dex file [21]. This file is decompiled by the dex2jar tool to obtain the jar file. Using the jad tool, we obtain the Java file from the jar file. API calls are extracted from this Java file [20] (Figure 7).

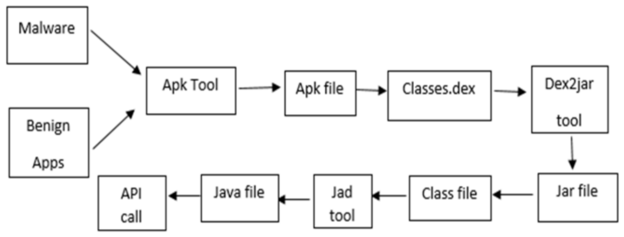

FIGURE 7. API Call.

3) FEATURE SELECTION

Among the total 145 permissions set [41], permissions are selected as desired features to verify malicious activities. To select the permissions, the authors have used the information gain technique [20]. Permissions with low gain value are marked as safe and others are considered risky permissions. Another research determined risky permissions using risk factors [42]. They ranked permissions using three ranking methods. The ranking methods used were correction coefficient, mutual information, and T-tests. According to [42], the riskiest permissions are given in Table 5.

Another research proposed the top twenty permission sets [31]. They considered twenty permissions that were used in a previous research work. All permissions are covered in [42] except WRITE_APN_SETTINGS and VIBRATE.

The research work in [5] mentioned twenty-six permissions that are mostly used by malware. Most are also used in [31] and [42]. Only three permissions are new here: RECORD_AUDIO, ACCESS_LOCATION_EXTRA, and INSTALL_PACKAGES.

Reference [21] selected a total of 34 permissions. Among these permissions, most are used in other research papers. DELETE_PACKAGES, MOUNT_UNMOUNT_FILESYSTEMS, INSTALL_SHORTCUT, READ_LOGS, CALL

TABLE 5. Risky permission set.

READ_SMS	RECEIVE_BOOT_COMPLETED	WRITE_CONTACTS
RECEIVE_SMS	CHANGE_WIFI_STATE	CHANGE_NETWORK_STATE,Also, used in [21]
SEND_SMS	WAKE_LOCK	INTERNET
WRITE_SMS	DISABLE_KEYGUARD	READ_HISTORY_BOOKMARKS
SET_ALARM	ACCESS_NETWORK_STATE	CHANGE_CONFIGURATION
RECEIVE_WAP_PUSH	WRITE_SETTINGS	PROCESS_OUTGOING_CALLS
READ_PHONE_STATE	READ_CONTACTS	GET_PACKAGE_SIZE
READ_EXTERNAL_STORAGE Also, used in [21]	RECEIVE_MMS	PERSISTANT_ACTIVITY
RESTART_PACKAGES	WRITE_EXTERNAL_STORAGE	ACCESS_WIFI_STATE
SYSTEM_ALERT_WINDOW Also, used in [21]	EXPAND_STATUS_BAR	READ_CALL_LOG
CAMERA	CALL_PHONE	GET_ACCOUNTS Also, used in [21]
WRITE_HISTORY_BOOKMARKS	SET_WALLPAPER_HINTS	GET_TASKS
WRITE_CALL_LOG	ADD_SYSTEM_SERVICE	ACCESS_FINE_LOCATION
ACCESS_MOCK_LOCATION		

PREVILIGED, RECEIVE_WAP_PUSH, BAIDU LOCATION_SERVICE, CLEAR_APP_CACHE, ACCESS FIND_LOCATION, DEVICE_POWER, INTERNAL SYSTEM_WINDOW, and PROCESS_OUTGOING_CALL are not selected in [31] and [42].

Finally, we obtain a list containing 58 suspicious permissions from the above-mentioned references. They are listed below:

All permissions from [42] plus DELETE_PACKAGES, PROCESS_OUTGOING_CALL, WRITE_APN_SETTINGS, MOUNT_UNMOUNT_FILESYSTEMS, READ_LOGS, VIBRATE, INSTALL_SHORTCUT, CALL PREVILIGED, RECORD_AUDIO, RECEIVE_WAP_PUSH, BAIDU LOCATION_SERVICE, ACCESS_LOCATION_EXTRA, INTERNAL SYSTEM_WINDOW, DEVICE_POWER, INSTALL_PACKAGES, ACCESS FIND_LOCATION and CLEAR_APP_CACHE.

The permission is the mostly used feature in the Android malware detection process. Another feature, API call, is also used in Android mobile malware detection. According to [20], [30], and [31] the API calls that are mostly used by malware are listed in Table 6.

From the above details, we can summarize that mostly used feature selection methods are Information Gain, Mutual Information, T-test, correlation-based feature selection, selected features from previous research, and Consistency Subset Evaluator. The research paper [21] namely uses gain ratio, RELIEF attribute evaluator [43], correlation-based [44], and consistency subset evaluator [45] as feature selection methods. Among these feature selection methods, correlation-based feature selection showed the better performance.

TABLE 6. Suspicious API calls.

Privacy related API	getSimSerialNumber(), getSubscriberID(),getImei(),getDeviceId(), getLine1Number(), getNetworkOperator(),New_outgoing_call
SMS Related API	sendTextMessage(), SendBroadcast(), sendDataMessage(),Smsto, content:sms,Telephony SMS-Received
Network Related API	httpclient.execute(),getOutputStream(),GetInputStream(),getNetworkInfo(), HttpURLConnection.connect(), execHttpRequest(),Pdsu,mailto, openStream(), openConnection(),setDataAndType()setRequestMethod()
Location Related API	getLastKnownLocation(), getCellLocation(), getLatitude() getLongitude(), requestLocationUpdates(), getFromLocation()
Wi-Fi Related API	getWifiState(), setWifiEnabled(), getWifiState()
File Related API	URL(),openFileOutput(), getAssets(), Browser: Bookmarks_URL
Component Related API	startService(), setComponent()
Application Related API	getInstalledApplications(), installPackage()
API function calls for execution of external or particular commands	like Runtime.exec(), andLjava/lang/Runtime; -exec(), Android_intent_action_run
API calls frequently used for obfuscation and loading of code	DexClassLoader.Loadclass(), Cipher.getInstance()

TABLE 7. Static features with detection method.

Ref.	Static Features	Accuracy	Best Detection Method	#of Malware
[1]	Permission	91.75%	RF	200
[2]		81%	C4.5, SVM	480
[5]		88.2%	HMNB	378
[9]		-	AHP	1242
[25]		98.6%	2layered J48	1536
[41]		92.79%	RF	4868
[21]		94.90%	RF	1446
[20]	Permission, API call	92.36%	RF	175
[3]	Permission, API call, Intent	97.87%	KNN	238
[19]	Permission, intent	-	Naïve Bayes	45
[4]	ICC related features	97.4%	SVM	5264
[6]	API call	99%	KNN	3987
[8]		93.04%	Signature matching	238
[24]		96.69%	SVM	3368
[26]	API call, command, Permission	98.6%	Parallel classifier	2925
[22]	Opcode sequence	96.83%	SVM	6405
[7]	Hardware component, requested permission, app components, filtered intent, restricted API, used permission, Suspicious API call, Network address	93.9%	SVM	5560

4) DETECTION METHOD

Most research uses machine learning approaches. A very few of them have tried different detection methods. For example, the Analytic Hierarchy process is used as a detection method in MAETRIOD [9]. A two-layer detection method using machine learning was applied in [25] and showed improved performance. Parallel use of detection methods is demonstrated in [26] and exhibits excellent performance. However, most research to date involves single classifiers as shown in Table 7.

From this comparison table, we note that the best accuracy is 99% and second best is 98.6%. The detection method using K-nearest Neighbor (KNN) achieves best performance in accuracy. Random Forest (RF) and Support Vector Machine (SVM) are mostly used as detection methods and also exhibit high accuracy. The two layered and parallel classifiers achieved second best accuracy. Permission is the most commonly used feature among all static features. Permission along with other features (combinations of features) show improvement in performance at the cost of higher complexity.

KNN has the best overall performance. However, KNN's speed decreases with large data sets. Random Forest is the mostly widely used detection method. A random forest contains a set of tree-structured classifiers and each tree classifies an object. The result depends on the majority vote count [47]. The computational complexity of a random forest of size T is O (T.D) where D is the maximum depth of each tree and the space complexity is $O(2^D)$ [46].

The time complexity of the SVM is $O(m^3)$ and the space complexity is $O(m^2)$ for m elements [48]. The time complexity of the j48 decision tree is $O(nm^2)$ where n is the number of training examples and m is the number of features.

The two-layered classifier [25] has the second-best accuracy. However, this two-layered classifier involves decision trees in two layers and requires extensive computation compared to single layer classifiers. The parallel classifier [26] also shows better performance. This parallel classifier involves four types of classifiers, which also require high computational complexity.

The most common feature of Android Mobile Malware Detection is the permission, but research using other features can be found in the literature. The research in [6] achieves best performance using API call as a feature. The second-best performance in [25] uses the requested permission and the used permissions. The equal performance in [26] uses the permission, command, and API call as features. Research in [25] and [26] uses two layered and parallel classifiers as detection methods. The research in [7] uses more features, but the best accuracy is only 93.9%. Using more features would intuitively lead to higher accuracy. However, the detection method used, it exhibits lower accuracy. Moreover we have to consider that using more features will result in higher computation, as the mobile platform has limited resources.

B. DYNAMIC ANALYSIS ON ANDROID MOBILE MALWARE DETECTION

In this section, we discuss the dynamic analysis of mobile malware detection. Dynamic analysis requires execution of the malware. Dynamic analysis has been introduced to overcome the limitations of static analysis.

1) DATA SET

The data sets that are used in the Dynamic Analysis involve a small range of malware and benign ware. As the dynamic analysis involves execution of the apps, it is difficult to use a large number of apps. However, a larger data set can more accurately reflect the accuracy of the detection engine so it is desirable to use a large data set in the detection process. If we compare the number of malware and benign applications used the previous research, Table 8 is the result.

TABLE 8. Comparison based on the number of applications used in dynamic analysis.

Ref.	#Malware	#Benign apps
[10]	230 applications From Android market	
[11]	30 Self-written malwares	150 Self-written App
	2 from PJAPPs	6
	5 from HongTouTou	15
[12]	1050 malware	1160 benign apps
[23]	350 apps from Amazon Android market	
[32]	1260	-
[33]	3524	11268
[34]	1260	-

From this table, the number of malware used in dynamic analysis is comparatively lower than in static analysis. The research in [33] uses the highest number of malwares, which are 3,524. Dynamic analysis involves high computational cost. The number of malware is relatively low for this reason. However, it is necessary to use a large data set to evaluate the detection system more precisely.

2) FEATURE EXTRACTION

Feature extraction in dynamic analysis requires execution of the mobile application. The malware or the application is executed in a virtual machine or emulator, which generates a log file. Then, from the log file the features are extracted using a log analyzer or parser. The system call is extracted by using the Strace tool and features are selected by using different selection algorithms (Figure 8).

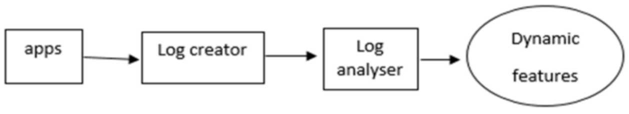

FIGURE 8. Dynamic Feature Extraction.

TABLE 9a. Suspicious system calls.

Name	Used in malicious	Used in benign
PTRACE	Most frequently used [27,32]	Used in Benign apps [32]
SIGPROCMASK	Most frequently used [27,32]	Used in Benign apps [32]
CLOCK	Most frequently used [27,28]	
CLOCK-GETTIME	Used in Malicious Apps [32]	Used in Benign apps [32]
RECV	Most frequently used [28,32]	Not used [32]
RECVFROM	Most frequently used [27,28,32]	Not used [32]
WRITE	Most frequently used [27,28,32]	Used in Benign apps [32]
WRITEV	Most frequently used [28,32]	Used in Benign apps [32]
WAIT4	Most frequently used [27]	
SEND	Most frequently used [28]	
SENDTO	Most frequently used [27,28]	
MPROTECT	Most frequently used [27,28,32]	Used in Benign apps [32]
FUTEX	Most frequently used [27,32]	Used in Benign apps [32]
IOCTL	Most frequently used [27,32]	Used in Benign apps [32]
FCNTL64	Used in Malicious Apps [32]	Used in Benign apps [32]
GETPID	Most frequently used [27,32]	Used in Benign apps [32]
GETUID32	Most frequently used [27,32]	Used in Benign apps [32]
EPOLL	Used in Malicious Apps [32]	Used in Benign apps [32]
EPOLL_CTL	Used in Malicious Apps [32]	Used in Benign apps [32]
EPOLL-WAIT	Most frequently used [28,32]	Used in Benign apps [32]
CACHEFLUSH		
READ	Most frequently used [27,28]	
READV	Most frequently used [28]	
STAT64		
GETTIMEOFDAY	Used in Malicious Apps [32]	Used in Benign apps [32]
ACCESS	Most frequently used [28,32]	Used in Benign apps [32]
PREAD		
UMASK	Most frequently used [32]	Not used [32]
CLOSE	Used in Malicious Apps [32]	Used in Benign apps [32]
OPEN	Most frequently used [28,32]	Used in Benign apps [32]
MMAP2	Used in Malicious Apps [32]	Used in Benign apps [32]
MUNMAP		
MADVISE	Used in Malicious Apps [32]	Used in Benign apps [32]
FCHOWN32	Most frequently used [32]	Not used [32]
PRCTL	Not used [32]	Only used in Benign apps [32]
BRK	Most frequently used [32]	Not used [32]
LSEEK	Used in Malicious Apps [32]	Used in Benign apps [32]
DUP	Used in Malicious Apps [32]	Used in Benign apps [32]
GETPRIORITY	Used in Malicious Apps [32]	Used in Benign apps [32]
PIPE		
CLONE	Used in Malicious Apps [32]	Used in Benign apps [32]
FSYNC	Most frequently used [32]	Not used [32]
GETDENTS64	Used in Malicious Apps [32]	Used in Benign apps [32]
GETTID	Used in Malicious Apps [32]	Used in Benign apps [32]
LSTAT64	Used in Malicious Apps [32]	Used in Benign apps [32]
FORK		
NANOSLEEP	Not used [32]	Only used in Benign apps [32]
RECVMSG		
CHMOD	Used in Malicious Apps [32]	Used in Benign apps [32]
SENDMSG	Most widely used [28]	
FLOCK	Not used [32]	Only used in Benign apps[32]
MKDIR	Most frequently used [32]	Not used [32]
CONNECT	Most frequently used [32]	Not used [32]
POLL	Not used [32]	Only used in Benign apps [32]
RENAME	Most widely used [28]	Not used [32]
SETPRIORITY		
SETSOCKOPT	Most frequently used [32]	Not used [32]
SOCKET	Most frequently used [32]	Not used [32]
UNLINK		
GETSOCKOPT		

TABLE 9b. *(Continued).*

BIND	Most frequently used [32]	Not used [32]
FTRUNCATE	Used in Malicious Apps [32]	Used in Benign apps [32]
GETSOCKNAME		
INOTIFY		
RESTART		
SCHED	Used in Malicious Apps [32]	Used in Benign apps [32]
GETRLIMIT		
LGETXATTR		
READLINK		
SOCKETPAIR		
STATFS64	Used [32]	Not used [32]
FDATASYNC	Used [32]	Not used [32]
GETPPID		
KILL		
PWRITE	Used [32]	Not used [32]
MSGGET	Used [32]	Not used [32]
RMDIR	Used [32]	Not used [32]
SELECT	Used [32]	Not used [32]
SEMGET	Used [32]	Not used [32]
SEMOP	Used [32]	Not used [32]
CHDIR	Not used [32]	Only used in Benign apps [32]
GETCWD	Not used [32]	Only used in Benign apps [32]
RT_SIGRETURN	Used [32]	Only used in Benign apps [32]
SIGACTION	Not used [32]	Only used in Benign apps [32]
SYS_281	Not used [32]	Only used in Benign apps [32]
SYS_283	Not used [32]	Only used in Benign apps [32]
SYS_224	Used [32]	Not used [32]
SYS_248	Used [32]	Not used [32]
SYS_290	Used in Malicious Apps [32]	Used in Benign apps [32]
SYS_292	Used in Malicious Apps [32]	Used in Benign apps [32]
SYSCALL_983042	Used in Malicious Apps [32]	Used in Benign apps [32]
FSTAT64	Used in Malicious Apps [32]	Used in Benign apps [32]

3) FEATURE SELECTION

The system calls, used in malicious apps according to [27], [28], and [32] are listed in Table 9.

Dimjasevic *et al.* [29] also chose the system call as a desired feature. Examples of selection algorithms used in dynamic analysis are information gain, similarity function, and using suspicious activity lists from previous research. Other researchers have used other dynamic features in their research work. Ham and Choi [33] have used the following dynamic features:

- Memory related features: size, shared, allocated, pss, vss, free for Native, and Dalvik
- Virtual memory related features: VmPeak, VmHwm, VmData, VmSize, and VmLib
- Power: Level and temperature
- Network: TxBytes, RxBytes, RxPacket, and TxPacket
- CPU: CPU Usage

4) DETECTION METHOD

Mainly, machine learning approaches are used as detection methods. Both supervised and unsupervised methods have been used. The detection methods that are commonly used in dynamic analysis are K-means Clustering, Random Forest, Naïve Bayes, Logistic regression (LR), SVM, KNN_E, KNN_M, and Bayesian network (BN). Weka tool is used for the detection methods. Table 10 illustrates the dynamic features, accuracy level, and detection methods used in dynamic analysis:

TABLE 10. Dynamic features and detection methods.

Ref.	Dynamic Features	Accuracy	Detection	#of Malware
[10]	System call	-	Signature matching	230
[11]		100% for self-written apps 85% for others	K-means	37
[32]		-	frequency	1260
[34]		-	Pattern matching	1260
[12]	API call	97.66%	KNN_M	1050
[23]		-	-	350
[33]	Native_size, Native_shared, Other_shared, VMpeak, VMlib, Dalvik_RSS, Rxbytes, VmData, Send_SMS and CPU_Usage.	TPR 99.9%	RF SVM	3524

C. HYBRID ANALYSIS FOR ANDROID MOBILE MALWARE DETECTION

In this section, we discuss hybrid analysis for mobile malware detection. Hybrid analysis involves both static and dynamic features. Researchers have introduced hybrid analysis to overcome the limitations of both static and dynamic analysis. Hybrid analysis helps to improve the accuracy level in the Android mobile malware detection process.

1) DATA SET

Hybrid analysis uses both static and dynamic features. A proper data set is essential for hybrid analysis. Malware are usually collected from Gnome Project, Contagio Mobile, Virus Share, AppChina, etc. Benign apps are collected from Google Play, Wandoujia App market, AppChina, etc. Table 11 provides a brief list of data sets used in hybrid analysis.

TABLE 11. Number of applications used in hybrid analysis.

Ref.	#Malware	#Benign apps
[13]	500	500
[14]	15741	135823
[15]	110	893
[16]	1260	-
[17]	-	-
[18]	2784	-

From this table, the number of malware apps varies from 110 to 15,741. The research in [14] used a large number of malware and benign apps.

2) FEATURE EXTRACTION

Hybrid analysis uses static as well as dynamic features. The static features are extracted from the APK file. Dynamic features are extracted from the Androidmanifest.xml and classes.dex. By executing the apps in the virtual machine or emulator, dynamic features are extracted. Figure 9 depicts the feature extraction process for hybrid analysis.

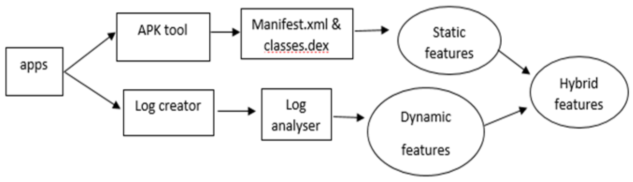

FIGURE 9. Hybrid feature extraction.

Hybrid analysis requires both static and dynamic feature extraction tools. The APK tool, dex2jar, Soot, WALA, etc. are needed to perform the static feature extraction. The Starce and Rnadoop tools are used to extract dynamic features. Dynamic analysis involves the use of a virtual machine or emulator. Feature extraction in hybrid analysis requires high computation, which is a concern for the mobile platform.

3) FEATURE SELECTION

Feature selection is extremely important in hybrid analysis. Because of the many features available, feature selection must be precise and accurate. The time and space complexity of the selection method should also be considered. The appropriate feature set is selected using different selection methods such as information gain, mutual information, Fisher score, etc. The feature selection methods used are the same as for static or dynamic analysis. However, the importance of proper selection in hybrid analysis is much higher than in static or dynamic analysis. Unnecessary and irrelevant features can hamper the overall performance of the detection engine. Information gain and Fisher score are used as the selection methods in the research papers that are discussed here.

The static features that are involved in hybrid analysis are Permission, API call, Java Package name, Intent receivers, Class structure, Crypto operations (cryptographic API) Services, Receivers, and Publisher ID. The most commonly used static feature is the permission in hybrid analysis.

The more common dynamic features involved in hybrid analysis are System call, File operations, Network activity, dynamically loaded code, and dynamically registered broadcast receivers, Phone activity, and Crypto operations (cryptographic API). The most commonly used dynamic feature is the system call in hybrid analysis.

4) DETECTION METHOD

The detection method must handle many features. The main objective here is to achieve more accurate performance than static or dynamic analysis. For this reason, it is necessary to design and implement a detection method that will improve the accuracy level. Different detection methods are used in hybrid analysis such as J48 decision tree, Linear L1, Linear L2, SVM, KNN, LDC, QDC, MLP, PARZC, Bayes, and others. Weka tool is used here. Table 12 indicates the static and dynamic features used in hybrid analysis, accuracy level, the used detection method, and the number of malware used in hybrid analysis.

TABLE 12. Feature set with detection method in hybrid analysis.

Ref.	Static features	Dynamic Features	Accuracy	Detection	#of Malware
[13]	Permission, API call	System call	93.33-99.28%. SVM shows better result for static features. (99.28%) Naïve Bayes shows better accuracy for dynamic. (90%)	KNN, SVM, Bayes	500
[15]	Permission	API	-	J48	110
[14]	Java package name, intents, Requested permission, ID	File op., Network, SMS, runtime code, runtime broadcast receivers.	TPR 98.24% FPR 0.04%	L1, L2, SVM	15741
[16]	Permission, services, receivers, package name	System call, Network	-	Reports from six plug ins are evaluated	1260
[17]	permission	System call, internet traffic	-	Used cloud computing.	-
[18]	permission	Critical API, User activity, SMS, system call	96.9% KNN shows best result	KNN, LDC, QDC, MLP, PARZC	2784

From this table, the system call is the most commonly used dynamic feature and the permission is the most commonly used static feature. Other static and dynamic features are also used in many research works to improve accuracy. Among these research works, the SVM shows the best performance in the scale of TPR, which is 98.24% [14] for the static analysis and provides low FPR. However, the overall accuracy of hybrid analysis was not discussed in this reference. Another research paper showed 96.9% accuracy by using KNN as the detection method [18].

VIII. COMPARATIVE ANALYSIS OUTCOMES

A comparative analysis has been carried out based on static, dynamic, and hybrid analysis for the Android mobile malware detection process. This comparative analysis has provided a broad sense of the Android Mobile Malware Detection process. We have analyzed the challenges and the security loop holes of Android applications, which attract the malware developers. We have also analyzed the features, feature extraction techniques, feature selection techniques, and the detection methods. We evaluate them considering the number of used malware, feature extraction process, selected features, detection method, and accuracy. Most research works choose static features and static analysis, as static analysis is easier to carry out on a mobile platform. However, static analysis suffers from specific limitations such as code obfuscation, downloading malicious contents from remote servers, and others. Dynamic analysis can handle some of these limitations, but not all of them. Moreover, dynamic analysis for Android platforms is not feasible if we consider the resources necessary to carry out he analysis. To overcome this limitation, Cloud computing and use of remote servers have been introduced [17]. As a third option, hybrid analysis involves both static and dynamic analysis. Hybrid analysis provides the freedom to choose both static and dynamic features to improve accuracy in the detection process. However, Hybrid solutions also involve operational complexity and limited resources inherent in mobile devices. Compared to static and dynamic analysis, hybrid analysis is the best option for research work because new techniques can be introduced by the malware developer. Analyzing only static or dynamic features will not be sufficient to provide full detection capabilities; we need both. Hybrid analysis is the method leading to higher accuracy and lower false positive rates.

In this literature review, we have analyzed a large number of research works that involve recent static, dynamic, and hybrid analysis for Android mobile malware detection. While analyzing these, we have noted the following literature gaps.

A. FAILURE TO DETECT UNKNOWN/NEW VARIANT OF MALWARE, NEED TO ENSURE ZERO-DAY DETECTION

Today, malware developers are using newer and innovative techniques to change the internal architecture of malware and to change its procedures to avoid detection. They have used code obfuscation techniques, polymorphism, and metamorphism to change the behavior of the malware to act more like benign ware by hiding the malicious activities. Normal conventional malware detection techniques are not capable of detecting these new forms of malware. Most of the research carried out to detect between malware or benign ware depend on app behaviors and features. They have not focused on the detection of new malware. In the literature review, static analysis [1]–[9], [19]–[22], [24]–[26], [41] did not focus on the detection of new malware. Moreover, dynamic analysis [10]–[12], [23], [32]–[34] and hybrid analysis [13]–[17] provided little or no discussion on the detection of new malware. In addition, malware changes behavior or feature sets very frequently. If behavioral or signature based detection techniques attempt to detect new variants of malware, the attempt will likely result in failure. Analyses based on only the behaviors of past malware will not be able to detect future malware. If our detection process is not able to detect new variants of malware, then the apps market will face both financial and existential problems. Ability in detecting a new variant is an essential role for a detection engine. However, this ability is absent in the above-mentioned literature review. We must design detection engines that can detect the unknown and new malwares introduced in the Apps Market to ensure Zero-day malware detection.

A few research works discuss Zero-day detection in order to detect unknown and new malware. Brief descriptions of these along with their limitations are discussed here.

A noble and effective approach has been proposed using data mining and several detection algorithms to ensure Zero-Day Malware Detection for the Windows system [58]. Supervised learning methods are used to detect unknown malware. The method extracts API calls as the features. A large number of data sets are evaluated using classifiers such as Naïve Bayes, KNN, SMO- normalized poly kernels, SMO-Poly kernels, SMO-Puk, SMO-Radial Basis Function (RBF), Backpropagation neural network algorithms, and the J48 decision tree. A total of 51,223 recent malware datasets are used and a signature set for malware and for benign ware has been generated. The authors achieved 98.5% TPR and 0.025 FPR. However, in the results they did not mention Zero-day detection. How many unknown malwares have been detected is not discussed. Moreover, malware developers use code obfuscation techniques to hide the malicious behavior of a malware. A signature matching approach will not be able to detect a new malware under these conditions. Thus, Zero-Day Malware detection will not be achieved by signature detection.

A heuristic-based filtering approach is introduced in a permission-based behavioral footprint scheme to ensure Zero-Day Malware detection for Android [57]. They have named the system DroidRanger. The study collected a very large number of apps from Android's official market place and also from four other third party app markets named eoeMarket [59], alcatleclub [60], gfan [61], and mmoovv [62]. In this heuristic-based filtering approach, only two heuristics are applied; however, two heuristics are not sufficient. More heuristics are needed in order to detect new unknown malwares.

For Zero-day detection of unknown malware, an approach has been introduced which combines static analysis and machine learning techniques [52]. They have used ensemble learning to obtain better performance using Simple Logistic, Decision tree, Random tree, and Random Forest as the detection methods. A total of 179 features are used to perform the classification. The experiment was run on 2,925 malware and 3,938 benign ware samples. To improve the accuracy and to detect unknown malware they used a large number of features and four detection methods. However, they did not mention how to incorporate the features of unknown malware, and presented no proof that unknown malware could be detected. Large numbers of features and four detection methods can improve accuracy but cannot guarantee Zero-day detection.

Risk Ranker is a malware engine that introduced a proactive approach to detect unknown malware to ensure Zero-day detection [50]. They proposed a proactive technique, as reactive techniques will not be able to detect unknown malware. Rather than using a signature based detection technique, Risk ranker uses a ranking approach to determine high, medium, and low risk apps. Based on dangerous behavior, potential risks were analyzed. A total of 118,318 apps were analyzed, and among them a total of 281 were identified as risky apps. They found 718 malwares and 322 qualified as Zero-day detection. However, this risk ranker also has limitations such as using the javax.crypto libraries to detect encryption. Malware developers can adopt new techniques that avoid encryption detection. Moreover, dynamic loading of the malicious code can be accomplished with the help of the internet, making it easy for malware developers to hide their malicious activities and avoid detection. The heuristics used in the Risk ranker to detect malware can be avoided by malware developers, as there are thousands of avoidance techniques.

MADAM (Multi-Level Anomaly Detector for Android Malware) is a proposed method to ensure Zero-day detection [18]. MADAM uses multiple levels of features. The levels are kernel level, application level, user level, and package level. In MADAM, there are three main blocks named App Risk Assessment, Global Monitor, and per-app Monitor. MADAM has achieved 96.9% accuracy. As MADAM has used multiple levels of features and three different blocks to detect malicious behavior of malware, it has achieved high accuracy and low overhead. However, behavioral or signature based detection suffer from a major disadvantage, which is that they will not be able to detect unknown and new behaviors of malware. MADAM used signature based detection and it may not be able to detect new behaviors of unknown malware.

B. SELECTION OF PROPER FEATURE SET TO INCORPORATE UNKNOWN BEHAVIORS OF NEW MALWARE

Feature selection is an essential task for any detection engine. Proper feature selection will eliminate unnecessary and irrelevant features. This will be immensely helpful in the detection process. Moreover, selection of features should be carried out in such a way that the selected features will incorporate both known and unknown malware and help to detect unknown and new malware to ensure Zero-day detection. Most research has focused on feature selection. It is essential to develop a feature selection method that will select the correct features to ensure Zero-day detection.

Thus, the research question is: how to select the appropriate feature set that will incorporate known and unknown behaviors of malware to ensure Zero-day detection. Moreover, must develop a feature selection approach that will reduce feature extraction during runtime detection.

Yerima *et al.* [52] extracted features by using the APK tool. They have considered the permission, API call, and commands as appropriate features, but did not mention their reasons for selecting only these three categories of features. They extracted 179 features and used all of them as their ensemble learning and demonstrated improvement in performance with various ranges of features. Among these 179 features, 130 are permission features. Others are critical API calls and commands. They used SMA related API, Telephony related API, packaged related API, and others. They have selected these API types as these were also used in other research work. They selected the top 20 and top 50 features using Mutual Information ranking. However, how these selected features would help to detect unknown malware is not explained.

MADAM [18] proposed a multi-level approach. Features are extracted by using a hybrid feature extraction technique. Features were classified into different levels, for a total of four levels of features. In the package level, some features describe the app metadata such as permissions, market rating, developer's reputation, and number of downloads. They used these features to create a suspicious-apps list. The application level contains Critical API call and SMS related features. Critical API calls are used for signature-based detection. There are two other levels: user and kernel levels. User activity features, SMS related features, and system calls are used to classify the malware and benign ware. Thus, different features are used in different levels in the detection process. In MADAM, features have been selected based on different types of malware behavior. Feature selection is performed based on the previous behavior of malware. This feature selection methodology is not suitable for detection of unknown malware. How to incorporate new behaviors of unknown malware is not made clear. The feature selection mechanism should accommodate the characteristics of both known and unknown malware.

Zhou *et al.* [57] used a heuristic-based filtering approach to detect malware to ensure Zero-day detection. First, they used SMS related permissions for filtering. Receive_SMS and Send_SMS permissions are used here as the features. For behavioral footprint matching, they used an API call and permission list. They chose the API call and permission list based on the heuristics used and on the behavior of known malware. However, the feature selection mechanism can select certain types of features which may not be able to detect unknown and new malware with different types of behavior.

Past research mostly used permissions as the selected features in the Android malware detection process. Permission was selected as the feature in [1], [2], [5], [9], [25], [21], and [41]. Some authors selected the API call as the feature by using static feature extraction techniques [6], [8], [24]. Information gain was used as the feature selection method to obtain a permission list [20]. Other selection methods such as correlation coefficient, mutual information, and T-test have been used to detect suspicious system calls [42].

Multiple categories of static features are extracted and selected in some work [3], [4], [19], [20] to improve accuracy. For example, the authors used multiple categories of static features: permission, API call, and Intent in a research work that achieved 97.87% accuracy [3]. Although static features are easy to extract, they suffer from mimicry attacks and their inability to detect code obfuscation techniques.

To overcome the limitations of static features, dynamic feature extraction has been introduced. Dynamic features are able to depict the behaviors of malware more accurately. In the dynamic analysis, many researchers used only one category of features, which is the system call [10], [11]. Others used only the API call as the extracted feature [12], [23]. Using a single category of features makes the feature extraction system simple and the computational cost is comparatively lower than using multiple categories of features. Other authors used multiple categories of features. For example, the permission, API call, and command related features which illustrate the use of features from multiple categories were used in [26] and they achieved a higher level of accuracy of 98.6%. They combined these multiple categories of features in order to improve the accuracy. However, these combined features have shown obvious improvement of accuracy but at the same time, the feature extraction process become costly regarding required computation.

More recently, researchers have become interested in using hybrid features, as hybrid features can illustrate the actual behavior of an application more accurately than static or dynamic features. Usually they choose a single category of static features and a single category of dynamic features to make the feature extraction simple and easy. Permission and API call are commonly extracted by using hybrid feature extraction and the authors combined the single-category features as hybrid features [15]. To achieve higher accuracy levels, most hybrid feature extraction techniques combine multiple categories of both static and dynamic features [13], [14], [16]–[18]. Permission, API call, and system call are commonly extracted and used as hybrid features, as these have shown a significant improvement in accuracy [13]. However, the hybrid feature extraction technique suffers from higher computational cost. More resources and time are needed, which increase the performance overhead.

The rapid increase of Android malware has led to the necessity of proper detection engines [60]. From the previous discussions, it is apparent that without a proper feature selection mechanism, the detection engine will not be able to detect malware accurately. To detect the unknown and new malware it is necessary to propose and implement a proper feature selection mechanism that will select the appropriate features. These appropriate feature sets will lead to accurate and fast detection of malware to ensure Zero-day detection.

C. FAILURE TO PREDICT THE BEHAVIOR OF MALICIOUS APPS FROM THEIR PAST ACTIONS

Malware developers are using different techniques to hide malicious activities and mimic benign ware behavior. From the past behavior of the malware, it is very difficult to predict the future behaviors of malware. Behavior based and signature based detection methods fail to detect the newer malicious behaviors of malware. The crucial need is to predict future behavior from past actions of malicious applications to ensure Zero-day detection. This research work points out necessary research directions to increase the probability of detecting new and unknown malware and to ensure Zero-day detection.

From the literature review, we have not located any approach to predict the behavior of malware from past actions. Only some researchers used data bases to store app behaviors [50]. However, they did not mention any approach to predict future behaviors from past actions.

IX. CONCLUSION

We have analyzed the static, dynamic, and hybrid analysis methods for mobile malware detection. The analysis includes recent literature on Zero-day detection. The detection process, feature extraction and selection process, and detection algorithms are discussed in this study. We have found that machine learning approaches are commonly used to classify malware and benign ware. The suspicious permission list, API call list, and the system call list are also identified to assist application developers. In future, we plan to design and implement a framework that will be able to detect mobile malware with a high accuracy to ensure Zero-day detection.

REFERENCES

[1] Z. Aung and W. Zaw, "Permission-based Android malware detection," *Int. J. Sci. Technol. Res.*, vol. 2, no. 3, pp. 228–234, 2013

[2] C.-Y. Huang, Y.-T. Tsai, and C.-H. Hsu, "Performance evaluation on permission-based detection for Android malware," in *Advances in Intelligent Systems and Applications* (Smart Innovation, Systems and Technologies), vol. 2. Berlin, Germany: Springer, 2013, pp. 111–120.

[3] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "DroidMat: Android malware detection through manifest and API calls tracing," in *Proc. 7th Asia Joint Conf. Inf. Secur.*, 2012, pp. 62–69.

[4] K. Xu, Y. Li, and R. H. Deng, "ICCDetector: ICC-based malware detection on Android," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 6, pp. 1252–1264, Jun. 2016.

[5] H. Peng *et al.*, "Using probabilistic generative models for ranking risks of Android apps," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 241–252.

[6] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in Android," in *Security and Privacy in Communication Networks*. Cham, Switzerland: Springer, 2013, pp. 86–103.

[7] D. Arp *et al.*, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. NDSS*, 2014, pp. 23–26.

[8] A. Desnos and G. Gueguen, "Android: From reversing to decompilation," in *Proc. Black Hat Abu Dhabi*, 2011, pp. 77–101.

[9] F. D. Bene, F. Martinelli, I. Matteucci, and A. Saracin, "Risk analysis of Android applications: A multi-criteria and usable approach," Nat. Res. Council, Inst. Inform. Telematics, Pisa, Italy, Tech. Rep. TR-04-2015, 2015. [Online]. Available: http://www.iit.cnr.it/node/32795

[10] T. Isohara, K. Takemori, and A. Kubota, "Kernel based behavior analysis for the Android malware detection," in *Proc. 7th Int. Conf. Comput. Intell. Secur.*, 2011.

[11] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior based malware detection system for Android," in *Proc. 1st ACM Workshop Secur. Privacy Smartphones Mobile Devices (SPSM)*, New York, NY, USA, 2011, pp. 15–26.

[12] S. Wu, P. Wang, X. Li, and Y. Zhang, "Effective detection of Android malware based on the usage of data flow APIs and machine learning," *Inf. Softw. Technol.*, vol. 75, pp. 17–25, Jul. 2016.

[13] Y. Liu, Y. Zhang, H. Li, and X. Chen, "A hybrid malware detecting scheme for mobile Android applications," in *Proc. IEEE Int. Conf. Consumer Electron. (ICCE)*, Jan. 2016, pp. 155–156.

[14] 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. (COMPSAC)*, Jul. 2015, pp. 422–433.

[15] D.-U. Kim, J. Kim, and S. Kim, "A malicious application detection framework using automatic feature extraction tool on Android market," in *Proc. 3rd Int. Conf. Comput. Sci. Inf. Technol. (ICCSIT)*, 2013, pp. 1–4.

[16] T. Eder, M. Rodler, D. Vymazal, and M. Zeilinger, "ANANAS—A framework for analyzing Android applications," in *1st Int. Workshop Emerg. Cyber Threats Countermeasures (ECTCM)*, 2013, pp. 711–719.

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

[18] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and efficient behavior-based Android malware detection and prevention," *IEEE Trans. Depend. Sec. Comput.*, vol. 15, no. 1, pp. 83–97, Jan./Feb. 2018.

[19] F. Idrees and M. Rajarajan, "Investigating the Android intents and permissions for malware detection," in *Proc. IEEE 10th Int. Conf. Wireless Mobile Comput. Netw. Commun.*, Oct. 2014, pp. 354–358.

[20] P. P. K. Chan and W.-K. Song, "Static detection of Android malware by using permissions and API calls," in *Proc. Int. Conf. Mach. Learn.*, Jul. 2014, pp. 82–87.

[21] U. Pehlivan, N. Baltaci, C. Acartürk, and N. Baykal, "The analysis of feature selection methods and classification algorithms in permission based Android malware detection," in *Proc. IEEE Symp. Comput. Intell. Cyber Secur.*, Dec. 2014, pp. 1–8.

[22] Q. Jerome, K. Allix, R. State, and T. Engel, "Using opcode-sequences to detect malicious Android applications," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2014, pp. 914–919.

[23] L. Min and Q. Cao, "Runtime-based behavior dynamic analysis system for Android malware detection," in *Proc. 2nd Int. Conf. Comput. Inf. Appl.*, 2012, pp. 2220–2225.

[24] H.-Y. Chuang and S.-De Wang, "Machine learning based hybrid behavior models for Android malware analysis," in *Proc. IEEE Int. Conf. Softw. Quality, Rel. Secur.*, Aug. 2015, pp. 201–206.

[25] X. Liu and J. Liu, "A two-layered permission-based Android malware detection scheme," in *Proc. 2nd IEEE Int. Conf. Mobile Cloud Services Eng.*, Apr. 2014, pp. 142–148.

[26] S. Y. Yerima, S. Sezer, and I. Muttik, "Android malware detection using parallel machine learning classifierS," in *Proc. 8th Int. Conf. Next Generat. Mobile Appl., Services Technol.*, 2014, pp. 37–42.

[27] S. Malik and K. Khatter, "System call analysis of Android malware families," in *Indian J. Sci. Technol.*, vol. 9, p. 21, Jun. 2016, doi: 10.17485/ijst/2016/v9i21/90273

[28] F. Tchakounte and P. Dayang, "System calls analysis of malwares on Android," *Int. J. Sci. Technol.*, vol. 2, no. 9, pp. 669–674, Sep. 2013.

[29] M. Dimjašević, S. Atzeni, Z. Rakamaric, and I. Ugrina, "Evaluation of Android malware detection based on system calls," in *Proc. IWSPA*, Mar. 2016, pp. 1–8, doi: 10.1145/2875475.2875487

[30] O. Somarriba, U. Zurutuza, R. Uribeetxeberria, L. Delosières, and S. Nadjm-Tehrani, "Detection and visualization of Android malware behavior," *J. Elect. Comput. Eng.*, vol. 2016, Feb. 2016, Art. no. 8034967. [Online]. Available: http://dx.doi.org/10.1155/2016/8034967

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

[32] Y. J. Ham and H.-W. Lee, "Detection of malicious Android mobile applications based on aggregated system call events," *Int. J. Comput. Commun. Eng.*, vol. 3, no. 2, pp. 149–153, Mar. 2014.

[33] H.-S. Ham and M.-J. Choi, "Analysis of Android malware detection performance using machine learning classifiers," in *Proc. Int. Conf. ICT Converg. (ICTC)*, 2013, pp. 490–495.

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

[35] [Online]. Available: https://www.statista.com/statistics/232786/forecast-of-andrioid-users-in-the-us/

[36] Statistica.com. (2018). *Global Mobile OS Market Share in Sales to End Users From 1st Quarter 2009 to 2nd Quarter 2017*. Accessed: Mar. 20, 2018. [Online]. Available: http://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/

[37] [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf

[38] Idc.com. (2018). *Smartphone OS*. Accessed: Mar. 20, 2018. [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[39] W. Enck, M. Ongtang, and P. McDaniel, "Understanding Android security," *IEEE Security Privecy*, vol. 7, no. 1, pp. 50–57, Jan./Feb. 2009.

[40] C. Miller, "Mobile attacks and defense," *IEEE Security Privacy*, vol. 9, no. 4, pp. 68–70, Jul./Aug. 2011.

[41] Developer.android.com. (2018). *Manifest.Permission*. Accessed: Mar. 20, 2018. [Online]. Available: https://developer.android.com/reference/android/Manifest.permission.html

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

[43] I. kononenko, "Estimating attributes: Analysis and extensions of RELIEF," in *Machine Learning—ECML* (Lecture Notes in Computer Science), vol. 784. Berlin, Germany: Springer, 2005, pp. 171–182.

[44] A. M. Hall, "Correlation-based feature selection for machine learning," Ph.D. dissertation, Univ. Waikato, Hamilton, New Zealand, Apr. 1999.

[45] H. Liu and R. Setiono, "A probabilistic approach to feature selection—A filter solution," in *Proc. ICML*, vol. 96. 1996, pp. 319–327.

[46] X. Solé, A. Ramisa, and C. Torras, "Evaluation of random forests on large-scale classification problems using a bag-of-visual-words representation," in *Proc. CCIA*, 2014, pp. 273–276.

[47] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 1999.

[48] V. N. Vapnik, *Statistical Learning Theory*. New York, NY, USA: Wiley, 1998.

[49] P. D. Meshram and R. C. Thool, "A survey paper on vulnerabilities in Android OS and security of Android devices," in *Proc. IEEE Global Conf. Wireless Comput. Netw. (GCWCN)*, Dec. 2014, pp. 174–178.

[50] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: Scalable and accurate zero-day Android malware detection," in *Proc. ACM MobiSys*, Jun. 2012, pp. 281–294.

[51] L. Sayfullina *et al.*, "Efficient detection of zero-day Android malware using normalized Bernoulli Naïve Bayes," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2015, pp. 198–205.

[52] S. Y. Yerima, S. Sezer, and I. Muttik, "High accuracy Android malware detection using ensemble learning," *IET Inf. Secur.*, vol. 9, no. 6, pp. 313–320, 2015.

[53] S. Gold, "Android insecurity," *Netw. Secur.*, vol. 2011, no. 10, pp. 5–7, 2011.

[54] S. Mansfield-Devine, "Paranoid Android: Just how insecure is the most popular mobile platform?" *Netw. Secur.*, vol. 2012, no. 9, pp. 5–10, 2012.

[55] S. Mansfield-Devine, "Android architecture: Attacking the weak points," in *Netw. Secur.*, vol. 2012, no. 10, pp. 5–12, 2012.

[56] S. Mansfield-Devine, "Android malware and mitigations," *Netw. Secur.*, vol. 2012, no. 11, pp. 12–20, 2012.

[57] 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 *NDSS* vol. 25, no. 4, pp. 50–52, Feb. 2012.

[58] M. Alazab, S. Venkatraman, P. Watters, and M. Alazab, "Zero-day malware detection based on supervised learning algorithms of API call signatures," in *Proc. 9th Australasian Data Mining Conf.*, vol. 121. 2011, pp. 171–182.

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

[60] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proc. 1st ACM Workshop Secur. Privacy Smartphones Mobile Devices*, 2011, pp. 3–14.

[61] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Emerg. Artif. Intell. Appl. Comput. Eng.*, vol. 160, pp. 3–24, 2007.

[62] D. Dagon, T. Martin, and T. Starner, "Mobile phones as computing devices: The viruses are coming!" *IEEE Pervasive Comput.*, vol. 3, no. 4, pp. 11–15, 2014.

[63] J. Abawajy and A. Kelarev, ''Iterative classifier fusion system for the detection of Android malware,'' *IEEE Trans. Big Data*, Mar. 2017, doi: 10.1109/TBDATA.2017.2676100.

[64] R. Schlegel, ''Soundcomber: A stealthy and context-aware sound trojan for smartphones,'' in *Proc. NDSS*, 2011, pp. 17–33. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle: Soundcomber+:+A+Stealthy+and+Context-Aware+Sound+Trojan+ for+Smartphones#0

[65] F. M. David, E. M. Chan, J. C. Carlyle, and R. H. Campbell, ''Cloaker: Hardware supported rootkit concealment,'' in *Proc. Secur. Privacy*, 2008, pp. 296–310, accessed: Feb. 2, 2014. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4531160

SHAILA SHARMEEN was a Senior Lecturer with the Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh. She is currently pursuing the Ph.D. degree with the School of Information Technology, Deakin University, Australia, with a focus on IT Security under the supervision of Prof. Dr. J. H. Abawajy. She has authored three publications in well reputed international conferences. Her main research interests include machine learning, data mining, and IT security.

SHAMSUL HUDA was a Lecturer with Federation University, Australia. He was an Assistant Professor with the Computer Science Department, Khulna University of Engineering and Technology, Bangladesh. He is currently a Lecturer with the School of Information Technology, Deakin University, Australia. He has authored over 50 journal and conference papers in well reputed journals, including the IEEE Transactions. His main research interests include computational intelligence, cyber security for industrial control system, and optimization approaches to machine learning.

JEMAL H. ABAWAJY is currently a Full Professor with the Faculty of Science, Engineering and Built Environment, Deakin University, Australia. He has authored/co-authored over 250 refereed articles and supervised numerous Ph.D. students to completion. He has also served on the editorial board of numerous international journals, including the IEEE Transactions on Cloud C omputing. He has delivered over 50 keynote and seminars worldwide and has been involved in the organization of over international conferences in various capacity including chair and general co-chair.

WALAA NAGY ISMAIL is currently a Lecturer with the Department of Information systems, Minya University, Egypt. She is currently a Research Assistant with the Information Systems Department, College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia. Her research interests include pervasive health care, anomaly detection, cloud computing, mobile health care, and data mining.

MOHAMMAD MEHEDI HASSAN (M'12) received the Ph.D. degree in computer engineering from Kyung Hee University, South Korea, in 2011. He is currently an Associate Professor with the Information Systems Department, College of Computer and Information Sciences (CCIS), King Saud University (KSU), Riyadh, Saudi Arabia. He received the Excellence in Research Award from CCIS, KSU, in 2015 and 2016, respectively. His research interests include cloud computing, mobile cloud, sensor-cloud, Internet of Things, big data, and social network.