

# We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900

Open access books available

186,000

International authors and editors

200M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index  
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?  
Contact [book.department@intechopen.com](mailto:book.department@intechopen.com)

Numbers displayed above are based on latest data collected.  
For more information visit [www.intechopen.com](http://www.intechopen.com)



# Android Application Security Scanning Process

*Iman Almomani and Mamdouh Alenezi*

## Abstract

This chapter presents the security scanning process for Android applications. The aim is to guide researchers and developers to the core phases/steps required to analyze Android applications, check their trustworthiness, and protect Android users and their devices from being victims to different malware attacks. The scanning process is comprehensive, explaining the main phases and how they are conducted including (a) the download of the apps themselves; (b) Android application package (APK) reverse engineering; (c) app feature extraction, considering both static and dynamic analysis; (d) dataset creation and/or utilization; and (e) data analysis and data mining that result in producing detection systems, classification systems, and ranking systems. Furthermore, this chapter highlights the app features, evaluation metrics, mechanisms and tools, and datasets that are frequently used during the app's security scanning process.

**Keywords:** Android, application, scanning, security, malware

## 1. Introduction

This section introduces the Android operation system and its applications. Moreover, it defines Android malware and shares its recent statistics. Android permissions and security model are also presented. This section ends with discussing the security scanning framework for Android applications.

### 1.1 Android and application definition

Android is one of the most popular operating systems that provide open-source development environment based on Linux. It allows the development for mobile, tablets, smartwatches, and smart TVs. Android was established by Open Handset Alliance that started working in 2003, while Google released its first Software Development Kit (SDK) in 2007, but the first commercial version was released in September 2008 called as Android 1.0 [1] with the first device executed being HTC Dream. The sale of the Android phone was increased from 75% in 2013 [2] to 88% in 2018 [3]. **Table 1** lists the sales of smartphones from 2011 to 2018 which show a clear capture of the market over the years. This market penetration reveals the successful implementation of features as well as cheap price.

The Android system is composed of five important layers:

- **Applications** refer to the software stack of native as well as user-based applications.

- **Android runtime** allows the application to run on mobile devices by converting the Android code into DEX format or byte code. The conversion of DEX code into device-related code is done before compilation, and this kind of technique is referred to as ahead of time (AOT).
- **Application framework** manages and runs the applications using the services such as activity manager, content providers, telephony manager, package manager, location manager, etc.
- **Android libraries** are a set of Java-based development application programming interfaces (APIs) that can help in performing general purpose tasks, as well as location-based and string handling.
- **Android kernel** is based on the Linux 2.6 kernel and is used to provide abstraction between device hardware and other software layers [4, 5].

The efforts for making each of the component secure have been made. However, still there are issues due to open-source development, and every vendor and company following their own standards has led to serious security issues [6].

The Android application contains four types of components shown in **Figure 1** [7]:

- **Activities:** each activity represents a single screen with a user interface.
- **Services:** a service operates in the background to execute long-running operations. Services could be initiated by other components like activity or broadcast receiver.
- **Content providers:** to share data between different applications.
- **Broadcast receivers:** to listen for specific system-wide broadcast announcements and react to them.

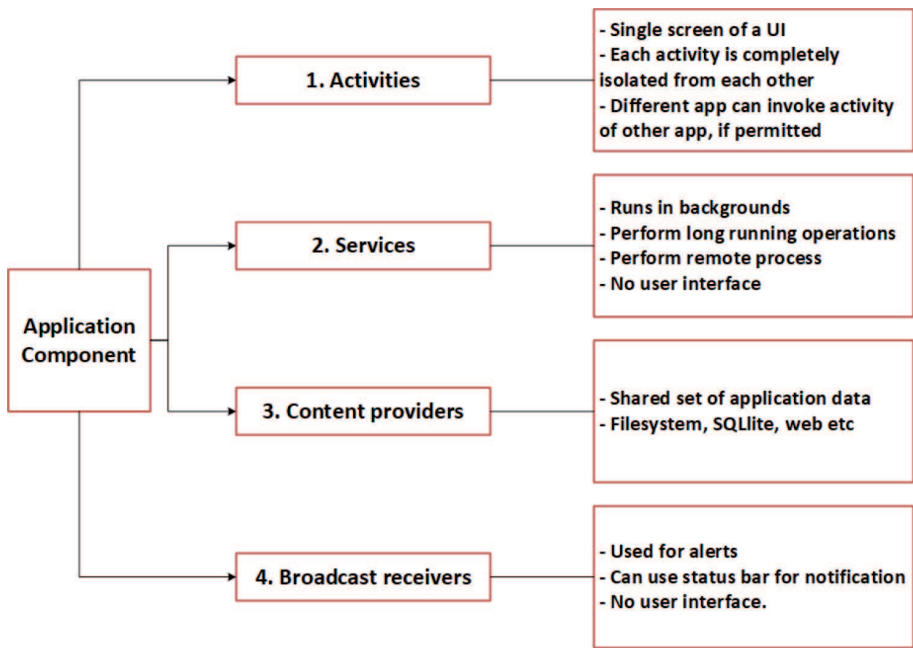
Android applications are written in Java programming language and distributed as .apk files. Android application package (APK) file is a ZIP compressed file that includes the following files:

- **AndroidManifest.xml file:** it describes the application's capabilities and informs the OS about the other components of the application. It identifies the needed hardware and software features such as the camera, in addition to, the minimum API level required by the application. The permissions requested by the app and the permissions required to access the application's interfaces/data are defined in its manifest file.
- **Dalvik executable or classes.dex file:** the Java classes and methods defined in the application code are grouped into one single file (classes.dex).
- **.xml files:** which are used to define the user interface of the application.
- **Resources:** the external resources that are associated with the application (e.g., images).

Android applications run in a virtual environment to improve security. However, they can be downloaded from any source whether trusted or not. After an application is initiated, it grants its own virtual environment, so the code will be isolated

Year	Android share	iOS shares	Other OS shares
2011	46.66	18.87	34.45
2012	66.34	19.11	14.53
2013	78.50	15.54	5.94
2014	80.70	15.37	3.91
2015	81.60	15.88	2.50
2016	84.79	14.44	0.75
2017	85.91	13.98	0.09
2018	88	11.75	0.03

**Table 1.**  
The detail of the Android phone compared to iOS and other smartphone sales shares from 2011 to 2018 retrieved from Statista.com [3].



**Figure 1.**  
The Android application components.

from other apps. Although the applications are isolated, still they can interact with the system and other applications through APIs. Meanwhile, Android assigns Linux user ID for each application.

API stands for application programming interface that refers to the set of tools providing interfaces for communication between different software components. APIs are used to access data and key features within Android devices. API framework consists of a set of API packages that include specific classes and methods. Additionally, it contains a set of XML elements and attributes for declaring a manifest file and accessing resources besides permissions and intents. Looking into API component calls in the executable file may allow exploring the behavior of an app and reporting its capabilities. However, in many cases, the attackers hide the API calls using cryptography, reflection, or dynamic code loading techniques to increase the difficulty of analysis.

1.2 Malware definition and statistics

Presently, mobile device apps are distributed through online marketplaces such as Google Play Store. Such marketplaces are considered hubs to allow developers

to publish their apps and distribute them as well. Today, there are more than 2.5 million applications available in the Google Play Store [8].

When downloading apps from unofficial markets, the user is usually at risk because there is no centralized control like official markets. As more users shift to Android devices, cybercriminals are also turning to Android to inflate their gain. However, many Android apps turn out to be malicious. The number of malicious software (malware) samples in the Android market has surged to an alarming number reaching over 5.49 million by the end of 2018 [9, 10].

A recent report from F-Secure [11] showed that over 99% of all malware programs that target mobile devices are designed for Android devices. Another report from the security firm G DATA shows that a new instance of Android malware pops up nearly every 10 seconds. Another report from AV-TEST [12] states very clearly that anyone seeking to make money by attacking mobile devices will choose Android devices as targets.

Malware is an umbrella term used to stand for an assortment of types of hostile or intrusive software, including viruses, worms, Trojan horses, ransomware, spyware, adware, and different malicious programs [13].

Ransomware is considered one of the most threatening malwares nowadays. There are two types of ransomware: crypto ransomware and lock screen ransomware. The crypto ransomware encrypts the information, while the locker ransomware hinders users from gaining access to their data by locking the device's screen. For both types, the attack demands a payment (ransom) to recover the files or access to the device. It is worth mentioning that paying the ransom money does not guarantee that the files will be back or that the ransomware will be removed from the device [14, 15].

According to Kaspersky, ransomware has taken place in most of the majority of notorious security attacks for the past decade. Also, 116.5 million attacks were noted in 2018, compared to 66.4 million in 2017, an increase of twofold in just 1 year [16].

Malicious apps, in general, are distributed mostly through phishing, drive-by attacks, and app stores. Phishing messages might comprise links to malicious apps and are sent over SMS or WhatsApp. Drive-by attacks are carried out by Web page exploits. When the user has a vulnerable browser, the exploit is able to execute a code. To infect users through app stores, malwares are submitted to them hiding as some legitimate app. In fact, in some cases some popular apps are modified to include malicious actions while keeping the app's main functionalities [17].

Therefore, a reliable tool is needed to test the trustworthiness of these apps before being installed. App risk scoring or rating should be empirically calculated according to different risk scoring techniques. The visualization of these risks should be easy enough for a normal user to recognize the risk associated with a specific app.

### **1.3 Android permissions and security model**

Android platform is very popular due to its available and comprehensive API framework [18]. Android API offers the developers of mobile apps the ability to gain access to hardware information, accessing user's data, knowing phone state, changing phone settings, etc. The developers are impacted by the permission model while developing mobile applications. To develop a mobile app, the engineers are required to determine, for each API functionality, what permissions are needed and how they are correctly activated. Android asks the developers to list publicly what permissions are used by the app; however, there are no mechanisms to know the exact purpose of such permissions and what kind of sensitive data they could use.

Android permissions mainly fall under four categories [19]:

- **Normal:** minimal risk permission is assigned automatically by the system and does not require an explicit declaration.
- **Dangerous:** the permission to private data, system process, and other hardware is referred to as dangerous and should be assigned explicitly at the time of installation or usage of the application.
- **Signature:** the applications get the same ID and the same access rights if the two application certificates are the same.
- **SignatureOrSystem:** the applications that are signed with the same certificate will get the same permission as the base system automatically.

Take the camera permission as an example; it belongs to the dangerous category. These permissions also ensure the safety of the system by keeping the user aware of what he is trying to do and what permissions have been requested. The issue with Android permission is that they are coarse-grained and violate the principle of least privileges (PoLP) that ensure that the only required thing is permitted. In contrast, Android allows overall permission about most of the features such as phone contact permission can allow checking phone state and other details.

The Android permission system obliges app's developers to state which security critical resources are needed. At runtime, the access requests are controlled by the permission checker component in order to secure the critical resources and operations.

In general, the security policy for the phones is delegated to their users. The lists of permissions will be shown to the users where they can accept or reject. It is essential and challenging to make sure that these apps appropriately deal with great value sensitive data [18].

Since Android 6.0, dangerous permissions are now asked explicitly to the user when requested the first time and then granted automatically. Android 6.0 changed some areas with regard to permissions. Two major changes were introduced. (1) Apps targeting SDK 23 (Software Development Kit) or higher can request permissions at run-time. (2) If an app requests a dangerous permission, with another permission from the same group that has been already granted, the system immediately grants it without any further interaction with the user.

The Android permission system received several criticisms [20]. The system is considered to be too coarse-grained since the user has to choose whether to accept all of the permissions declared by an app or to refuse to install the app. Users are usually not sure to determine if an app can be trusted or not. Actually, how Android is showing the required permissions is not very user-friendly and quite difficult to understand the risks associated with these permissions.

Android apps are allowed to define new custom permissions on Android. These permissions are used to protect an app's own resources from others. To define new custom permission, a permission name is needed, optionally including a permission group and a description regarding the permission purpose. Sixty-five percent of Google Play Store apps define their custom permissions, whereas 70% of these apps request them for their operation [21].

Mobile app history has shown that the users' privacy and security must be protected against benign applications not to mention malware ones. Actually, lots of widely used apps have been reported as requiring too many permissions or leaking user information to their servers intentionally [22].

Android uses both discretionary access control (DAC) and mandatory access control (MAC) to form a multilayer security model [23]. The model implements a

kernel-level application sandbox that uses Linux user identifiers (UIDs) and UNIX-style file permissions. Since version 4.3, Security-Enhanced Linux (SELinux) was introduced, and from version 4.4 it started being deployed in enforcing mode.

Android security has seen other improvements as well. In version 5.0, Google introduced smart lock, which allows users to unlock the phone using a trusted device, such as Bluetooth/NFC beacon, smartwatch, or facial recognition. In version 6.0, they introduced a fingerprint API. All these features are an extra step to make security easier for the average user. However, Android's security model is still based on a set of coarse-grained permissions.

Android builds its security basis on multiuser capabilities of Linux by assigning a unique ID to each application that will manage its own processes [24]. The runtime manager runs the applications in its sandbox that provides security as it does not allow:

- Inter-process communication
- Data access to other processes
- Hardware access such as camera, GPS, or network
- Access to local data of the phone such as media libraries and contacts

As a contrast to other OS platforms, the sandbox facility is provided by runtime manager for direct access to resources and hardware, while other operating systems provide sandboxing based on their kernel. This is based on features such as all kinds of requests outside the applications are by default denied and have to be permitted explicitly. When an application is installed, the permissions are allocated in addition to a unique, permanent identification (ID) that is also assigned to this app. This application ID is used to enforce the permissions for application, processes, and file system [25–27].

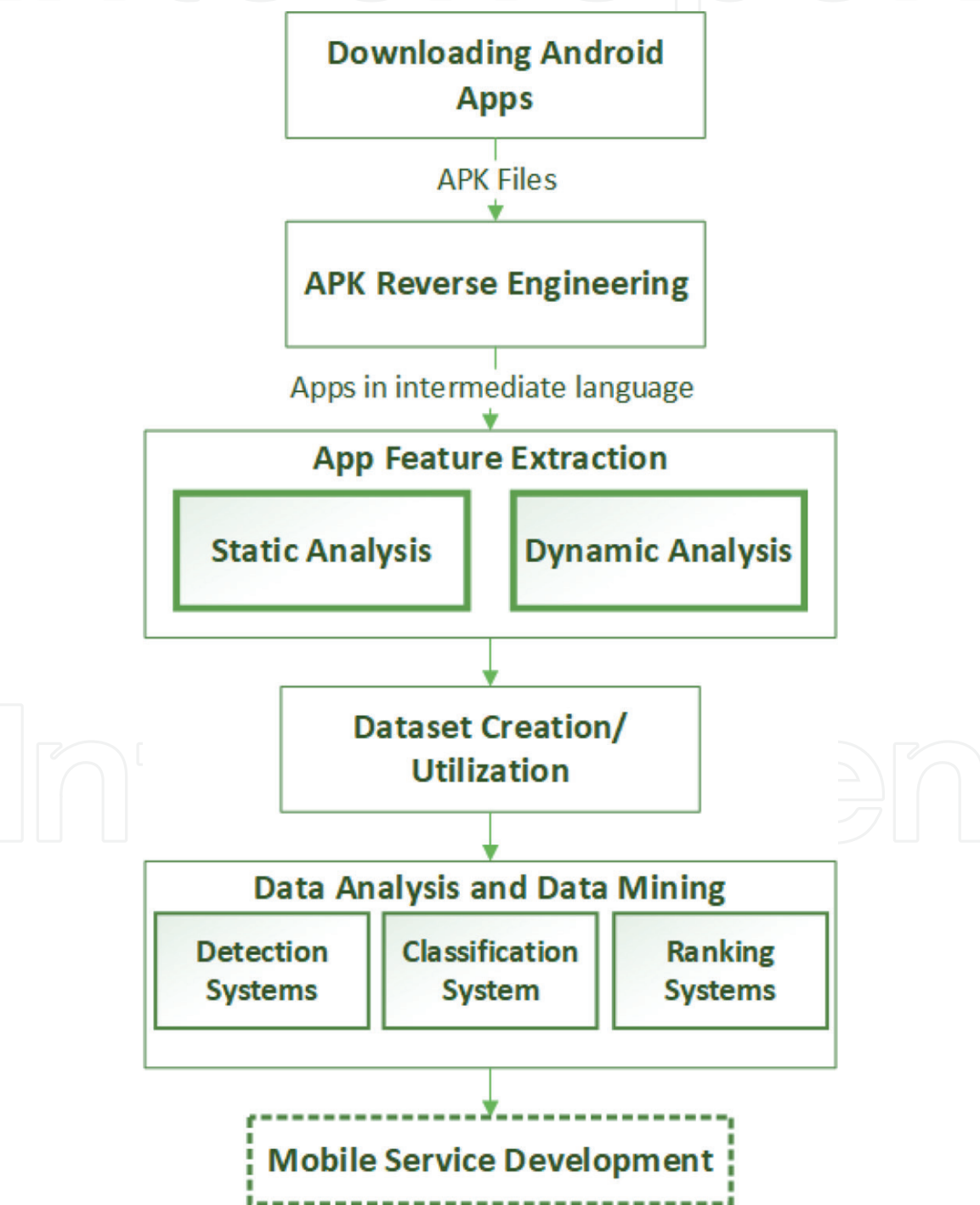
The files in the application are always private unless they are explicitly set to be shared using two modes, (1) readable and (2) writable. In order for two applications to share other's files, then the application ID must be the same for both applications, as well as the user ID. Additionally the public key infrastructure (PKI) certificate value must be shared to be considered as one application [27]. The paranoid network security mechanism is used to protect the network access by keeping all kinds of network access in separate groups such as WiFi, the Internet, and Bluetooth. Thus, if the application or process gets the permission to access a Bluetooth, then its application ID will be added to the group access list for Bluetooth and similarly for others. Consequently, one application can be assigned to one or more access groups [28].

Before any application distribution, Google that manages the main play store requires to sign the application using developers' personal certificate to make sure that the distributed copy is done through the right developer and no modification can be made to the application. If the two application matches the same certificate, Android will assign the same application ID to both applications and will access to private files for each application [27, 28].

Relying only on the current Android security model and permission levels to secure Android app is inefficient. Other more comprehensive security systems need to be considered and implemented to ensure efficient detection of malware apps. Consequently, the following sections present a reference model for Android application's security scanning process.

1.4 Android application scanning framework

A reference model for Android scanning process is shown in **Figure 2**. This model provides the core steps/phases vital to analyze Android apps and malware detection. The following sections highlight each one of these phases, starting from allocating the source of Android apps, downloading mechanisms, app’s source code generation process, app’s features extraction, applying static and dynamic analysis, generating datasets, detecting and classifying the app into benign or malware, and ranking its risk if it is detected as a malware app. Moreover, the mostly used mechanisms and tools utilized by researchers and developers at each process’s phase are also presented.



**Figure 2.**  
*Android application security scanning model [29].*

## 2. Android application

The app's source and how they are downloaded are presented in this section, in addition to the source code generation for Android applications.

### 2.1 App source and download

Android application collection process includes gathering APK files from different Android marketplaces. The main application sources include Google Play, Anzhi, and AppChina. For every potential free app, the crawler script must ensure that the app has not been downloaded before and then calculate the app's hash using the SHA256 algorithm [30]. Once the app is downloaded, it can be archived for future use. Chrome APK Downloader, a desktop version of APK downloader tool, can be used to download the APK files of the free Android applications into desktop from Google Play marketplace [31]. For the paid applications, the Raccoon APK Downloader can be used to download APK files from Google Play Store [32].

### 2.2 Source code generation

After downloading the apps, they need to be analyzed. In order to do that, APK reverse engineering process is required to decompile, rebuild, and convert the Android executable code (.apk file) into an intermediate language such as Smali, Jimple, and Jasmin [33]. The aim of reverse engineering is to retrieve the source file from the executable files in order to apply program analysis. Unzipping the APK files generates .dex files. By reassembling the dex files using an APK reverse engineering tool, the Java files can be retrieved. Three of the most popular tools that have been used in Android APK reverse engineering are Apktool, Dex2jar, and Soot. A comparison of Android reverse engineering tools was conducted in [33]. The results showed that Apktool which uses Smali reassembled 97% of the original code, whereas Soot which uses Jimple and Dex2jar which uses Jasmin preserve 73% and 69% of the app's original code, respectively.

## 3. Android application analysis

The process of analyzing Android apps to detect different types of malwares and the result of such analysis in terms of datasets are illustrated in this section.

### 3.1 Feature extraction

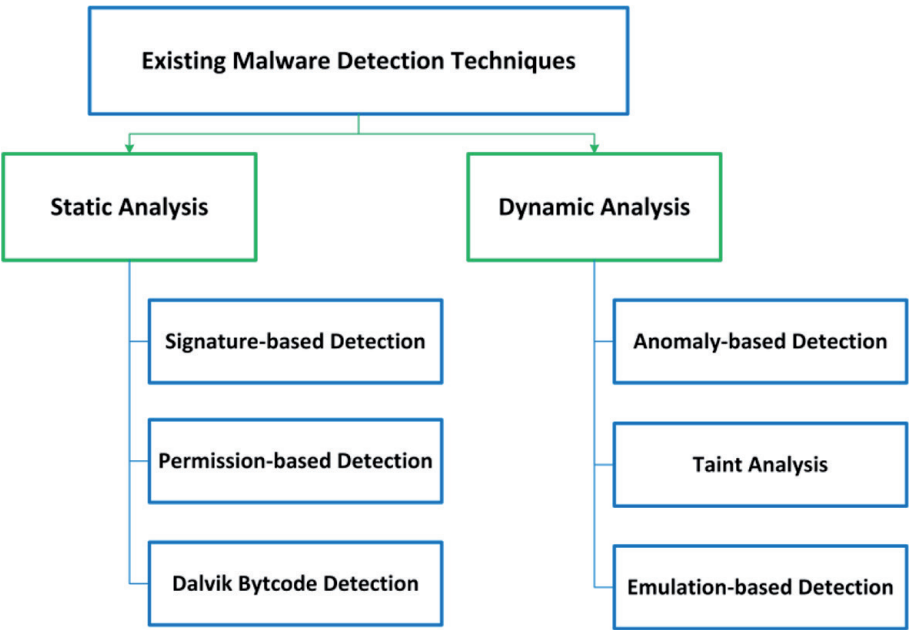
Once the app's source code is retrieved, the feature extraction process starts. The features that are usually extracted depend on the type of malware and the analysis mode whether static or dynamic. This will be explained in the following two sub-sections. **Table 2** lists the most commonly used static and dynamic features [34].

### 3.2 Static analysis

The static analysis aims to check the existence of malware by disassembling the source code without executing the application. Tools which perform static analysis are mainly categorized into three approaches as shown in **Figure 3**: (1) signature-based detection, (2) permission-based detection, and (3) Dalvik Bytecode detection. There is some limitation which is related to each static detection approach. The

Static features	Dynamic features
Permission	Network, SMS, power usage, CPU, process info, native and Dalvik memory
API calls	Data packets being sent, IP address, no. of active communications, system calls
String extracted	Process ID, system calls collected by strace, returned values, times between consecutive calls
Native commands	Network traffic, destination IP address
XML elements	System calls collected by strace, logs of system activities
Meta data	Data collected by logger, Internet traffic, battery percentage, temperature collected every minute
Opcodes from .dex file	
Task intents	

**Table 2.**  
*Most commonly used features in static and dynamic analyses [34].*



**Figure 3.**  
*Existing malware detection techniques.*

signature-based detection which relies on stored signatures for known malwares does not have the ability to detect unidentified malware signatures [35]. In the permission-based detection, the benign app could be considered, incorrectly, as a malware due to the minor variation of the requested permissions from the original and the malware application [30]. Finally, the Dalvik Bytecode detection, which assists in evaluating the applications actions, consumes more resources [36]. Several tools were discussed and analyzed in [35–41] such as FlowDroid [39], PScout [40], and ApkAnalyser [41]. Each of the aforementioned tools focuses on one or more features. The most generally extracted static features are the permissions [18], API calls [42], and source code metrics [43].

Years ago, different rival static approaches have been proposed like TaintDroid [43], DroidRanger [45], and RiskRanker [46] to detect malicious malware features. But all of them rely on manually crafted detection patterns which may not be able to detect new malware and come with significant device performance cost [47].

Authors in [48] proposed DREBIN as the first approach which provides detection of Android malicious code directly on the mobile device. They used

static analysis in a machine learning system to distinguish malware from trusted applications. They considered linear support vector machines for classification. This approach, however, cannot detect runtime loaded and obfuscated malicious applications because it relies on static analysis [47].

Yang [49] developed a prototype (AppContext) that detects malicious apps using static analysis. They mined 633 benign apps from Google Play and 202 malware apps from various malware datasets. AppContext identifies applications using machine learning based on the contexts that trigger security-sensitive behaviors (e.g., the conditions and events that cause the security-sensitive behaviors to occur). But this approach can be evaded by dynamic code loading and consumes huge human efforts in labeling each security-sensitive behavior [50].

Akhuseyinoglu and Akhuseyinoglu [51] have proposed an automated feature-based static malware detection system called AntiWare for Android devices. It is automated since it engages the machine learning method for detecting malicious applications by using the extracted apps' features. They took into consideration the requested permissions and Google market data, including developer name, download time, and user ratings from Google Play as a feature set. AntiWare is designed to predict the rank of an application inquired by the user as malicious or benign and then report the results to the user. The main disadvantages are primarily depending on market data on Google Play and the requested permissions. The market data is not reliable since a lot of applications are invented by different new developers every second. Additionally, the permissions by its own are not sufficient to assess the malicious behavior of an application.

### 3.3 Dynamic analysis

In dynamic analysis, the application actions are dynamically analyzed and monitored during the execution time. The unexecuted code might be missed by this approach, but it can effectively detect the malware behaviors which are not detectable by the static analysis. Since this approach occurs during runtime, it can be performed in a controlled environment to avoid damaging the device [52].

Android dynamic malware analysis detection techniques (see **Figure 3**) can be classified into [53, 54]:

- **Anomaly detection:** the anomaly-based detection has the ability to identify suspicious behaviors to indicate the presence of malware. A drawback for this technique is that it can sometimes flag a benign application as malware because it displayed similar behaviors of malware.
- **Taint analysis:** it is an efficient technique that checks and monitors sensitive information; however, a limitation is that the performance becomes very slow rendering it useless to be applied in real time.
- **Emulation-based detection:** it is a detection technique, where it scans the application behavior by simulating the conditions of its execution environment to determine if the application is a benign or malware application from the behavior. Similar to this technique is sandbox-based detection, but the main difference originates from the details of designing each approach. A major drawback for this approach is that it requires more resources.

Tam [55] applied dynamic analysis method and machine language to detect malware. They capture real-time system calls performed by the application as key information to discriminate between ransomware, malware, and trusted files and

called it CopperDroid. CopperDroid runs the Android application in the sandbox and records all system calls, in particular inter-process communications (IPC) and remote procedure call (RPC) interactions which are essential to understanding an application maliciousness behavior. However, some types of malware can detect the virtual environment and act differently (as a benign) which gives false positives.

Recent research [56] dynamically classifies Android applications to malicious or benign in the first launching of the app. The classification is applied based on the frequency of system calls as an indicator of suspicious behavior. They have built a syscall-capture system to capture and analyze the behavior of system call traces made by each application during their runtime. They have achieved an accuracy level of 85% and 88% using the decision tree algorithm and the random forest algorithm, respectively.

Also, Wang [57] proposed a dynamic analysis to analyze an application on the fly to detect malicious behavior. They developed a prototype called Droid-AntiRM to identify malware applications that employ anti-analysis techniques. The prototype identifies the condition statements in applications that could trigger the malicious acts of malware, which are unable to be recognized by static analysis. However, their prototype cannot handle dynamic code loading, encryption, or other various obfuscation techniques.

Many tools have been developed based on the dynamic perspective such as TaintDroid [44], Droidbox [58], and MobSF [59]. Additionally, some tools are considering both static and dynamic analysis in their solutions such as VirusTotal tool [60].

### **3.4 Ransomware detection**

Unfortunately, there were very few researches studying ransomware where the malicious app blocks access to the Android device or/and its data. In [61] the authors presented a tool called Cryptolock that focuses on detecting ransomware by tracking the changes in real-time user data. They have implemented the tool on Windows platform. However, Cryptolock may send a false-positive alert because it cannot differentiate whether the user or the ransomware is encrypting a set of files [62]. They focus on changes on user's data rather than trying to discover ransomware by investigating its execution (e.g., API call monitoring and access permissions).

HelDroid tool [63] was developed to analyze Android ransomware and to detect both crypto and locking ransoms. The tool includes a text classifier that uses natural language processing (NLP) features, a lightweight Smali emulation technique to detect the locking scheme, and the application of taint tracking for detecting file-encrypting flows. The primary disadvantage of this approach is that it highly depends on a text classifier as it assumes the availability of text. Also, it cannot be applied to some languages that have no specific phase structure like Chinese, Korean, and Japanese. This approach can be easily avoided by ransomware by applying techniques such as encryption and code obfuscation [63]. Moreover, like whatever machine learning approach, HelDroid trains the classifier in order to label an app as a ransomware. The detection capability of the model depends on the training dataset [64–66].

Another work in literature exploring the ransomware detection in Android mobiles was presented in [67]. The authors presented R-PackDroid as a static analysis approach that classifies Android applications into ransomware, malware, or benign using random forest classifier. The classification employed was based on information taken from the system API packages. An advantage over the previous approach (HelDroid) is its ability to detect ransomware regardless of the application language. Also, it flags the applications that were recognized as ransomware with very low confidence by the VirusTotal service. However, R-PackDroid cannot analyze applications with a feature code that is dynamically loaded at runtime or classes that are fully encrypted because it relies on static analysis.

Likewise, Mercaldo [68] focused on ransomware detection specifically in Android. They tested a dataset composed of 2477 samples with real-world ransomware and benign applications. The main issue of this approach is that it is manual and requires a lot of effort to analyze and build logic rules used for the classification [69].

Another automated detection approach was introduced in [70] to analyze and penetrate the malicious ransomware. They have introduced some features of static and dynamic analysis of malware. In static analysis, malicious features can be discovered with permissions, API calls, and APK structure, while malicious features in the dynamic analysis may include access to sensitive data or sensitive paths, access to the HTTP server, and user charge without notification and bypass permissions. The aim was to produce a better performance apparatus that supports ransomware detection in Android mobiles which they have designed but did not implement. The authors analyzed one malware and listed the steps of APK analysis as a concept but did not implement the proposed design. Therefore, there are no results that can prove the effectiveness of their approach.

In [71], the authors experimentally presented a new framework called DNADroid which is a hybrid of static and dynamic techniques. This framework employs a static analysis approach to classify apps into suspicious, malware, or trusted. Only suspicious classified applications are then inspected by dynamic analysis to determine if it is ransomware or not. The main weakness is that dynamic analysis is only applied to suspicious applications leaving the possibility of having malware not successfully recognized by static analysis.

### **3.5 Dataset creation and utilization**

Datasets are mainly in two types. The first type is the Android application datasets. These include both benign apps and malicious apps. For the benign apps, the majority of researchers are collecting them from the app stores like Google Play Store [30, 37, 60]. For malicious apps, it depends on malicious behavior under study. For example, for malware Android apps, VirusTotal was one of the main sources for many researchers [38, 60]. For ransomware apps, HelDroid project [63] and RansomProper project [38] were also used.

The second type is the datasets generated after extracting the app's features. The researchers can either use existing constructed datasets considering the features under study or build new ones by screening the apps and extracting their features. The main concerns regarding the use of existed datasets are (1) absence of up-to-date apps and operating system version (2) including many duplicate samples (3) and not being accessible. These reasons could motivate researchers to build their own up-to-date and labeled datasets.

## **4. Android malware application detection and ranking**

Many previous works have considered the problem of ranking Android apps and classify them to either malware or benign apps. The majority of these solutions have relied mainly only on the permission model and what types of permissions are requested/used by the application. They used different ways and depth of analysis in this regard.

The work presented in [72] studied the permission occurrences in the market apps and the malware apps. Also, the authors analyzed the rules (a combination of permissions) defined in Kirin [73] in order to calculate the risk signals and to reduce the warning rates. Gates et al. in [74] have compared work presented in [72, 73], naïve-based algorithms and two proposed methods for risk scoring. These methods

consider the rarity of permissions as the primary indicator that contributes to raising a warning. The performance comparison was in terms of the detection rate.

The authors in [75] used similar hypotheses of listing the permissions in each app and count occurrences of permissions in similar apps (a game category in their case) excluding the user-defined permissions that are not affecting the privacy. In their solution, they gave the user the choice to turn off the permission(s). In [76], the authors used the combination of features (permissions) to compare the clean app values with the malware values to come up with thresholds that will be used to classify new Android apps. Within the same context, the idea presented in [77] was to construct a standard permission vector model for each category, which can be used as a baseline to measure and assess the risk of applications within the same category. For each downloaded app, the permission vector will be extracted and compared with the standard one; the amount of deviation from the baseline will calculate the app's risk.

While discussing the approaches in the existing risk scoring systems and their main dependency on the Android permissions, it is worth asking how many of them have considered the involvement of the user with the scoring results and, if they decided to involve the user, how the risk was displayed and communicated to the user. The empirical study conducted in [78], which implemented an intensive study on top, used permissions with a high-risk level. They calculated the risk level based on the type of permissions and the probability they will be requested by the app. The risk value for each permission in addition to its technical name and description was transmitted to the users. Although a coloring code was used to indicate the level of risk, still users are involved in technical details which will not help them to take proper decisions regarding the apps' installation. The work presented in [79] has utilized fuzzy logic to measure the risk score. Also, in addition to the permissions and their categories, they took input from different antivirus tools to calculate the score. Their system allowed the user to upload the app's APK through the browser and provided them with a risk report. This report showed the risk score, permission usage rate, and unnecessary permission usage rate in addition to the list of permissions, their categories, and risk level. On the other hand, the authors in [80] have considered the statistical distribution of the Android permissions in addition to the probabilistic functions. The declared but not exploited permissions and vice versa were all considered in their risk analysis. Machine learning was also utilized to measure risk.

In terms of visualizing the permissions and their risks, the authors in [81] introduced Papilio to visualize Android application permissions graphically. This helped them to find the relations among the applications and applications' permissions as well. Papilio was able to find the permissions requested frequently by applications and permissions that either never requested or requested infrequently. The authors in [82] stressed the importance of visualizing the statistical information related to Android permissions. Having graphical representation for the permissions' statics within a specific category encouraged the users to choose more often apps with a lower number of permissions. A privacy meter was used in [83] to visualize the permissions' statistics through a slider bar which outperformed the existing warning system like Google's permission screens. Visualizing app activities enhances user's awareness and sensitivity to the privacy intrusiveness of mobile applications [84]. Another attempt to visualize the permissions statistics was also introduced in [85] using lifelog analysis views in terms of risk history and app's risk view.

From the above-related work, we can observe that the majority of the previous solutions have mainly relied on permissions either statistically or based on probability to analyze Android apps, to classify them as malwares, and to measure their risk level. Although permissions are important to analyze and classify Android applications. However, these permissions should be up-to-date. Also, other important static and dynamic metrics need to be considered to guarantee a comprehensive evaluation and consequently an accurate detection of malware apps.

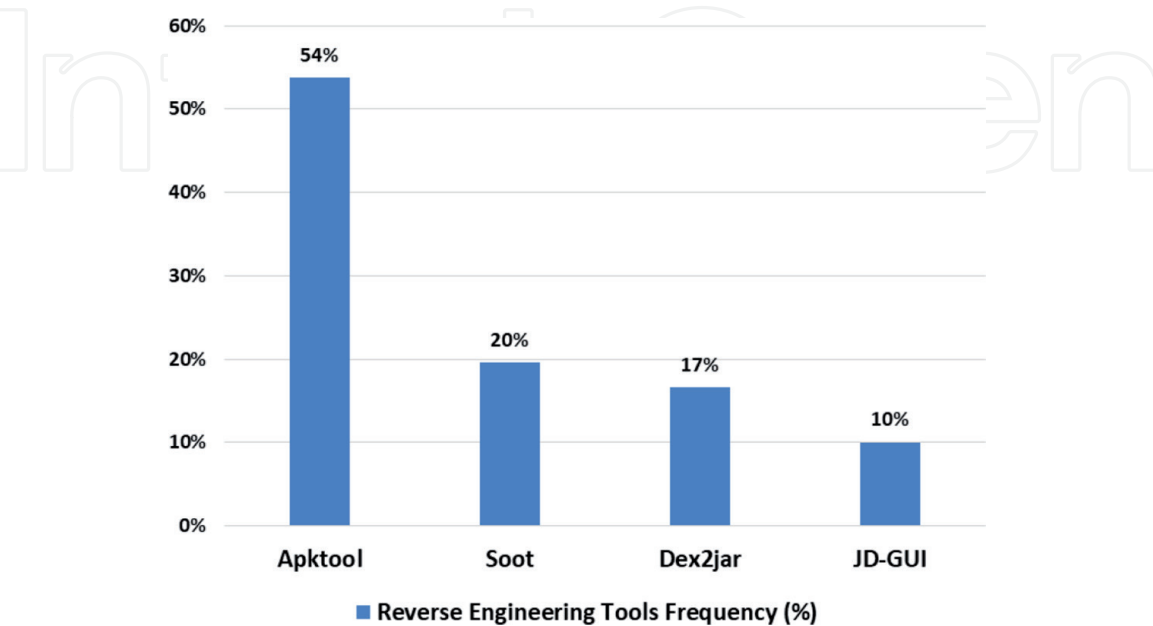
There have been many types of research on designing malicious detection approaches. Such approaches resort to static analysis of the malware, and others use dynamic analysis, while some methods utilize both static and dynamic analyses to get better detection of a malicious incident. Moreover, the generated datasets will be analyzed in order to detect any potential security threats, regardless whether these datasets were constructed based on static or dynamic tests or even both. Usually, data mining techniques could be used for the purpose of detecting and classifying attacks [42, 52]. Moreover, intelligence techniques could be utilized to even rank the risk by assigning the attack a risk score [42, 86].

The scanning service might fruit in developing a mobile application that is installed on user’s devices to examine the Android application and discriminate, if it is a clean app or a malicious app to warn the user and protect her/his Android device. DREBIN [87] is one of the malware detection systems available for smart-phones. One of the major features that DREBIN provides is instantaneous malware detection. When a new application is downloaded, DREBIN starts the analyzing process directly. As a result, the user is protected against any unreliable sources. Another example of anti-malware software is HinDroid [88] which has been integrated as one of Comodo’s mobile security scanning tools. HinDroid structures the APIs based on heterogeneous information network in order to make predictions about the tested application. Consequently, HinDroid can reduce the time and cost of analyzing Android apps.

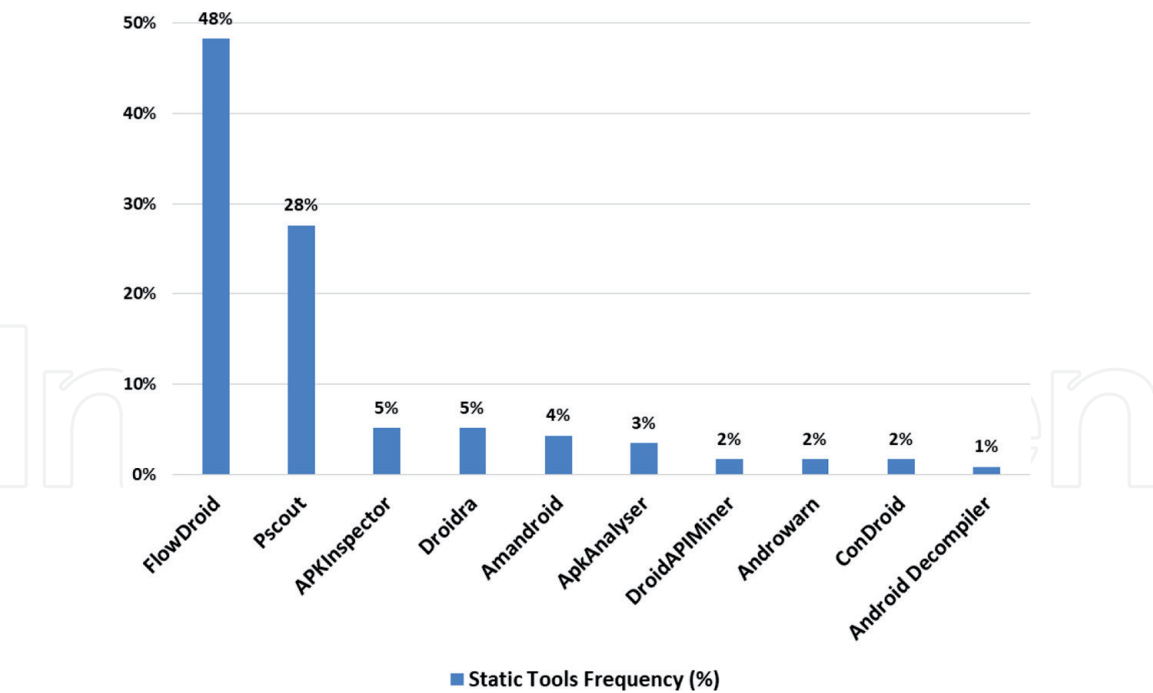
### 5. Statistical analysis

This section presents a statistical study to show the frequency of the used approaches, methods, datasets, and tools in the current systems. Various, related, recent, published solutions in 2017–2018 were considered in this study.

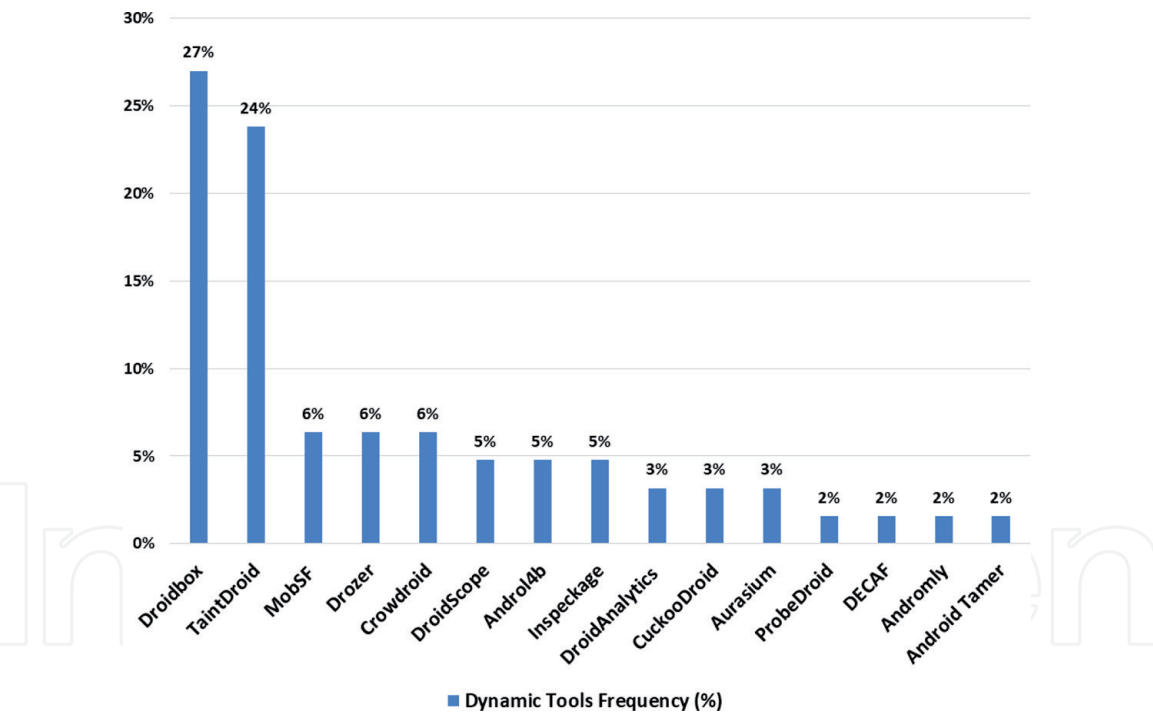
In regard to reverse engineering tools utilized by researchers, APKtool was heavily used by 54% in comparison with other tools (see **Figure 4**). Soot was next with 20% of usage.



**Figure 4.**  
*Reverse engineering tool usage in 2017–2018 research.*



**Figure 5.**  
*Static tool frequency in 2017–2018 research work.*

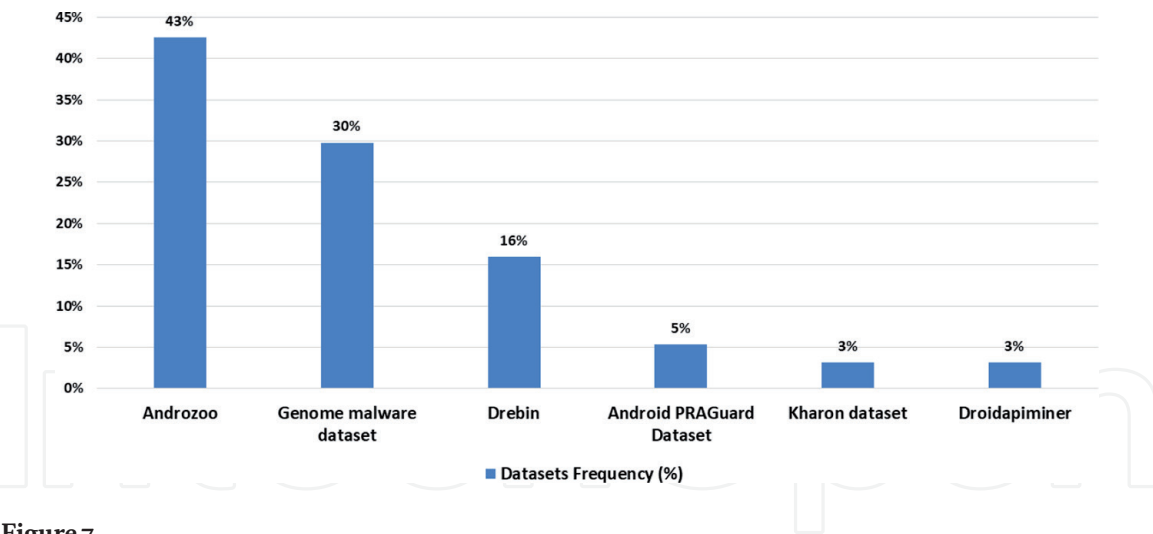


**Figure 6.**  
*Dynamic tool frequency in 2017–2018 research work.*

**Figure 5** shows a comparison among the static tools which were utilized by researchers. It can be observed that 48% of the static-based systems used FlowDroid tool in their solutions. PScout was the second most used with percentage reaching around 28%.

Dynamic analysis tool usage is illustrated in **Figure 6**. The majority of existing solutions used Droidbox with 27% and TaintDroid with 24% in comparison with other approaches. The rest of the results are shown in **Figure 6**.

The results in **Figure 7** reveal that AndroZoo was the most used dataset in 2017–2018. The percentage of usage reached 43%. Genome and DREBIN datasets came next with frequencies 30 and 16%, respectively.



**Figure 7.**  
*Dataset frequency in 2017–2018 research work.*

## 6. Conclusions

This chapter highlighted the booming of Android technologies and their applications which make them more attractive to security attackers. Recent statistics of Android malwares and their impact were presented. Additionally, this chapter has provided the main phases required to apply security scanning to Android applications. The purpose is to protect Android users and their devices from the threats of different security attacks. These phases include the way of downloading Android apps, decoding them to generate the source code, and how this code is screened to extract the required features to apply either static analysis or dynamic analysis or both. The feature extraction process resulted in constructing different datasets. Proper data analysis and data mining techniques could be applied to examine the app and classify it as benign or malware with high accuracy. The malware detection service could be implemented and provided in terms of a mobile application that will communicate the scanning results to the user in a friendly way. The chapter was concluded by presenting a statistical study that showed the most used tools and datasets throughout the scanning process for the last 2 years 2017 and 2018.

## Acknowledgements

We would like to acknowledge the Security Engineering Lab ([sel.psu.edu.sa](http://sel.psu.edu.sa)) team and Prince Sultan University for supporting this work. Special thanks go to Ms. Samah Alsoghyer and Ms. Aala Khayer.

## Conflict of interest

The authors declare that there is no “conflict of interest” in regard to publishing this book chapter.

IntechOpen

### **Author details**


Iman Almomani<sup>1,2\*</sup> and Mamdouh Alenezi<sup>1</sup>

1 Prince Sultan University, Riyadh, KSA

2 The University of Jordan, Amman, Jordan

\*Address all correspondence to: [imomani@psu.edu.sa](mailto:imomani@psu.edu.sa)

### **IntechOpen**

© 2019 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. 

## References

- [1] Alliance OH. Android overview. Open Handset Alliance. 2011;8:88-91
- [2] Faruki P, Bharmal A, Laxmi V, Ganmoor GM, Conti M. Android security: A survey of issues, malware penetration, and defenses. *IEEE Communications Surveys & Tutorials*. 2015;17(2):998-1022
- [3] Global smartphone sales to end users from 1st quarter 2009 to 2nd quarter 2018 [Internet]. Available from: <https://www.statista.com/statistics/266219/global-smartphone-sales-since-1st-quarter-2009-by-operating-system/> [Accessed: 2019-03-22]
- [4] Nimodia C, Deshmukh HR. Android operating system. *Software Engineering*. 2012;3(1):10
- [5] Brahler S. Analysis of the android architecture. *Karlsruhe Institute for Technology*. 2010;7(8):3-64
- [6] Drake JJ, Lanier Z, Mulliner C, Fora PO, Ridley SA, Wicherski G. *Android Hacker's Handbook*. New Jersey, USA: John Wiley & Sons; 2014
- [7] Mithilesh Joshi BlogSpot. What is android application components and how we use it? 2015. <https://mithileshjoshi.blogspot.com/2015/06/CITATIONS-74-what-is-android-application-components.html> [Accessed November 27, 2017]
- [8] AppBrain. Number of Android applications on the Google Play store | AppBrain. 2019. Available from: <https://www.appbrain.com/stats/number-of-android-apps> [Accessed: 2019-03-02]
- [9] AV-TEST The IT-Security Institute. Malware Statistics and Trends Report | AV-TEST. The AV-TEST Institute; 2018. Available from: <https://www.av-test.org/en/statistics/malware/> [Accessed: 2019-03-02]
- [10] GData. Cyber attacks on Android devices on the rise. 2018. Available from: <https://www.gdatasoftware.com/blog/2018/11/31255-cyber-attacks-on-android-devices-on-the-rise> [Accessed: 2019-04-15]
- [11] F-Secure State of cyber security. Available from: <https://www.f-secure.com/documents/996508/1030743/cyber-security-report-2017> [Accessed: 2019-04-15]
- [12] Security Report 2016/17 [Internet]. Available from: [https://www.avtest.org/fileadmin/pdf/security\\_report/AV-TEST\\_Security\\_Report\\_2016-2017.pdf](https://www.avtest.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2016-2017.pdf) [Accessed: 2019-04-15]
- [13] Delac G, Silic M, Krolo J. Emerging security threats for mobile platforms, MIPRO. In: 2011 Proc. 34th Int. Conv. 2011. pp. 1468-1473
- [14] Savage K, Coogan P, Lau H. Security Response – The Evolution of Ransomware. Mountain View (CA): Symantec Corporation; 2015
- [15] Liska A, Gallo T. Ransomware: Defending Against Digital Extortion. California, USA: O'Reilly Media, Inc; 2016
- [16] (Kaspersky) Chebyshev V. Mobile Malware Evolution 2018 [Internet]. Available from: <https://securelist.com/mobile-malware-evolution-2018/89689/> [Accessed: 2019-04-16]
- [17] Grégio A, Abed R, Afonso V, Filho D, Geus P, Jino M. Toward a taxonomy of malware behaviors. *The Computer Journal*. 2015;58(10):2758-2777
- [18] Alenezi M, Almomani I. Abusing android permissions: A security perspective. *IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AECT)*; Amman, Jordan. 2017

- [19] Developer.Android. "Permission Types" [Internet]. Available from: <https://developer.android.com/guide/topics/manifest/permission-element> [Accessed: 2019-04-15]
- [20] Gianluca D, Martinelli F, Matteucci I, Petrocchi M, Saracino A, Sgandurra D. Risk analysis of android applications: A user-centric solution. *Future Generation Computer Systems*. 2018;**80**:505-518
- [21] Seray T, Demetriou S, Ganju K, Gunter C. Resolving the predicament of android custom permissions. In: *ISOC Network and Distributed Systems Security Symposium (NDSS)*. 2018
- [22] Arstechnica.com. Your iPhone Calendar isn't Private. 2012. Available from: <http://arstechnica.com/apple/2012/06/your-iphone-c>
- [23] Haining C, Li N, Enck W, Aafer Y, Zhang X. Analysis of SEAndroid policies: Combining MAC and DAC in android. In: *Proceedings of the 33rd Annual Computer Security Applications Conference*. Orlando, FL, USA: ACM; 2017. pp. 553-565
- [24] Ratazzi EP. Understanding and Improving Security of the Android Operating System. No. AFRL-RI-RS-TP-2018-001. New York, USA: Air Force Research Laboratory/Information Directorate Rome United States; 2016
- [25] Backes M, Bugiel S, Hammer C, Schranz O, von Styp-Rekowsky P. Boxify: Full-fledged app sandboxing for stock android. In: *Proceedings of the 24th {USENIX} Security Symposium ({USENIX} Security 15)*. 2015. pp. 691-706
- [26] Bennet Y, Sehr D, Dardyk G, Chen J, Muth R, Ormandy T, et al. Native client: A sandbox for portable, untrusted x86 native code. In: *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*. Berkeley, CA, USA: IEEE; 2009. pp. 79-93
- [27] Elenkov N. *Android Security Internals: An in-Depth Guide to Android's Security Architecture*. San Francisco, California, USA: No Starch Press; 2014
- [28] Georgios P, Homburg P, Anagnostakis K, Bos H. Paranoid android: Versatile protection for smartphones. In: *Proceedings of the 26th Annual Computer Security Applications Conference*. Austin, Texas, USA: ACM; 2010. pp. 347-356
- [29] Almomani I, Alkhayer A. Android applications scanning: The guide. In: *Proceedings of the IEEE International Conference on Computer and Information Sciences (ICCIS)*; 3-4 April 2019; Saudi Arabia. Joutf: IEEE; 2019
- [30] Allix K, Bissyandé F, Klein J, Traon, L. AndroZoo. In: *Proceedings of the 13th International Workshop on Mining Software Repositories—MSR 16*. 2016. DOI: 10.1145/2901739.2903508
- [31] Abubaker H. Analytics on malicious android applications. *International Journal of Advanced Software Computer*. 2018;**10**:106-118. ISSN 2074-8523
- [32] Ikram M, Kaafar M. A first look at mobile ad-blocking apps. In: *Proceedings of the 2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*. 2017. DOI: 10.1109/nca.2017.8171376
- [33] Arnatovich L, Wang L, Ngo N, Soh C. A comparison of android reverse engineering tools via program behaviors validation based on intermediate languages transformation. *IEEE Access*. 2018;**12**:12382-12394. DOI: 10.1109/access.2018.2808340
- [34] Baskaran B, Ralescu AA. Study of android malware detection techniques and machine learning. In: *Proceedings of the Mod. Artif. Intell. Cogn. Sci. Conf*. 2016. pp. 15-23

- [35] Arshad S, Ali M, Khan A, Ahmed M. Android malware detection & protection: A survey. *International Journal of Advanced Computer Science and Applications*. 2016;7(2):463-475. DOI: 10.14569/ijacsa.2016.070262
- [36] Zachariah R, Yousef M, Chacko A. Android malware detection and prevention. *International Journal of Recent Trends in Engineering and Research*. 2017;3(2):213-217. DOI: 10.23883/ijrter.2017.3028.0uhbl
- [37] Baskaran B, Ralescu A. A study of android malware detection techniques and machine learning. In: *Proceedings of the IEEE International Conference on Smart Internet of Things (SmartIoT)*. 2018. DOI: 10.1109/smartiot.2018.00034
- [38] Chen J, Wang C, Zhao Z, Chen K, Du R, Ahn G. Uncovering the face of android ransomware: Characterization and real-time detection. *IEEE Transactions on Information Forensics and Security*. 2018;13(5):1286-1300
- [39] Arzt S, Rasthofer S, Fritz C, Bodden E, Bartel A, Klein J, et al. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices*. 2014;49(6):259-269
- [40] Au K, Zhou Y, Huang Z, Lie D. Pscout: Analyzing the android permission specification. In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*; October 2012. Raleigh, North Carolina, USA: ACM; 2012. p. 217-228
- [41] Mujahid S, Abdalkareem R, Shihab E. Studying permission related issues in android wearable apps. In: *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME)*; September 2018. Madrid, Spain: IEEE; 2018. pp. 345-356
- [42] Tao G, Zheng Z, Guo Z, Lyu M. MalPat: Mining patterns of malicious and benign android apps via permission-related APIs. *IEEE Transactions on Reliability*. 2018;67(1):355-369. DOI: 10.1109/tr.2017.2778147
- [43] Alenezi M, Almomani I. Empirical analysis of static code metrics for predicting risk scores in android applications. In: *Proceedings of the 5th Symposium on Data Mining Applications (SDMA2018)*; 21-22 March, 2018; Riyadh, KSA
- [44] Enck W. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In: *Proceedings of the ACM Transactions on Computer Systems (TOCS)* 32.2. 2014. p. 5
- [45] Zhou Y “H. You, get o\_ of my market: Detecting malicious apps in official and alternative android markets. In: *Proceedings of the NDSS*. 2012. pp. 50-52
- [46] Grace M. Riskranker: Scalable and accurate zero-day android malware detection. In: *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*. ACM; 2012. pp. 281-294
- [47] Security Affairs. DREBIN Android app detects 94 percent of mobile malware [Internet]. Available from: <http://securityaffairs.co/wordpress/29020/malware/drebin-android-av.html> [Accessed: 2017-12-01]
- [48] Arp D. DREBIN: Effective and explainable detection of android malware in your pocket. *Proceedings of the NDSS*. 2014
- [49] Yang W. Appcontext: Differentiating malicious and benign mobile app behaviors using context. In: *Proceedings of the Software Engineering (ICSE)*; 2015 IEEE/ACM 37th IEEE International Conference. IEEE. 2015. pp. 303-313

- [50] Yang W. Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps. *Proceedings of the Proc. ACSAC*. 2017
- [51] Akhuseyinoglu N, Akhuseyinoglu A. AntiWare: An automated Android malware detection tool based on machine learning approach and official market metadata. In: *Proceedings of the 2016 IEEE 7th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*. October 2016. pp. 1-7. DOI: 10.1109 /UEMCON.2016.7777867
- [52] Allix K, Bissyandé T, Klein J, Traon Y. AndroZoo: Collecting millions of android apps for the research community. In: *Proceedings of the 13th International Workshop on Mining Software Repositories—MSR*. 2016. DOI: 10.1145/2901739.2903508
- [53] Zachariah R, Akash K, Yousef M, Chacko A. Android malware detection a survey. In: *Proceedings of the 2017 IEEE International Conference on Circuits and Systems (ICCS)*. 2017. pp. 238-244
- [54] Kaspersky Lab. “Emulator.” [Internet]. Available from: <https://www.kaspersky.com/enterprise-security/wiki-section/products/emulator> [Accessed: 2019-04-01]
- [55] Tam K. CopperDroid: Automatic reconstruction of android malware behaviors. *Proceedings of the NDSS*. 2015
- [56] Bhatia T, Kaushal R. Malware detection in android based on dynamic analysis. In: *Proceedings of the 2017 International Conference on Cyber Security And Protection Of Digital Services (Cyber Security)*. 2017. pp. 1-6. DOI: 10.1109/CyberSecPODS.2017.8074847
- [57] Wang ZD-ARM. Taming control flow anti-analysis to support automated dynamic analysis of android malware. In: *Proceedings of the 33rd Annual Conference on Computer Security Applications (ACSAC'17)*. 2017
- [58] Huang H, Zheng C, Zeng J, Zhou W, Zhu S, Liu P, et al. A large-scale study of android malware development phenomenon on public malware submission and scanning platform. *IEEE Transactions on Big Data*. 2018; 15-23. DOI: 10.1109/tbdata.2018.2790439
- [59] Kaur R, Li Y, Iqbal J, Gonzalez H, Stakhanova NA. Security assessment of HCE-NFC enabled E-wallet banking android apps. In: *Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*; July 2018. Tokyo, Japan: IEEE; 2018. pp. 492-497
- [60] VirusTotal Malware Intelligence Services [Internet]. n.d. Available from: <https://www.virustotal.com/learn/> [Accessed 2018-12-15]
- [61] Scaife N. Cryptolock (and drop it): Stopping ransomware attacks on user data. In: *Proceedings of the Distributed Computing Systems (ICDCS)*, 2016 IEEE 36th International Conference on. Nara, Japan: IEEE; 2016. pp. 303-312
- [62] Tripwire. Early-Warning Ransomware Detection Tool Could Help Protect Users Despite Drawbacks [Internet]. Available from: <https://www.tripwire.com/state-of-security/security-data-protection/cyber-security/early-warning-ransomware-detection-tool-could-help-protect-users-despite-drawbacks/> [Accessed 2017-12-02]
- [63] Mercaldo F, Nardone V, Santone A. Ransomware inside out. In: *Proceedings of the 2016 11th International Conference on Availability, Reliability and Security. ARES*; 2016. pp. 628-637. DOI: 10.1109/ARES.2016. 35
- [64] Li J, Sun L, Yan Q, Li Z, Srisa-an W, Ye H. Significant Permission

Identification for Machine-Learning-Based Android Malware Detection, *IEEE Transactions on Industrial Informatics*; July 2018;14(7):3216-3225

[65] Mercaldo F, Nardone V, Santone A. Ransomware inside out. *Proceedings of the 11th International Conference on Availability, Reliability and Security (ARES)*; August. 2016:628-637. DOI: 10.1109/ARES.2016.35

[66] Sun L, Wei X, Zhang J, He L, Yu PS, Srisa-an W. Contaminant removal for Android malware detection systems. Boston, MA, USA: 017 IEEE International Conference on Big Data (Big Data); 11-14 Dec 2017. pp. 1053-1062

[67] Maiorca D. R-PackDroid: API package-based characterization and detection of mobile ransomware. In: *Proceedings of the Symposium on Applied Computing*. Marrakech, Morocco: ACM; 2017. pp. 1718-1723

[68] Mercaldo F. Ransomware steals your phone. Formal methods rescue it. In: *Proceedings of the International Conference on Formal Techniques for Distributed Objects, Components, and Systems*. Heraklion, Greece: Springer; 2016. pp. 212-221

[69] Mercaldo F. Extinguishing ransomware-a hybrid approach to android ransomware detection. In: *Proceedings of the 10th International Symposium on Foundations Practice of Security*. 2017

[70] Yang T. Automated detection and analysis for android ransomware. In: *Proceedings of the High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conference on Embedded Software and Systems (ICSS), 2015 IEEE 17th International Conference on*. IEEE; 2015. pp. 1338-1343

[71] Gharib A, Ghorbani A. DNA-droid: A real-time android ransomware detection framework. In: *Proceedings of the International Conference on Network and System Security*. Helsinki, Finland: Springer; 2017. pp. 184-198

[72] Sarma BP, Li N, Chris Gates C, Potharaju R, Nita-Rotaru C, Molloy I. Android permissions: a perspective combining risks and benefits. In: *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*. Newark, New Jersey, USA; 20-22 June 2012. pp. 13-22

[73] Enck W, Ongtang M, McDaniel P. On lightweight mobile phone application certification. In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. Vol. 2009. pp. 235-245

[74] Gates C, Li N, Peng H, Sarma B, Qi Y, Potharaju R, et al. Generating summary risk scores for mobile applications. *IEEE Transactions on Dependable and Secure Computing*. 2014;11(3):238-251

[75] Mathew J, Joy M. Efficient risk analysis for android applications. In: *Proceedings of the IEEE Recent Advances in Intelligent Computational Systems (RAICS)*; 10-12 December 2015. Trivandrum, India: IEEE; 2015. pp. 382-387

[76] Guo C, Xu G, Liu L, Xu S. Using association statistics to rank risk of android application. In: *Proceedings of the IEEE International Conference on Computer and Communications (ICCC)*; 10-11 October 2015. IEEE; 2015. pp. 1-5

[77] Hao H, Li Z, Yu H. An effective approach to measuring and assessing the Risk of android application. *Proceedings of the International Symposium on Theoretical Aspects of Software Engineering (TASE)*; 12-14 Sept. 2015:31-38

- [78] Wang Y, Zheng Y, Sun C, Mukkamala S. Quantitative security risk assessment of android permissions and applications. In: Proceedings of the Lecture Notes in Computer Science, LNCS-7964. Newark, NJ, USA: Springer; 2013. pp. 226-241
- [79] Yuksel A, Yuksel E, Sertbasa A, Zaim A. Implementation of a web-based service for mobile application risk assessment. Turkish Journal of Electrical Engineering & Computer Sciences. 2017;25(2):976-994
- [80] Merlo A, Georgiu G. RiskInDroid: Machine learning-based risk analysis on android. In: Proceedings of the IFIP International Conference on ICT Systems Security and Privacy Protection (SEC). 2017. pp. 538-552
- [81] Hosseinkhani M, Fong P, Papilio CS. Visualizing android application permissions. In: Proceedings of the Eurographics Conference on Visualization. 2014. pp. 1-10
- [82] Kraus L, Wechsung I, Möller S. Using statistical information to communicate android permission risks to users. Workshop on Socio-Technical Aspects in Security and Trust (STAST). Vienna, Austria: Co-located with 27th IEEE Computer Security Foundations Symposium (CSF); 18 July 2014. pp. 1-9
- [83] Kang J, Kim H, Cheong YG, Huh JH. Visualizing privacy risks of mobile applications through a privacy meter. In: Information Security Practice and Experience. Lecture Notes in Computer Science. Vol. 9065. Beijing, China: Springer; 2015. pp. 548-558
- [84] Eze C, Nurse J, Happa J. Using visualizations to enhance users' understanding of app activities on android devices. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications. 2016:39-57
- [85] Yoo S, Ryu H, Yeon H, Kwon T, Jang Y. Personal visual analytics for android security risk lifelog. In: Proceedings of the 10th International Symposium on Visual Information Communication and Interaction. 2017. pp. 29-36
- [86] Dash S, Suarez-Tangil G, Khan S, Tam K, Ahmadi M, Kinder J, et al. DroidScribe: Classifying android malware based on runtime behavior. In: 2016 IEEE Security and Privacy Workshops (SPW). 2016. DOI: 10.1109/spw.2016.25
- [87] Arp D. Drebin: Effective and explainable detection of android malware in your pocket. In: Proceedings of the 2014 Network and Distributed System Security Symposium. 2014. DOI: 10.14722/ndss.2014.23247
- [88] Hou S. HinDroid. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining—KDD 17. Halifax, NS, Canada: ACM; 2017. DOI: 10.1145/3097983.3098026