electronics

MDPI

Review

Android Malware Family Classification and Analysis: Current Status and Future Directions

Fahad Alswaina and Khaled Elleithy *

Department of Computer Science & Engineering, University of Bridgeport, Bridgeport, CT 06604, USA; falswain@my.bridgeport.edu

* Correspondence: elleithy@bridgeport.edu

Received: 18 April 2020; Accepted: 1 June 2020; Published: 5 June 2020

Abstract: Android receives major attention from security practitioners and researchers due to the influx number of malicious applications. For the past twelve years, Android malicious applications have been grouped into families. In the research community, detecting new malware families is a challenge. As we investigate, most of the literature reviews focus on surveying malware detection. Characterizing the malware families can improve the detection process and understand the malware patterns. For this reason, we conduct a comprehensive survey on the state-of-the-art Android malware familial detection, identification, and categorization techniques. We categorize the literature based on three dimensions: type of analysis, features, and methodologies and techniques. Furthermore, we report the datasets that are commonly used. Finally, we highlight the limitations that we identify in the literature, challenges, and future research directions regarding the Android malware family.

Keywords: android malware family; malicious application; android security; android application; machine learning; classification; smartphone

1. Introduction

Android Operating system has become the dominant mobile OS in the market capturing 86% in 2017, Gartner [1]. Regarding Android malware, and based on McAfee's report, the malware app increased to 22 million in Q3 of 2017 [2]. Symantec also reported that one on every five Android apps is malware [3]. This has put Android application security at risk and encourages researchers to increase efforts on defending users from malicious developers.

As we investigate the scientific databases on Android malware, we found that most of the current detection techniques are focusing on malware detection. Detecting malware in the sense of labeling an application as one of two labels: benign or malware [4–27]. For example, in [26], they study the dangerous permissions in malware. However, the number of malware samples and its variants are rapidly increasing. Although malware detection is essential to anti-virus (AV) software, studying malware families and identifying/categorizing a malware to its family is even more important. Identifying malware families help AV companies and security researchers focusing more on family (group level) than malware (member level). For example, in order to identify a malware family, researchers analyze common static and dynamic characteristics in a large number of malware samples. If malware families are identified, researchers can focus more on widely spread and highly threaten families rather than in individual samples or less risky families. As a result, identifying the risky families can help detection systems identifying more malware by recognizing its associate family and seizing its effect on the users.

In our survey, we focus on reviewing the literature for the past ten years based on what has been published on the scientific databases regarding Android malware families. Our contributions are stated below:

- We conduct a comprehensive survey of the state-of-the-art in Android malware families, which is one of the first surveys in this topic.
- We introduce a novel taxonomy that categorizes all the related work in familial classification in terms of the type of analyses, features, and techniques that has been used. The complete taxonomy is shown in Figure 1 and Table 1.
- We highlight the limitations of the related works as well as future trends.

The rest of this paper is organized as follows: taxonomy and related work is discussed in Section 2. The type of analyses that are applied is presented in Section 3. In Section 4, we discuss the techniques that are implemented. In Section 5, we talk about the features that are used in the literature. Furthermore, we discuss limitations and future directions in Section 6. Finally, conclusions are presented in Section 7 of the paper.

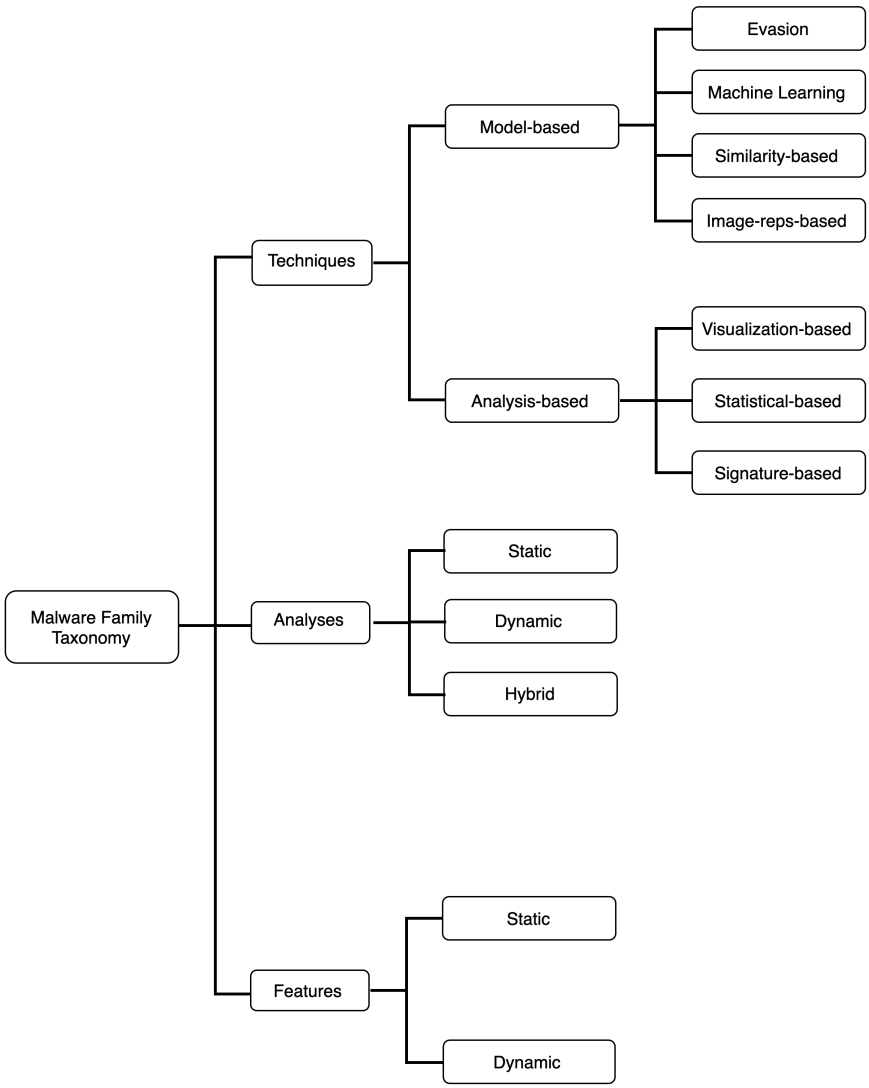

Figure 1. Android malware families taxonomy.

Table 1. Taxonomy table of the literature.

Index	Publication Year	Reference	Analysis	Features	Technique
1	2020	Fang et al. [28]	Static	Static	Image-reps-based
2	2019	Qiu et al. [29]	Static	Static	Similarity-based and Machine Learning
3	2019	Zhang et al. [30]	Static	Static	Signature-based and Machine Learning
4	2019	Zhiwu et al. [31]	Static	Static	Visualization-based and Machine Learning
5	2019	Mirzaei et al. [32]	Static	Static	Visualization-based
6	2019	Vega et al. [33]	Static	Static	Visualization-based
7	2019	Vega et al. [34]	Static	Static	Visualization-based
8	2019	Jiang et al. [35]	Static	Static	Machine Learning
9	2019	Fasano et al. [36]	Static	Static	Machine Learning
10	2019	Blanc et al. [37]	Static	Static	Machine Learning
11	2019	Xie et al. [38]	Static	Static	Statistical-based and Machine Learning
12	2019	Turker et al. [39]	Static	Static	Statistical-based and Machine Learning
13	2018	Atzeni et al. [40]	Hybrid	Dynamic and Static	Signature-based
14	2018	Kim et al. [41]	Hybrid	Dynamic and Static	Visualization-based and Machine Learning
15	2018	Fan et al. [42]	Static	Static	Visualization-based and Machine Learning
16	2018	Sun et al. [43]	Dynamic	Dynamic	Visualization-based
17	2018	Martin et al. [44]	Dynamic	Dynamic	Machine Learning and Statistical-based
18	2018	Aktas et al. [45]	Hybrid	Dynamic and Static	Machine Learning
19	2018	Garcia et al. [46]	Static	Static	Machine Learning
20	2018	Calleja et al. [47]	Static	Static	Evasion and Machine Learning
21	2018	Alswaina et al. [48]	Static	Static	Machine Learning
22	2017	Massarelli et al. [49]	Dynamic	Dynamic	Signature-based and Machine Learning
23	2017	Zhou et al. [50]	Static	Static	Visualization-based and Similarity-based
24	2017	Chakraborty et al. [51]	Hybrid	Dynamic and Static	Machine Learning
25	2017	Sedano et al. [52]	Static	Static	Statistical-based
26	2016	Battista et al. [53]	Static	Static	Signature-based
27	2016	Hsiao et al. [54]	Dynamic	Dynamic	Visualization-based
28	2016	Gonzale et al. [55]	Static	Static	Visualization-based
29	2016	Fan et al. [56]	Static	Static	Visualization-based and Machine Learning
30	2016	Kang et al. [57]	Static	Static	Similarity-based
31	2016	Malik et al. [58]	Dynamic	Dynamic	Statistical-based
32	2016	Sedano et al. [59]	Static	Static	Statistical-based
33	2016	Feng et al. [60]	Hybrid	Dynamic and Static	Visualization-based, Machine Learning, and Signature-base
34	2015	Aresu et al. [61]	Dynamic	Dynamic	Signature-based and Similarity-based
35	2015	Lee et al. [62]	Static	Static	Signature-based and Similarity-based
36	2015	Li et al. [63]	Static	Static	Visualization-based and Machine Learning
37	2015	Garcia et al. [64]	Static	Static	Machine Learning
38	2014	Deshotels et al. [65]	Static	Static	Visualization-based and Similarity-based
39	2014	Suarez et al. [66]	Static	Static	Statistical-based and Machine Learning
40	2013	Kang et al. [67]	Static	Static	Statistical-based and Machine Learning

2. Taxonomy and Related Work

In this section, we discuss the Android operating system, Android malware as well as the related work.

2.1. Android and Malware

In this section, we discuss Android in general as an operating system. We address the main components inside the app and define some technologies and fundamentals. Then, we discuss Android malware and attacks they use to harm the user.

2.1.1. Android Operating System

Android is a Google product that is designed for smartphones and mostly written in Java language. Android uses a Linux kernel to communicate with the hardware. The updated overall architecture of the Android in [68] is shown in Figure 2.

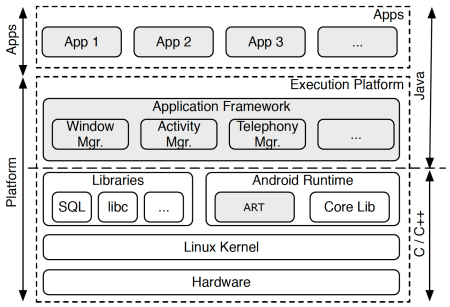

Figure 2. Android architecture.

Android contains four main components that form the building blocks of the app [69]: Activities, Services, Broadcast receiver, and Content providers. *Activity* is a Java class (a single screen) and entry point that the user interacts with. For example, in a contact app is an instance of an activity that shows a list of contacts. *Services* are background processes that process long-running jobs. An example of a service is running some updates for the application. *Broadcast receiver* is a component that responds to system announcements or delivers broadcasts to another or within the same app. An example of this component is when the user notified that the battery is low. Finally, *Content provider* manages data stored in a database, i.e., SQLite, or in the file system and allows other apps to query such data if they have the permissions. An example is a content provider response to a user clicking on a person's contact in a contact app. It is also important to talk about an important message event called Intent. *Intent* is a message object that is used to perform some operations such as starting an activity or a service, or delivering a broadcast message to broadcast receivers. The intent object contains a set of information such as component name, action to be performed, data type, category type, extras, and flag.

Android applications, either system or third-party app, communicate with the Android platform via defined Application Programming Interfaces (APIs). Android framework provides a list of APIs that a developer can call to extend the functionality of the hardware without direct use of lower layers of the architecture. Such functionalities are managing user interface (UI) elements, accessing shared data storage, and passing messages between application components. As in Linux, the Android app is assigned a unique user id (UID) and group id (GID). Each app runs in a separate process to identify and isolate each app's resources from each other. Using UID, Android creates kernel-level application sandbox to enforce kernel security.

Android application is compressed in an archive format file, like any other known formats such as ZIP and JAR, called Android Application Package (APK). APK contains seven files: asset, lib, meta-info, res, androidmanifest.xml, classes.dex, and resources.arsc. In this section, we limit our discussion on two main files: the manifest file (Androidmanifest.xml) and the code file (classes.dex).

Android manifest. The manifest file is an XML format file that provides beforehand a set of information about the app and declaration of the app components. Information such as the app's package name and version number, permissions required by the application, app entry points, and registered intents.

Dalvik executable (DEx). The file Classes.dex contains a set of files (bytecodes). Those files are a special type of bytecode called Dalvik Bytecode that are compiled from normal Java classes. In Figure 3, we show the steps of converting Java classes and the generation of a DEx file [70].

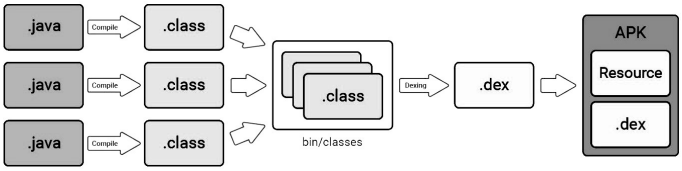

Figure 3. Dexing Java classes into classes.dex.

2.1.2. Android Malware

In this section, we discuss the most recognized type of malware attacks in the literature such as: repackaged, update attack, and drive-by download as listed in [71]. Furthermore, we discuss the way that malicious payload is executed. Finally, we end by talking about malware families and characteristics.

Attack techniques. One of the most common techniques is to piggyback a known app with a malicious payload. This technique is known as *repackaging* as the malicious disassemble an app, insert the malicious code (payload), and repack the app. Examples of such malware families are *ADRD*, *AnserverBot*, and *BgServ*. [71]. An alternative way of the same technique is *update attack*. This is in order to repackage the application when performing updates. A victim installs the modified app, without the payload, to avoid detection. When it is time for the update, a payload will be installed with the new version. Families such as *BaseBridge*, *DroidKungFuUpdate*, and *Plankton* are some examples of families adopting this technique.

Another technique is called *Drive-by download*. In this technique, the victim installs an app that advertises another app that is either standalone or repackaged malware. In addition, instead of advertising, the download request can happen without user notification. This could happen when the user grants certain permissions to the app to download when the user first installs the main application. *GGTracker*, *Jifake*, *Spitmo*, and *ZitMo* are some of the families using this type of attack.

Obfuscation techniques. Some malware encrypts strings in the code. Strings such as method name, class name, or URLs are encrypted via some obfuscators to avoid static analysis detection. The obfuscated strings are hard to reverse engineer and then hard to read. Common Java obfuscators are ProGuard [72] and DexGuard [73], which are widely used.

Activation techniques. This technique associated with Android events. BOOT_COMPLETED event, for example, is triggered when the device finishes the booting process. Malware uses this event to be notified when the device is up and running to activate the malicious process. Other events such as SMS_RECEIVED that is triggered when an SMS is received is utilized by *zSone* family. Another example is a ACTION_MAIN event that is triggered when an app's icon that is clicked is adopted by a *DroidDream* family.

There are many papers that contribute to detection such techniques such as [74–78]. For example, Tian et al. [78] designed a repackaged detection technique. Their technique based on partitioning the code into two levels, class-levels dependency graph (regions), and method-level call graphs. They utilize machine-learning to recognize internal behavior using three types of features: permissions, sensitive API calls, and user interaction.

Malware families. A family of malware is a group of malware that shares common characteristics and behavior. Adopting an attack or malicious behavior by inserting a payload (or more than one payload) requires using the same package names used for the attack. By frequent use of package names (or other common characteristics), this becomes one identity (signature) of a group of malware (family). For example, *AnserverBot* family, a popular malware family, uses *com.sec.android.provider.drm* the package name in the code. Another example is that malware in *DroidKungFu* family contain a package named *com.google.ssearch* [71]. Other common malware families are listed in Table 2 [79].

Table 2. Common Android malware families.

Ackposts	Counterclank	FakeRegSMS	JollyServ	Photsy/Phopsy	SpamBot
Acnetdoor	Crusewind	FakeTaoBao	Jsmshider/Xsider	Pincer	Spitmo
Adsms	Dogowar	FakeTimer	Kidlogger	Pjapps	SPPush
Airpush/StopSMS	Dougalek	FakeUpdate/Apkqug	KMIN	Placms	SpyBubble
AnServer/Answerbot	DroidDeluxe	FakeVertu	Ksapp	Plankton	SpyOO
Antares/Antammi	DroidDream	Find and Call/Fidall	LeNa	Podec	Ssucl
Arspam	DroidDreamLight	Finspy	Lien/	PoisonCake	Steek/Fatakr
AVPass	DroidJack/SandoRAT	Fjcon	Locker/SLocker Ransomware	ProxyTrojan/NotCompatible/NioServ	TapSnake/Droisnake
BackFlash/Crosate	DroidKungfu	Flexispy	Loicdos	Qicsomos	Tascudap
Badaccents	DroidSheep	Fokange/Fokonge	Loozfon	Raden	Tetus
Badnews	DSEncrypt	Foncy	Lovetrap/Luvrtrap	Repane	TGloader/Stiniter
BankBot	Extension/Monad	Fonefee/Feejar	Luckycat	Roidsec/Sinpon	TigerBot
Basebridge	FaceNiff	Gamex	Maistealer	RootSmart/Bmaster	Titan
BeanBot	FakeAngry	Gazon	Malap	RuFraud	Tonclank
Beita	FakeApp.AL	Geinimi	Mania	Saiva	Tracer
BgServ	FakeAV	GGTracker	MMarketPay	Samsapo	TypStu
Biige	FakeBank	GingerBreak	MobiDash	SaveMe/SocialPath	UpdtBot
Binv	FakeDaum/vmwol	GingerMaster/GingerBreaker	MobileSpy/Godwon	Scavir	UpdtKiller
Booster	FakeDefender	Godwon	MobileTx	Scipiex	Uracto
Boxer	FakeDoc	GoldenEagle/GlodEagl	Mobinauten	SeaWeth	USBcleaver
Cajino	FakeFlash	GoneIn60Seconds	Moghava	Selfmite	Uten
Carberp	FakeInst	GPspy	Nandrobox	Skullkey	Uxipp
Cawitt	FakeJobOffer	HeHe	Netisend	Smack	Vdloader
Cellspy	FakeMarket	HideIcon	Nickispy	SMSilence/SMSCatcher	Walkinwat/Pirater
Chulli	FakeMart	HippoSMS	Obad	SMSpacem	Waps/Simhosy
Code4hk/xRAT	FakeNefix	HongTouTou/Adrd	Oldboot/MouaBad	SMSreg	Wroba/HijackRAT
Coogos	FakeNotify	Iconosys	OpFake	SMSsniffer	YZHC
CopyCat	FakePlay	Imlog	PDAspy	SMSspy	
Cosha	FakePlayer	Jifake	Penetho	Sndapps/Snadapps	
ZertSecurity	Zitmo/Citmo	Zsone	ZergRush	Zeahache	

How anti-virus works. Malware signatures, as they have been manually analyzed or detected, are saved in an AV database to be compared against files under scanning. When a match is found in the file, the file (or app) is considered malicious, and it will be quarantined.

2.2. Android Malware Related Work

In this section, we review the survey papers on Android malware. Most of the surveys focus on malware detection, including [80–88]. The most recent survey has reviewed papers on malware detection while focusing on their approaches; they discussed the advantages and disadvantages of each detection approaches and methods [81].

The following survey has proposed a taxonomy to categorize Android malware detection techniques; they highlighted the trends and the challenges [83]. The following two survey papers have provided an outline of the methodologies used in classifying malware based on work surveyed [82,87]. The authors in [84] have focused on the state-of-the-art papers in identifying malware behaviors based on a diverse set of features; they highlighted the effective features in detecting malware. Yan and Yan have surveyed the related work in dynamic malware detection; they focused on the performance evaluation criteria on malware detection [85].

Souri and Hosseini have conducted a systematic survey on the state-of-the-art papers in utilizing data mining techniques in malware detection; they categorize the techniques into signature-based and behavioral-based. Furthermore, they discuss the importance of data mining techniques in malware detection [86]. Riasat et al. have provided a comprehensive survey on the tools and methods used on malware detection; they highlighted the various types of tools used in the research field [88]. Arshad et al. categorize the antimalware and penetration techniques proposed by state-of-the-art research to protect the Android system; they highlighted their limitation and benefits [80].

The previous surveys on malware detection have focused on malware detection. In this survey, our focus is on malware familial classification, detection, and analysis, which will introduce a baseline for future work in this domain.

In order to conduct our review, we followed an exploratory research approach. We looked into more than a thousand papers published in journals and conferences. To filter out the selected papers, we considered keywords. The following respectable scientific databases are explored: IEEE Xplore [89], ACM Digital Library [90], MDPI [91], ScienceDirect [92], Hindawi [93], Springer [94], and arXiv [95], and we also used reputable literature search engines such as Microsoft Academic [96], Semantic Scholar [97], and Google Scholar [98]. Keyword criteria for selecting a literature contain main and optional keywords. Main keywords are Android malware and malware family. Optional keywords are malware detection, familial classification, malware family identification, and malware family categorization. We have classified the related work according to their type of analysis, techniques, and features.

3. Analysis

In this section, we discuss the type of analysis followed by the state-of-the-art. They are static, dynamic, and hybrid analysis.

3.1. Static Analysis

Static analysis is applied while the app is in a static state. It basically collects information about the app such as the app's name, size, permissions, code, and programing pattern. Some of the information requires reverse engineering the app from machine code to a readable format to analyze the code. The advantage of performing such analysis is that it is fastest and cheapest since it doesn't require executing the application nor does it require monitoring activities. A drawback of the analysis is that many malware launch their attack at runtime. In addition, other malware use an obfuscation technique or encrypted methods which cannot be read or decrypted unless the app is

executed. A set of papers [28–39,42,46–48,50,52,53,55–57,59,62,63,65–67] used static analysis. Details on the static features used by the papers were discussed in Section 4, Features.

3.2. Dynamic Analysis

This type of analysis (also known as behavioral analysis) performed during the execution of an app. It monitors the inside and outside action, connections, calls, and clicks that happen while the app is being executed. Such analysis has the advantage of detecting wide-range and sophisticated malware. Malware families that are bound to an event that were mentioned earlier can only be detected while the app is running. The disadvantage of such analysis is that it is time-consuming. In addition, it requires a priori knowledge of the malware technique to monitor. Several papers have applied dynamic analysis such as [43,44,49,54,58,61]. Details on the dynamic features used by the papers were discussed in Section 4, Features.

3.3. Hybrid Analysis

Hybrid analysis is a combination of both static and dynamic analysis. Although hybrid analysis has the advantage of covering both analyses, it has a major drawback. Such analysis is a time-consuming process considering the huge number of malware samples to be detected and analyzed. Papers such as [40,41,45,51,60] have used hybrid analysis and the details on the features used were discussed in Section 4, Features.

4. Techniques

In this section, we discuss the techniques used by the state of the art to address the familial malware problem. There are two main techniques used: model-based and analysis-based.

4.1. Model-Based

In a model-based technique, a model is created to classify malware into families. There are four main categories of techniques used, which are machine learning, similarity analysis and image processing, and evasion.

Machine learning. The literature use machine learning to classify malware samples into families.

In [31], the authors classify the malware using Deep Learning (DL) techniques. In [44], the authors classify malware into families using classical machine learning such as Support Vector Machine (SVM) and DL algorithms such as CNN and RNN. In [66], the authors use a Nearest Neighbor classifier (NN) to classify malware into families. In [35], the authors preprocess the data and extract the sensitive opcode sequence. For the minor families, they use the oversampling technique to overcome this issue. To represent the semantic features of the sensitive opcode sequence, they use text mining (i.e., Doc2Vec algorithm [99]). Finally, they train their model using nine machine learning algorithms such as SVM and Randomforest. In [49], the authors feed the fingerprint to an SVM algorithm to classify malware into families. In [63], the authors construct the feature vector and feed it to several machine learning algorithms such as Randomforest. In [38], the authors used SVM to classify the samples into families. In [67], the authors feed the features to several machine learning classifiers such as Decision Tree and Association rules. In [47], the authors build a framework to train the classifier algorithm with a set of samples to drive the heuristic search using a Genetic algorithm. In [42,56], the authors use frequency graphs (FreGraph) as their features to be fed into several machine learning algorithms such as SVM, Decision Tree, and Randomforest to classify the malware into families. In [37], the authors feed the Android-oriented matrices to several machine learning algorithms such as SVM, KNN, and Decision Tree. In [46], the authors apply machine learning algorithms to extract complex features and used them to classify malware into families. In [39], the authors use three machine learning techniques: standard classifier such as SVM, ensemble classifier, and Neural Network to classify malware into families. In [48], Alswaina et al. use two models to perform familial classification. The authors use the binary representation of the features and weighted importance. Then, they use six

machine learning algorithms to predict malware families. In [45], the authors apply three filters to filter the features. The dynamic and static features are combined and fed to machine learning algorithms, such as Randomforest and KNN for classification. In [29], the authors apply Linear SVM, DT, and DL algorithms. Fene et al. [60] utilize the SVM algorithm.

In [51], the authors use supervised algorithms such as Randomforest. Moreover, the authors use unsupervised learning such as K-means and mean-shift due to unbalanced samples in each family. They also propose ensemble clustering and classification techniques, which integrate the results generated from the supervised and unsupervised algorithms. In [41], the authors optimize the weight of features using community detection algorithms. They further classify the malware into families using machine learning. In [30], the authors use the fingerprints to classify malware into families using online passive-aggressive (PA) classifiers. Further details of PA can be found in [100]. In [36], the authors extract features from the apps and create code metrics. Then, they binary classify (coarse-grain) the samples. The malware is further classified into families (fine-grain).

Evading detection. In this technique, the goal is to evade detection or elude classifiers into misclassification. In [47], the authors build a framework to alter the malware to perform an attack and misclassify the results.

Similarity analysis. Literature computes the distance between any malware and the family.

In [61], the authors use the token-subsequence algorithm to extract and generate signatures from each family based on network traffic analysis. In [57], the authors represent opcode as a vector of binary and frequency to compute the similarity between the malware and families. In [50], the authors evaluate their approach by performing similarity analysis. In [65], the authors perform two tests. The first test is used to binary classify malware. In the second test, they apply the agglomerative clustering algorithm to cluster the apps into families. To evaluate their model, they compute the distance between the malware and the clusters' centroids to validate which family the sample belongs to. In [62], the authors cluster the families based on the most frequent key terms used by each family. Then, they use the dictionary search method for classification. In [29], the authors use TF-IDF to represent the frequency of the features.

Image representation. Some literature classifies the malware to malware families based on image representation. In [28], the authors convert the DEX file into an image and plain text. Then, they extract the color and the texture feature from the image. For the three features: color, texture, and text, they feed them into the feature Fusion algorithm to classify malware into families.

4.2. Analysis-Based

In the analysis-based technique, an analysis is carried to analyze and construct features to observe families' characteristics. There are three sub techniques under this approach, which are signature-based, statistical analysis, and visualization analysis.

Signature-based. They construct a signature for each family to identify the families. In [61], the authors use a multi-step clustering approach: First, they apply coarse-grained clustering and then apply fine-grained clustering. In [30], the authors construct the fingerprint of the malware families using n-grams analysis and features hashing. In [49], the authors generate a fingerprint for each family. In [62], the authors construct a signature of each malware family based on the collected features. Feng et al. [60] propose an approximate signature matching algorithm to generate signature for malware families.

Statistical analysis. They applied statistical tools to analyze and identify the family's characteristics and the important features. In [66], the authors use statistical analysis and text mining to extract the features. In [44], the authors use Markov chain to represent the features. In [38], the authors eliminate unimportant features using the frequency-based approach. In [67], the authors compute the bytecode frequency. In [39], the authors apply a feature ranking algorithm to identify the most important features.

Visualization analysis. They visualize the characteristics of families using graph mining and PCA. In [31], the authors extract DFG and CFG. Then, they encode the graphs into a matrix. In [41], the authors represent the features using a network graph. In [50], the authors collect the sensitive API calls and then construct graphs based on sensitive API calls. Then, they characterize malware families based on the subgraph isomorphism. In [63], the authors construct a short and long APIs dependency path to perform context and constant analysis. In [65], the authors disassemble the app into Smali files. Then, they create class dependency graph (CDG) to group the classes into modules to identify which module contains malicious code. In [42,56], the authors use community detection, subgraph matching, and subgraph clustering to generate the FreGraph. Feng et al. [60] utilize an inter-component call graph (ICCG) to represent the communication in the app to construct the features.

5. Features

In this section, we discuss the types of features used by works of literature to classify malware into families. They are classified into static and dynamic features.

5.1. Static Features

Static features are any features that can be recognized or utilized without the execution of the application. Some examples of static features are package name, application size, permissions, and list of APIs.

A set of papers [33,34,48,52,55] uses features that are related to malware installation such as repackage and update, payload activation such as on booting and receiving calls, and privilege escalation attack such as *asroot* and *exploid* families [71]. Moreover, in [33,34,52], they include other features related to financial charges such as SMS and phone calls. Vega et al. in [33,34], also include features related to personal information stealing such as phone number.

In [29,32,35,38,39,42,46,47,50,56,59,60,63–65], a subset of sensitive or suspicious API calls are utilized in their feature set. Permissions used in the app are included as features in [29,38,39,48,59]. Moreover, in [35], sensitive opcode sequence, actions, and strings are utilized in their features. Garcia et al. [46,64] added native code-based to their set of features.

Fasano et al. [36] and Blanc et al. [37] use a set of metrics generated from Smali files to measure the quality code of the app to be used as features. However, in [35,53,57,62,67], code-based analysis such as Java bytecode, bytecode frequency, opcode, or opcode sequence are used as features.

Other papers such as [31,66] use data-flow graph (DFG) and control-flow graph (CFG) as features. In [28], the authors extract the texture, color, and text features from the DEX file. Zhang et al. [30] use features extracted from DEX as n-gram and hash code.

Finally, some works of literature have applied a set of static features in addition to dynamic features. In [51], the authors use 190 static features such as permissions. In [45], the authors use static features such as the number of services and receivers. In [41], the static features such as permissions, filename, and activity name are utilized. In the paper [40], a set of static features from the Android manifest in addition to an APK file that is generated from Androguard [101] tool, a Python code to reverse engineer Android files.

5.2. Dynamic Features

Features that require execution of the application are considered dynamic. For example, network traffic, send/receive SMS, resource consumption, system logs, and I\O operations.

In [58], the author traced the system calls during the execution of the application. Aresu et al. [61] utilize network traffic (HTTP) in their classification. Martin et al. [44] depend on the features that are generated by a DroidBox [102] tool, an Android sandbox for dynamic analysis, which is represented as operations and function of time. In [54], the authors record the API calls that are performed during application execution. In [49], resources' consumption is utilized as features for their classification. In [43], the authors use sensitive and permission-related API calls.

Finally, a group of literature works has applied a set of dynamic features in addition to static features. In [51], the authors use around 2048 dynamic features logs such as file I/O, network usages, and cryptographic usage. In [45], the authors use dynamic features that are generated using a DroidBox tool [102] such as the number of open/closed connections and the number of sent/received network packets. In [41], the dynamic features such as API call sequence are utilized. In [40], a set of dynamic features uses DroidBox [102] and CuckooDroid [103]. Feng et al. [60] use suspicious API call behaviors such as sendSMS API and data leakage.

6. Discussion

In this section, we highlight the datasets that have been used, the limitation of literature, the general challenges related to malware families, and we also report future directions.

6.1. Experimental Datasets

There are many datasets used in the literature that contain a collection of Android malware grouped into families such as: Android Malware Genome Project (Malgenome) [71], Drebin [104], the AMD [105] Project, and AndroZoo [106]. Some papers collected the malware samples from the Android market such as Anzhi, or a repository such as VirusTotal [107] and VirusShare [108].

The datasets differ in the number of samples and number of families. For example, AMD [105] contains 4354 malware samples grouped in 42 families. While Drebin [104] has 5560 samples grouped in 179 families, other datasets such as AndroZoo [106] contain many more samples and families, where the number of samples is 10.7 million grouped into more than 3000 families.

In Figure 4, we show the number of publications that uses each dataset found in the literature. Furthermore, Table 3 shows detailed information where the publications are included. The repository category includes sites like VirusTotal, VirusShare, and Koodous, for which there is no fixed set to be used as benchmarks. Collection category refers to either an unknown collection performed by the author or sites such as HelDroid, FalDroid, and the Anzhi app market. As we see from the figure, the most used datasets are Drebin [104] and Genome [71]. More details on the commonly used datasets are reported in Table 4.

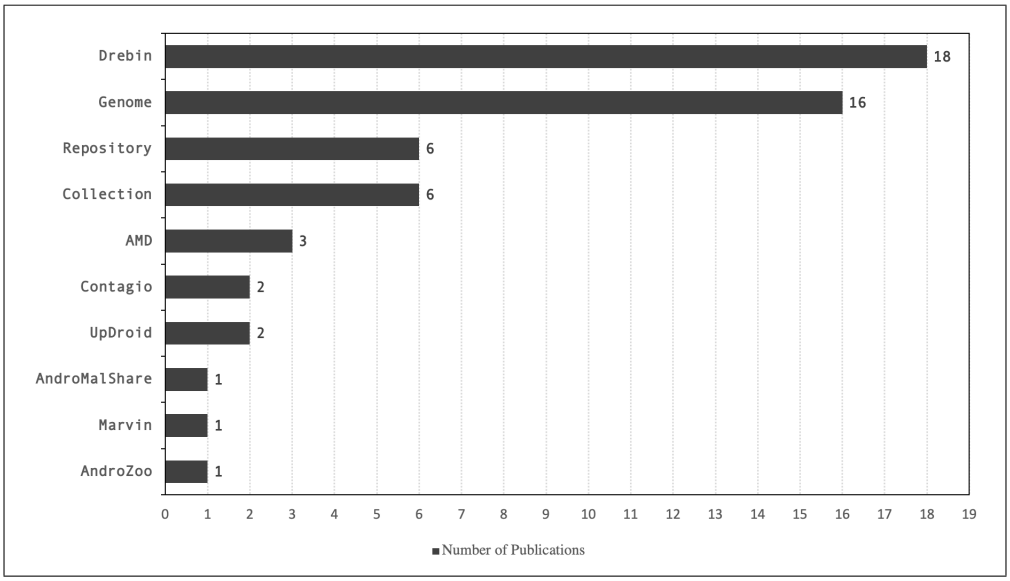

Figure 4. Dataset Reported by Publications, where the *x*-axis represents the number of publications and the *y*-axis represents the datasets.

Table 3. Dataset, number of publications and publication details

Dataset	Number of Publications	Publications
Drebin	18	[29–31,35–37,39,42–44,46,47,49,51,53,59–61]
Genome	16	[30,33,34,46,48,50,52,54,55,57,60,61,63,65–67]
Collection	6	[36,38,41,42,58,62]
Repository	6	[31,38,40,46,51,56]
AMD	3	[28,29,39]
UpDroid	2	[39,45]
Contagio	2	[31,61]
AndroZoo	1	[32]
Marvin	1	[31]
AndroMalShare	1	[50]

Table 4. Commonly used Datasets and their details.

Dataset	No. of Samples	No. of Families
AMD [105]	4354	42
Drebin [104]	5560	179
Malgenome [71]	1260	49
AndroZoo [106]	10.7M	3K+

6.2. Limitations

As we surveyed forty research papers, we summarize the limitations to the following: First, most of the literature uses small datasets such as a a few numbers of families or a few malware samples for studying families. Moreover, they use outdated or discontinues datasets such as Contagiodump [109] and Malgenome Project [71]. In addition, several papers build their experiments on manually collected data without testing their model on benchmarked data. Several papers lack the disclosure of the list of features applied to reproduce the work.

6.3. Challenges

Family naming. One of the challenges that we observe is that there are no naming schemes (conventions) for the malware family. Naming a family is varied depending on the AV company. Families such as *BaseBridge* (or *adSMS*), *Smssend* (or *fakeplayer*), and *DroidDream* (or *DORDRAE*) are some of the families that have multiple names. One of the reasons is that one company names a family based on different share characteristics than other companies. Characteristics such as installation methods, activation, or the name of their malicious file name are discussed in [71,110,111]. Attempts have been made by [106,112,113] to establish naming standards. Sebastian et al. [114] address the issue of inconsistent labeling (naming) of malware family and contribute the AVclass tool, an auto-labeling, as an effort to unify labeling. In addition, Euphony is a system proposed by [106] to unify different AV companies.

Imbalance Dataset. Some of the malware families contain hundreds of samples, while others contain as little as one sample i.e., a DroidKungFuUpdate family in the Malgenome dataset [71]. The whole list is shown in Table 5. This cause identifies the characteristics of a family as challenging. In case of standalone malware (not repackaged), the identification is almost impossible.

Table 5. Malgenome dataset families with one sample.

Family
SMSReplicator
Walkinwat
Endofday
GGTracker
GamblerSMS
Lovetrap
Zitmo
CoinPirate
DogWars
NickyBot
DroidCoupon
DroidDeluxe
Spitmo
DroidKungFuUpdate
FakeNetflix
Jifake

6.4. Future Directions

Advanced machine learning. Malware families should be deeply analyzed and identified. Deep learning technology has been adapted to address various research problems including voice recognition, image processing, and text analysis. One of the advanced techniques of Deep Learning is reinforcement learning, which can be utilized to better understand the families' characteristics. Reinforcement learning has shown very promising results, especially in dynamic analysis. Another technique that should be adopted is transferred learning, which can be utilized to address the lack of samples in families.

Big data handling. Since the amount of malware is increasing exponentially, as it was reported by GDATA that almost 9K of new malware programs are reported daily [115], a scalable solution should be considered. For example, the AndroZoo [116] dataset has millions of samples that can be handled using big data technologies. One of the most important tools are Hadoop [117] and Spark [118]. They can handle a huge amount of malware data with fast processing.

Crowdsourcing. Beside Big data technologies, a group of malware family analyzers can be utilized to better identify and characterize the families. For example, a source can use a subset of features, while other sources investigate other feature sets. A malware repository VirusTotal [107] and VirusShare [108] are some examples.

Automated detection. The huge number of generated malware necessitate the call for automated analysis and classification of malware family rather than performing such tasks manually [41,119].

7. Conclusions

Malware family detection and analysis have been a problem for many years. With the escalation in the amount of malware, especially on Android devices, researchers have studied malware deeply using various tools, such as machine learning, graph mining, image processing, and statistical analysis. Most of the literature is focused on detecting malware rather than detecting families. Detecting malware families can help us to better understand the characteristics of the malware family.

In this paper, we surveyed a total of forty research papers on Android malware familial detection, classification, and categorization from various scientific databases. We classified the literature according to their type of analysis, type of features, and the techniques applied. We further report the datasets that have been used and include details about each of them. Moreover, we discussed the limitations of the literature approaches, challenges faced by the researchers, and future trends for the research community.

Our findings show that most of the limitations circulate around the availability and the size of benchmarked datasets. In addition, some common challenges are the lack of samples and standardization of family naming. As for the future directions, investment in advanced artificial intelligence techniques such as machine learning and big data technologies should be considered. Moreover, crowdsourcing and automated detection should be utilized to better address malware family identification problems.

Author Contributions: The work has been primarily conducted by F.A. under the supervision of K.E. Extensive discussions about the algorithms and techniques presented in this paper took place among the authors over the past year. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: The authors acknowledge the University of Bridgeport for providing the necessary resources to carry this research conducted under the supervision of Khaled Elleithy.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ML	Machine Learning
DL	Deep Learning
SVM	Support Vector Machine
CNN, RNN	Convolutional, Recurrent Neural Networks
NN	Nearest Neighbor or Neural Network
App	Application
Malware	Malicious Application
DT	Decision Tree
DFG	Data-Flow Graph
CFG	Control-Flow Graph
CDG	Class Dependency Graph
APK	Android Application Package
DEX	Dalvik Executable
PA	Passive-Aggressive Algorithm
AV	Anti-Virus
UI	User Interface
UID, GID	User, Group ID.
API	Application Programming Interface
FreGraph	Frequency Graph

References

1. Gartner Says Worldwide Sales of Smartphones Recorded First, Ever Decline During the Fourth Quarter of 2017. Available online: https://www.gartner.com/newsroom/id/3859963 (accessed on 1 April 2018).
2. McAfee Mobile Threat Report Q1. 2018. Available online: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2018.pdf (accessed on 1 March 2020).
3. Internet Security Threat Report Volume 20. Available online: https://www.symantec.com/content/en/us/enterprise/other_resources/21347933_GA_RPT-internet-security-threat-report-volume-20-2015.pdf (accessed on 22 August 2017).

4. Grace, M.; Zhou, Y.; Zhang, Q.; Zou, S.; Jiang, X. Riskranker: Scalable and accurate zero-day android malware detection. In Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, Low Wood Bay, Lake District, UK, 25–29 June 2012; pp. 281–294.
5. Wu, D.J.; Mao, C.H.; Wei, T.E.; Lee, H.M.; Wu, K.P. Droidmat: Android malware detection through manifest and api calls tracing. In Proceedings of the IEEE 2012 7th Asia Joint Conference on Information Security, Tokyo, Japan, 9–10 August 2012; pp. 62–69.
6. Sahs, J.; Khan, L. A machine learning approach to android malware detection. In Proceedings of the IEEE 2012 European Intelligence and Security Informatics Conference, Odense, Denmark, 22–24 August 2012; pp. 141–147.
7. Isohara, T.; Takemori, K.; Kubota, A. Kernel-based behavior analysis for android malware detection. In Proceedings of the IEEE 2011 7th International Conference on Computational Intelligence and Security, Sanya, Hainan, China, 3–4 December 2011; pp. 1011–1015.
8. Zarni Aung, W.Z. Permission-based android malware detection. *Int. J. Sci. Technol. Res.* **2013**, *2*, 228–234.
9. Yuan, Z.; Lu, Y.; Wang, Z.; Xue, Y. Droid-sec: Deep learning in android malware detection. In Proceedings of the 2014 ACM Conference on SIGCOMM, Chicago, IL, USA, 17–22 August 2014; pp. 371–372.
10. Yerima, S.Y.; Sezer, S.; McWilliams, G.; Muttik, I. A new android malware detection approach using bayesian classification. In Proceedings of the 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), Barcelona, Spain, 25–28 March 2013; pp. 121–128.
11. Aafer, Y.; Du, W.; Yin, H. Droidapiminer: Mining api-level features for robust malware detection in android. In Proceedings of the International Conference on Security and Privacy in Communication Systems, Sydney, Australia, 25–28 September 2013; Springer: Berlin, Germany, 2013; pp. 86–103.
12. Peiravian, N.; Zhu, X. Machine learning for android malware detection using permission and api calls. In Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, 4–6 November 2013; pp. 300–305.
13. Gascon, H.; Yamaguchi, F.; Arp, D.; Rieck, K. Structural detection of android malware using embedded call graphs. In Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security, Berlin, Germany, 4 November 2013; pp. 45–54.
14. Yuan, Z.; Lu, Y.; Xue, Y. Droiddetector: Android malware characterization and detection using deep learning. *Tsinghua Sci. Technol.* **2016**, *21*, 114–123. [CrossRef]
15. Saracino, A.; Sgandurra, D.; Dini, G.; Martinelli, F. Madam: Effective and efficient behavior-based android malware detection and prevention. *IEEE Trans. Depend. Secure Comput.* **2016**, *15*, 83–97. [CrossRef]
16. Yerima, S.Y.; Sezer, S.; Muttik, I. High accuracy android malware detection using ensemble learning. *IET Inf. Sec.* **2015**, *9*, 313–320. [CrossRef]
17. Millar, S.; McLaughlin, N.; Martinez del Rincon, J.; Miller, P.; Zhao, Z. DANdroid: A Multi-View Discriminative Adversarial Network for Obfuscated Android Malware Detection. In Proceedings of the 10th ACM Conference on Data and Application Security and Privacy, New Orleans, LA, USA, 16–18 March 2020; pp. 353–364. [CrossRef]
18. Han, Q.; Subrahmanian, V.; Xiong, Y. Android Malware Detection via (Somewhat) Robust Irreversible Feature Transformations. *IEEE Trans. Inf. For. Secur.* **2020**. [CrossRef]
19. Ma, Z.; Ge, H.; Wang, Z.; Liu, Y.; Liu, X. Droidetec: Android Malware Detection and Malicious Code Localization through Deep Learning. *arXiv* **2020**, arXiv:2002.03594.
20. Mantoo, B.A.; Khurana, S.S. Static, Dynamic and Intrinsic Features Based Android Malware Detection Using Machine Learning. In Proceedings of the ICRIC 2019, Jammu, India, 8–9 March 2019; Springer: Berlin, Germany, 2020; pp. 31–45.
21. Amin, M.; Tanveer, T.A.; Tehseen, M.; Khan, M.; Khan, F.A.; Anwar, S. Static malware detection and attribution in android byte-code through an end-to-end deep system. *Future Gen. Comput. Syst.* **2020**, *102*, 112–126. [CrossRef]
22. Rana, M.S.; Sung, A.H. Evaluation of Advanced Ensemble Learning Techniques for Android Malware Detection. *Vietnam J. Comput. Sci.* **2020**, 1–15. [CrossRef]
23. Alazab, M.; Alazab, M.; Shalaginov, A.; Mesleh, A.; Awajan, A. Intelligent mobile malware detection using permission requests and api calls. *Future Gen. Comput. Syst.* **2020**, *107*, 509–521. [CrossRef]

24. Chen, X.; Li, C.; Wang, D.; Wen, S.; Zhang, J.; Nepal, S.; Xiang, Y.; Ren, K. Android HIV: A study of repackaging malware for evading machine-learning detection. *IEEE Trans. Inf. Forensics Secur.* **2019**, *15*, 987–1001. [CrossRef]
25. Zhang, M.; Duan, Y.; Yin, H.; Zhao, Z. Semantics-aware android malware classification using weighted contextual api dependency graphs. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, 3–7 November 2014; pp. 1105–1116.
26. Arora, A.; Peddoju, S.K.; Conti, M. PermPair: Android Malware Detection using Permission Pairs. *IEEE Trans. Inf. For. Secur.* **2019**. [CrossRef]
27. Jordaney, R.; Sharad, K.; Dash, S.K.; Wang, Z.; Papini, D.; Nouretdinov, I.; Cavallaro, L. Transcend: Detecting concept drift in malware classification models. In Proceedings of the 26th {USENIX} Security Symposium ({USENIX} Security 17), Vancouver, BC, Canada, 16–18 August 2017; pp. 625–642.
28. Fang, Y.; Gao, Y.; Jing, F.; Zhang, L. Android Malware Familial Classification Based on DEX File Section Features. *IEEE Access* **2020**, *8*, 10614–10627. [CrossRef]
29. Qiu, J.; Zhang, J.; Luo, W.; Pan, L.; Nepal, S.; Wang, Y.; Xiang, Y. A3CM: Automatic Capability Annotation for Android Malware. *IEEE Access* **2019**, *7*, 147156–147168. [CrossRef]
30. Zhang, L.; Thing, V.L.; Cheng, Y. A scalable and extensible framework for android malware detection and family attribution. *Comput. Secur.* **2019**, *80*, 120–133. [CrossRef]
31. Zhiwu, X.; Ren, K.; Song, F. Android Malware Family Classification and Characterization Using CFG and DFG. In *2019 International Symposium on Theoretical Aspects of Software Engineering (TASE)*; IEEE: Piscataway, NJ, USA, 2019; pp. 49–56.
32. Mirzaei, O.; Suarez-Tangil, G.; De Fuentes, J.M.; Tapiador, J.; Stringhini, G. Andrensemble: Leveraging api ensembles to characterize android malware families. In Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, Auckland, New Zealand, 7–12 July 2019; pp. 307–314.
33. Vega Vega, R.; Quintián, H.; Calvo-Rolle, J.L.; Herrero, Á.; Corchado, E. Gaining deep knowledge of Android malware families through dimensionality reduction techniques. *Logic J. IGPL* **2019**, *27*, 160–176. [CrossRef]
34. Vega Vega, R.; Quintián, H.; Cambra, C.; Basurto, N.; Herrero, Á.; Calvo-Rolle, J.L. Delving into android malware families with a novel neural projection method. *Complexity* **2019**, *2019*. [CrossRef]
35. Jiang, J.; Li, S.; Yu, M.; Li, G.; Liu, C.; Chen, K.; Liu, H.; Huang, W. Android Malware Family Classification Based on Sensitive Opcode Sequence. In *2019 IEEE Symposium on Computers and Communications (ISCC)*; IEEE: Piscataway, NJ, USA, 2019; pp. 1–7.
36. Fasano, F.; Martinelli, F.; Mercaldo, F.; Santone, A. Cascade Learning for Mobile Malware Families Detection through Quality and Android Metrics. In Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 14–19 July 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–10.
37. Blanc, W.; Hashem, L.G.; Elish, K.O.; Almohri, M.H. Identifying Android Malware Families Using Android-Oriented Metrics. In Proceedings of the 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 9–12 December 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 4708–4713.
38. Xie, N.; Wang, X.; Wang, W.; Liu, J. Fingerprinting Android malware families. *Front. Comput. Sci.* **2019**, *13*, 637–646. [CrossRef]
39. Türker, S.; Can, A.B. AndMFC: Android Malware Family Classification Framework. In *2019 IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops)*; IEEE: Piscataway, NJ, USA, 2019; pp. 1–6.
40. Atzeni, A.; Diaz, F.; Marcelli, A.; Sánchez, A.; Squillero, G.; Tonda, A. Countering android malware: A scalable semi-supervised approach for family-signature generation. *IEEE Access* **2018**, *6*, 59540–59556. [CrossRef]
41. Kim, H.M.; Song, H.M.; Seo, J.W.; Kim, H.K. Andro-simnet: Android malware family classification using social network analysis. In Proceedings of the 2018 16th Annual Conference on Privacy, Security and Trust (PST), Belfast, UK, 28–30 August 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–8.
42. Fan, M.; Liu, J.; Luo, X.; Chen, K.; Tian, Z.; Zheng, Q.; Liu, T. Android malware familial classification and representative sample selection via frequent subgraph analysis. *IEEE Trans. Inf. Forensics Secur.* **2018**, *13*, 1890–1905. [CrossRef]
43. Sun, Y.S.; Chen, C.C.; Hsiao, S.W.; Chen, M.C. ANTSdroid: Automatic malware family behaviour generation and analysis for Android apps. In Proceedings of the Australasian Conference on Information Security and Privacy, Wollongong, Australia, 11–13 July 2018; Springer: Berlin, Germany, 2018; pp. 796–804.

44. Martín, A.; Rodríguez-Fernández, V.; Camacho, D. CANDYMAN: Classifying Android malware families by modelling dynamic traces with Markov chains. *Eng. Appl. Artif. Intell.* **2018**, *74*, 121–133. [CrossRef]
45. Aktas, K.; Sen, S. Updroid: Updated android malware and its familial classification. In *Nordic Conference on Secure IT Systems*; Springer: Berlin, Germany, 2018; pp. 352–368.
46. Garcia, J.; Hammad, M.; Malek, S. Lightweight, obfuscation-resilient detection and family identification of android malware. *ACM Trans. Softw. Eng. Method. (TOSEM)* **2018**, *26*, 1–29. [CrossRef]
47. Calleja, A.; Martín, A.; Menéndez, H.D.; Tapiador, J.; Clark, D. Picking on the family: Disrupting android malware triage by forcing misclassification. *Exp. Syst. Appl.* **2018**, *95*, 113–126. [CrossRef]
48. Alswaina, F.; Elleithy, K. Android malware permission-based multi-class classification using extremely randomized trees. *IEEE Access* **2018**, *6*, 76217–76227. [CrossRef]
49. Massarelli, L.; Aniello, L.; Ciccotelli, C.; Querzoni, L.; Ucci, D.; Baldoni, R. Android malware family classification based on resource consumption over time. In Proceedings of the 2017 12th International Conference on Malicious and Unwanted Software (MALWARE), Fajardo, Puerto Rico, 11–14 October 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 31–38.
50. Zhou, H.; Zhang, W.; Wei, F.; Chen, Y. Analysis of Android malware family characteristic based on isomorphism of sensitive API call graph. In Proceedings of the 2017 IEEE Second International Conference on Data Science in Cyberspace (DSC), Shenzhen, China, 26–29 June 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 319–327.
51. Chakraborty, T.; Pierazzi, F.; Subrahmanian, V. Ec2: Ensemble clustering and classification for predicting android malware families. *IEEE Trans. Depend. Secure Comput.* **2017**. [CrossRef]
52. Sedano, J.; González, S.; Chira, C.; Herrero, Á.; Corchado, E.; Villar, J.R. Key features for the characterization of Android malware families. *Logic J. IGPL* **2017**, *25*, 54–66. [CrossRef]
53. Battista, P.; Mercaldo, F.; Nardone, V.; Santone, A.; Visaggio, C.A. Identification of Android Malware Families with Model Checking. In Proceedings of the 2nd International Conference on Information Systems Security and Privacy - ICISSP 2016, Rome, Italy, 19–21 February 2016; pp. 542–547. [CrossRef]
54. Hsiao, S.W.; Sun, Y.S.; Chen, M.C. Behavior grouping of Android malware family. In Proceedings of the 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, Malaysia, 22–27 May 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 1–6.
55. González, A.; Herrero, Á.; Corchado, E. Neural visualization of android malware families. In Proceedings of the International Joint Conference SOCO'16-CISIS'16-ICEUTE'16, San Sebastián, Spain, 19–21 October 2016; Springer: Berlin, Germany, 2016; pp. 574–583.
56. Fan, M.; Liu, J.; Luo, X.; Chen, K.; Chen, T.; Tian, Z.; Zhang, X.; Zheng, Q.; Liu, T. Frequent subgraph based familial classification of android malware. In Proceedings of the 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), Ottawa, ON, Canada, 23–27 October 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 24–35.
57. Kang, B.; Yerima, S.Y.; McLaughlin, K.; Sezer, S. N-opcode analysis for android malware classification and categorization. In Proceedings of the 2016 International Conference On Cyber Security In addition, Protection Of Digital Services (Cyber Security), London, UK, 13–14 June 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 1–7.
58. Malik, S.; Khatter, K. System call analysis of android malware families. *Ind. J. Sci. Technol.* **2016**, *9*. [CrossRef]
59. Sedano, J.; Chira, C.; González, S.; Herrero, Á.; Corchado, E.; Villar, J.R. Characterization of android malware families by a reduced set of static features. In Proceedings of the International Joint Conference SOCO'16-CISIS'16-ICEUTE'16, San Sebastián, Spain, 19–21 October 2016; Springer: Berlin, Germany, 2016; pp. 607–617.
60. Feng, Y.; Bastani, O.; Martins, R.; Dillig, I.; Anand, S. Automated synthesis of semantic malware signatures using maximum satisfiability. *arXiv* **2016**, arXiv:1608.06254.
61. Aresu, M.; Ariu, D.; Ahmadi, M.; Maiorca, D.; Giacinto, G. Clustering android malware families by http traffic. In Proceedings of the 2015 10th International Conference on Malicious and Unwanted Software (MALWARE), Fajardo, Puerto Rico, 20–22 October 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 128–135.
62. Lee, J.; Lee, S.; Lee, H. Screening smartphone applications using malware family signatures. *Comput. Secur.* **2015**, *52*, 234–249. [CrossRef]

63. Li, Y.; Shen, T.; Sun, X.; Pan, X.; Mao, B. Detection, classification and characterization of android malware using api data dependency. In Proceedings of the International Conference on Security and Privacy in Communication Systems, Dallas, TX, USA, 26–29 October 2015; Springer: Berlin, Germany, 2015; pp. 23–40.
64. Garcia, J.; Hammad, M.; Pedrood, B.; Bagheri-Khaligh, A.; Malek, S. Obfuscation-resilient, efficient, and accurate detection and family identification of android malware. *Dep. Comput. Sci. George Mason Univers. Tech. Rep.* **2015**, *202*. Available online: https://cs.gmu.edu/media/techreports/GMU-CS-TR-2015-10.pdf (accessed on 1 April 2020).
65. Deshotels, L.; Notani, V.; Lakhotia, A. Droidlegacy: Automated familial classification of android malware. In Proceedings of the ACM SIGPLAN on Program Protection and Reverse Engineering Workshop 2014, San Diego, CA, USA, 22–24 January 2014; pp. 1–12.
66. Suarez-Tangil, G.; Tapiador, J.E.; Peris-Lopez, P.; Blasco, J. Dendroid: A text mining approach to analyzing and classifying code structures in android malware families. *Exp. Syst. Appl.* **2014**, *41*, 1104–1117. [CrossRef]
67. Kang, B.; Kang, B.; Kim, J.; Im, E.G. Android malware classification method: Dalvik bytecode frequency analysis. In Proceedings of the 2013 Research in Adaptive and Convergent Systems, Montreal, QC, Canada, 1–4 October 2013; pp. 349–350. [CrossRef]
68. Shiraishi, S. SDK-Based Quality Assurance Framework for Third Party Apps of IVI Systems. In Proceedings of the First International Workshop on Software Development Lifecycle for Mobile (DeMobile13), Saint Petersburg, Russia, 19 August 2013.
69. Android Developers. Available online: https://developer.android.com/guide/components/fundamentals. (accessed on 1 April 2020).
70. Dangizyan, A. [AAR to DEX] Loading and Running Code at Runtime in Android Application. 2019. Available online: https://medium.com/@artyomdangizyan/aar-to-dex-loading-and-running-code-at-runtime-in-android-application-69089a30c715 (accessed on 1 April 2020).
71. Zhou, Y.; Jiang, X. Dissecting android malware: Characterization and evolution. In *2012 IEEE Symposium on Security and Privacy*; IEEE: Piscataway, NJ, USA, 2012, pp. 95–109.
72. ProGuard. 2020. Available online: https://www.guardsquare.com/en/products/proguard (accessed on 1 April 2020).
73. DexGuard. 2020. Available online: https://www.guardsquare.com/en/products/dexguard (accessed on 1 April 2020).
74. Zhou, W.; Zhou, Y.; Jiang, X.; Ning, P. Detecting repackaged smartphone applications in third-party android marketplaces. In Proceedings of the Second ACM Conference on Data and Application Security and Privacy, San Antonio, TX, USA, 7–9 February 2012. [CrossRef]
75. Hu, W.; Tao, J.; Ma, X.; Zhou, W.; Zhao, S.; Han, T. MIGDroid: Detecting APP-Repackaging Android malware via method invocation graph. In Proceedings of the 2014 23rd International Conference on Computer Communication and Networks (ICCCN), Shanghai, China, 4–7 August 2014; pp. 1–7. [CrossRef]
76. Lin, Y.D.; Lai, Y.C.; Chen, C.H.; Tsai, H.C. Identifying android malicious repackaged applications by thread-grained system call sequences. *Comput. Secur.* **2013**, *39*, 340–350. [CrossRef]
77. Shao, Y.; Luo, X.; Qian, C.; Zhu, P.; Zhang, L. Towards a scalable resource-driven approach for detecting repackaged Android applications. In Proceedings of the 30th Annual Computer Security Applications Conference, New Orleans, LA, USA, 8–12 December 2014.
78. Tian, K.; Yao, D.; Ryder, B.G.; Tan, G.; Peng, G. Detection of Repackaged Android Malware with Code-Heterogeneity Features. *IEEE Trans. Depend. Secur. Comput.* **2020**, *17*, 64–77. [CrossRef]
79. Sullivan; Roberts, L.; Marvin; Cj; Malviya, V.; Indra; Saarj; Dickson, J. Current Android Malware. 2018. Available online: https://forensics.spreitzenbarth.de/android-malware/ (accessed on 1 April 2020).
80. Arshad, S.; Shah, M.A.; Khan, A.; Ahmed, M. Android Malware Detection & Protection: A Survey. *Int. J. Adv. Comput. Sci. Appl.* **2016,** *7*, 463–475.
81. Aslan, Ö.; Samet, R. A Comprehensive Review on Malware Detection Approaches. *IEEE Access* **2020**, *8*, 6249–6271. [CrossRef]
82. Gandotra, E.; Bansal, D.; Sofat, S. Malware Analysis and Classification: A Survey. *J. Inf. Secur.* **2014**, *5*, 56–64. [CrossRef]
83. Wu, T.; Deng, X.; Yan, J.; Zhang, J. Analyses for specific defects in android applications: A survey. *Front. Comput. Sci.* **2019**, 1–18. [CrossRef]

84. Wang, W.; Zhao, M.; Gao, Z.; Xu, G.; Xian, H.; Li, Y.; Zhang, X. Constructing Features for Detecting Android Malicious Applications: Issues, Taxonomy and Directions. *IEEE Access* **2019**, *7*, 67602–67631. [CrossRef]
85. Yan, P.; Yan, Z. A survey on dynamic mobile malware detection. *Softw. Qual. J.* **2017**, *26*, 891–919. [CrossRef]
86. Souri, A.; Hosseini, R. A state-of-the-art survey of malware detection approaches using data mining techniques. *Hum. Centric Comput. Inf. Sci.* **2018**, *8*, 1–22. [CrossRef]
87. Shaerpour, K.; Dehghantanha, A.; Mahmod, R. Trends In Android Malware Detection. *JDFSL* **2013**, *8*, 21–40. [CrossRef]
88. Riasat, R.; Sakeena, M.; Wang, C.; Sadiq, A.H.; Wang, Y.J. A Survey on Android Malware Detection Techniques. *DEStech Trans. Comput. Sci. Eng.* **2017**. [CrossRef]
89. IEEE Xplore. Available online: https://ieeexplore.ieee.org/ (accessed on 1 April 2020).
90. ACM Digital Library. Available online: https://dl.acm.org/ (accessed on 1 April 2020).
91. Publisher of Open Access Journals. Available online: https://www.mdpi.com/ (accessed on 1 April 2020).
92. Explore Scientific, Technical, and Medical Research On ScienceDirect. Available online: https://www.sciencedirect.com/ (accessed on 1 April 2020).
93. Hindawi. Home. Available online: https://www.hindawi.com/ (accessed on 1 April 2020).
94. Springer—International Publisher Science, Technology, Medicine. Available online: https://www.springer.com/gp (accessed on 1 April 2020).
95. arXiv. Available online: https://arxiv.org/ (accessed on 1 April 2020).
96. Microsoft Academic. Available online: https://academic.microsoft.com/home (accessed on 1 April 2020).
97. Semantic Scholar: AI-Powered Research Tool. Available online: https://www.semanticscholar.org/ (accessed on 1 April 2020).
98. Google Scholar. Available online: https://scholar.google.com/ (accessed on 1 April 2020).
99. Lau, J.H.; Baldwin, T. An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv* **2016**, arXiv:1607.05368.
100. Crammer, K.; Dekel, O.; Keshet, J.; Shalev-Shwartz, S.; Singer, Y. Online passive-aggressive algorithms. *J. Mach. Learn. Res.* **2006**, *7*, 551–585.
101. Desnos, A. Androguard-Reverse Engineering, Malware and Goodware Analysis of Android Applications. 2013. Available online: com/p/androguard (accessed on 1 April 2020).
102. Lantz, P.; Desnos, A.; Yang, K. *DroidBox: An Android Application Sandbox for Dynamic Analysis*; GitHub: San Francisco, CA, USA, 2011.
103. Github: Quoscient/Cuckoo-Droid. Available online: https://github.com/quoscient/cuckoo-droid (accessed on 1 April 2020).
104. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Siemens, C. Drebin: Effective and explainable detection of android malware in your pocket. *Ndss* **2014**, *14*, 23–26.
105. Wei, F.; Li, Y.; Roy, S.; Ou, X.; Zhou, W. Deep ground truth analysis of current android malware. In Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Bonn, Germany, 6–7 July 2017; Springer: Berlin, Germany, 2017; pp. 252–276.
106. Hurier, M.; Suarez-Tangil, G.; Dash, S.K.; Bissyandé, T.F.; Traon, Y.L.; Klein, J.; Cavallaro, L. Euphony: Harmonious unification of cacophonous anti-virus vendor labels for Android malware. In Proceedings of the 14th International Conference on Mining Software Repositories, Buenos Aires, Argentina, 20–21 May 2017; IEEE Press: Berlin, Germany, 2017; pp. 425–435.
107. VirusTotal. Available online: https://www.virustotal.com/gui/home (accessed on 13 November 2018).
108. VirusShare. Available online: https://www.virusshare.com/ (accessed on 13 November 2018).
109. Contagio. Available online: http://contagiodump.blogspot.com/ (accessed on 1 April 2020).
110. Levinec. Malware Names—Windows Security. Available online: https://docs.microsoft.com/en-us/windows/security/threat-protection/intelligence/malware-naming (accessed on 1 April 2020).
111. Le Thanh, H. Analysis of malware families on android mobiles: Detection characteristics recognizable by ordinary phone users and how to fix it. *J. Inf. Secur.* **2013**, *4*, 4. [CrossRef]
112. Bontchev, V.; Skulason, F.; Solomon, A. CARO Virus Naming Convention. 1991. Available online: http://www.caro.org/articles/naming.html (accessed on 1 April 2020).
113. Beck, D.; Connolly, J. The Common Malware Enumeration Initiative. In Proceedings of the Virus Bulletin Conference, Montreal, QC, Canada, 11–13 October 2006. Available online: https://www.virusbulletin.com/conference/vb2006/abstracts/common-malware-enumeration-initiative/ (accessed on 1 April 2020).

114. Sebastián, M.; Rivera, R.; Kotzias, P.; Caballero, J. Avclass: A tool for massive malware labeling. In *International Symposium on Research in Attacks, Intrusions, and Defenses*; Springer: Berlin, Germany, 2016; pp. 230–253.
115. G DATA Software AG. News. Available online: https://www.gdatasoftware.com/news/2017/02/threat-situation-for-mobile-devices-worsens (accessed on 13 November 2018).
116. Allix, K.; Bissyandé, T.F.; Klein, J.; Le Traon, Y. AndroZoo: Collecting Millions of Android Apps for the Research Community. In Proceedings of the 13th International Conference on Mining Software Repositories, MSR '16, Austin, TX, USA, 14–15 May 2016; ACM: New York, NY, USA, 2016; pp. 468–471. [CrossRef]
117. Lam, C. *Hadoop in Action*; Manning Publications Co.: Shelter Island, NY, USA, 2010.
118. Armbrust, M.; Xin, R.S.; Lian, C.; Huai, Y.; Liu, D.; Bradley, J.K.; Meng, X.; Kaftan, T.; Franklin, M.J.; Ghodsi, A.; et al. Spark sql: Relational data processing in spark. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, VIC, Australia, 31 May–4 June 2015; pp. 1383–1394.
119. Ahmadi, M.; Ulyanov, D.; Semenov, S.; Trofimov, M.; Giacinto, G. Novel feature extraction, selection and fusion for effective malware family classification. In Proceedings of the 6th ACM Conference on Data and Application Security and Privacy, New Orleans, LA, USA, 9 March 2016; pp. 183–194. [CrossRef]