MalRadar: Demystifying Android Malware in the New Era

LIU WANG, Beijing University of Posts and Telecommunications, China
HAOYU WANG*, Huazhong University of Science and Technology, China
REN HE, Beijing University of Posts and Telecommunications, China
RAN TAO, Beijing University of Posts and Telecommunications, China
GUOZHU MENG, SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences, China
XIAPU LUO, The Hong Kong Polytechnic University, China
XUANZHE LIU, Peking University, China

Mobile malware detection has attracted massive research effort in our community. A reliable and up-to-date malware dataset is critical to evaluate the effectiveness of malware detection approaches. Essentially, the malware ground truth should be manually verified by security experts, and their malicious behaviors should be carefully labelled. Although there are several widely-used malware benchmarks in our community (e.g., MalGenome, Drebin, Piggybacking and AMD, etc.), these benchmarks face several limitations including out-of-date, size, coverage, and reliability issues, etc.

In this paper, we first make efforts to create MalRadar, a growing and up-to-date Android malware dataset using the most reliable way, i.e., by collecting malware based on the analysis reports of security experts. We have crawled all the mobile security related reports released by ten leading security companies, and used an automated approach to extract and label the useful ones describing new Android malware and containing Indicators of Compromise (IoC) information. We have successfully compiled MalRadar, a dataset that contains 4,534 unique Android malware samples (including both apks and metadata) released from 2014 to April 2021 by the time of this paper, all of which were manually verified by security experts with detailed behavior analysis. Then we characterize the MalRadar dataset from malware distribution channels, app installation methods, malware activation, malicious behaviors and anti-analysis techniques. We further investigate the malware evolution over the last decade. At last, we measure the effectiveness of commercial anti-virus engines and malware detection techniques on detecting malware in MalRadar. Our dataset can be served as the representative Android malware benchmark in the new era, and our observations can positively contribute to the community and boost a series of research studies on mobile security.

CCS Concepts: • **Security and privacy → Malware and its mitigation**.

Additional Key Words and Phrases: Android malware; dataset; malware evolution; security reports

ACM Reference Format:
Liu Wang, Haoyu Wang, Ren He, Ran Tao, Guozhu Meng, Xiapu Luo, and Xuanzhe Liu. 2022. MalRadar: Demystifying Android Malware in the New Era. *Proc. ACM Meas. Anal. Comput. Syst.* 6, 2, Article 40 (June 2022), 27 pages. https://doi.org/10.1145/3530906

*Corresponding Author: Haoyu Wang (haoyuwang@hust.edu.cn).

Authors' addresses: Liu Wang, Beijing University of Posts and Telecommunications, Beijing, China; Haoyu Wang, Huazhong University of Science and Technology, Wuhan, China, haoyuwang@hust.edu.cn; Ren He, Beijing University of Posts and Telecommunications, Beijing, China; Ran Tao, Beijing University of Posts and Telecommunications, Beijing, China; Guozhu Meng, SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China; Xiapu Luo, The Hong Kong Polytechnic University, Hong Kong, China; Xuanzhe Liu, Peking University, Beijing, China.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2022 Association for Computing Machinery.
2476-1249/2022/6-ART40 $15.00
https://doi.org/10.1145/3530906

1 INTRODUCTION

Mobile malware detection has attracted massive research effort in our research community. In the past years, there are over 15k papers focus on Android malware analysis [26] based on the trends of Google Scholar. Various advanced techniques, from signature-based approaches, to deep learning based approaches, were proposed to address all-around challenges in the malware detection, and almost all of them were reported to achieve promising results.

One major gap between malware detection research and practice is that, the number of malicious apps in the wild is massive and their malicious behaviors and evasion techniques used are evolving all the time, while our research community does not maintain the most up-to-date malware samples to evaluate and improve their techniques. Thus, *the aforementioned research efforts are adversely affected by the lack of clear understanding of the latest mobile malware trends.*

A reliable and up-to-date malware dataset is essential to evaluate the effectiveness of malware detection. However, labelling a reliable malware dataset is non-trivial. Although there are several widely-used malware datasets in our community (e.g., MalGenome [73], Drebin [42], Piggybacking [52], AMD [69]), these benchmarks face limitations including *size*, *coverage*, *out-of-date malware*, and *the reliability of the methods* to label the ground truth, etc.

Existing Malware Labelling Methods. In general, there are mainly two ways used by the research community to create the malware datasets. The first one is to create malware dataset by *resorting to the analysis reports of security experts*. The Android Malware Genome (MalGenome for short) [73] is the first well-labeled dataset. It has over 1,200 malware samples, ranging from 2010 to 2011. This dataset was created by carefully examining related security announcements, threat reports, and blog contents from existing mobile antivirus companies and security experts. As a result, this dataset is reliable and hundreds of research papers use this dataset as the ground truth to perform malware detection and analysis.

The second way is to *rely on the scanned results of anti-virus engines*. Most existing malware datasets were created by resorting to the detection results of VirusTotal[1], a widely-used online malware scanning service aggregating over 70 anti-virus engines. As VirusTotal does not provide researchers with binary labels per app, i.e., malicious or benign, researchers have to label the scanned apps based on the results of over 70 engines. In this context, researchers usually use some *ad-hoc* methods to label malware, i.e., *defining a threshold*, and the scanned apps will be regarded as malware as long as the number of flagged engines exceeds the threshold. *However, different researchers would choose different thresholds, without convincing reasons.* For example, Drebin [42] was created based on the results of 10 famous engines on VirusTotal, i.e., one sample was selected as long as *at least two of the 10 selected engines* flagged the sample as malicious. The Piggybacking [52] dataset used 1 engine as threshold, TESSERACT [58] [50] used 4 engines as threshold to label malware from AndroZoo [41], and AMD [69] used 28 engines (over 50% of the engines) as threshold. Wang et al. [66] believe a threshold of 10 engines is a reliable choice. *Thus, there are no standards on how to take advantage of the detection results to label malware.* More importantly, the detection results of VirusTotal would change all the time, which may introduce bias towards labelling a trustworthy malware dataset [74]. Our preliminary exploration suggests that, for over 876K Android apps that were flagged by at least one engine on VirusTotal in the AndroZoo dataset, over 200K of them (roughly 1/4) are not flagged as positive by any engines on VirusTotal in the latest results.

Essentially, a reliable malware dataset should be manually verified by security experts and their malicious behaviors should be carefully labelled. We investigate the existing malware datasets commonly used by the research community, and characterize in Table 1 the number of malware samples, the number of families, the collection time and the malware labelling method used. As can

[1]www.virustotal.com

be seen, besides MalGenome, all the other representative malware datasets are hard to guarantee that they are reliable. They largely rely on the VirusTotal scan results, inheriting the limitations including both the thresholds and the volatility of the results. However, MalGenome dataset is old (created in 2011) and the MalGenome authors have stopped supporting the dataset sharing. Although the authors of AMD [69] seek to provide a reliable dataset by manually analyzing the samples, they are only able to analyze 405 samples due to the huge effort involved, while the remaining samples are not verified.

Table 1. Existing malware datasets commonly used by the research community.

Dataset	# of Malware	# of Families	Collection time	Malware Labelling Method
MalGenome[73]	1200	49	2010.8-2011.10	Examining the related security announcements, threat reports, and blog contents
Drebin[42]	5560	179	2010.8-2012.10	VirusTotal: at least 2 out of 10 specific engines report malware
AMD[69]	24650	71	2010-2016	VirusTotal: at least 50% of anti-virus engines report malware
Rmvdroid[67]	9133	56	2014-2018	VirusTotal and Google Play's app maintenance
Piggybacking[52]	1497	29	2009.9-2014.7	VirusTotal and Machine learning

This Work. In this paper, we first make efforts to create ***MalRadar***, a growing and up-to-date Android malware dataset using the most reliable way, i.e., by collecting malware samples based on the analysis reports of security experts (see **§ 2**). To this end, we first search security websites that usually report new Android malware samples, covering a list of well-known anti-virus companies including *TrendMicro*, *McAfee*, *Kasperky*, etc. Then, we used an automated method to examine all the security reports from 2014 to April 2021 (967 in total), and we have collected 178 related Android malware security reports. Note that, we only select the security reports that provide IoC information (e.g., MD5 or SHA256) of the malware samples. In addition, we resort to Koodous [35] to download these samples and collect all their related information. At last, we have collected 4,534 representative malware samples in total, which belong to 148 malware families. Then we investigate *MalRadar* from a variety of perspectives, including *distribution channels*, *installation methods*, *malware activation*, *malicious behaviors* and *anti-analysis techniques* (see **§ 3**). We next investigate the malware evolution by comparing *MalRadar* with MalGenome, and we further analyze the malware evolution across years (see **§ 4**). At last, we apply the commercial anti-virus engines and well-known malware detection techniques to MalRadar, to measure the effectiveness of existing techniques on the up-to-date dataset (see **§ 5**).

We believe that our efforts can positively contribute to community and boost a series of research studies related to Android malware analysis and detection. The MalRadar dataset has been released to the research community at: https://malradar.github.io/.

2 MALRADAR

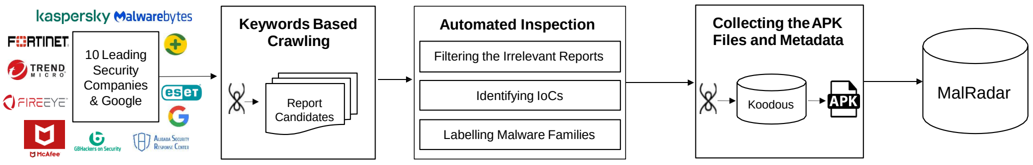

10 Leading Security Companies & Google
Keywords Based Crawling
Report Candidates
Automated Inspection
Filtering the Irrelevant Reports
Identifying IoCs
Labelling Malware Families
Collecting the APK Files and Metadata
Koodous
APK
MalRadar

Fig. 1. The pipeline to construct MalRadar.

Method Overview. To build a highly reliable Android malware dataset, we resort to security reports published by leading security companies. Usually, the security companies publish security reports against emerging security threats to reveal the *new identified malware*. These reports are manually analyzed by security experts with detailed code/runtime analysis and the evidence of the malicious behaviors (e.g., screenshots). Besides, the security reports often list the *Indicators of Compromise (IoCs)*, which could be used to create the MalRadar dataset. *Note that, most of the malicious samples revealed in the security reports are new emerging malware (i.e., the security companies who first identify them), which can reflect the trends of malware evolution.*

As aforementioned, labelling a reliable malware dataset based on security reports is non-trivial. It requires searching for high-quality security reports, reading them carefully to identify any valuable information therein (e.g.,distribution channels, families, malicious behaviors), collecting corresponding malware samples, and so on. Thus, we seek to automate this process in order to save time and labor. As shown in Figure 1, we use an automated approach consisting of four main steps: (1) use a keyword-based method to automatically search and crawl the security reports from major security company websites; (2) filter the irrelevant reports and remove the reports without IoCs; (3) collect the IoCs, family labels and investigate the behaviors of malware; (4) collect the binary files (i.e., apks) and other information. This automated pipeline we implemented helped a lot in crawling security reports and collecting APK files/metadata (using web crawler technique), filtering the reports, identifying the IoCs (using regular expression technique), etc. However, it still cost a lot of human effort (e.g., manually verifying results). Our data collection process can be updated monthly or quarterly, to keep MalRadar an up-to-date dataset. We next describe each step in detail.

2.1 Collecting the Security Reports

Source Selection. To collect reliable security reports, we first refer to the list of 20 leading security companies launched by AV-Comparatives [25]. We manually searched the security reports using keywords (e.g., "Android Malware", "Mobile Malware" and "Mobile Security") on the websites of these 20 leading security companies. Finally, we have identified 7 leading security companies that have posted Android malware analysis reports, including TrendMicro [39], ESET [30], Kaspersky [34], McAfee [37], Fortinet [32], FireEye [31] and Malwarebytes [36]. To expand the list, we further resort to Google search, use the same keywords to identify other reliable sources that usually publish security reports related to Android malware. We add GBHackers [33] and Check Point [29], two advanced cyber security online platforms which include the security reports released by security expert teams, and Qihoo 360 [38], a leading Chinese security company. Thus, we collected security reports from these 10 leading security platforms.
Crawling Related Reports For the selected 10 major security websites, we have created an automated crawler to search the candidates of Android malware analysis reports using the aforementioned keywords. Moreover, as some security experts occasionally release high-quality reports of new Android malware on personal blogs, we also crawl the first 20 pages in the search results of Google using the same keywords periodically. Our latest crawling of security websites was performed in April 2021. After removing the redundant reports, we have 967 unique mobile security reports left.

2.2 Filtering the Irrelevant Reports

Note that not all the 967 reports could meet our requirement, as some of them may only describe mobile security techniques, and some of them did not provide any IoCs that could be used to create our own dataset. Thus, we take an automated approach to filter and inspect all the 967 reports. To ensure the accuracy, we adopt the following criteria to filter the reports: (1) the content must be Android malware analysis, and provide sufficient details of the malware (e.g., their malicious behaviors); (2) the reports should provide at least one IoC. To be specific, the information contained in an indicator including at least one item: file hash (MD5/SHA1/SHA256), package name or app name. The file hash is known to be the unique code representation of an app, but the app names and package names are not. Thus, we only selected the report containing the apk file hash to ensure consistency and accuracy. Based on these two criteria, we create a *Filtering* to remove reports that do not have 'Android' or any IoCs, and then manually confirm the results. Finally, we harvested 178 high-quality security reports that meet our demands, which were released from 2014 to 2021[2]. The distribution of these reports is shown in Table 2.

Table 2. Overview of the MalRadar dataset.

Security Webiste	# Reports	# Apps	# Families	Time
TrendMicro[39]	43	2,348	46	2016-2020
ESET[30]	27	160	22	2014-2020
Kaspersky[34]	23	155	22	2016-2020
GBHackers[33]	15	128	14	2018-2020
Qihoo[38]	12	143	16	2017-2019
McAfee[37]	8	47	8	2016-2021
Fortinet[32]	7	22	7	2017-2019
FireEye[31]	4	338	11	2014-2016
Check Point[29]	4	142	5	2020-2021
Malwarebytes[36]	3	12	5	2017-2020
Google	32	1039	40	2016-2021
Total	178	4,534	148	2014-2021

2.3 Identifying and Labelling IoCs

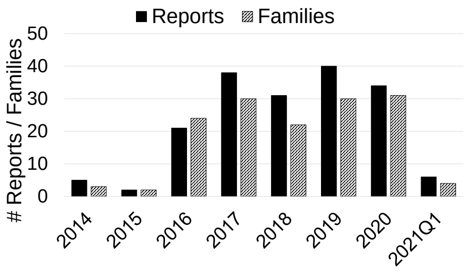
Then, for each security report, we seek to take an automated approach for identifying and labelling malware samples. We implement an *Extractor* to retrieve the IoCs and families of the samples in reports, and further manually confirm the results. IoCs can be easily identified using regular expression in the reports, as they usually have explicit labels (e.g., MD5 and SHA256) in the reports. We sampled *Extractor's* results and found that it was able to extract all the IoCs (MD5 or SHA256) included in the reports.

Labelling the family of a malware is essential for creating a comprehensive dataset. To this end, we first examined and summarized the forms in which the family names appear in the reports. We discovered that sometimes family names appear in a separate proper noun format and is not a regular word (e.g., Monokle, Hqwar), sometimes it is included in a formatted label (e.g., ANDROIDOS_ LIBSKIN.A). Thus, on the one hand, we located the sentence that describes malware family (i.e., containing keyword 'family', 'variant', etc) if exists, and then filtered out all regular words[3] as well as security words (e.g., Trojan) to extract the family candidates. On the other hand, we extracted the formatted label from the reports (if exists) and parsed the family from the label[4]. Further we manually verified the labelling results, and found the *Extractor* helped us successfully identify the family names in 135 reports. For the remaining 43 reports, we then manually extracted the family names by reading them carefully. As a result, we found that there were 27 reports indeed did not clearly specify the malware families. The other 16 reports' family extraction failed partly because the sentence or the formatted label containing family was not successfully identified, and partly because the family name was detected as a regular word (e.g., Joker). Overall, we found that most of the security reports have explicitly labelled the malware families of the analyzed samples,

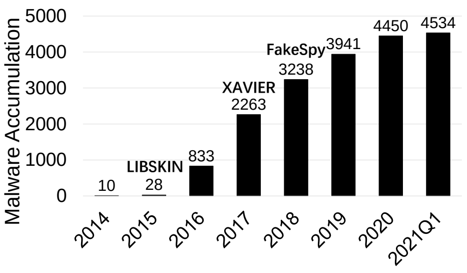
[2]We found that the number of qualified reports/samples prior to 2014 was too small (e.g., only 2 reports and 3 samples in 2013). Considering that such a small sample size is not practically meaningful and may even bias the trend analysis, we retained the results for 2014 and later.

[3]we use PyEnchant, a spellchecking library for Python based on the excellent Enchant library.

[4]We design regular expressions in some common formats to match, then tokenize and remove generic tokens (refer to AVClass's generic token list).



(a) The number of security reports and malware families over time



(b) The cumulative growth of new malware samples

Fig. 2. The Android malware growth from 2014 to 2021 in MalRadar dataset.

and a large portion of the security reports were the first to identify such malware, which makes the dataset more credible. To address the issue of the 27 reports providing no family information, we use AVClass [61], a widely used malware labelling tool, to label the malware families for these unknown reports.

2.4 Collecting the APK Files and Metadata

Although these security reports have provided the information of IoCs, no malware samples could be downloaded directly from the security websites, in case of distributing malware for malicious purposes. Thus, we further resort to other channels to harvest the binary files of these samples. To the best of our knowledge, Koodous [35] is one of the most popular platforms that provide mobile malware downloading services. It is designed specifically for Android malware, which hosts over 76 million Android apps by the end of May 2021 and it grows rapidly everyday. Thus, we purchased the premium services of Koodous and fed all the APK file hashes to it for downloading. Furthermore, we have tried our best to collect the corresponding metadata of these malware from Koodous and VirusTotal, including app names, package names, the first seen time on these platforms (e.g., first scan time on VirusTotal and upload time on Koodous) and developer/signature, which are used to facilitate our analysis in § 3.

2.5 Overview of Dataset

Table 2 shows MalRadar dataset, including 4,534 malware collected from 178 security reports, ranging from 2014 to 2021. Among them, 4,503 samples are labelled to 148 malware families[5]. In our collection, *TrendMicro* is the most active security company on Android malware analysis, and we have collected 2,348 malware from its 43 security reports. Besides, we have collected over 100 representative malware samples from FireEye, ESET, Kaspersky, Qihoo, Check Point and GBHackers respectively. Figure 2(a) shows the number of security reports and malware families captured per year in MalRadar. Figure 2(b) shows the cumulative growth of Android malware samples in MalRadar (and representative families). Most of the malicious apps are distributed in 2016-2020, and over 3,700 (approximately 81.6%) of them were released after 2017 (inclusive). Considering the existing accessible malware datasets (e.g., Drebin and AMD) are mostly out-dated (created before 2016), *MalRadar can serve as the reliable and up-to-date Android malware dataset.*

[5]Note that 31 samples could not be labeled as any family.

3 MALWARE CHARACTERIZATION

In this section, we present a systematic characterization of Android malware in MalRadar, ranging from *distribution* (**§ 3.1**), *installation* (**§ 3.2**), *activation* (**§ 3.3**), *malicious behaviors* (**§ 3.4**), and *anti-analysis techniques* (**§ 3.5**). Specifically, observations in § 3.1 and § 3.4 are completely summarized from the reports, since there is no other way for us to know the malware distribution channel, and each report has provided detailed descriptions of malicious behaviors. While § 3.2, § 3.3 and § 3.5 are drawn from a combination of the reports and experimental analysis. This is because some of the reports analyzed these elements and some did not, thus we conducted some further analysis to expand the results on the basis of the reports.

Table 3. Distribution channels and installation methods of the top-50 families in MalRadar.

Family Name	# reports	# apps	Distribution Channel			Installation				
			GPlay	3rd-party	Others	Repackaging	Update	Drive-by	Library	Standalone
RuMMS	2	796			Website;SMS;Email			√		
Xavier	1	593	√						√	
LIBSKIN	1	290		√					√	
HiddenAd	7	289	√	√						√
GhostClicker	1	248	√						√	
MilkyDoor	1	210	√			√				
EventBot	1	124		√	Website					√
GhostCtrl	1	109			Email					√
Lucy	1	82			Social Media;IM Apps					√
FakeBank	1	80			SMS	√				
FakeSpy	2	74			SMS			√		√
Joker	4	73	√	√					√	√
SpyNote	1	63		√						√
solid	1	61	√							√
ZNIU	1	59			Website					√
KBuster	1	54								√
hiddad	4	54	√							√
Hqwar	4	52								√
Monokle	1	51				√				
hawkshaw	1	48			Forum					√
TOASTAMIGO	1	46	√							√
AdDisplay	1	38	√	√						√
LokiBot	1	33			Website			√		
TERRACOTTA	1	33	√						√	
Click	1	33	√		Website					√
Necro	1	31	√	√	Forum;Blog				√	
Xloader	3	29			Website;SMS			√		
GnatSpy	2	28	√							√
Bahamut	1	27	√		Website	√		√		
meftadon	1	26	√							√
Shopper	2	25	√							√
SpyMax	1	25			Forum					√
Slocker	2	23			Forum					√
Dmisk	1	22			Website			√		
smsthief	1	21			SMS			√		
Maikspy	1	20			Website;Social Media			√		√
Svpeng	2	20			Website					√
donot	2	19		√	SMS					√
campys	1	19			Website;Email;SMS	√				
Banker	6	18	√	√						√
mazarbot	1	18			SMS			√		
ROOTSTV	1	16			Website			√		
Fanta	1	15		√	Website		√			
PhantomLance	1	15	√	√						√
Mapin	1	15	√	√			√			√
raddex	1	15			Website;Email;SMS;Social Media	√				
Exobot	1	14			Website;SMS/MMS					√
SpyAgent	4	14		√						√
Agent	4	13	√	√	Website;Social Media					√
Inazigram	1	12	√							√
Others(98)	127	410	40	9	37	8	5	8	5	75
Total (families)		4,503	61	23	64	14	7	18	11	106
Total (apks)		4,503	2,108	1,073	1,957	443	49	1,100	1,284	1,855

3.1 Malware Distribution Channels

Previous work (e.g., AndroZoo [41] and RmvDroid [67]) mainly crawled apps from app markets to construct the malware dataset, including both Google Play and third-party app markets. However, app market is not the only source to distribute malware. Thus, we first analyze the distribution channels of the malware samples in MalRadar according to the security reports. In general, the security reports would mention how the security researchers identify this kind of malware. Over 94% of malicious apps in our dataset could be traced to their distribution channels. By detailedly analyzing the security reports, we have classified the malware distribution channels into the following six categories, and Table 3 (see Column 4-6) shows the distribution channels of the top-50 malware families due to space limitation. The features and behaviors of all the 148 families will be available in our dataset website.

(1) Google Play (GPlay). Google Play is indeed the primary channel for releasing apps, and even for malware [63–67]. In MalRadar, 2,108 malware samples (47%) and 61 malware families (41%) were distributed through Google Play. Malware distributed through Google Play could gain a great number of app downloads. For example, 111 malware belonged to the family *HiddenAd* were found to disguise as popular games and camera apps in June 2019. Before being removed, they had gained over 9 millions app installs in total [18].

(2) Alternative App Markets. Existing studies [49, 66] have analyzed a number of third-party markets beyond Google Play, and observed that the presence of malware in third-party app markets is significant and prevalent. In MalRadar dataset, 1,073 malware samples (24%) and 23 malware families (15.5%) were distributed through alternative app markets, including ApkPure [23], 1Mobile [21], Aptoide [24], etc.

(3) Malicious Websites. Social engineering tricks are widely used to deceive victims to download malware [68]. Attackers can elaborately create fake websites to lure unsuspecting users to download malware. For example, the malicious developers took advantage of the domain squatting attack to create a flash player update website[6] to distribute malware [1]. It masqueraded as the real Flash Player, with a legitimate-looking icon, which was flagged as *Agent* malware. In MalRadar, 1,318 malware (29%) and 31 malware families (21%) were distributed through malicious websites.

(4) SMS. SMS is another widely exploited channel for spreading malware. Attackers with ulterior motives distribute various SMS messages containing a malicious URL, such as "*Congratulations on winning the grand prize, click the link to receive it*" to lure the potential victims to download the malware. For example, the *Exobot* malware primarily uses SMS/MMS phishing attacks to send messages to victims that contain a link to a fake version of a popular appp [6]. In MalRadar, 1,172 malware samples (26%) and 26 malware families (17.6%) were distributed through the SMS channel.

(5) Email. Similar to SMS, the attackers could add a download link or attach a malicious file to spread the malware via spam/phishing Emails. For example, the *AzoRult* malware samples are attached to the Email, claiming that products can be purchased by clicking on the link, which caused great affect in Japan [13]. In MalRadar dataset, 960 malware samples and 9 malware families were distributed through Emails.

(6) Social Media. Attackers can post attractive contents on social media platforms (e.g., Twitter, forums and blogs), to lure users to download malware. For example, attackers posted fake news on Twitter to claim that users can play an updated version of *Virtual Girlfriend Game* by clicking the download link, which is actually a malicious link to download *MaikSpy* malware [16]. In MalRadar, 303 malware (6.7%) and 18 families (12%) were distributed through social media promotion.

[6]www.flashplayeerupdate.com, "playeer" with double "e".

Observation #1: *MalRadar contains malware from a variety of distribution channels. A large portion of the malware samples (43.5%) and families (43.2%) in MalRadar were distributed beyond Google Play and third-party markets, which might be overlooked by previous studies in our community as most of previous work focused on app markets. Researchers and regulators should pay more attention to app release channels beyond app markets.*

3.2 Malware Installation

Then, we investigate how malicious apps (more specifically, malicious payloads) are installed into users' devices. The malware installation methods are classified into the following five categories according to previous work [69, 73]. Table 3 (see Column 7-11) presents the results for each family.

(1) App Repackaging. Previous work [52, 73] suggested that app repackaging is the most popular way to create malware, i.e., over 80% of malware samples in MalGenome were created based on app repackaging. Malware developers piggyback malicious payload into legitimate apps to trick users into downloading and installing them. To label whether a malware sample is repackaged or not, we first analyze each report to see whether the security analysts mentioned it. For the unmentioned samples, we further follow the same method used in MalGenome [73] to enforce fair comparison: if a sample shares the same package with an app in the official Market (here we resort to the AndroZoo [41], which contains roughly 10 million apps, and most of them were downloaded from Google Play), we then download the official apps and manually compare the difference, which typically contains the malicious payload piggybacked. *In MalRadar, app repackaging is no longer the most widely used technique to create malware, only 10% malware (443 samples) used app repackaging.*

(2) Update Attack. Update attack is the scenario that the original apps have not been embedded any malicious payloads in the binaries, which allows them to evade the detection of anti-virus engines or bypass the security verification of app markets. However, they would be subsequently updated with a malicious version. The tactics covered in the reports include the following three scenarios: 1) the fraudulent apps gain access to the store by submitting a clean version of the app for review and then introducing the malicious code via an update to the app later. For example, the creator of *Dvmap* malware uploaded a clean app to the Google Play and would then update it with a malicious version shortly afterwards. Usually they would upload a clean version back to the store on the same day and they do this over and over again [5]. 2) When an app is installed into user's devices, it will lure the user to update new version, and the malicious payloads will be attached to the new package. For example, there was a sample in *Fakewhatsapp* family that advertised as a simple update of the official app, but in reality it downloaded a modified version of the Whatsapp messenger app, and offered additional features that are not available on the official version [9]. 3) Some malicious apps are capable of updating themselves. If the app is not granted the permissions it wants, it will still perform part of the possible commands such as updating itself, opening the door to new payloads. Note that this behavior is not allowed by Google[7], and the malware was basically distributed from sources (e.g., social media, malvertising) other than the official store. In MalRadar, 49 malware samples and 7 malware families used update attack for installation.

(3) Drive-by Download Drive-by download is the scenario that malicious developers exploit the vulnerability of web browsers and domains to download malware without the user's authorization, or entice users to visit a malicious website and download malware. For example, *RuMMS* malware was spread in Russia by propogating the malicious URL via SMS phishing. When unsuspecting

[7]As stated in Google Developer Policy: "An app distributed via Google Play may not modify, replace, or update itself using any method other than Google Play's update mechanism. Likewise, an app may not download executable code (e.g. dex, JAR, .so files) from a source other than Google Play."

users click the seemingly innocuous link, their devices would be infected with *RuMMS* malware [2]. In MalRadar, 1,100 malware samples (24%) were installed in this way, involving 18 families.

(4) Third-party Library. The malicious payloads are usually hidden in third-party libraries (e.g., advertising libraries and '.so' native libraries), and the malware loads the malicious payload dynamically at runtime. For example, the malicious payload of *LIBSKIN* malware family was hidden in `libskin.so`. Once installed, the app would load `libskin.so` and trigger malicious behaviors of downloading other malware and uninterrupted pop-up advertisements [11]. In MalRadar, 1,284 malware samples (28%) loaded the malicious payloads from libraries, involving 11 families.

(5) Standalone. It means the developers have not taken other disguise methods to insert malicious payloads to apps [73]. The downloaded app is indeed malware that contains malicious payloads. In MalRadar, standalone is the most widely used installation method with 1,855 malware (41%) and 106 families in total. Specifically, most of standalone apps belong to *fake apps* that masquerade as the legitimate apps by mimicking the look or functionality but stealthily perform malicious actions. They usually have identical or extremely similar app names, package names or app icons to the original ones. Alternatively they trick users into believing it is legitimate with professional-looking brand and a sophisticated user interface. One *FlokiSpy* sample is an example that poses as the bank's service for mobile banking tokens (used in identity management and transaction authorization) but does not offer any of the functionality it claims to have. All it does is to serve as a spyware collecting private data including device identifiers, SMS messages, phone numbers, etc [14]. In addition, a number of standalone apps are pretending to be normal apps that also intentionally include malicious functionality. They can provide the functionality they claimed, while they also exhibit certain misbehaviors, e.g., ad fraud [20].

Observation #2: *Our investigation suggests that app repackaging is no longer the most popular way to create Android malware, and standalone malware become the most mainstream malicious apps. The possible reason is that most popular apps are now paying more attention to protection (e.g., packing), making repackaging not as easy. This also coincides with the increase in fake apps, which is inherently similar to repackaging (i.e., it borrows from popular apps to do cheating, though the technical implementation is simplified). Besides, it is a trend that malware developers are injecting the malicious payload into third-party libraries and load them at runtime.*

3.3 Malware Activation

MalGenome [73] reported event-based activation, i.e., by registering for the related system-wide events, a malware can rely on the built-in support of automated event notification and callbacks on Android to flexibly trigger or launch its payloads. Wei et al. [69] complemented the *scheduling* option of activation method. The *scheduling* means that malware starts their monitoring or data collection in a periodic manner [69]. For a more comprehensive analysis, we take into account both types of activation methods. On the one hand, we examine the system-wide Android events of interest to existing Android malware in accordance with MalGenome [73]. On the other hand, we can label the *scheduling* method by reviewing the security reports. Table 4 shows the statistical results of the activation methods in MalRadar, and the comparison with MalGenome will be discussed in § 4.

(1) System-wide Events. Among all available events, `Boot Completed` is the most interested one to existing Android malware. The particular event will be triggered when the system finishes its booting process, and malware will complete self-triggering with system turn-on. `Network` comes second with nearly half of the malware interested in it. This event is related to the malware

Table 4. Activation methods of the top-50 families.

Family Name	Event									Scheduling
	BOOT	SMS	NET	CALL	USB	PKG	BATT	SYS	MAIN	
RuMMS	√	√		√				√		
Xavier	√		√			√				
LIBSKIN	√		√		√	√		√		√
HiddenAd	√		√	√		√		√		√
GhostClicker	√		√							√
MilkyDoor	√					√		√		
EvenBot	√	√	√			√		√		
GhostCtrl										
Lucy	√									
FAKEBANK	√	√								√
FakeSpy	√	√	√	√			√	√		
Joker	√								√	√
SpyNote	√	√		√		√	√			
solid	√		√			√	√	√		
ZNIU	√	√	√				√	√		
KBuster	√			√						
hiddad	√		√			√	√			
Hqwar	√	√								
Monokle	√		√	√		√		√		
hawkshaw	√			√						
TOASTAMIGO	√		√			√		√		
AdDisplay	√							√		√
LokiBot	√	√		√		√	√	√		
TERRACOTTA										
Click	√		√	√			√			
Necro						√				
Xloader	√	√				√		√		
GnatSpy		√	√	√						
Bahamut	√		√	√						√
meftadon	√									
Shopper	√					√		√		
SpyMax	√	√		√		√	√			
Slocker									√	
Dmisk	√	√								
smsthief	√	√								
Maikspy	√			√						√
Svpeng	√	√	√				√	√		
donot	√	√	√	√						
campys	√						√			
Banker	√	√		√		√		√		
mazarbot	√	√								
ROOTSTV										
Fanta	√	√	√							
PhantomLance	√		√			√	√			
Mapin			√			√				
raddex	√	√	√	√			√			
Exobot	√	√								
SpyAgent	√	√		√						
Agent	√		√			√				
Inazigram	√									
Other (98)	77	37	36	24	0	29	23	32	1	8
Total (families)	120	58	57	41	1	48	35	48	3	16
Total (apks)	3,443	1,464	2,063	1,091	13	1,984	706	2,044	103	1,106

communicating with their C&C servers. The malware transmits the stolen privacy data to the server, or executes the commands of the remote server to achieve remote control. Some malware families only transfer data in the `WIFI network` state to prevent users from being alert due to the consumption of mobile cellular traffic. In addition, a number of malicious apps are keen in intercepting or responding incoming SMS messages as well as `phone calls`.

(2) Scheduling. The *Scheduling* method is frequently used by malicious apps to start their monitoring or data collection in a periodic manner. When the scheduling time is reached, the malicious app will start to perform the predetermined behavior or communicate with the C&C server to execute the new commands. In MalRadar, 1,106 apps (16 families) have adopted *Scheduling* to start their malicious behaviors, and a large portion of them are ransomware or adware. For example, a later version of adware *GhostClicker* will pop up interstitial advertisements for a certain period of time if the device is connected to a network with data [7].

Observation #3: *System event based activation is remaining the main trick to activate malicious payload. Boot Completed and Network are two most popular system events that are listened to by malware. Scheduling is the emerging way to active malicious payload, adopted by 24% of malware in MalRadar.*

3.4 Malicious Behaviors

As shown in Table 5, we classify the malicious behaviors into 12 categories. Note that, multiple security reports may analyze malware variants that belong to a same family, which may show diverse malicious behaviors. Thus, we reflect the behavior variants in Table 5. The '◯' indicates that the malicious behavior only exists in some variants, rather than the entire malware family.

(1) Privacy Stealing. Privacy stealing is one of the most common malicious behaviors. In MalRadar, about 90% (4,088) malware and 105 malware families steal users' private information, e.g., *phone number*, *contacts*, *SMS messages*, and *location information* etc.

(2) Abusing SMS/CALL. The malware can abuse SMS/ CALL related permissions to secretly commit malicious actions including sending, blocking, deleting SMS or making phone calls without the user's awareness. They usually take advantage of the communication function of SMS and calls to conduct some nasty transactions, such as stealing credential access, sneaking bank operations and spreading adverse information. For example, the malware detected as *Agent* can even bypass 2FA (two-factor authentication) by sending all incoming text messages to the server if requested. This enables the attacker to intercept all SMS text messages from the bank and immediately delete them from the client device, thus raising no suspicion [1]. In MalRadar, 2,287 samples (68 families) have been found to have malicious behavior of sending, intercepting, deleting SMS or making phone calls.

(3) Remote Control. Roughly 85% of malicious apps in MalRadar (3,840 samples and 107 families) are able to communicate with remote C&C servers and receive commands from the server to perform malicious activities. For example, the *XLoader* malware abused the WebSocket protocol to communicate with its C&C servers. The malware executed the corresponding behavior after receiving the remote command, such as "sendSms" meant to send SMS to a specified address, "show_fs_float_window" meant to show a full-screen window for phishing [17].

(4) Bank/Financial Stealing. Trojan-banker apps are designed to steal users' accounts of online banking systems, e-payment systems and credit or debit cards. There are 1,522 apps and 45 families in our dataset showing the banking stealing activities. *Exobot* shows a customized phishing window as long as a targeted app is launched on the device. The window is usually indistinguishable from the expected screen (e.g., a login screen of a banking app) and is designed to steal the victim's banking credentials. It targets more than 30 banks [6].

(5) Ransom. Ransomware is a type of malware that threatens to publish the victim's data or perpetually block access to it unless the ransom is paid. Ransomware attacks are typically carried

Table 5. Malicious behaviors of the top-50 families.

Family Name	Privacy Stealing	SMS/CALL	Remote Control	Bank Stealing	Ransom	Abusing Accessibility	Privilege Escalation	Stealthy Download	Ads	Miner	Tricky Behavior	Premium Service
RuMMS	√	√	√	√		○	○				√	
Xavier	√		√					√	√			
LIBSKIN	√	√	√				√	√	√			
HiddenAd	√								√		√	
GhostClicker	√		√				√		√			
MilkyDoor	√		√									
EventBot	√	√	√	√		√						
GhostCtrl	√	√	√		○							
Lucy			√		√	√	√	√			√	
FAKEBANK	√	√	√	√			√				√	
FakeSpy	√	√	√	√								
Joker	√	√	√					○	○			√
SpyNote	√	√	√								√	
solid	√								√		√	
ZNIU	√	√	√				√					
KBuster	√	√	√	√			√					
hiddad	√								√			
Hqwar				√	√							
Monokle	√	√	√			√						
hawkshaw	√	√	√			√	√	√			√	
TOASTAMIGO						√		√	√			
AdDisplay	√		√						√		√	
LokiBot	√	√	√		√							
TERRACOTTA			√				√		√			
Click	√		√				√		√		√	√
Necro			√						√			√
Xloader	√	√	√	√			√	√			√	
GnatSpy	√		√					○			○	
Bahamut	√		√								√	
meftadon	√	√	√	√				√			√	
Shopper	√		√			√		√			○	
SpyMax	√		√				√					
Slocker					√						√	
Dmisk	√	√	√								√	
smsthief	√	√	√	√								
Maikspy	√	√	√	√							√	
Svpeng	√	√	√	√		√						
donnot	√		√									
campys	√		√									
Banker	√	√	√	√		○	○	○			○	
mazabot	√	√	√	√								
ROOTSTV			√				√	√				
Fanta			√	√			√		√		√	
PhantomLance	√						√					
Mapin	√		√				√	√	√			
raddex		√				√	√					
Exobot	√	√	√	√			√					
SpyAgent	√	√				√					√	
Agent	○	○	○	○		○	○	○			○	
Inazigram	√		√									
Other (98)	64	43	66	29	12	18	30	21	17	5	31	4
Total (families)	105	68	107	45	17	30	50	35	31	4	52	7
Total (apps)	4,088	2,287	3,840	1,522	346	1,356	2,028	1,391	1,898	8	1,872	162

out using a Trojan that is disguised as a legitimate app. Once launched, it will encrypt the victim's files or force a lock screen and extort a high ransom. For example, *SLocker* disguises itself as a popular app in order to lure users into installing it. Once the ransomware runs, the app will encrypt files on the smartphone. These files will not be decrypted until the user has paid the money using Bitcoin [8]. There are 346 samples (17 families) that are recognized as ransomware in our dataset.

(6) Abusing Accessibility. Accessibility services are used to assist users with disabilities to use Android devices and apps. But malicious app developers have violated the original intention of designing this service, and use its high-level permissions to steal information, hijack, install maliciously, etc. For instance, *TOASTAMIGO* poses as legitimate app lockers that supposedly protect the device with a PIN code. Once installed, these apps will notify the user that they need to be granted Accessibility permissions in order to work. After granting permissions, these malicious apps will silently perform malicious actions or commands, including stealing users' private information, and downloading unauthorized apps [10]. In MalRadar, there are 1,356 samples and 30 families are reported to abuse Accessibility service.

(7) Privilege Escalation. Sophisticated malware can exploit Android system vulnerabilities to gain elevated privileges. Through carefully reading the reports, we can summarize the privilege escalation behavior into two common types: 1) gain `ROOT` access on the targeted system. Malicious apps can root the phone by downloading root modules (e.g.,*LIBSKIN* downloads the root module *right_core.apk* to replace the original system file [11]), or by exploiting vulnerabilities on the Android platform (e.g., *ZNIU* rootkit exploits Dirty COW [12]), etc. Malware with rooting capability can pose bigger threats as rooting Android phones can have a lot of repercussions. 2) Ask for device administration privileges. Some malware asks for the administration privileges when it first runs, and even if the user rejects or kills the process, it will reappear until they accept the request. With the administration privileges, malware can make itself more difficult to be uninstalled, and perform more sensitive malicious actions. In MalRadar, 2,028 (45%) malware samples and 50 families have the behavior of privilege escalation.

(8) Stealthy Downloading. Malware could use the `ROOT` privilege or use Accessibility to silently install apps without users' awareness. By downloading and installing other apps, the attacker can implant more malware into the infected smartphone, or download apps from the 'Pay-per-Install' ad networks to make money. For example, the *LIBSKIN* malware can download and install other apps without the user's awareness after being granted the `ROOT` privilege [11]. The *TOASTAMIGO* downloaded and installed another malware by abusing the accessibility service. 1,391 malicious apps and 35 families in our dataset have the malicious downloading behaviors.

(9) Aggressive Advertising. Malware can show diverse aggressive advertising behaviors [46, 54, 55]. First, malware can create a large number of fake clicks in the background to make a profit. Second, malware displays aggressive ads that cannot be closed on the device interface. Furthermore, some popped ads are malicious, i.e., will lead users to a malicious websites. For example, *GhostClicker* adware inserted its malicious code into Admob library (Google's mobile advertising network). Then, the malware calculated the coordinate position that needs to be clicked based on the screen dimensions and used the `dispatchTouchEvent` API to simulate clicking. Using this method, adware generated fake network traffic to earn revenue [7]. *HiddenAd* pops up full-screen ads every interval, which will seriously affect the normal use of users. The users cannot even close the ads [18]. More than 42% of malicious apps (1,898 samples and 31 families) in MalRadar have such aggressive ad behaviors.

(10) Miner. Mining malware consumes the processing power of smartphones to generate revenue from cryptocurrency mining, which will increase device wear and tear, reduce battery life, comparably slower performance. For example, *HiddenMiner* uses the device's CPU power to mine `Monero` until the device's resources are exhausted [15]. *JSMiner*, a malware family with hidden cryptocurrency mining capabilities, loads the JavaScript library code from Coinhive and starts mining with the attacker's own site key. When the malicious JavaScript code is running, the CPU usage will be exceptionally high [4]. Although Coinhive has stopped service in 2019, there are a number of other coin-mining services available. In MalRadar, only 8 malware samples and 5 families have this malicious behavior.

(11) Tricky Behavior. There are a number of malicious apps that use a variety of tricks to prevent themselves from being uninstalled, such as hiding or changing the icon once executed, emptying app label, blocking the user access to the app detail page, launching a transparent activity background to hide malicious content from the user, etc. They can dismiss notifications and change the device's settings (e.g., set the ringer mode to silent, switch off the screen) which can prevent victims from noticing fraudulent transactions happening. For example, the *Maikspy*-carrying app will display "Error: 401. App not compatible. Uninstalling.." after being launched to trick the user into thinking that the app has been removed from the device. However, the spyware is simply hiding itself and running in the background [16]. In MalRadar, 1,872 (41.5%) samples and 52 families exploit such tricks.

(12) Premium Service. Interestingly, we discover that some malicious apps are able to subscribe the users to premium services without their knowledge or consent, which is a type of malicious behavior that has rarely been mentioned in previous studies. They automatically processes the user's premium service subscription, which can cost the victim money. The malware usually misuse the `WAP-Click`, a technology that simplifies the subscription to various premium services without letting users know and there is no permission required to subscribe to the unwanted services. For example, *Joker* is a type of Android malware that subscribes users to premium phone services with a tactic known as WAP fraud. The apps impersonate legitimate apps, such as virtual keyboards, camera apps, and games. However, the hidden malicious code would operate behind the scenes to open a browser window and subscribe users to premium phone numbers, enabling the operators to earn commissions [28]. There are 162 samples and 7 families in MalRadar with malicious actions for premium service subscriptions.

Observation #4: *The malicious behaviors of Android malware have become diverse. Although Privacy Stealing, Remote Control and Privilege Escalation are remaining the most popular behaviors, a number of new emerging activities (e.g., ransomware, mining and premium services subscription) are becoming rampant.*

3.5 Anti-analysis Techniques

To evade detection, malware developers are taking advantage of sophisticated anti-analysis techniques. To investigate the landscape of anti-analysis techniques used in malware, we first carefully analyze the related description in the security reports to label anti-analysis techniques that malware adopted. Then we take advantage of APKiD [22], a widely used static analysis tool to identify the packers, obfuscators, and other anti-analysis techniques used in an Android app. As shown in Table 6, we have classified the anti-analysis techniques into seven categories.

Over 96% of malware samples in MalRadar use at least one anti-analysis technique. The most widely used technique is *Environment check*, with 85% of malware samples. To evade dynamic analysis (e.g., sandbox), malware developers have carefully designed the malicious payloads, by checking the device information and runtime environment before triggering their malicious behaviors. If a malware detects that it is not running on a real device, malicious behaviors will be hidden. For example, *Xavier* masks its aggressive ad behaviors by detecting whether the system is running in an emulator to evade dynamic detection [3]. Additionally, the *Anubis* malware determines whether the app is running in a sandbox environment by checking the *motion sensor* data [19]. *Obfuscation* is the second most popular evasion technique, i.e., over 52% of malware samples use this technique to either obfuscate the identifiers (e.g., class name and method name) in the code, or obfuscate the code structure (e.g., control flow). Followed by *Communication encryption*, a sophisticated technique that usually employs the https protocol and converts the constant strings into ciphertext using encryption algorithms to increase the efforts of reverse engineering. In MalRadar, there are 2,222 malicious apps that encrypt the communication between them and the `C&C` servers. Besides, some malicious apps use more aggressive methods to kill the anti-virus process or uninstall the anti-virus apps after they have been granted the `ROOT` privilege. The Trojan *Shopper* will check

Table 6. Anti-analysis techniques used in families.

Family Name	Obfuscation	Communication Encrypt	Packer	Environment Check	Disable Anti-virus	Anti Disassembly	Anti Debug
RuMMS		√		√	√		
Xavier	√	√		√			○
LIBSKIN	○		○	√			
HiddenAd	○			√	○	○	○
GhostClicker				√			○
MilkyDoor	√	√		√			
EvenBot	√	√		√			
GhostCtrl	√	√	√				
Lucy	√						
FAKEBANK				√			
FakeSpy	○	√	○	○			
Joker	√	√	○	√			○
SpyNote				√			
solid				√			
ZNIU		√	√	√			
KBuster				√			
hiddad	○			√		○	
Hqwar	√						
Monokle	√	√		√			
hawkshaw				√	√		
TOASTAMINGO			√	○			
AdDisplay				√			○
LokiBot				√			
TERRACOTTA	√			√			
Click	○			√			○
Necro	○		○	√		○	○
Xloader		○			√		
GnatSpy	√	√		√			
Bahamut				√			
meftadon	○		○	√			○
Shopper	√			√			
SpyMax				√			
Slocker	√		○				
Dmisk	√	√	○	√			
smsthief							
Maikspy				√			
Svpeng				√	√		
donot				√			
campys				√			
Banker	√			√	√		
mazarbot				√			
ROOTSTV				√			
Fanta				√			
PhantomLance				√			○
Mapin				√			
raddex				√			
Exobot	√						
SpyAgent				√			○
Agent	√		○	○		○	○
Inazigram				√			
Other (98)	21	8	6	16	10	7	14
Total (families)	44	20	17	59	16	11	25
Total (apks)	2,362	2,222	797	3,845	1,253	419	1,440

the availability of Accessibility or ROOT privileges first. If found these privileges were not granted, they will periodically issue a request (with phishing messages) to lure users to provide them. Once granted, the malware will disable Google Play Protect and other anti-virus process [27]. This behavior is also common to see in ransomware families.

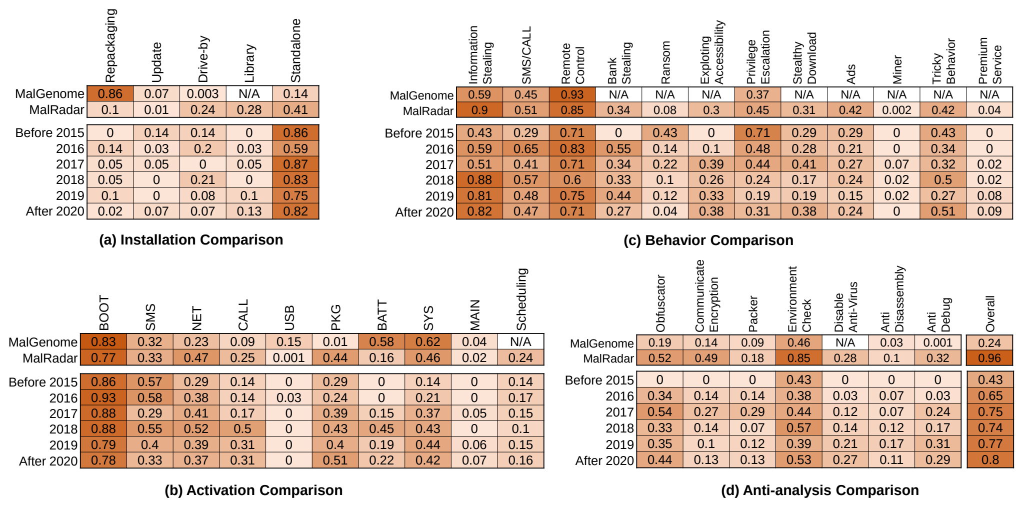
> **Observation #5:** *96% of malware samples in MalRadar use at least one anti-analysis technique. Environment check, obfuscation and communicate encryption are the most widely used techniques to evade detection. Sophisticated techniques are emerging in the malware, which pose challenges to malware detection and analysis.*

4 MALWARE EVOLUTION

To understand the Android malware evolution, we first compare MalRadar (created in 2021, malware ranges from 2014-2021) with MalGenome (created in 2011, malware ranges from 2010-2011) on all the aforementioned characteristics, in order to reveal the malware evolution over a decade. We compare with MalGenome because MalGenome is labelled using a similar approach to ours and it is, to the best of our knowledge, the most reliable reference. There is a small gap from MalGenome (2010-2011) to MalRadar (2014-2021), which we consider conducive to highlighting the evolution. Further, we analyzed in detail the year-by-year malware evolution.

4.1 MalRadar VS. MalGenome

First we compare MalRadar with MalGenome in the above four aspects. Here, the MalGenome dataset (i.e., 1,260 samples) was obtained from one of the authors of that work. The data/results below were retrieved directly from the previous paper of MalGenome [73], and we have confirmed that the analysis method we used was consistent with it through some experiments (e.g., reproducing its result). As such, we calculate the number of apps in MalRadar based on the same metrics, and compare them accordingly.



	Repackaging	Update	Drive-by	Library	Standalone
MalGenome	0.86	0.07	0.003	N/A	0.14
MalRadar	0.1	0.01	0.24	0.28	0.41
Before 2015	0	0.14	0.14	0	0.86
2016	0.14	0.03	0.2	0.03	0.59
2017	0.05	0.05	0	0.05	0.87
2018	0.05	0	0.21	0	0.83
2019	0.1	0	0.08	0.1	0.75
After 2020	0.02	0.07	0.07	0.13	0.82

(a) Installation Comparison

	Information Stealing	SMS/CALL	Remote Control	Bank Stealing	Ransom	Exploting Accessibility	Privilege Escalation	Stealthy Download	Ads	Miner	Tricky Behavior	Premium Service
MalGenome	0.59	0.45	0.93	N/A	N/A	N/A	0.37	N/A	N/A	N/A	N/A	N/A
MalRadar	0.9	0.51	0.85	0.34	0.08	0.3	0.45	0.31	0.42	0.002	0.42	0.04
Before 2015	0.43	0.29	0.71	0	0.43	0	0.71	0.29	0.29	0	0.43	0
2016	0.59	0.65	0.83	0.55	0.14	0.1	0.48	0.28	0.21	0	0.34	0
2017	0.51	0.41	0.71	0.34	0.22	0.39	0.44	0.41	0.27	0.07	0.32	0.02
2018	0.88	0.57	0.6	0.33	0.1	0.26	0.24	0.17	0.24	0.02	0.5	0.02
2019	0.81	0.48	0.75	0.44	0.12	0.33	0.19	0.19	0.15	0.02	0.27	0.08
After 2020	0.82	0.47	0.71	0.27	0.04	0.38	0.31	0.38	0.24	0	0.51	0.09

(c) Behavior Comparison

	BOOT	SMS	NET	CALL	USB	PKG	BATT	SYS	MAIN	Scheduling
MalGenome	0.83	0.32	0.23	0.09	0.15	0.01	0.58	0.62	0.04	N/A
MalRadar	0.77	0.33	0.47	0.25	0.001	0.44	0.16	0.46	0.02	0.24
Before 2015	0.86	0.57	0.29	0.14	0	0.29	0	0.14	0	0.14
2016	0.93	0.58	0.38	0.14	0.03	0.24	0	0.21	0	0.17
2017	0.88	0.29	0.41	0.17	0	0.39	0.15	0.37	0.05	0.15
2018	0.88	0.55	0.52	0.5	0	0.43	0.45	0.43	0	0.1
2019	0.79	0.4	0.39	0.31	0	0.4	0.19	0.44	0.06	0.15
After 2020	0.78	0.33	0.37	0.31	0	0.51	0.22	0.42	0.07	0.16

(b) Activation Comparison

	Obfuscator	Communicate Encryption	Packer	Environment Check	Disable Anti-Virus	Anti Disassembly	Anti Debug	Overall
MalGenome	0.19	0.14	0.09	0.46	N/A	0.03	0.001	0.24
MalRadar	0.52	0.49	0.18	0.85	0.28	0.1	0.32	0.96
Before 2015	0	0	0	0.43	0	0	0	0.43
2016	0.34	0.14	0.14	0.38	0.03	0.07	0.03	0.65
2017	0.54	0.27	0.29	0.44	0.12	0.07	0.24	0.75
2018	0.33	0.14	0.07	0.57	0.14	0.12	0.17	0.74
2019	0.35	0.1	0.12	0.39	0.21	0.17	0.31	0.77
After 2020	0.44	0.13	0.13	0.53	0.27	0.11	0.29	0.8

(d) Anti-analysis Comparison

Fig. 3. Malware evolution across the decade.

Installation. As shown in Figure 3 (a) (see the first two rows), *app repackaging* was the most popular way for malicious developers to deliver malware in MalGenome. Roughly 86% of malware samples and over 51% of malware families were installed by repackaging. While in MalRadar, the proportion of repackaged malware has dropped greatly, i.e., only 10% of malware are repackaged apps. On the contrary, the standalone malware has gradually risen to become dominant, from

14% to 41%. In addition, new installation method of loading malicious payloads from third-party libraries are emerging (28% in MalRadar).

Activation Methods. As shown in Figure 3 (b) (see the first two rows), system event based activation is remaining the main trick to active malicious payloads in both MalGenome and MalRadar. Compared with MalGenome, the proportion of *Network*, *Phone events (CALL)* and *Package* events have increased, while the events of *Boot Completed*, *USB Storage* and *Power/Battery* have declined. Overall, the *Boot Completed* event has been staying at the top, being listened to by 83% of malware in MalGenome and 77% in MalRadar. This is not surprising, as this particular event will be triggered when the system completes the boot process, which is the perfect time for malware to initiate its background services. Meanwhile, the *Scheduling* method is used by a quarter of samples in MalRadar to show malicious behaviors in a periodic manner.

Malicious Behavior. As shown in Figure 3 (c) (see the first two rows), *remote control* and *privacy stealing* were two most popular malicious behaviors in MalGenome, i.e., 93% and 59% of malicious apps have such behaviors respectively. More than that, these two behaviors remain strong and vibrant in MalRadar, with about 85% and 90% of malware having the two nasty actions. Additionally, in MalRadar, there are a number of new malicious behaviors that were not available in MalGenome, such as *ransomware*, *exploiting accessibility*, *miner*, *tricky behavior*, etc.

Anti-analysis Techniques. Anti-analysis techniques used in MalGenome have not been characterized. To enforce a fair comparison, we use the same tool, APKiD [22], to analyze all the 1,260 samples in MalGenome. Besides, we searched through the relevant reports of each family in MalGenome and manually extracted the anti-analysis techniques used in them. As shown in Figure 3 (d) (see the first two rows), only 24% of malware in MalGenome used anti-analysis techniques (19% of malware used obfuscation and 14% encrypted the communication). In MalRadar, about 96% of malware samples (4X of those in MalGenome) used evasion techniques to increase the detection and analysis efforts. Runtime environment check and obfuscation are two most popular techniques to evade detection. Disable anti-virus started to spread its wings, 28% aggressive malware directly attacks the anti-virus software (if already installed on the victim's phone) and forcibly shuts down the protection process. Some complex anti-analysis techniques are becoming more and more prevalent, such as Anti-debug to evade debugger.

4.2 Year-by-year Evolution

We then analyze the year-by-year malware evolution in MalRadar. Since considering that the number of malware samples provided by different reports varies greatly, with some reports providing a large number of samples and some providing only a few, which may introduce bias into the calculation of the proportion of each behavior, we normalized the number of samples provided by the same report for the same family (i.e., set it as 1) and show the evolution.

Installation. We look in detail on a per-year basis (see row 3-7 in Figure 3 (a)). In general, standalone apps have been rampant over the years, with a majority of samples coming in every year. The proportion of repackaging has gradually decreased, and the method of installing hidden malicious payloads in third-party libraries is gradually expanding.

Activation Methods. Looking in detail on an annual basis (see row 3-7 in Figure 3 (b)), we can observe the *Boot Completed* event has remained popular over the years, but there seems to be a slight downward trend. On the contrary, the *Package* and *System events* have a tendency to increase year by year. Besides, there are several other system-wide events that have experienced fluctuations in their annual proportion, such as *Network*, *Phone events (CALL)* and *Power/Battery* which gradually increased from 2014 to 2018, yet have declined since. A certain proportion (10%-20%) of malicious apps using *Scheduling* method appear every year.

Malicious Behavior. We further look in detail on a per-year basis (see row 3-7 in Figure 3 (c)) of malicious behaviors in MalRadar. Malicious actions of *information stealing* and *remote control* have been leading the way for all these years, and in particular, *information stealing* has become increasingly prevalent in malicious apps. The proportion of malware that exploiting vulnerabilities to obtain privileges is declining, mainly due to the more and more secure Android system and the latest system has been widely adopted. *Ransomware* also shows a gradual decline. *Miner* and *Premium service subscription* are the nascent malicious behaviors of malware in recent years. Although small in number, it represents a possible evolutionary trend that should make us alert.

Anti-analysis Techniques. At last, we take a look at the anti-analysis techniques on an annual basis (see row 3-7 in Figure 3 (d)). As expected, the proportion of malware using anti-analysis techniques (at least one) has increased year by year, from 43% (before 2015) to 80% (after 2020). In 2015 and before, only one anti-analysis technique, i.e., *runtime environment checking*, was used in the samples of our dataset, and from 2016 onwards various techniques have been unfolded. Particularly, the use of *Disable Anti-Virus* technique is becoming increasingly popular among malware, which may indicate a trend of malware targeting anti-virus software that warrants our attention.

Observation #6: *The malware has undergone a marked evolution over the decade. The characteristics of malware have changed greatly. Existing out-dated malware dataset cannot reflect the new trends of Android malware in the new era. Thus, it is important for our community to update the malware dataset in a timely manner.*

5 MALWARE DETECTION

Although a large number of malware detection techniques have been proposed, existing research efforts have been adversely affected by the lack of clear understanding of the latest Android malware trends and dataset. In this section, we seek to measure the effectiveness of existing anti-virus engines and malware detection techniques on MalRadar. Note that, as MalRadar is created without relying on the information of VirusTotal like other datasets, thus we first measure the effectiveness of anti-virus engines on VirusTotal (**Section 5.1**). Then, we select three kinds of state-of-the-art malware detection approaches from the research community and measure their effectiveness on MalRadar (**Section 5.2**). Note that, we do not intend to devise our own approach for malware detection in this paper.

5.1 Measurement of Anti-virus Engines

There are 75 commercial anti-virus engines on VirusTotal that provide malware detection for Android by the time of this study. As the samples in MalRadar are high-quality malware verified by security researchers, we use them to measure the effectiveness of the anti-virus engines.

5.1.1 Overall Result. We upload all these samples to VirusTotal in May 2021 and get their detection results. It represents the up-to-date results. Only 24% of them could be flagged by over 30 engines. It suggests that, based on the threshold defined by AMD (half of the total anti-virus engines), 76% of them will not be regarded as malware. Roughly 97% of them could be flagged by over 10 engines, which suggests that 3% of the samples would not be labelled as malware using the methods in previous work [67]. Moreover, we observe that a number of malicious apps can only be flagged by a few engines. In Table 7, we list the top 10 most evasive malware samples. Most of them belong to the *TERRACOTTA* and *GhostClicker* family. For comparison, we also upload the MalGenome apps to VirusTotal and get their most recent results. All samples in MalGenome could be flagged by at least 17 engines and roughly 83% of malware could be flagged by over 30 engines. It suggests that the malware samples in MalRadar are more evasive than those in MalGenome.

Table 7. The top-10 most evasive malware samples.

MD5	Package Name	AV-Rank	Family
8aa182e6f4780caf5f33f03fb36f5ed0	com.mfs15.myfreeshoes	0	TERRACOTTA
3fe1e712c8b80cb1c3dee749afea7c03	com.yfs16.yourfreeshoes	0	TERRACOTTA
b734cc1830862cf7ada685add0e3645a	com.yfs19.yourfreesneakers	0	TERRACOTTA
652326bd3e48e391101c4a1fcdbb3cfc	com.yss25.yourstarshoes	0	TERRACOTTA
fbf12a77e8d2b35991dafc131844eaf9	com.voicerecorderprotb.vnm	1	GhostClicker
5e6923cc2ee4c47f8d143960e3c600ed	com.smartcompasstvc.vnm	1	GhostClicker
cbaa22ad4d564b444ca0148416fbc247	com.mycompassungt.vn	1	GhostClicker
6ffc0f559f79217889f00f1c1852032d	com.smartcompass.compass.brt	1	GhostClicker
4f47dbecf090846575eef45aef712b8c	games.bubbleshot.shooterbubble	1	GhostClicker
101715c13f4f37dd7a6cb883b3a9781e	com.penfour.taptaplock	2	Xavier

5.1.2 Per-engine Analysis. For each anti-virus engine, we define the *malware coverage ratio* as the proportion of malware that can be accurately flagged by each engine. As shown in Figure 4, we ranked the 75 anti-virus engines based on their *malware coverage ratio*. ESET [30] achieves the best coverage, i.e., 4,349 malware (96%) can be identified in MalRadar. Only 17 engines have the coverage ratio over 80%. Interestingly, only 27 engines (36%) detected more than 50% of malware. Specially, there are 17 engines cannot detect any malware (even if they claim to support the Android app detection). We further observed that some major IT companies performed poorly. For example, *Microsoft* flagged roughly half (52.8%) of the malware. *BitDefender* and *TrendMicro* only identified 22% and 13% respectively. Note that, even if some engines (e.g., ESET) are more accurate in identifying these malicious apps, we cannot fully rely on their results to label malware, as they usually report many false positives. The detection results in VirusTotal are constantly updated, and the preliminary studies we mentioned earlier showed that over 200K of the previously flagged malicious apps in Androzoo failed to be flagged by any engine after several months.

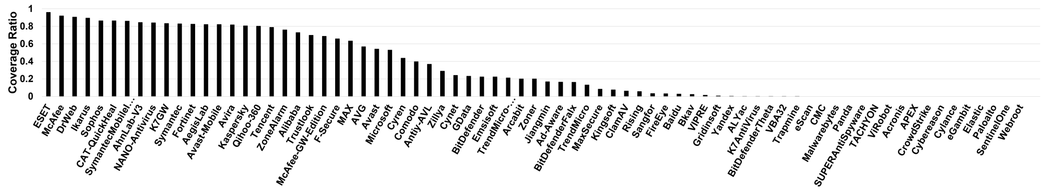


Fig. 4. The malware coverage ratio for each anti-virus engine

5.1.3 Per-family Analysis. We further investigate which malware families are more *evasive*, i.e., could evade the detection for most of the engines. Thus, for each family, we calculate the average number of flagged engines for all the samples in this family, as the *detected ratio*. As shown in Figure 5, we ranked the malware families based on their *detected ratio*, from low to high. The *TERRACOTTA* family has the lowest *detected ratio*, i.e., detected by 5.2 engines on average. 16 families have the *detected ratio* less than 20. Some malware families are quite popular, while they also have low *detected ratio*. For example, *GhostClicker* family covers 248 apps, while its *detected ratio* was 14.7, which means that a large number of samples in *GhostClicker* could evade detection

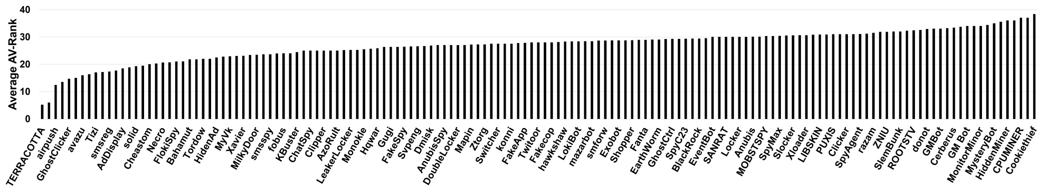


Fig. 5. The detected ratio for each malware family

of most of the anti-virus engines. Specifically, 100 samples in this family are flagged by less than 15 engines on VirusTotal and all samples are flagged by no more than 25 engines.

5.2 Measurement of Academic Techniques

Then, we further measure whether the existing malware detection models that can be adversely affected by the lack of clear understanding of the latest malware dataset. By resorting to the existing work published in top-tier security and software analysis venues, we select three widely-adopted tools for evaluation. (1)**Drebin** [42] is a lightweight method for detection of Android malware. It extracts the features (such as permissions, Android components, API calls, network addresses, etc.) from an app's code and Manifest file. Drebin further feeds all the features into Support Vector Machine (SVM) for malware detecion. (2)**CSBD** [40] builds the Control-Flow Graph(CFG) of app's bytecode, extracts the textual features and uses machine learning to classify. Besides, CSBD provides various machine learning classifier algorithms, including SVM, Random Forest, RIP-PER and C4.5. Their work suggested the Random Forest classifier had the best result. Thus we chose Random Forest classifier. (3)**MamaDroid** [56], a static-analysis-based system that abstracts app's API calls to their class, package, or family, and builds a model from their sequences obtained from the call graph of an app as Markov chains. In our experiment, we use the family mode for malware detection.

5.2.1 Evaluation Setup. Our evaluation is based on the following datasets and configurations.

Datasets. We use the following three datasets for evaluation. *(1) MalGenome*, the most popular malware datasets in our community, has 1,260 malware samples collected between 2010-2011. *(2) Benign Apps.* We collected and downloaded 5,000 benign apps from Androzoo [41]. *(3) MalRadar*, our crafted dataset contains 4,534 labelled malicious apps from security reports. They are distributed in 2014-2021 and can represent the up-to-date Android malware.

Configurations. We designed three groups of comparative experiments.

- **Experiment A**: We trained the three selected tools on *MalGenome* and *Benign datasets.* To eliminate the bias introduced by the unbalanced distribution of dataset, we set the proportion of malicious apps and benign apps as 50% and 50%. The default ratio of training set and test set for Drebin and CSBD is 70% and 30%, so we followed this splitting method to evaluate these three tools.
- **Experiment B**: We used the model trained in Experiment A (MalGenome & Benign) to predict samples in MalRadar, in order to evaluate whether it can detect new malware.
- **Experiment C**: We leveraged the report release dates to build a timeline and used samples from previous years to predict samples from later years in MalRadar. This more granular experiment can help us understand the capability of old malware samples to identify new malware samples as time progresses. Due to the small number of samples collected in some years (2014, 2015 and 2021), we took 2016, 2017, 2018 and 2019 as the division year respectively, i.e., the samples from 2016/2017/2018/2019 and earlier were used for training and the samples from 2017/2018/2019/2020 and later were used for testing.

Table 8. Measurement of academic techniques.

Exp A: MalGenome & Benign Dataset					Exp B: Train (MalGenome) Test (MalRadar)	
	Precision	Recall	F-Measure(F1)	Accuracy		Recall
Drebin	0.98	0.99	0.98	0.98	Drebin	0.68
CSBD	0.98	0.95	0.97	0.96	CSBD	0.58
MamaDroid	0.88	0.85	0.86	0.86	MamaDroid	0.32

Exp C: Train (samples from prior years in MalRadar) Test (samples from later years in MalRadar)				
	Recall (2016-division)	Recall (2017-division)	Recall (2018-division)	Recall (2019-division)
Drebin	0.64	0.84	0.85	0.92
CSBD	0.04	0.09	0.29	0.36
MamaDroid	0.22	0.39	0.78	0.75

5.2.2 Detection Results. Table 8 shows the results. The result of *Experiment A* suggests the high precision and recall of the selected three tools on MalGenome dataset. Drebin achieves the best result, with a F1-score of 98%. In *Experiment B*, we apply the model trained using MalGenome dataset to detect the malware in MalRadar. It is surprising to see that, all of these three models achieve really poor results. For example, MamaDroid can only detect 32% of the samples. It suggests that machine learning models trained using the out-dated samples in MalGenome cannot well flag the more evasive samples in MalRadar. Besides, the results of *Experiment C* present the recall of detecting later samples with earlier samples under different division years (samples in and before the division year act as the training set and samples after the division year act as the test set). It can be seen that the performance of detecting newer malware samples using older malware samples is quite poor for CSBD, with recall ranging from 4% to 36%. Drebin and MamaDroid achieved relatively good results but were far from their upper bounder (if refer to the results in Experiment A). This indicates the rapid evolution of malware and the great challenge for existing academic techniques to resist this evolution. Besides, there seems an upward trend in the recalls as the division year draws near. This suggests, to some extent, that adding newer samples to the training set can improve the detection performance of the model. This further demonstrates the imperative for the research community to update the malware datasets. As aforementioned, we are not intended to propose techniques for detecting malware in MalRadar or study the model aging problem, while we only want to show that *existing research efforts can be adversely affected by the lack of clear understanding of the latest mobile malware trends.*

Observation #7: *The commercial anti-virus engines and the widely-used techniques proposed in the academia cannot well identify the malware in the new era (at least cannot achieve the claimed promising results). As Android malware is evolving all the time and the malware in the new era are greatly different from the old ones, it poses more challenges to achieve a promising result.*

6 THREAT TO VALIDITY

Despite the encouraging contributions, this work has three potential threats to validity. First, our observations rely heavily on commercial reports from security companies. However, these reports are likely to be skewed towards the types of clients that security companies have. For example, quite a lot reports/samples in our analysis involve SMS abuse, but this does not mean that SMS abuse is more prevalent in the wild. The second threat relates to the definition of repackaging. In this work, we used the same methodology as in MalGenome (if a sample shares the same package name with an app in the official Market, we then manually compare the difference) to identify the repackaging for samples that were unmentioned by security reports. Although this ensures comparability with MalGenome, there is a real limitation, since it is possible that an attacker piggybacks a rider into a benign app and changes the package name. Considering this, we performed an additional check for several samples being flagged as repackaging in the reports. Specifically, if the report gives both the repackaged app and the original app, we check whether their package names are the same. It turned out that their package names remained the same. We suspect that the repackaged apps keeping the package name unchanged may be a trick to deceive users, i.e., make users easier to believe and download it. However, as it is difficult to identify the targeted/original apps for each repackaged app, we are not sure if there are other cases in MalRadar where the package name has changed. Third, the implementation of MalRadar's automated pipeline seems somewhat straightforward and still requires manual effort. This is because we did not focus too much on the innovative and technical aspects of the tool design in this study. In the future, we will try some advanced techniques (e.g., Natural Language Processing) to improve our automated collection/labelling approach.

7 RELATED WORK

In this section, we discuss some related work of building the Android malware dataset and detecting the Android malware.

7.1 Android Malware Dataset

There are a number of Android malware datasets in our community available for researchers. MalGenome [73] was the first available dataset to the community, which was created in 2011. They manually examined the existing security reports and collected the malware sample. Wet et al. [69] created AMD dataset in 2018, which contains 24,650 malware samples from various sources. Wang et al. [67] collected the removed apps from Google Play and identified the malware to create RmvDroid. Besides, there are some other datasets created using the similar approach, such as Drein [42], Piggybacking [52], Kharon [51], Andrubis [53], etc. AndroZoo [41] is a growing app repositories with providing the detection results from VirusTotal. However, researchers have to define their heuristics (e.g., VirusTotal thresholds) to flag malware when using AndroZoo [58]. As aforementioned, they face the limitations including ad-hoc labelling method, out-dated samples, threshold issue, etc.

7.2 Malware Detection

Many research efforts were focused on Android malware detection in our research community, mainly could be classified as signature-based approaches [47, 48, 71, 72], behavior-based approaches [43, 45, 60, 62], and machine-learning based approaches [44, 57, 59, 70], etc. Besides, a number of studies have analyzed the sustainability (concept drift) and fairness issues of machine learning based Android malware detection [50, 58]. For example, TESSERACT considers two kinds of bias, including 1) spatial bias caused by distributions of training and testing data that are not representative of a real-world deployment and 2) temporal bias caused by incorrect time splits of training and testing sets, leading to impossible configurations. However, it does not consider the dataset bias introduced by the malware labelling method. As aforementioned, a reliable and up-to-date Android malware dataset is critical to evaluate the effectiveness of these approaches.

8 CONCLUSION

In this paper, we have made efforts to create MalRadar, a reliable and up-to-date Android malware dataset by collecting malware samples from security reports released by leading security companies and security researchers. We then performed a systematic study of MalRadar from a number of perspectives, and will release all the labelled features and behaviors of the apps with MalRadar to the community. Based on the labelled characteristics, we further studied the evolution of Android malware across the last decade, and reveal a number of trends. Finally, we used the crafted dataset as the ground truth to measure the effectiveness of existing commercial anti-virus engines and malware detection techniques. We believe that our research efforts can positively contribute to the community, help improve existing malware detection techniques, and boost a series of research studies on mobile malware detection and analysis.

ACKNOWLEDGMENT

We sincerely thank our shepherd Prof. Niklas Carlsson (Linkoping University) and all the anonymous reviewers for their valuable suggestions and comments to improve this paper. This work was supported by the National Natural Science Foundation of China (grants No.62072046, 61873069, 62102042, 61902395) and Hong Kong RGC Projects (PolyU15223918, PolyU15224121). Xuanzhe's work was supported by Alibaba Group through Alibaba Innovative Research (AIR) Program.

REFERENCES

[1] 2016. Android banking trojan masquerades as Flash Player and bypasses 2FA. https://www.welivesecurity.com/2016/03/09/android-trojan-targets-online-banking-users/.

[2] 2016. RuMMS: The Latest Family of Android Malware Attacking Users in Russia Via SMS Phishing. https://www.fireeye.com/blog/threat-research/2016/04/rumms-android-malware.html.

[3] 2017. Analyzing Xavier: An Information-Stealing Ad Library on Android. https://blog.trendmicro.com/trendlabs-security-intelligence/analyzing-xavier-information-stealing-ad-library-android/.

[4] 2017. Coin Miner Mobile Malware Returns, Hits Google Play. https://blog.trendmicro.com/trendlabs-security-intelligence/coin-miner-mobile-malware-returns-hits-google-play/.

[5] 2017. Dvmap: the first Android malware with code injection. https://securelist.com/dvmap-the-first-android-malware-with-code-injection/78648/.

[6] 2017. Exobot - Android banking Trojan on the rise. https://www.threatfabric.com/blogs/exobot_android_banking_trojan_on_the_rise.html.

[7] 2017. GhostClicker Adware is a Phantomlike Android Click Fraud. https://blog.trendmicro.com/trendlabs-security-intelligence/ghostclicker-adware-is-a-phantomlike-android-click-fraud/.

[8] 2017. New WannaCry-Mimicking SLocker Abuses QQ Services. https://blog.trendmicro.com/trendlabs-security-intelligence/new-wannacry-mimicking-slocker-abuses-qq-services/.

[9] 2017. The Strange Case of Play Policy for Copyright and Security. https://www.fortinet.com/blog/threat-research/the-strange-case-of-play-policy-for-copyright-and-security.

[10] 2017. Toast Overlay Weaponized to Install Several Android Malware. https://blog.trendmicro.com/trendlabs-security-intelligence/toast-overlay-weaponized-install-android-malware-single-attack-chain/.

[11] 2017. User Beware: Rooting Malware Found in 3rd Party App Stores. https://blog.trendmicro.com/trendlabs-security-intelligence/user-beware-rooting-malware-found-in-3rd-party-app-stores/.

[12] 2017. ZNIU: First Android Malware to Exploit Dirty COW. https://www.trendmicro.com/en_us/research/17/i/zniu-first-android-malware-exploit-dirty-cow-vulnerability.html.

[13] 2018. Analysis of Smoke Loader in New Tsunami Campaign. https://unit42.paloaltonetworks.com/analysis-of-smoke-loader-in-new-tsunami-campaign/.

[14] 2018. Fake Banking App Found on Google Play Used in SMiShing. https://www.trendmicro.com/en_us/research/18/k/fake-banking-app-found-on-google-play-used-in-smishing-scheme.html.

[15] 2018. Monero-Mining HiddenMiner Android Malware Can Potentially Cause Device Failure. https://blog.trendmicro.com/trendlabs-security-intelligence/monero-mining-hiddenminer-android-malware-can-potentially-cause-device-failure/.

[16] 2018. Windows, Android Users Targeted by Maikspy Spyware. https://www.trendmicro.com/en_us/research/18/e/maikspy-spyware-poses-as-adult-game-targets-windows-and-android-users.html.

[17] 2018. XLoader Android Spyware and Banking Trojan Distributed via DNS Spoofing. https://blog.trendmicro.com/trendlabs-security-intelligence/new-wannacry-mimicking-slocker-abuses-qq-services/.

[18] 2019. Adware Campaign Identified From 182 Game and Camera Apps on Google Play and Third-Party Stores Like 9Apps. https://blog.trendmicro.com/trendlabs-security-intelligence/adware-campaign-identified-from-182-game-and-camera-apps-on-google-play-and-third-party-stores-like-9apps/.

[19] 2019. Google Play Apps Drop Anubis Banking Malware, Use Motion-based Evasion Tactics. https://blog.trendmicro.com/trendlabs-security-intelligence/google-play-apps-drop-anubis-banking-malware-use-motion-based-evasion-tactics/.

[20] 2019. Tracking down the developer of Android adware affecting millions of users. https://www.welivesecurity.com/2019/10/24/tracking-down-developer-android-adware/.

[21] 2020. 1Mobile. https://www.1mobile.com.

[22] 2020. APKiD. https://github.com/rednaga/APKiD.

[23] 2020. ApkPure. https://apkpure.com.

[24] 2020. Aptoide. https://www.aptoide.com.

[25] 2020. List of Mobile Security Vendors (Android). https://www.av-comparatives.org/list-of-mobile-security-vendors-android/.

[26] 2020. Publication Trends. https://app.dimensions.ai/discover/publication.

[27] 2020. Smartphone shopaholic. https://securelist.com/smartphone-shopaholic/95544/.

[28] 2021. Android malware found on Huawei's official app store. https://therecord.media/android-malware-found-on-huaweis-official-app-store/.

[29] 2021. Check Point. https://research.checkpoint.com/.

[30] 2021. ESET. https://www.welivesecurity.com/.

[31] 2021. FireEye. https://www.fireeye.com/blog.

[32] 2021. Fortinet. https://www.fortinet.com.

[33] 2021. GBHackers. https://gbhackers.com/.

[34] 2021. Kaspersky. https://securelist.com/.

[35] 2021. Koodous. https://koodous.com.

[36] 2021. Malwarebytes. https://blog.malwarebytes.com.

[37] 2021. McAfee. https://www.mcafee.com/blogs/.

[38] 2021. Qihoo. https://ti.360.net/blog/.

[39] 2021. TrendMicro. https://blog.trendmicro.com/.

[40] Kevin Allix, Tegawendé F Bissyandé, Quentin Jérome, Jacques Klein, Yves Le Traon, et al. 2016. Empirical assessment of machine learning-based malware detectors for Android. *Empirical Software Engineering* 21, 1 (2016), 183–211.

[41] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2016. Androzoo: Collecting millions of android apps for the research community. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 468–471.

[42] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. Drebin: Effective and explainable detection of android malware in your pocket.. In *Ndss*, Vol. 14. 23–26.

[43] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. 2011. Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. 15–26.

[44] Sen Chen, Minhui Xue, Zhushou Tang, Lihua Xu, and Haojin Zhu. 2016. Stormdroid: A streaminglized machine learning-based system for detecting android malware. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. 377–388.

[45] Santanu Kumar Dash, Guillermo Suarez-Tangil, Salahuddin Khan, Kimberly Tam, Mansour Ahmadi, Johannes Kinder, and Lorenzo Cavallaro. 2016. Droidscribe: Classifying android malware based on runtime behavior. In *2016 IEEE Security and Privacy Workshops (SPW)*. IEEE, 252–261.

[46] Feng Dong, Haoyu Wang, Li Li, Yao Guo, Tegawendé F Bissyandé, Tianming Liu, Guoai Xu, and Jacques Klein. 2018. Frauddroid: Automated ad fraud detection for android apps. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 257–268.

[47] Parvez Faruki, Vijay Ganmoor, Vijay Laxmi, Manoj Singh Gaur, and Ammar Bharmal. 2013. AndroSimilar: robust statistical feature signature for Android malware detection. In *Proceedings of the 6th International Conference on Security of Information and Networks*. 152–159.

[48] Yu Feng, Saswat Anand, Isil Dillig, and Alex Aiken. 2014. Apposcopy: Semantics-based detection of android malware through static analysis. In *Proceedings of ICSE 2014*. 576–587.

[49] Yangyu Hu, Haoyu Wang, Ren He, Li Li, Gareth Tyson, Ignacio Castro, Yao Guo, Lei Wu, and Guoai Xu. 2020. Mobile app squatting. In *Proceedings of The Web Conference 2020*. 1727–1738.

[50] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting concept drift in malware classification models. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 625–642.

[51] Nicolas Kiss, Jean-François Lalande, Mourad Leslous, and Valérie Viet Triem Tong. 2016. Kharon dataset: Android malware under a microscope. In *The {LASER} Workshop: Learning from Authoritative Security Experiment Results ({LASER} 2016)*. 1–12.

[52] Li Li, Daoyuan Li, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, David Lo, and Lorenzo Cavallaro. 2017. Understanding android app piggybacking: A systematic study of malicious code grafting. *IEEE Transactions on Information Forensics and Security* 12, 6 (2017), 1269–1284.

[53] Martina Lindorfer, Matthias Neugschwandtner, Lukas Weichselbaum, Yanick Fratantonio, Victor Van Der Veen, and Christian Platzer. 2014. Andrubis–1,000,000 apps later: A view on current Android malware behaviors. In *BADGERS Workshop*. IEEE, 3–17.

[54] Tianming Liu, Haoyu Wang, Li Li, Guangdong Bai, Yao Guo, and Guoai Xu. 2019. DaPanda: Detecting Aggressive Push Notifications in Android Apps. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 66–78.

[55] Tianming Liu, Haoyu Wang, Li Li, Xiapu Luo, Feng Dong, Yao Guo, Liu Wang, Tegawende F. Bissyande, and Jacques Klein. 2020. MadDroid: Characterising and Detecting Devious Ad Content for Android Apps. In *Proceedings of the Web Conference 2020 (WWW'20)*.

[56] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. 2017. MAMADROID: Detecting Android Malware by Building Markov Chains of Behavioral Models. In *Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS)*.

[57] Niall McLaughlin, Jesus Martinez del Rincon, BooJoong Kang, Suleiman Yerima, Paul Miller, Sakir Sezer, Yeganeh Safaei, Erik Trickel, Ziming Zhao, Adam Doupé, et al. 2017. Deep android malware detection. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. 301–308.

[58] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 729–746.

[59] Justin Sahs and Latifur Khan. 2012. A machine learning approach to android malware detection. In *2012 European Intelligence and Security Informatics Conference*. IEEE, 141–147.

[60] Andrea Saracino, Daniele Sgandurra, Gianluca Dini, and Fabio Martinelli. 2016. Madam: Effective and efficient behavior-based android malware detection and prevention. *IEEE Transactions on Dependable and Secure Computing* 15, 1 (2016), 83–97.

[61] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. Avclass: A tool for massive malware labeling. In *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 230–253.

[62] Mingshen Sun, Xiaolei Li, John CS Lui, Richard TB Ma, and Zhenkai Liang. 2016. Monet: a user-oriented behavior-based malware variants detection system for android. *IEEE Transactions on Information Forensics and Security* 12, 5 (2016), 1103–1112.

[63] Haoyu Wang, Hao Li, and Yao Guo. 2019. Understanding the evolution of mobile app ecosystems: A longitudinal measurement study of google play. In *The World Wide Web Conference*. ACM, 1988–1999.

[64] Haoyu Wang, Hao Li, Li Li, Yao Guo, and Guoai Xu. 2018. Why are Android Apps Removed From Google Play? A Large-scale Empirical Study. In *The 15th International Conference on Mining Software Repositories (MSR 2018)*.

[65] Haoyu Wang, Zhe Liu, Yao Guo, Xiangqun Chen, Miao Zhang, Guoai Xu, and Jason Hong. 2017. An explorative study of the mobile app ecosystem from app developers' perspective. In *Proceedings of the 26th International Conference on World Wide Web*. 163–172.

[66] Haoyu Wang, Zhe Liu, Jingyue Liang, Narseo Vallina-Rodriguez, Yao Guo, Li Li, Juan Tapiador, Jingcun Cao, and Guoai Xu. 2018. Beyond google play: A large-scale comparative study of chinese android app markets. In *Proceedings of IMC 2018*. 293–307.

[67] Haoyu Wang, Junjun Si, Hao Li, and Yao Guo. 2019. Rmvdroid: towards a reliable android malware dataset with app metadata. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 404–408.

[68] Liu Wang, Ren He, Haoyu Wang, Pengcheng Xia, Yuanchun Li, Lei Wu, Yajin Zhou, Xiapu Luo, Yulei Sui, Yao Guo, et al. 2021. Beyond the virus: a first look at coronavirus-themed Android malware. *Empirical Software Engineering* 26, 4 (2021), 1–38.

[69] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. 2017. Deep ground truth analysis of current android malware. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 252–276.

[70] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. 2014. Droid-sec: deep learning in android malware detection. In *Proceedings of SIGCOMM 2014*. 371–372.

[71] Mu Zhang, Yue Duan, Heng Yin, and Zhiruo Zhao. 2014. Semantics-aware android malware classification using weighted contextual api dependency graphs. In *Proceedings of the CCS 2014*. 1105–1116.

[72] Min Zheng, Mingshen Sun, and John CS Lui. 2013. Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware. In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 163–171.

[73] Yajin Zhou and Xuxian Jiang. 2012. Dissecting android malware: Characterization and evolution. In *2012 IEEE symposium on security and privacy*. IEEE, 95–109.

[74] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. 2020. Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines. In *Proceedings of USENIX Security 2020*.

Received February 2022; revised March 2022; accepted April 2022