A Large-scale Temporal Measurement of Android Malicious Apps:
Persistence, Migration, and Lessons Learned

Yun Shen
Norton Research Group

Pierre-Antoine Vervier
Norton Research Group

Gianluca Stringhini
Boston University

Abstract

We study the temporal dynamics of potentially harmful apps (PHAs) on Android by leveraging 8.8M daily on-device detections collected among 11.7M customers of a popular mobile security product between 2019 and 2020. We show that the current security model of Android, which limits security products to run as regular apps and prevents them from automatically removing malicious apps opens a significant window of opportunity for attackers. Such apps warn users about the newly discovered threats, but users do not promptly act on this information, allowing PHAs to persist on their device for an average of 24 days after they are detected. We also find that while app markets remove PHAs after these become known, there is a significant delay between when PHAs are identified and when they are removed: PHAs persist on Google Play for 77 days on average and 34 days on third party marketplaces. Finally, we find evidence of PHAs migrating to other marketplaces after being removed on the original one. This paper provides an unprecedented view of the Android PHA landscape, showing that current defenses against PHAs on Android are not as effective as commonly thought, and identifying multiple research directions that the security community should pursue, from orchestrating more effective PHA takedowns to devising better alerts for mobile security products.

1 Introduction

Millions of malicious Android apps have been observed over the years [1], performing a variety of malicious activity from sending premium SMS messages [23], to displaying annoying advertisements [31], to enabling stalking [5]. Malicious apps on Android often come in the form of repackaged apps, where a useful Android app is modified to contain hidden malicious functionality to entice users into installing it [20, 31, 39]. To cover the variety of malicious apps that tar-

get Android, Google has coined the term *Potentially Harmful Apps* (PHAs).[1]

A large body of research has been published measuring the threat of PHAs on Android. Previous studies have mostly relied on crawling app markets to retrieve malicious applications [1, 23, 31, 34, 40]. Alternative approaches include downloading firmware from public repositories to study pre-installed Android apps [11] and setting up public analysis infrastructures relying on third parties to submit apps that they suspect are malicious [21]. These approaches then analyze the collected apps by either performing static or dynamic analysis. While useful to shed light on the functionalities of malicious Android apps, these approaches do not have visibility on the population of infected devices and on how users interact with PHAs. An alternative approach relied on users installing an app able to monitor network traffic on devices, looking for security and privacy sensitive information [26]. This solution solves the aforementioned problem, but it is challenging to recruit a large and representative population of users; in fact, previous studies relied on 11k users to perform their measurements [26]. A third approach that researchers followed is monitoring the network traffic of a mobile ISP and identifying malicious connections based on blacklist information [18]. This approach provides a real-time view of malicious activity from a large number of devices but is limited to monitoring connections to known malicious hosts. Additionally, this method is limited by the pervasive use of encryption, and does for example allow to observe when a device connects to an app store, but not to inspect what specific PHA a user is installing.

In this paper, we present the first large-scale study to understand the temporal dynamics of PHA installations on Android. We collect anonymized information about PHA installations from users who installed a popular mobile security product and opted into data collection. Between 2019 and 2020 we observed over 8.8M PHAs installed on over 11.7M devices from

---

[1] https://developers.google.com/android/play-protect/potentially-harmful-applications

Dataset	Data	Count
Mobile PHA detection log (01/2019 - 02/2020)	Total records	3.2B
	Days	416
	Countries and regions	201
	Devices	11.7M
	Distinct app names	2.3M
	Distinct app SHA2s	8.8M
VT	Total reports	8.8M
	PHA SHA2s (detections $\geq$ 4)	7M
	Singleton SHA2s (w/o family)	1.3 M
	PHA SHA2s w/ family	5.7 M
	PHA families	3.2K

Table 1: Summary of datasets.

across the globe. This data allows us to develop a number of metrics and answer the following key research questions:

How long do devices stay infected with PHAs? Mobile security products on Android run as regular users without root privileges and are therefore limited in the actions they can take after they detect a malicious program. Typically, they just raise an alert informing the user about it, and relying on them uninstalling the malicious app. Our study shows that users do not act promptly on these alerts, and that PHAs persist on devices for approximately 20 days once detected.

How long do PHAs survive on app markets? By observing millions of mobile devices installing malicious apps from app stores, we can estimate when a certain PHA is removed from the store. We find that, on average, PHAs persist on Google Play for 77 days, while they persist on alternative marketplaces for 34 days on average.

Do PHAs migrate to other app markets once removed? We observe 3,553 PHAs that exhibit inter-market migration. However, those PHAs have on average shorter lifespans in these markets compared to the average persistence time.

Do PHAs persist on devices for longer if migrating from backup/clone services? Android devices allow users to backup their apps and automatically install them on a new device when the user gets a new phone. We discover that these PHAs on average persist on these devices for longer periods. For example, we find 14K PHAs that migrated to 35.5K new Samsung devices by using the Samsung smart switch mobile app (`com.sec.android.easyMover`). These apps persist in the new devices for 93 days on average.

Implications for Android malware research. Our study has a number of implications for the computer security research community. We show that malicious apps can survive for long periods of time on app markets, and that the Android security model severely limits what mobile security products can do when detecting a malicious app, allowing PHAs to persist for many days on victim devices. Furthermore, our results show that the current warning system employed by mobile security programs is not effective in convincing users to promptly uninstall PHAs. This could be due to usability issues such as alert fatigue [28], and calls for more research in this space. We also show that malicious app developers
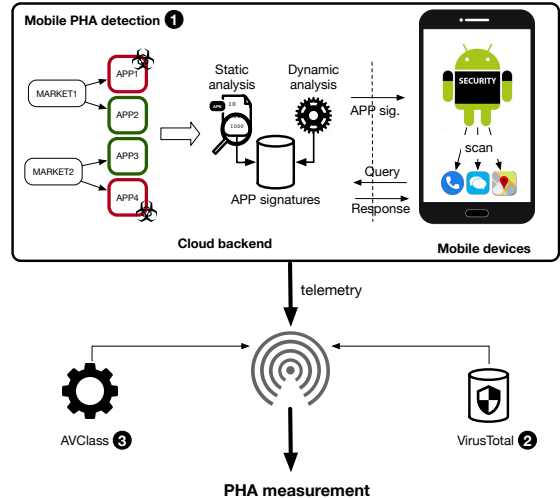

Figure 1: Data Collection Architecture.

often move their PHAs to alternative marketplaces after they have been removed. This suggests that an effective mitigation strategy should include cooperation between multiple marketplaces.

2 Datasets

This section summarizes our data collection approach (see Figure 1) and the datasets used in this study (see Table 1).

Mobile PHA reputation data ❶. In this paper, we use mobile PHA reputation data collected from real-world Android devices by NortonLifeLock's mobile security product, which covers over 200 countries and regions in the world. Similar to the device geolocation distribution discussed in Kotzias *et al.* [16], 25% of the devices used in our study were from the United States, 28% of the devices were from European countries, and 31% of the devices were from the APJ area (although this distribution was skewed towards Japan and India).

NortonLifeLock has an elastic infrastructure to collect and carry out systematic static (e.g., flow and context-sensitive taint analysis, fine-grained permission analysis) and dynamic analysis (e.g., apk fuzzing, UI-automation, examining network traces, behaviors, etc. in a sandbox environment) of mobile apps from multiple markets and partners at scale. During the process, nefarious activities and their triggering conditions (such as reflection, dynamic code loading, native code execution, requesting permissions not related to its advertised description, etc.) are analyzed and fingerprinted. NortonLifeLock employs state-of-the-art commercial products to deal with challenges such as emulator/motion evasion, obfuscated code/libraries, and other evasive techniques, as well as to trigger the critical execution paths in apps. The results are then included in NortonLifeLock's detection engine and deployed in its mobile security product to scan and identify suspicious

apps on the mobile endpoints. NortonLifeLock also builds machine learning models from the static and dynamic analysis results of known PHAs and applies these models to inspect unknown or low-prevalence apps. Also, apps are periodically re-inspected by the analysis infrastructure.

At runtime, the mobile security product periodically scans newly installed apps on a device and can perform a full device scan when requested by the end-user. When having network access, the security engine queries a cloud backend to obtain the verdicts of the apps installed on a device. The query contains certain metadata including timestamp, app hash, package name, certificate information, etc. The response from the backend includes the reputation scores of the on-device apps together with other proprietary data to guide further actions. When network access is not available, the security engine leverages the locally stored signatures to scan and identify suspicious apps on the mobile endpoints. The corresponding scan metadata will then be sent back once network access is restored.

From the backend telemetry data lake, we extract the following information: anonymized device identifier, device country code, detection timestamp, app SHA2, app package name, and installer package name. This way, we are able to tell the time at which a PHA is detected, on which device it is installed, and which package installed it. We collected 416 days of detection data between January 1, 2019 and February 20, 2020. On average, we collect 8M raw events daily (i.e., 3.2B events in total). Note that to carry out the temporal measurement, we only select apps (per SHA2) that we observe at least twice on the same device. This way, we can reliably calculate their lifespan both on-device and in-market (see Section 3.2). In total, our dataset covers 2.3M unique package names with 8.8M unique SHA2s from 11.7M devices. We provide a detailed discussion of bias potentially incurred by our dataset in Section 9.

VirusTotal ❷. Note that different security companies have different policies when flagging PHAs (especially adware). That is, a PHA flagged by NortonLifeLock that collaborated on this study may not have the consensus from other security companies. To minimize false positives and bias potentially incurred by our dataset, we query the 8.8M SHA2s corresponding to the PHAs in our dataset on VirusTotal. We consider an app as a PHA if VirusTotal returns a minimum of four detections in this paper. This is in line with the best practices recently proposed in the malware research community [16, 41]. We refer the audience to Kotzias *et al.* [16] and Zhu *et al.* [41] for in-depth analysis of the impact of different detection threshold values of VirusTotal reports. In total, we identify 7M unique malicious SHA2s, and 3.5K PHA families.

AVclass ❸. In our study, reliable PHA labeling is a necessary condition to guarantee the quality of malware family attribution. To this end, we use AVclass [29] to extract family information from AV labels. This tool selects the top ranked family corresponding to a majority vote from the VirusTotal report of a given PHA, effectively removing noise in the labels. In total, the observed PHAs belong to 3.2K families. Not all PHAs are equally harmful. While some apps are clearly malicious (i.e., mobile malware including ransomware, Trojans, spyware, etc), others are merely an annoyance to users (e.g., adware). Google groups these apps into Mobile unwanted software (MUwS) as "apps that are not strictly malware, but are harmful to the software ecosystem" [12]. To investigate differences in how malware and MUwS behave, we use the feature provided by AVclass to classify a sample as Mobile unwanted software (MUwS) or mobile malware (see Section 5). Note that EUPHONY [13] also mines AV labels and analyzes the associations between all labels given by different vendors to unify common samples into family groups. Due to their comparable labeling accuracy in terms of family attribution and the lower memory required by AVclass, we opt for AVclass in this paper.

Data distillation and measurement data selection. To study the provenance of PHAs, and in particular, which marketplaces they are installed from, we need to collect information on the installer package names of the detected PHAs. The mobile security product uses the Android API to record a PHA's installer package name when a detection event is triggered. However, due to the well known fragmentation from Android device manufacturers and limitations of our measurement infrastructure (e.g., we cannot identify an installer package's certificate via Android API), it is hard to accurately extract and attribute the installer packages of all detected PHAs. For instance, if an app was already installed on a device before the observation period started, our approach would not be able to attribute it to the app that installed it. Similarly, if an updated version of an existing PHA was installed, this would be identified as being installed by an update component and not by a marketplace (e.g., `com.google.android.packageinstaller`). To mitigate this issue, we first identify 3.7M out of 11.7M devices that have at least one PHA installed. We then distill the aforementioned datasets by selecting 2.46M devices in which we can attribute their on-device PHAs to the respective installer packages with high confidence. In total, we identify 197K PHAs from 2.46M devices that we use in Section 6 and 7 to study the dynamics between PHA, devices, and markets. These PHAs account for 22% of all installations recorded by our dataset during the observation period. We provide a detailed discussion on the limitations of this approach in Section 9.

Ethics and Data Privacy. The data used in this paper is privacy sensitive. NortonLifeLock offers end users the possibility to explicitly opt-in to its data sharing program to help improve the security product's detection capabilities. This dialog is shown during the setup process when the app is run

Notation	Description
$p \in \mathbf{P}$	a PHA
$d \in \mathbf{D}$	a device
$m \in \mathbf{M}$	a market
$f \in \mathbf{F}$	a PHA family
x_{i^y}	x on/in y, e.g., p_{i^d} denotes a PHA p_i detected on device d.
(F)	first seen timestamp, e.g., $p_{i^d}^{(F)}$ denotes first seen timestamp of a PHA p_i on device d.
(L)	last seen timestamp, e.g., $p_{i^m}^{(L)}$ denotes last seen timestamp of a PHA p_i in market m.
δ_{x^y}	lifespan of x on/in y, e.g., $\delta_{p_i d}$ denotes the lifespan of p_i on a device d

Table 2: Summary of the notations used in this paper. We use lowercase letters to denote an item and bold uppercase letters to denote sets.

Figure 2: Abstract model of the relations between PHAs, installers, devices, and markets as observed in our dataset.

for the first time, and it informs the end-user about the purpose of the telemetry collection, and how the global privacy policy of NortonLifeLock safeguards the data. For instance, the license agreement specifies that the telemetry "is processed for the purposes of delivering the product by alerting you to potentially malicious applications, malware, and links" and "for the purpose of understanding product usage to further develop and improve the product performance as well as telemetry." Since the analysis performed in this paper allows the community to get a better understanding of the Android PHA ecosystem and guide mitigation techniques, this falls under the primary use of the data that users agreed to. The telemetry data collection, storage, and process are guarded by NortonLifeLock's rigorous privacy policies. To preserve the anonymity of users and their devices, client identifiers are anonymized and it is not possible to link the collected data back to the users and the mobile devices that originated it. Also, NortonLifeLock does not track the devices or profile user behavior nor has the capability to inspect network data. For our measurement study, the anonymized device identifier is only used to compute device-based prevalence rates. As such, we are not using any PII and the risks to the users are minimal.

3 Approach

In this section, we first introduce the overall relationships among PHAs, installer packages, devices, and markets. We then describe our overall measurement design philosophy and methods together with examples.

3.1 Relationships

For the reader's convenience, we summarize the notations introduced here and in the following sections in Table 2. We provide a detailed description of the relations observed in our dataset to form the foundation of our measurements in the rest of the paper. Figure 2 shows an example to illustrate the complex dynamic relations among PHAs $\mathbf{P}$, installer packages Ψ, devices $\mathbf{D}$, and markets $\mathbf{M}$, coupled with a timeline. Each
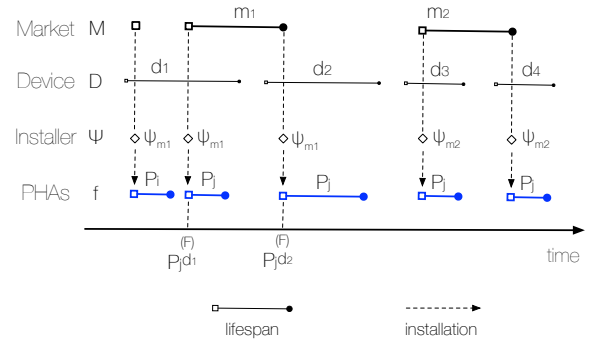
device d can have multiple PHAs installed (e.g., d_1 has two PHAs p_i and p_j in Figure 2). A PHA p_j can be present in multiple devices (e.g., p_j is installed in all four devices). Additionally, multiple PHAs can belong to a PHA family. For example, as we can see in Figure 2, $\mathbf{P}_f$ includes p_i in d_1 and p_j in all four devices. In addition, the Android API allows the mobile security product to retrieve the package name (i.e., ψ) of the application that installed a PHA. This enables us to identify which market a PHA came from if the package name of ψ matches the name of the market. For example, p_j on device d_1 is installed by a package ψ_{m_1} from market m_1 at timestamp $p_{j^{d_1}}^{(F)}$ (see Figure 2). Aggregating all installation events of the *same* PHA p_i in all devices $\mathbf{D}$, we can estimate the lifespan $\delta_{p_i m}$ in market m as $[p_{j^{d_1}}^{(F)}, p_{j^{d_2}}^{(F)}]$ (see Figure 2).

3.2 Design Philosophy

Measuring the in-market presence of PHAs (e.g., how fast PHAs are removed) is a challenging task as we are not the app market owners. One solution is to crawl known app markets and track all apps on a daily basis [35]. However, crawling results cannot be correlated with the device installation data since not all markets offer precise device installation information. In this study, we adopt an outside-in design philosophy to perform our market presence measurements. That is, we treat mobile devices as *sensors* and their PHA installation events as the *probing results* of a PHA's existence. We then use the information on the installer packages of apps to identify the origin markets of installed PHAs (see the above section for relations). By correlating this information with on-device detection timestamps we can calculate PHA in-market persistence and prevalence in a non-intrusive, outside-in way. Similarly, we can also calculate PHA on-device persistence using the detection timestamps. In this study we use different metrics to study the PHA ecosystem along three axes: on-device persistence, in-market persistence, and PHA migration across markets. In this section, we define the metrics that we will later use to measure these three aspects.

3.2.1 Measurement of PHA On-device Persistence

The mobile security product runs periodically in the background and sends telemetry data to the backend if PHAs are detected. If a PHA was not removed from the device after the user was displayed an alert, the mobile security product records this recurrent detection at different timestamps until the PHA is removed from the device. Given this series of detection events, we are able to tell the first seen and last seen timestamps of a PHA p_i on a device d, consequently enabling us to estimate the lifespan of p_i on a device d (i.e., $\delta_{p_{id}}$). Following this observation, we use Eq 1 to measure the persistence period a PHA family f on a device d.

$$persistence(f,d) = \sum_{p_i \in \mathbf{P}_f} (\delta_{p_{id}})/|\mathbf{P}_f| \qquad (1)$$

That is, we calculate the mean lifespan of all PHAs belonging to a family f on device d. For example, in Figure 2, family f has two PHAs (p_i and p_j) on device d_1, hence $persistence(f,d_1) = (\delta_{p_{id_1}} + \delta_{p_{jd_1}})/2$. We then use Eq 2 to measure the mean persistence period per PHA family f on all devices $\mathbf{D}$.

$$persistence(f,\mathbf{D}) = \sum_{d \in \mathbf{D}} persistence(f,d)/|\mathbf{D}| \qquad (2)$$

For example, family f has presence in all four devices in Figure 2. Following Eq 2, we can calculate $persistence(f,\mathbf{D})$ as $[persistence(f,d_1) + persistence(f,d_2) + persistence(f,d_3) + persistence(f,d_4)]/4$.

3.2.2 Measurement of PHA In-market Persistence

Given a single device d, when the mobile security product detects a PHA on the mobile device, it also records the installer package name of this PHA. Correlating this with the official package names of the markets, we can identify if a PHA was installed from a certain market m at a certain timestamp. For example, if we observe the installer package name of a PHA is `com.android.vending`, we can tell that this PHA comes from the Google Play store. Note that malicious apps can impersonate the legitimate apps on Android devices (e.g., `com.android.vending` may not be the legitimate Google Play app). To avoid false attributions, we check the detection telemetry data of the same device and verify if any detection records match the same package names of the known marketplaces. By doing so, we are able to verify the legitimacy of the market apps in this measurement study. We provide a detailed discussion of the limitations of this approach in Section 9. Note that first seen timestamp of a PHA on device d can reliably prove that a PHA exists in a market at the time of installation. By aggregating the first detection events of a PHA p_i across all devices $\mathbf{D}$, we can represent a PHA's in-market appearances using Eq 3.

$$\Omega_{p_im} = \{p_{i d_j}^{(F)}\}, \forall d_j \in \mathbf{D}, p_{i d_j} \in \mathbf{P}_{i^m} \qquad (3)$$

Essentially, Ω_{p_im} represents a series of timestamps where p_i was first seen on all devices D. Take the relations in Figure 2 as an example, we have two detections of a PHA p_j respectively on d_1 and d_2 installed from market m_1. In turn, we have $\Omega_{p_{j}m_1} = \{p_{j d_1}^{(F)}, p_{j d_2}^{(F)}\}$. Following the above observation, we use Eq 4 to measure the persistence period of a PHA p_i in a market m.

$$persistence(p_i,m) = max(\Omega_{p_im}) - min(\Omega_{p_im}) \qquad (4)$$

It is straightforward to observe $persistence(p_j,m_1) = p_{j d_2}^{(F)} - p_{j d_1}^{(F)}$ following Eq 3 and Eq 4. Note that we rely the on-device detection to measure a PHA's in-market persistence. It is possible that a PHA still exists in a market but our dataset did not reflect its existence. Consequently, we measure the *lower bound* of the PHA in-market persistence. Finally, we use Eq 5 to measure the persistence period a PHA family f in a market m.

$$persistence(f,m) = \sum_{p_i \in \mathbf{P}_f} persistence(p_i,m)/|\mathbf{P}_f| \qquad (5)$$

3.2.3 Measurement of PHA Inter-market Migration

Recall that the mobile security product records that a PHA p was installed on a device d at a timestamp t by an installer package ψ. By aggregating the telemetry data about a specific PHA p and mapping its installer package names to marketplaces across all devices $\mathbf{D}$, we can track the appearance of a PHA p_i across all marketplaces $\mathbf{M}$. Take PHA p_j in Figure 2 for example, it was detected in four devices (d_1, d_2, d_3, and d_4) from two marketplaces (m_1 and m_2). Following Eq 3, the lifespan of p_j in m_1 and m_2 are respectively $\delta_{p_jm_1} = [min(\Omega_{p_jm_1}), max(\Omega_{p_jm_1})]$ and $\delta_{p_jm_2} = [min(\Omega_{p_jm_2}), max(\Omega_{p_jm_2})]$. As we observe in Figure 2 that $max(\Omega_{p_jm_1})$ is less than $min(\Omega_{p_jm_2})$, we define that a PHA p_j exhibits *inter-market migration* from m_1 to m_2. Following the observation, we use Eq 6 to represent the appearances of a PHA p_i across the marketplaces $\mathbf{M}$.

$$appearance(p_i,\mathbf{M}) = \{\delta_{p_im}\}, \forall m \in \mathbf{M} \qquad (6)$$

Note that each δ_{p_im} is an interval (i.e., $[min(\Omega_{p_im}), max(\Omega_{p_im})]$). In turn, we sort $appearance(p_i,\mathbf{M})$ by $min(\delta_{p_im})$, then identify sequentially non-overlapping intervals from $appearance(p_i,\mathbf{M})$ to measure PHA inter-market migration across the marketplaces $\mathbf{M}$.

3.3 Right Censored Data

Censoring occurs when incomplete information is available about the survival time of some individuals. Recall that our observation period is between January 1, 2019 and February 20,

Rank	Family	Total SHA2s	Active SHA2s/Month (01/19 - 02/20)	# month ≥ avg
1	jiagu (U)	671K		7
2	smsreg (M)	438K		2
3	hiddad (U)	308K		6
4	airpush (U)	164K		6
5	revmob (U)	132K		6
6	dnotua (U)	105K		6
7	dowgin (U)	87K		6
8	leadbolt (U)	75K		7
9	mobidash (U)	74K		5
10	kuguo (U)	72K		6
11	locker (M)	60K		6
12	ewind (M)	57K		7
13	secapk (U)	51K		7
14	inmobi (U)	44K		5
15	tencentprotect	44K		5
16	koler (M)	42K		7
17	domob (U)	40K		8
18	secneo (U)	29K		7
19	autoins (M)	25K		6
20	datacollector (M)	15K		7

Table 3: Summary of the temporal patterns of the top 20 PHA families. These families are ordered by the total number of SHA2s. **M** denotes Malware and **U** denotes MUwS/Adware.

2020. There exist a number of PHAs that we cannot observe if they have been removed from the markets after our study ends on February 20, 2020. Such PHA data is defined as right censored in survival analysis [14]. In our study, we assume that censoring is independent or unrelated to the likelihood of developing the event of interest (i.e., PHA removal). We, therefore, keep these right censored data to avoid estimation bias. More details can be found in Section 6.

4 Temporal Characteristics of PHA Families

Miscreants either repackage multiple apps with the same malicious code or modify their code to avoid being detected by security measures [23, 39]. These related malicious apps are commonly referred to as *families*. In this section, we study the temporal patterns of PHA families (e.g., do we see the same top families all the time or do they gradually reduce their operation due to increased detection efforts from mobile security companies?). As discussed in Section 2, we use AVclass [29] to group apps into families. Table 3 shows the temporal prevalence of the top 20 PHA families and summarizes our findings. These top 20 PHA families are detected in 2.01 million devices, approximately 67% of the aforementioned 3.7M devices. Recall that we select PHAs (per SHA2) that we observe at least twice on the same device to carry out the measurements (see Section 2), hence our dataset is purposely designed to measure the temporal behavior. Our results are consistent with the most recent Android PHA device prevalence study [16]: 15 of the 20 PHA families in Table 3 were also among the top 20 PHA families found by [16]. As we can see in Table 3, 16 out of the 20 top PHA families manage to exceed their average number of monthly active SHA2s for at least 6 months. Also, we see that majority of

Overall		Malware		MUwS	
# Devices	Avg. Persistence	# Devices	Avg. Persistence	# Devices	Avg. Persistence
3.7M	20.2 D	2.93M	20.3 D	1.97M	13.1 D

Table 4: Overall PHA on-device persistence.

the PHA families exhibit bell shaped temporal patterns. This shows that these PHA families may eventually reduce their operation due to increased detection efforts from mobile security companies and marketplaces. We will further analyze this finding in Section 6 (e.g., how rapid takedown can disrupt PHA operations) and, consequently, how miscreants may move their PHAs to alternative markets in Section 7. Besides, we notice that smsreg and smspay show an upward pattern towards the end of our observation period (i.e., January and February 2020). In light of the recent discussion of the limitation of SMS-based 2FA authentication[2], our findings indicate that the possibility of such breaches still exists in the wild and has attracted the attention of cybercriminals.

5 PHA On-Device Persistence

As we explained, mobile security products are limited by the Android security model, and they lack the ability to delete PHAs once they detect them. Instead, they typically inform the user about the newly discovered threat, asking them to manually remove the app. This leaves the question of how promptly users remove identified PHAs from their devices. In this section, we first study the PHA on-device persistence to understand how long PHAs can persist on devices once installed. We then study the consequences of PHA on-device persistence and, for example, whether this leads to additional PHA installations on devices.

5.1 On-Device Persistence of Different PHA Types

As discussed, not all Android PHAs are equally harmful, but some are merely annoying to users (MUwS). It is therefore possible that users will react differently when the security product informs them that they have installed malware compared to mobile unwanted software (MUwS), possibly not uninstalling the latter. To better understand this, we use AVClass [29] to distinguish mobile malware from MUwS among PHAs. We then follow the approach outlined in Section 3.2 and measure the overall PHA on-device persistence. Our findings are summarized in Table 4. We find that PHAs persist on devices for approximately 20 days once installed. On average, mobile malware can persist longer than MUwS (respectively 20.3 days and 13.1 days). It is surprising that end users do not promptly remove PHAs once detected. The prolonged persistence of PHAs on devices leaves a window of oppor-

---
[2]https://krebsonsecurity.com/2018/08/reddit-breach-highlights-limits-of-sms-based-authentication/

Family	Avg. Persistence	Max Persistence	# Devices (≤ Avg.)	# Devices (> Avg.)
jiagu	4.77 D	414.0 D	1132K	303K
smsreg	2.37 D	413.63 D	471K	34K
hiddad	5.54 D	415.08 D	611K	85K
airpush	20.9 D	413.5 D	183K	35K
revmob	3.54 D	413.82 D	354K	13K
dnotua	2.93 D	414.36 D	261K	12K
dowgin	7.24 H	412.11 D	239K	1K
leadbolt	5.74 D	413.45 D	243K	12K
mobidash	1.8 D	415.09 D	294K	10K
kuguo	6.18 H	408.89 D	155K	1K
locker	4.13 H	413.13 D	195K	322
ewind	8.72 D	414.11 D	118K	22K
secapk	12.98 H	412.03 D	215K	1K
inmobi	10.66 D	413.77 D	213K	23K
tencentprotect	5.27 D	413.58 D	173K	23K
koler	0.49 H	360.01 D	160K	1K
domob	1.87 D	409.91 D	174K	2K
secneo	3.32 D	413.15 D	157K	11K
autoins	22.45 D	414.0 D	174K	43K
datacollector	15.47 D	413.59 D	176K	54K

Table 5: Summary of the top 20 PHA family on-device persistence. **D** denotes days and **H** denotes hours.

5.2 PHA Family On-Device Persistence

We then follow the approach outlined in Section 3.2 to understand the on-device persistence of the top 20 largest PHA families ranked by their device prevalence ratios. Our findings are summarized in Table 5. It is interesting to observe that 15 out of the 20 top PHA families can persist on devices for several days, and only five PHA families are removed by end users in less than two days. For example, ewind, a Trojan family, persisted on 118K devices for up to 8.72 days on average and on 22K devices for even longer. This is interesting because it indicates that users choose not to uninstall the malicious app after they are warned by the mobile security product. In light of this, It is interesting to observe that end users removed locker/koler rapidly after detection. We can only speculate that the reason behind this might be the degradation in user experience (i.e., screen lockdown by locker, fake FBI warnings by koler). We hope that our findings would enable mobile security companies to devise effective notification systems to nudge the end users to remove PHAs upon detection, taking for example into account alert fatigue [28].

5.3 PHA Multiple-Instance Persistence

The fact that end users do not remove the detected PHAs promptly creates a window of opportunity (as shown in Table 4) that enables attackers to update the installed PHAs, install additional malicious apps without user consent, or entice them to install apps via full screen ads. We call this phenomenon *multiple instance persistence*. Figure 3 shows the
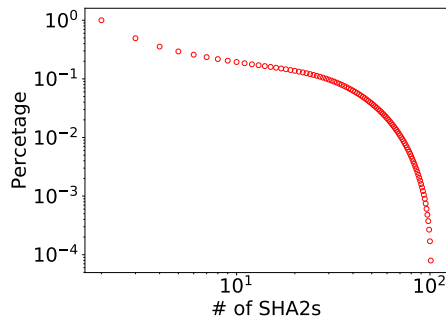

Figure 3: CCDF of the number of SHA2s on devices in our dataset (log scale).

Family	# Devices	# Multi-inst. Persistence Devices	Ratio
jiagu	1.33M	849K	0.64
smsreg	499K	237K	0.48
hiddad	670K	208K	0.31
airpush	214K	43K	0.2
revmob	367K	186K	0.51
dnotua	272K	82K	0.3
dowgin	241K	80K	0.33
leadbolt	255K	85K	0.33
mobidash	304K	112K	0.37
kuguo	155K	34K	0.22
locker	196K	53K	0.27
ewind	139K	17K	0.13
secapk	216K	75K	0.35
inmobi	235K	62K	0.27
tencentprotect	194K	42K	0.22
koler	161K	45K	0.28
domob	176K	45K	0.26
secneo	167K	34K	0.21
autoins	209K	33K	0.16
datacollector	211K	45K	0.21

Table 6: Summary of multiple-instance persistence per PHA family.

complementary cumulative distribution function (CCDF) of the number of unique PHAs installed on devices in our dataset. A large fraction of the devices that installed PHAs installs more than one during the observation period. For instance, 810K mobile devices (21.6% of 3.7M devices that have at least one PHA) installed more than 7 PHAs. In this section, we investigate to what extent the presence of a PHA on a device facilitates the installation of additional malicious components. Our findings are shown in Table 6. 18 out of the top 20 PHA families exhibit multiple-instance persistence on at least 20% of the mobile devices they infected. For example, 237K out of 499K mobile devices that installed PHAs from smsreg family have at least two other PHAs from the same family within the 14-month observation period. Even for locker and koler whereas the end users act swiftly (see Table 5), we observe 53K (27% of locker infected devices) and 45K (28% of koler infected devices) exhibiting multiple-instance persistence. Note that our data does not allow us to infer the causality relationship of PHA installations. Our results only

tunity during which attackers can cause harm to the victims and their devices (e.g., displaying intrusive full screen ads, collect private information, install additional malicious apps without user consent).

Market	# Total PHAs	#Apps	#Families	# Avg. Active PHAs	Active PHA (01/19 - 02/20)
Google Play (com.android.vending)	81K	56K	642	26K	
Huawei Market (com.huawei.appmarket)	24K	10K	175	3K	
Xiaomi Market (com.xiaomi.market)	11K	5K	226	2K	
Samsung Market (com.sec.android.app.samsungapps)	10K	5K	206	2K	
Bazaar Market (com.farsitel.bazaar)	5K	5K	74	1K	
Oppo Market (com.oppo.market)	3K	2K	143	462	

Table 7: Summary of active PHAs in the top 6 Android Markets.

Market	# Total PHAs	# Total Removed	%Removed	# Avg. Removed	PHA Removal (01/2019 - 01/2020)
Google Play	81K	74K	91.4%	5.28K	
Huawei Market	24K	22K	91.5%	1.58k	
Xiaomi Market	11K	9.8K	92.2%	705	
Samsung Market	10K	8.9K	91.3%	637	
Bazaar Market	5K	4.7K	95.4%	337	
Oppo Market	3K	3K	92.3%	223	

Table 8: Summary of PHAs removed by the top 6 Android markets.

demonstrate the fact that two PHAs from the same family **that** are installed on the same device are highly correlated.

6 PHA In-Market Persistence

In this section, we first quantify the active PHAs in six app markets, by leveraging the dataset of 197K apps for which we could reliably establish their market provenance (see Section 2). As discussed, this number is lower than the total number of PHAs installed for a number of reasons (e.g., PHAs that were already installed on the devices before our study started, and SHA2s that do not belong to newly installed apps but are rather updates), but it still covers 66% of all devices that installed any PHA and 22% of all PHA installations in our dataset. We then study how markets react to the presence of PHAs (e.g., how many PHAs the markets suspend or remove, etc). Finally, we study the PHA in-market persistence (i.e., how long can PHAs persist in different markets once published) and PHA in-market evolution (i.e., how PHAs may evolve to evade app vetting systems).

6.1 PHA In-Market Prevalence

The mobile security product records the installer package names of PHAs observed on mobile devices (see Section 2). This enables us to track the origin market of installed PHAs. We first measure the in-market prevalence of PHAs, serving as the foundation to understand PHA in-market persistence in Section 6.3. We investigate the active PHAs in six popular Android markets (i.e., `Google Play`, `Huawei Market`, `Samsung Market`, `Xiaomi Market`, `Bazaar Market` and `Oppo Market`). Our results are summarized in Table 7. Google Play is the Android market hosting most PHAs: 81K unique SHA2s from 642 PHA families, with the largest monthly active PHA population (i.e., 26K per month) on average. This makes sense, due to the fact that Google Play is the largest Android market with approximately 2.87 million apps[3], and consequently becoming the de facto target of PHA makers. Overall, all markets demonstrate persistent monthly presence of PHAs as we can see from the temporal patterns in Table 7. Following this finding, we will

discuss how these markets deal with the PHAs in the next section.

6.2 Marketplace Actions against PHAs

When apps are submitted to an Android marketplace, they are usually automatically analyzed for presence of malicious activity. If undetected, a PHA will be published on the marketplace, but it may later be suspended or removed, for example after the PHA is reported as malicious by users or security researchers. Google discloses the percentage of PHA installations in its annual Android security and privacy reports [12]. It remains however unclear how many PHAs are suspended or removed by all marketplaces. As we show in the previous section, all markets demonstrate persistent monthly presence of PHAs. For example, `Google Play` has 26K monthly active PHAs and 81K total PHAs in 14 months. This implies that the markets do remove PHAs, but not in a prompt manner. To better understand this phenomenon, we follow the approach in Section 3 to measure the number of PHAs removed or suspended by the top 6 Android markets. Note that we define a PHA p_i as removed/suspended if we do not observe the same SHA2 for the rest of our observation period after its last appearance (i.e., $max(\Omega_{p_i m})$). This way, we exclude all SHA2s that appeared in February 2020 to minimize false positives and discuss the limitation of this strategy in Section 9. Our findings are summarized in Table 8. Overall, each Android marketplace removes at least 91.3% of the PHAs published on it during our observation period. For example, Google Play removed 74k PHAs (91.4% of 81K PHAs) while Bazaar market performed the best removing 4.7K PHAs (95.4% of 5K PHAs) from its market. Unlike previous work [35], we find that Chinese marketplaces like Huawei, Xiaomi, and Oppo also remove most of the PHAs published on them (91.5%, 92.2%, 92.3%). A reason for this discrepancy might be that our observation period is later than the ones used in previous work (2019-2020 vs 2017), and these markets might have changed their security posture after coming under scrutiny. The temporal removal patterns of each marketplace are shown in Table 8, indicating that all marketplaces consistently remove PHAs. It is important to note that Table 8 does not indicate that Google Play and Huawei Market are not trustworthy. Rather, due to the popularity of these markets and their huge user base, they consequently become the de facto targets of PHA makers. For instance, Google Play removed 74K PHAs

---

Market	Average Persistence	Malware Persistence	MUwS Persistence
Google Play	77.64 D	78.72 D	77.11 D
Huawei Market	30.02 D	28.70 D	37.61 D
Xiaomi Market	29.93 D	27.40 D	37.27 D
Samsung Market	52.56 D	48.44 D	81.01 D
Bazaar Market	65.76 D	65.73 D	65.43 D
Oppo Market	28.29 D	26.32 D	36.47 D

Table 9: Summary of PHA in-market persistence in the Top 6 Android Markets.

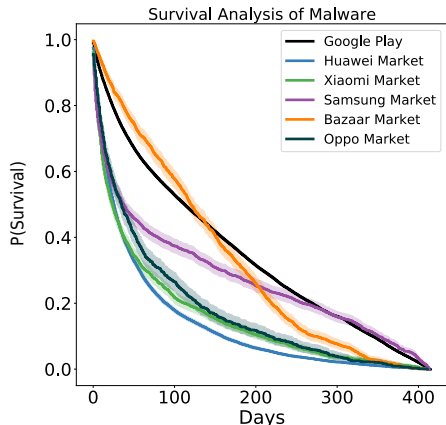Figure 4: Survival analysis of PHAs, malware and MUwS in the six markets.

Figure 5: Survival analysis of malware and MUwS persistence in the six markets.

during our observation period, which is respectively 8 times and 9 times more than Xiaomi Market and Samsung Market. This is in line with the findings by Lindorfer *et al*. [22].

6.3 In-Market Persistence of Different Types of PHAs

An important yet unanswered question is how long PHAs can persist in different markets before being taken down, since the longer they persist the more devices may be infected. To answer this question, we follow the approach outlined in Section 3.3 to measure PHA in-market persistence in these Android markets. Our findings are summarized in Table 9. We observe that PHAs, on average, can persist in `Google Play` for 77.64 days and on other markets for at least 24 days. This leaves a large window of opportunity for miscreants to exploit mobile devices putting the users and their data at risk. To further investigate the significance of our findings on mean PHA in-market persistence, we use the Kaplan-Meier Estimate [14]. Recall that our methodology allows us to include censored data (see Section 3), hence our estimates is not biased nor under-estimated. The survival distributions of PHAs in the six markets are shown in Figure 4. It is visually evident that, at any point across the timeline, we can see that the survival probability of the PHAs in Google Play is more than the other markets (except Bazaar Market). We further carry out the pair-
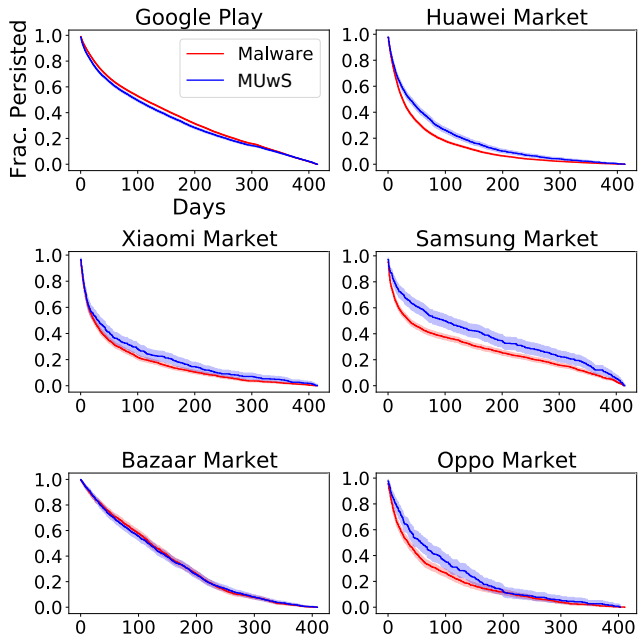
wise Peto-Prentice test to compare the survival distributions of the PHAs between Google and the other markets to establish the fact that the PHAs in Google Play persist longer than those in the other markets. The degrees of freedom are the number of groups minus one, hence always 1 in our tests. A χ^2 test shows that these differences are statistically significant as the test statistic values are significantly larger than 3.841 (from standard χ^2 distribution table) and the *p*-values are all less than 0.005. These results further validate our observation. Our observation is at odds with Lindorfer *et al*. [22]. We hypothesize two factors that may lead to our results. First, Android accounts for 87% of the global smart phone market, and, inevitably, has become the de facto target for mobile malware. In turn, some PHAs may end up on the Google Play Store despite of Google tightening Android's security and app review. Second, Google Play may have different policies to address PHAs (e.g., it may offer a longer grace period for these PHAs to remove offending libraries/code). Nevertheless, Google Play removed 74K PHAs during our observation period, which is far more than those removed by the other markets.

We then use AVClass [29] to distinguish between malware and MUwS among PHAs, investigating if there exists any in-market persistence difference between these two types of PHAs. Mobile MUwS have a slightly shorter persistence period in `Google Play` (77.11 days on average) and `Bazaar market` (65.43 days on average), while in the other four markets MUwS shows longer persistence than malware. In particular, the in-market persistence period of mobile MUwS is almost *twice* longer than that of malware in `Samsung market`

Google Play		Huawei Market		Xiaomi Market		Samsung Market		Bazaar Market		Oppo Market	
Family	Avg. Persistence	Family	Avg. Persistence	Family	Avg. Persistence	Family	Avg. Persistence	Family	Avg. Persistence	Family	Avg. Persistence
airpush	153.3 D	jiagu	32.3 D	jiagu	30.6 D	jiagu	39.9 D	adpush	92.2 D	**jiagu**	26.2 D
jiagu	153.5 D	smsreg	46.9 D	smsreg	32.1 D	airpush	188.2 D	hiddad	98.7 D	hiddad	91.9 D
revmob	159.1 D	tencentprotect	44.3 D	umpay	100.1 D	revmob	191.8 D	toofan	83.5 D	smsreg	65.4 D
leadbolt	159.9 D	secneo	51.8 D	datacollector	58.2 D	leadbolt	173.37D	**privacyrisk**	65.4	datacollector	46.3 D
inmobi	125.9 D	datacollector	32.8 D	tencentprotect	53.2 D	smsreg	56.2 D	ewind	129.0 D	tencentprotect	52.4 D
anydown	191.6 D	autoins	41.2 D	secneo	31.9 D	mobby	194.0 D	dnotua	92.9 D	utilcode	63.0 D
hiddad	165.9 D	**utilcode**	26.0 D	hiddad	82.2 D	tencentprotect	56.44 D	hiddenapp	80.4 D	badiduprotect	39.3 D
plankton	136.1 D	baiduprotect	83.5 D	utilcode	52.5 D	anydown	183.7 D	hiddapp	99.7 D	beitaad	45.9 D
datacollector	152.2 D	autoinst	35.1 D	baiduprotect	48.9 D	**wapron**	8.4 D	notifyer	75.2 D	airpush	47.5 D
dnotua	115.2 D	smspay	62.4 D	**wapron**	13.5 D	baiduprotect	84.9 D	airpush	123.8 D	revmob	106.6 D

Table 10: Summary of the top 10 families (ranked by the number of SHA2s) in-market persistence in the top 6 Android marketplaces. A family name is in bold if its in-market persistence period is below average (see Table 9).

(81.01 days on average) and `Oppo market` (48.44 days on average). This suggests that different markets apply different policies when vetting for PHAs, and might prioritize certain types of threats over others. To further validate our findings on the in-market persistence difference between these two types of PHAs, we again use the Kaplan-Meier Estimate. The survival curves of mobile malware and MUwS in the six markets are shown in Figure 5. We further carry out the Peto-Prentice test to compare the survival distributions of malware and MUwS within each market. A χ^2 test shows that these differences are statistically significant as the test statistic values are significantly larger than 3.841 (from standard χ^2 distribution table) and the p-values are all less than 0.005. The only exception is `Bazaar market`, where the test statistic is not significant. Hence, we cannot conclude if `Bazaar market` applies different policies when vetting for PHAs.

In Section 5 we showed that the overall number of devices infected is correlated with the number of SHA2s. Following this finding, we further study if PHA families with a large number of PHAs can persist longer in the marketplaces. Our hypothesis is that these large families may persist in the markets longer since app vetting systems require both machine and human inspection. Our findings on the top 10 largest families in the top six markets are shown in Table 10. We observe that most of the large families in the top six marketplaces persist longer than the mean persistence time (see Table 9). For example, all top 10 families in `Google Play` have in-market persistence of at least 115 days, which is 38 days longer than the mean 77.64 days persistence time. These results show that there is a need for more comprehensive app vetting measures.

6.4 PHA In-Market Evolution

In the previous sections, we showed that PHAs can persist in a market for weeks. In this section we aim to further understand how PHA families may evolve in the markets. For example, PHA makers may proactively switch ad libraries in response to market policy changes or gain better incentives from ads, or they may modify their malicious code to evade market app vetting systems, etc. Note that each app has a unique package name in a given market, by correlating the SHA2s belonging
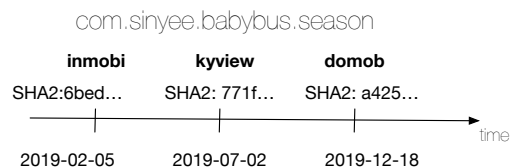

Figure 6: Example of PHA in-market evolution (`com.sinyee.babybus.season`).

Market	#Apps	#SHA2s	Approximate SHA2s per PHA	Avg. In-market Persistence	Avg. Evolution Gap
Google Play	1,349	7,883	~6	250.1 D	66.5 D
Huawei Market	320	1779	~5	276.6 D	116.2 D
Xiaomi Market	89	443	~5	247.8 D	98.8 D
Samsung Market	70	383	~5	238.9 D	86.8 D
Bazaar Market	40	129	~3	213.9 D	120.5 D
Oppo Market	43	234	~5	227.0 D	98.4 D

Table 11: Characteristics of PHA in-market evolution.

to a certain package name and the AVClass results of their VT reports, we can track and measure if an app evolves over time (i.e., if the SHA2s of a certain PHA belong to at least 2 PHA families over the time). We show an example in Figure 6 where SHA2s from `com.sinyee.babybus.season` in `Google Play` are associated with three different PHA families (i.e., `inmobi`, `kyview` [36], and `domob`) during our observation period.[4] Their overall evolution distribution is illustrated in Figure 7. As we can see, the majority of the PHAs exhibiting in-market evolution are observed in `Google Play` and `HuaWei Market` (1,340 and 320 PHAs respectively in these two markets). There are a limited number of PHAs in

---

[4]Note that `inmobi` is Google's preferred ad SDK partner. However, this library is flagged by multiple mobile security products as MUwS, and has leaked sensitive user data in the past. In fact, inMobi was charged by the FTC for COPPA violations in 2016. Therefore we flag `inmobi` as PHA in this paper even though we acknowledge that the definition of MUwS varies by platforms.
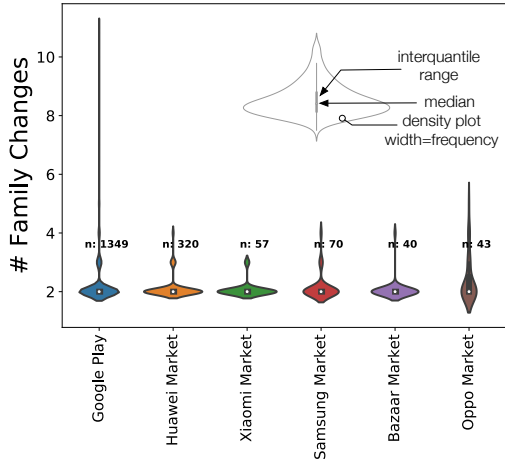
Figure 7: Violet plot summarizing PHA in-market evolution. The white dot in the middle is the median value, the thick black bar in the centre represents the interquartile range and the contour represents the distribution shape of the data.

the rest of the markets exhibiting in-market evolution. For example, we identify 10K PHAs in `Samsung Market` (see Table 8), yet only 70 of them exhibit in-market evolution. On average, these PHAs belong to two PHA families over time. Additional characteristics of these PHAs exhibiting in-market evolution are summarized in Table 11. Overall, these PHAs exhibiting in-market evolution show longer in-market persistence (i.e., over 200 days) in the top 6 markets. For example, the PHAs exhibiting in-market evolution persists in `Google Play` for 250.1 days compared to the average 77.6 days (see Table 9). The average gap between PHAs switching families in `Google Play` is 66.5 days, which is more frequent than the other markets. We believe that the shorter gap in `Google Play` is partially due to the stringent app vetting system and security policies applied by `Google Play`. As such, miscreants must be proactive to deal with the scrutiny from Google.

7 PHA Migration

When their PHAs are removed from a marketplace, miscreants might migrate to alternative ones to keep their operation going. In this section, we study how PHAs migrate among markets.

7.1 PHA Inter-Market Migration

An important unanswered question is if miscreants actively move PHAs among markets to infect more devices or after such PHAs were removed from a market. For example, the miscreants may move a PHA to alternative markets after Google Play takes it down. Alternatively, the miscreants may move a PHA from alternative markets to Google Play to profit from its massive end users even for a short period of time. We show an example in Figure 8. We have a repackaged app `com.avatar.star` with SHA2 `79e6...`, which origi-
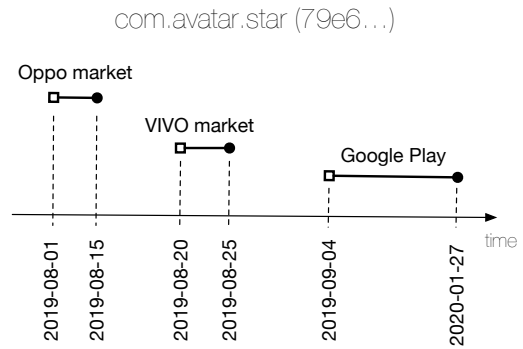

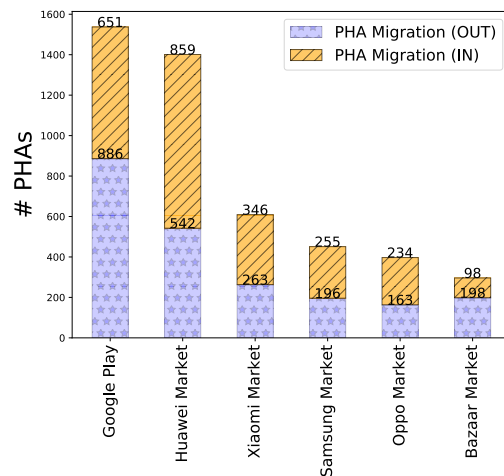Figure 8: Example of PHA inter-market migration (`com.avatar.star`).

Figure 9: PHA inter-market migration. Markets are ranked by the total number of inter-market activities.

nally appeared in `Oppo Market` between August 1, 2019 and August 15, 2019 (3 devices infected), then moved to `VIVO Market` between August 20, 2019 and August 25, 2019 (2 devices infected), and finally settled down in `Google Play` between September 4, 2019 and January 27, 2020 (215 device infected). Recall that we can track a SHA2 across markets and time to identify sequentially non-overlapping time intervals. To quantify the aforementioned phenomenon, we use the app package names and their associated SHA2s as seeds (similar to previous work [22]), and use the approach detailed in Section 3 to measure PHA inter-market migration. In total, we observe 3,533 PHAs that exhibit inter-market migration. The results for the top six markets are summarized in Figure 9. We see that `Google Play` exhibits the most inter-market migration activities with 1,404 PHA migrations, while `Google Play` also exhibits the most outward PHA migrations with 886 outward migration activities.

We next investigate whether mobile malware and MUwS present different migration activity on the various markets. We again use AVClass [29] to identify mobile malware and

Market	Total PHA Migration (in)	# Malware	# MUwS	Avg. Persistence	# Device Infected (upstream)	# Device Infected (current)
Google Play	651	447	204	57.9 D	964	3,003
Huawei Market	859	747	112	63.5 D	1,039	1,543
Xiaomi Market	346	292	54	10.44 D	4,065	471
Samsung Market	255	218	37	23.66 D	1,599	394
Oppo Market	234	186	58	10.3 D	3,364	296
Bazaar Market	107	63	44	63.84 D	121	284

Table 12: Market response to PHA migration.

Service	#PHAs	#Malware	#MUwS	Dev Infected	Avg. Persistence
com.sec.android.easyMover (Samsung)	14,038	10,960	3,078	35,557	93.38 D
com.samsung.android.scloud (Samsung)	5,088	3,835	1,253	8589	56.41 D
com.hicloud.android.clone (Huawei)	3,653	2,953	700	3,079	32.53 D
com.oneplus.backuprestore (Oneplus)	1,072	794	278	1,361	22.69 D
com.coloros.backuprestore (Oppo)	972	695	277	1,267	21.98 D
com.miui.cloudbackup (Xiaomi)	1,243	928	315	1,235	33.23 D

Table 13: PHA migration from data backup/clone services. Those services are ranked by the device prevalence ratios.

MUwS from the package names that migrated. Lindorfer *et al*. [22] found initial evidence that malicious apps jump from market to market, possibly for survival. For instance, the authors identified 131 apps that migrated to alternative markets, but didn't carry out further analysis of how long these apps would **survive** after the migration. To fill this gap, we then measure specifically the PHAs that migrated into the markets to understand their in-market persistence. The results are summarized in Table 12. More mobile malware migrates into the markets compared to MUwS for all top 6 markets. Our hypothesis is that the ecosystem of MUwS usually leverages ad libraries and can be more adaptable to market takedowns, while the miscreants behind mobile malware use more sophisticated methods (e.g., code obfuscation, environment awareness, etc) hence reusing the same PHAs across different markets to maximize the number victims is more desirable. To verify our hypothesis, we measure the device prevalence ratios of these PHAs migrating into the markets and compare this prevalence ratios to those of the immediate upstream markets they migrated from. Our results are summarized in Table 12. As it can be seen, PHAs migrating into `Google Play` and `Huawei Market` (which have large user bases) manage to infect at least 50% more devices than those from the immediate upstream markets. However, PHAs migrating into the rest of the markets (which have smaller user bases) do not reach more devices. Nevertheless, those PHAs, on average, have short lifespans in these markets compared to the average persistence time (see Table 9, Section 6) except `Huawei Market`. Our hypothesis is that this is partially due to the fact that these PHAs have been detected in the upstream markets, therefore signatures were made available for the downstream markets to detect them. At the same time, the exception of `Huawei Market` shows that markets must be responsible and rigorously vet the apps submitted. Our study only measures the *lower bound* of the PHA in-market persistence since it is possible that a PHA still exists in a market but our dataset did not reflect its existence. The issue could be addressed if our dataset is augmented with the method proposed by Lindorfer *et al*. [22]. We leave such task as part of our future work.

7.2 PHA Persistence After Migration via Backup/Clone Services

Android phones typically offer backup functionality to their users, allowing them to restore their apps and configuration when they purchase a new device. This mechanism allows users to quickly restore their data (e.g., contacts, settings, apps) in the new devices without manual reinstallation efforts. However, such services may inadvertently migrate existing PHAs to the new device too, and compromise the security and privacy of the new phones, *even though these PHAs may have been removed by the markets* and therefore the user might not be able to manually install them anymore. Kotzias *et al*. [16] showed that backup restoration is an unintended unwanted app distribution vector responsible for 4.8% of unwanted installs. Following this direction, we further investigate how long PHAs can persist after migrating via backup/clone services. Recall that the mobile security product captures an app's installer package name (see Section 2). This enables us to identify apps that were installed by backup/clone services in our dataset. To this end, we first identify the top six data backup/clone services in our dataset and understand how many PHAs migrate from backup/clone services, and consequently how long these PHAs may persist on the devices. To accurately identify the data backup/clone services, we first remove all known market installer packages and rank the rest of the installers by the device prevalence ratio. We then investigate these apps on `Google Play` and on the Web to understand the functions of the installers.

Our findings are shown in Table 13. Overall, we observe that a considerable number of PHAs are not removed by end users and consequently are migrated from the old phones and backups. For example, 14K PHAs migrated to 35.5K new Samsung models in our dataset. At the same time, it is interesting to see that there is three times more mobile malware than MUwS migrating via backup/clone services. In addition, these PHAs persist longer than the average 20.2 days persistence period (see Table 4). For example, PHAs migrated via Samsung smart switch mobile app (`com.sec.android.easyMover`) persist in the new devices for an averaged 93 days.
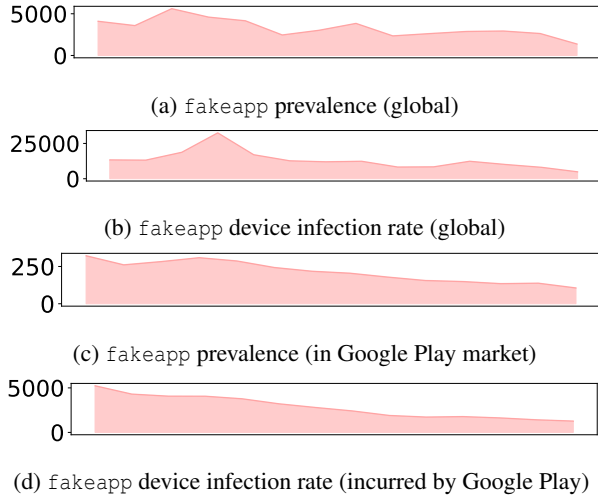
(a) `fakeapp` prevalence (global)

(b) `fakeapp` device infection rate (global)

(c) `fakeapp` prevalence (in Google Play market)

(d) `fakeapp` device infection rate (incurred by Google Play)

Figure 10: Case study: the `fakeapp` family

8 `Fakeapp` Case Study

In this section, we provide a case study on the `fakeapp` Android malware family to demonstrate that our approach can measure how a malicious campaign spread, persisted, and later was removed by the markets. `fakeapp` is a family of malicious apps that masquerades as popular legitimate apps, by using a similar package name and icons as AV apps, banking apps, etc. Some of the apps from the `fakeapp` family may engage in malicious activities such as sending/receiving premium SMS messages and downloading other apps [17,40].

Figure 10 shows the prevalence rate and the device infection rate of `fakeapp` from a global and a local perspective (we use the `Google Play` market in our case study). During our observation period, at the global level, `fakeapp` had approximately 3K active SHA2s infecting 13K devices on a monthly basis (see Figure 10a and 10b). From a local perspective, `fakeapp` had approximately 210 active SHA2s persisting in the `Google Play` market and infecting 3K devices on a monthly basis (see Figure 10c and 10d. This is rather interesting because apps from the `fakeapp` family in the `Google Play` market represent 7% of monthly active SHA2s belonging to this family, and yet accounted for approximately 25% of the global device infection. Our study shows that the `Google Play` market ramped up its removal efforts over our observation period. The market had 322 active `fakeapp` SHA2s in January 2019 and reduced the number to 106 active SHA2s in February 2020, representing a twofold decrease (Figure 10c). This, in turn, lead to a four-fold decrease in the number of device infection rate (Figure 10d). However, the `fakeapp` family, on average, persisted in the `Google Play` market for 86.6 Days before removal. This is about 9 days longer than the average PHA persistence period on that market (see Table 9, Section 6.2). At the same time, `fakeapp` apps installed from `Google Play` persist 29.41 days on devices, which is 14 days longer than the average 15.17 days `fakeapp` on-device

persistence period. We believe that this is due to the fact that `Google Play` is the *de facto* trusted source of Android apps, hence the end users may keep the PHAs from `Google Play` longer. Additionally, the fact that `fakeapp` apps come disguised as useful apps might hide the fact that these apps are malicious and lure users into keeping them on their devices for longer, despite being warned by the mobile security product. Our case study highlights the importance of the `Google Play` market in fighting PHAs and demonstrates the in-depth analysis that our measurement methodology can achieve.

9 Limitations and Discussion

Biases. While this paper presents the largest measurement of on-device Android PHA to date, our dataset is biased towards the users of a single mobile security product, and therefore still presents some biases. For example, our device population is skewed towards the United States and European countries. It is possible that end users in the United States and Europe tend to keep this mobile security app installed for longer, hence more likely that these devices fit in our data selection criteria (see Section 3). At the same time, we cannot observe the behavior of users that do not use mobile security products, and those who did not opt-in this data collection scheme. Besides, we cannot observe certain events from the devices protected by Google Play Protect. Nonetheless, we believe that our dataset is representative of the worldwide mobile users, and we do our best to minimize this bias, for example, by using percentages when looking at per country infection rates. In terms of the representativeness of the analyzed apps, it is challenging to ascertain the coverage of our study since it is infeasible to determine the total number of all Android apps, given such a fragmented ecosystem and many alternative markets. Still, by analyzing 8.8M unique apps, this study is covering one of the largest sets of apps to date, and is in line with the largest datasets collected by the academic community [1].

Data Limitations. It is important to note that the PHA detection data is collected passively. That is, a PHA detection event is recorded when the security product detects a potentially harmful application that matches a pre-defined signature including its behavior, communications, and policy violations. Any mobile PHAs preemptively blocked by other security products (e.g., application store link blacklists, cloud-based app reputation systems) cannot be observed. Additionally, any PHAs that do not match the predefined signatures on devices are also not observed. Inferring the last seen timestamp of a PHA in a market is practically hard since the mobile security data is collected passively. Our inference therefore relies upon the deduction that if we do not observe a given PHA from billions of events generated by 11.7M devices following its last observation time, we consider that this PHA was removed by a market. It is possible that this PHA could still remain in that market and our dataset simply did not capture its existence

(i.e., this PHA is not installed by the 11.7M devices after its last observation timestamp). Consequently, we measure the *lower bound* of the PHA in-market persistence in our study.

The Android API enables the mobile security product to identify the installer package name of a PHA. Correlating this with the official package names of the markets, we can identify if a PHA comes from a certain market at a certain timestamp. However, miscreants or end users can install apps via `ADB` and impersonate the official package names of the markets. In this case, the mobile security product can wrongly attribute a PHA as originating from a certain market. To minimize this risk, our study only selects a PHA observed in at least two devices. We believe that such false positives incurred by such impersonated official market package names are statistically ignorable. In addition, if an app was installed before our observation period started, we cannot obtain market information for it. If an already installer app is consequently updated, our system sees the updating software as the installer and not the original marketplace the app came from. We therefore exclude the PHAs that we cannot confidently attribute to certain markets. Still, this allows us to cover 66% of the devices in our dataset and 22% of all PHA installations.

Implications for mobile security research. Our study shows that many PHAs can persist on devices and in app markets for many days once installed or approved. We hope that our study can inspire better notification systems to nudge the end users to remove PHAs once detected, and, ideally, devise a prevention system able to convince users not to install PHAs in the first place.

Implications to Android markets. Our study shows that PHAs can persist in a market for at least 24 days. At the same time, while we recognize the efforts from the Android markets, not all PHAs are removed by them (e.g., `Google Play` removes 5.28K PHAs per month and, in total, removes 74K out of 81K PHAs). We hope that our findings will enable Android markets to ramp up their app vetting systems and takedown PHAs in a timely manner to minimize their in-market persistence. In addition, despite of the transparency report from Google Play, we hope that the markets can be more transparent and disclose the performance figures relating to PHA removal (e.g., the number of PHA removed monthly, the average time to remove a PHA, etc.). Our study shows that PHAs may evolve over time to survive in the markets for longer and be able to reach more victims. We hope that our findings can encourage app markets to make end users aware of the security and privacy issues incurred by the previous versions of an app if any. For example, certain versions of the popular app `com.intsig.camscanner` in `Google Play` were affected by the Trojan dropper `necro` due to the integration of a 3rd party SDK from AdHub. As the app remains in `Google Play` after the removal of the 3rd party library, a historical briefing of the security and privacy incidents as-

sociated with such apps would offer end users an informed decision when installing them on their devices in the future.

10 Related Work

There is an enormous amount of research on mobile security and privacy. In this section, we specifically review previous measurement studies on malware characterization and mobile app ecosystem. We refer the readers to [6,8,19,24,32,37] for overviews and surveys on securing Android devices.

Mobile PHA characterization. The security research community has been actively investigating the ever-changing characteristics of mobile PHAs for years [6, 8, 19, 24, 32, 37]. Previous efforts mainly focused on analyzing apps and systematically characterizing them from various aspects. From a high level, these research center on installation methods [40], evasion mechanisms [7], repackaging mechanisms [22,31,39], malicious payloads [40], behaviors [21,38], monetization [9], etc. In recent years, Faruki *et al.* [7] summarized Android security issues, malware growth (during 2010-13), their penetration, stealth techniques, and strength as well as weaknesses of some of the popular mitigation solutions. Mirzaei *et al.* [23] introduced Andrensemble, a system to characterize Android malware families by leveraging API ensembles. These efforts collectively shed lights on how Android malware operates in the wild, the main incentives of mobile malware, the weaknesses of some of the popular mitigation solutions, etc. However, they did not discuss potential threats posed by PHA persistence in both mobile devices and markets as these efforts center on app analysis and offer a less comprehensive view of the real device prevalence.

Measurement studies on Android permission system. The Android permission system has been extensively covered in the previous literature [2, 3, 8, 24]. We only review the work relating to our study in this paper. Felt *et al.* [8] built the Stowaway system to detect overprivileged apps which could result in privacy violations. Felt *et al.* [10] later showed that current Android permission warnings do not help most users make correct security decisions. Sarma *et al.* [27] discussed the risks incurred by the Android permission system and outlined 13 permissions that may critically invade users' privacy. Qu *et al.* [25] designed AutoCog to measure the description-to-permission fidelity in Android apps and assist the end users to understand the security and privacy implications when granting permissions.

Measurement studies on mobile PHA. From a device perspective, Shen *et al.* [30] carried out a detailed quantitative analysis on 6.14 million Android devices comparing rooted and non-rooted Android devices across a broad range of characteristics including PHA installations and network behavior. Suarez-Tangil *et al.* [31] carried out a systematic study of 1.28M repackaged apps spanning between 2010 and 2017 to understand how Android malware has evolved over time.

More recently, Gamba *et al*. [11] collected 82K pre-installed apps (424K files in total) on Android devices from more than 200 vendors and carried out a measurement study to understand how the stakeholders primarily build their relationship around advertising and data-driven services. From an app market perspective, Lindorfer *et al*. [22] proposed the AndRadar system to discover multiple instances of a malicious Android application in a set of alternative application markets using a set of package names as seeds. Wang *et al*. [35] leveraged 6M Android apps downloaded from 16 Chinese app markets and Google Play and provided a large-scale comparative study to understand various aspects and dynamics relating to apps (including PHAs), their behavior and the developers. These efforts collectively shed lights on the overall picture of how PHA evolves over time. Different from these previous efforts, our study focuses on the potential threats posed by PHA persistence in both mobile devices and markets as these efforts center on app analysis and offer a comprehensive view of the real device prevalence.

Desktop PUP PPI ecosystem study. Another loosely connected research line is related to measuring the PUP PPI ecosystem in the PC environment. Caballero *et al*. [4] provided the first large scale measurement of blackmarket pay-per-install services in the wild. Kotzias *et al*. [15] leveraged file dropping graphs to build a *publisher graph* and identify specific roles in the ecosystem, in turn revealing the relationship between PUP prevalence and PUP distributors. Thomas *et al*. [33] performed a similar study on unwanted software on desktop computers.

Comparison with Close Work. The closest work is a recent mobile unwanted app distribution study by Kotzias *et al*. [16]. Their study focuses on understanding who-installs-who relationships between installers and child apps, and uncovering the main unwanted app distribution vectors. Similar to the findings by Kotzias *et al*. [16], our study also shows that Google Play remains the main app distribution vector of PHAs, but also has the best defenses against PHAs (e.g., removing most of the PHAs). Kotzias *et al*. [16] also identifies many other distribution vectors such as bloatware, browsers, instant messaging, etc. Our study does not cover these distribution vectors as we focus on the temporal behavior of PHAs. Concretely, leveraging a longer observation period of PHA installation events across 11M devices, our study offers a large-scale temporal measurement study of Android PHAs to comprehend the characteristics their on-device and in-market persistence, and consequent inter-market migration after taken down. In summary, Kotzias *et al*. [16] cover where the PHAs come from while our study addresses the temporal dynamics of PHA installations on Android.

11   Conclusion

We presented the largest on-device study to date of Android PHAs installed in the wild. Our results show that PHAs on Android are a pervasive problem, and that malicious apps can persist for long periods of time both on devices and on markets. Our results suggests that current measures against malicious apps on Android are not as effective as commonly thought, and that more research from the security community is needed in this space.

References

[1] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *MSR*, 2016.

[2] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. Pscout: analyzing the android permission specification. In *ACM CCS*, 2012.

[3] David Barrera, H Güneş Kayacik, Paul C Van Oorschot, and Anil Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In *ACM CCS*, 2010.

[4] Juan Caballero, Chris Grier, Christian Kreibich, and Vern Paxson. Measuring pay-per-install: The commoditization of malware distribution. In *USENIX Security*, 2011.

[5] Rahul Chatterjee, Periwinkle Doerfler, Hadas Orgad, Sam Havron, Jackeline Palmer, Diana Freed, Karen Levy, Nicola Dell, Damon McCoy, and Thomas Ristenpart. The spyware used in intimate partner violence. In *IEEE S&P*, 2018.

[6] Zheran Fang, Weili Han, and Yingjiu Li. Permission based android security: Issues and countermeasures. *computers & security*, 43, 2014.

[7] Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan. Android security: a survey of issues, malware penetration, and defenses. *IEEE communications surveys & tutorials*, 17(2), 2014.

[8] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *ACM CCS*, 2011.

[9] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. A survey of mobile malware in the wild. In *SPSM*, 2011.

[10] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. In *SOUPS*, 2012.

[11] Julien Gamba, Mohammed Rashed, Abbas Razaghpanah, Juan Tapiador, and Narseo Vallina-Rodriguez. An analysis of pre-installed android software. In *IEEE S&P*, 2020.

[12] Google. Android Security & Privacy 2018 Year In Review. 2019.

[13] Médéric Hurier, Guillermo Suarez-Tangil, Santanu Kumar Dash, Tegawendé F Bissyandé, Yves Le Traon, Jacques Klein, and Lorenzo Cavallaro. Euphony: Harmonious unification of cacophonous anti-virus vendor labels for android malware. In *MSR*, 2017.

[14] David G Kleinbaum and Mitchel Klein. *Survival analysis*. Springer, 2010.

[15] Platon Kotzias, Leyla Bilge, and Juan Caballero. Measuring PUP Prevalence and PUP Distribution through Pay-Per-Install Services. In *USENIX Security*, 2016.

[16] Platon Kotzias, Juan Caballero, and Leyla Bilge. How did that get in my phone? unwanted app distribution on android devices. In *IEEE S&P*, 2021.

[17] Su Mon Kywe, Yingjiu Li, Robert H Deng, and Jason Hong. Detecting camouflaged applications on mobile application markets. In *ICISC*, 2014.

[18] Charles Lever, Manos Antonakakis, Bradley Reaves, Patrick Traynor, and Wenke Lee. The core of the matter: Analyzing malicious traffic in cellular carriers. In *NDSS*, 2013.

[19] Li Li, Tegawendé F Bissyandé, and Jacques Klein. Rebooting research on detecting repackaged android apps: Literature review and benchmark. *IEEE Transactions on Software Engineering*, 2019.

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

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

[22] Martina Lindorfer, Stamatis Volanis, Alessandro Sisto, Matthias Neugschwandtner, Elias Athanasopoulos, Federico Maggi, Christian Platzer, Stefano Zanero, and

Sotiris Ioannidis. Andradar: fast discovery of android applications in alternative markets. In *DIMVA*, 2014.

[23] Omid Mirzaei, Guillermo Suarez-Tangil, Jose M de Fuentes, Juan Tapiador, and Gianluca Stringhini. Andrensemble: Leveraging api ensembles to characterize android malware families. In *ASIACCS*, 2019.

[24] Mohammad Nauman, Sohail Khan, and Xinwen Zhang. Apex: extending android permission model and enforcement with user-defined runtime constraints. In *ASIACCS*, 2010.

[25] Zhengyang Qu, Vaibhav Rastogi, Xinyi Zhang, Yan Chen, Tiantian Zhu, and Zhong Chen. Autocog: Measuring the description-to-permission fidelity in android applications. In *ACM CCS*, 2014.

[26] Abbas Razaghpanah, Rishab Nithyanand, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Mark Allman, Christian Kreibich, and Phillipa Gill. Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem. In *NDSS*, 2018.

[27] Bhaskar Pratim Sarma, Ninghui Li, Chris Gates, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Android permissions: a perspective combining risks and benefits. In *SACMAT*, 2012.

[28] Angela Sasse. Scaring and bullying people into security won't work. *IEEE Security & Privacy*, 13(3):80–83, 2015.

[29] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. Avclass: A tool for massive malware labeling. In *RAID*, 2016.

[30] Yun Shen, Nathan Evans, and Azzedine Benameur. Insights into rooted and non-rooted android mobile devices with behavior analytics. In *SAC*, 2016.

[31] Guillermo Suarez-Tangil and Gianluca Stringhini. Eight years of rider measurement in the android malware ecosystem. *IEEE Transactions on Dependable and Secure Computing*, 2020.

[32] Darell JJ Tan, Tong-Wei Chua, Vrizlynn LL Thing, et al. Securing android: a survey, taxonomy, and challenges. *ACM Computing Surveys (CSUR)*, 47(4), 2015.

[33] Kurt Thomas, Juan A Elices Crespo, Ryan Rasti, Jean-Michel Picod, Cait Phillips, Marc-André Decoste, Chris Sharp, Fabio Tirelo, Ali Tofigh, Marc-Antoine Courteau, et al. Investigating commercial pay-per-install and the distribution of unwanted software. In *USENIX Security Symposium*, 2016.

[34] Haoyu Wang, Hao Li, Li Li, Yao Guo, and Guoai Xu. Why are android apps removed from google play? a large-scale empirical study. In *MSR*, 2018.

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

[36] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. Deep ground truth analysis of current android malware. In *DIMVA*, 2017.

[37] Meng Xu, Chengyu Song, Yang Ji, Ming-Wei Shih, Kangjie Lu, Cong Zheng, Ruian Duan, Yeongjin Jang, Byoungyoung Lee, Chenxiong Qian, et al. Toward engineering a secure android ecosystem: A survey of existing techniques. *ACM Computing Surveys (CSUR)*, 49(2), 2016.

[38] Chao Yang, Zhaoyan Xu, Guofei Gu, Vinod Yegneswaran, and Phillip Porras. Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications. In *ESORICS*, 2014.

[39] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *CODASPY*, 2012.

[40] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *IEEE S&P*, 2012.

[41] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. Measuring and modeling the label dynamics of online anti-malware engines. In *USENIX Security Symposium*, 2020.