Peeking into the Gray Area of Mobile World: An Empirical Study of Unlabeled Android Apps

Sen Chen[1], Lingling Fan[2]*, Cuiyun Gao[3], Fu Song[4], and Yang Liu[5]
[1]College of Intelligence and Computing, Tianjin University, China
[2]College of Cyber Science, Nankai University, China [3]Harbin Institute of Technology (Shenzhen), China
[4]ShanghaiTech University, China [5]Nanyang Technological University, Singapore

Abstract—**For the real-world dataset collected by our industrial partner, Pwnzen Infotech Inc., one of the leading industrial security companies, there are a large number of unlabeled Android applications (called unlabeled apps in this paper) that are unlikely to belong to known Android malware families nor ordinary benign apps according to the industrial black-list (i.e., signatures) and white-list (i.e., certificates). However, such apps have rarely been studied previously, but are important to peek into the gray area of mobile world. It is a time-consuming task for software analysts to understand the negative characteristics of these samples, which would lead to potential security or privacy threats for app users, significantly negative impacts on mobile system performance, and bad user experience, etc.**

To investigate the characteristics of these industrial unlabeled apps in a large-scale in practice, and provide insights to industrial software analysts as well as research communities, we collect a large-scale dataset of unlabeled apps (i.e., 22,886 in total) from our industrial partners. Given the common industrial perception of software analysts that a high percentage of these unlabeled apps could have some similar behaviors, we leverage the popular community-detection techniques based on widely-used app features in malware detection to cluster these unlabeled apps. After that, we investigate the common behaviors for different clusters with substantial human efforts and also conduct cross-validation across co-authors to check the results. Our manual analysis unveils the characteristics of these unlabeled apps by sampling data from different clusters, and discovers 11 categories, some of which have never been discovered by previous grayware research. Besides, from our exploration, we find that the community-based techniques are not effective enough in clustering unlabeled apps, so that manual analysis is encouraged. Manual analysis is an important first step towards studying unlabeled apps and understanding their characteristics. Finally, we highlight the lessons learned through real case studies, comparison study with existing malware/grayware research, in-depth discussion with industrial partners, and feedback from industrial partners.

I. Introduction

Android apps now have become the most popular way of performing daily tasks [1], [2], however, more and more mobile users suffer from Potentially Harmful Apps (PHAS) (e.g., Android malware [3] and Android grayware [4]). Consequently, a rapidly growing amount of software-engineering research focuses on studies and analysis of Android apps [5], [6], [7], [8], [9], [10]. Such research growth also thanks to the availability of real dataset of Android apps or Android malware from industry [11], [5], [9] such as Tencent Inc. [12] and Qihoo 360 [13]. Note that, the real-world samples collected by industrial companies are usually with concrete labels such as malicious and benign, which promotes the corresponding data-driven research in academia [14], [15], [16], [17].

Nevertheless, our industrial partner, Pwnzen Infotech Inc., one of the leading security companies with rich security experience and advanced technology, recently unveils that there are a large number of industrial unlabeled Android applications (called *unlabeled apps*) that are unlikely to belong to known families of Android malware nor ordinary benign apps according to the black-list (i.e., signatures) and white-list (i.e., certificates) of the industry. Note that, these unlabeled apps are crawled from various Android markets including the official market (Google Play) and other third-party markets. Some of them are collected by monitoring the network traffic. Such unlabeled apps can be regarded as a peephole to observe the gray area of mobile world. However, it is difficult to understand the characteristics of these samples for software analysts in a large-scale in practice due to the unknown property and possible diversity of categories. Therefore, there is a great demand to conduct an empirical study on these industrial unlabeled apps and provide insights to both academia and industry, avoiding introducing potential threats for app users (e.g., security, privacy, and negative impacts). Given that it is challenging and even infeasible to conduct a large-scale curation and analysis of these unlabeled apps without collaboration with industry, we have started collaborating with our industrial partner to get access to the unlabeled dataset and exchange insights with people from industry.

Our collaborator allows us to gain access to the large-scale industrial samples as well as the ensemble industrial scanning engine used to label samples. The scanning engine is shared across several security companies such as Qihoo 360 [13], and the signatures of samples stored in the scanning engine are updated timely for accurately filtering out Android malware and benign apps. For industry, it is a common practice to maintain a "black list" storing the signatures of known malware families and a "white list" storing known benign apps such as official certificates [5]. The others are then categorized as "industrial unlabeled apps", indicating that they are unlikely to belong to any known malware families or known benign apps in the official markets.

A common industrial perception of these unlabeled apps is that a high percentage of them share similar behaviors, therefore, unlabeled apps may be classified into different cate-

Lingling Fan is the corresponding author. Email: linglingfan@nankai.edu.cn gories based on their behaviors. According to this observation, we take these unlabeled apps collected from our industrial partner as our initial starting point to identify and analyze unlabeled apps, by employing the widely-used community-based technique in Android malware field [18]. Note that, different from the malware-specific features (e.g., semantic, security-related, and context-aware features) extracted for Android malware detection, whose goal is to distinguish malware from samples [19], [20], [14], [21], [22], the features extracted for unlabeled app analysis are more general since we aim to investigate the characteristics of these unlabeled apps as a whole. Fig. 1 shows that the data preprocessing and sampling phases are two initial phases for our study, while the core phase is to conduct an in-depth investigation of these unlabeled apps in order to understand their characteristics, which sheds light on the follow-up research towards unlabeled apps.

In this paper, we attempt to use the widely-used feature types [14], [21] and community-based techniques [18] in malware research to cluster these unlabeled apps. However, the clustering results are also not effective enough in our study, but a coarse-grained filtering can reduce the analysis efforts, to some extent. So in this paper, instead of relying on the clustering technique for accurate app grouping, we adopt the technique to select some representative samples for study, and manually classify these sampled apps based on their characteristics. We sample 375 apps from the unlabeled apps by the sampling criteria to conduct manual analysis (details in Section III-B). We observe that a lot of unlabeled apps share common characteristics. Based on our investigation, we finally summarize 11 categories of unlabeled apps based on their common characteristics.

We reach out the following findings from our manual analysis: **1)** We distill 11 categories of unlabeled apps. Some categories are similar to a number of previously-identified grayware categories [4], showing that our study is able to identify several grayware categories pointed out by the previous studies. The previous grayware study [4] only focused on the apps collected from Google Play Store, in this paper, we distill more categories and understanding of the gray area of the mobile world, which thanks to the real dataset from industry instead of only the official Android market. Consequently, 8 new categories and 2 more general categories are discovered based on our analysis. Further, our newly labeled data can be published as a benchmark dataset of grayware to foster further classification between grayware and other app types. **2)** Among our identified categories, apps in several categories have potential security threats for users, such as *Dialing/SMS-Managing Apps*, *Data Collection Apps*, *Porn Apps*, and *Background Service Apps*. To demonstrate the common behaviors in different categories of unlabeled apps, we also conduct a case study on several categories to provide more insights for academic researchers and industrial analysts. **3)** According to the results of our manual analysis, we also find that some behaviors of unlabeled apps may unveil special industrial profit chain in the real world. After that, we highlight the lessons learned through an in-depth discussion with industrial partner and feedback from industry.

In summary, this paper makes the following contributions:

- To the best of our knowledge, this is the first work systematically curating and analyzing industrial unlabeled Android apps based on a large-scale dataset collected from the industrial company instead of limited case studies.
- We discovered 11 categories of unlabeled apps based on their internal characteristics and community unit by leveraging community-based clustering and substantial manual analysis. Among these categories, 8 new categories and 2 more general categories are discovered compared to previous grayware studies, which can also be published as a new benchmark dataset of grayware.
- We also provide the corresponding case studies on interesting unlabeled app categories and highlight the lessons learned from the study, which motivates the follow-up research on unlabeled apps, and helps industrial companies better characterize unlabeled apps.

II. Related Work

A. *Analysis of Android Malware*

In practice, most of the researchers' and practitioners' efforts target Android malware detection and understanding. Consequently, various approaches have been exhibited in this area, such as signature-based [23], behavior-based [24], data-flow analysis-based (e.g., taint analysis) [25], [26], model checking-based [27], [28], and machine-learning-based techniques [14], [21], [17], [22], [29], [30], [31], [32]. However, in the initial research stage of Android malware, manual analysis and sample clustering are widely-used due to lack of enough labeled samples and insights of malware characteristics [3], [33], [18]. For example, in Genome project [3], Jiang et al. analyzed 1,260 malware samples and summarized their corresponding characteristics by an in-depth manual analysis study. Samra et al. [33] analyzed Android malware by leveraging clustering techniques. In this work, following the initial research of Android malware, we first perform an empirical investigation of industrial unlabeled Android apps through manual study. In practice, in a new research direction for both academia and industry, manual analysis is encouraged in an initial stage to analyze the characteristics of the objects. To facilitate our study, we also use a similar community-based technique used in [34] to cluster the unlabeled apps. Note that, the feature extraction mechanism is different from that in these two tasks (i.e., malware and clustering), because our goal of proposing the data sampling shown in Fig. 1 is not to achieve a high accuracy for malware detection, but to find a promotive way for manual analysis of unlabeled apps.

B. *Analysis of Android Grayware*

Andow et al. [4] primarily focused on the Android grayware collected from Google Play Store. They developed lightweight heuristics to identify Android grayware, which primarily combines text analytics by using app reviews. However, the published app dataset is limited to apps that pass the malware detection process of Google Play Store. In contrast, there

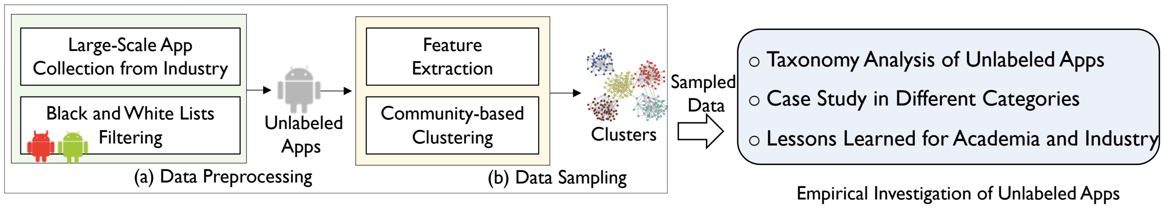

Large-Scale App Collection from Industry
Black and White Lists Filtering
Unlabeled Apps
Feature Extraction
Community-based Clustering
Clusters
Sampled Data
(a) Data Preprocessing
(b) Data Sampling
○ Taxonomy Analysis of Unlabeled Apps
○ Case Study in Different Categories
○ Lessons Learned for Academia and Industry
Empirical Investigation of Unlabeled Apps

Fig. 1: Workflow of our study

is also a prevailing opinion that simply treating grayware as malware can expose users' privacy [35]. Moreover, there also exists a neutral definition that grayware can legitimately collect user information [36], so the users are willing to trust the data collection when they use apps or they just ignore the potential risks induced by the apps. Overall, it is an initial study on Android grayware with several limitations. In this work, the dataset collected from our industrial partner is filtered by an ensemble industrial scanning engine, which is better initially processed than the samples only collected from Google Play Store directly. In addition, previous work lacks manual analysis to further understand the characteristics and behaviors of the apps. Our manual analysis on the industrial unlabeled apps unveils that several categories of unlabeled apps belong to grayware introduced in the first grayware work [4]. Furthermore, we also inspect several new categories to foster further research.

C. *Analysis of Android Apps*

A large number of studies focus on analysis of Android apps based on the large-scale dataset collected from industrial companies or commercial Android markets [6], [37], [38]. For example, Fan et al. [6] investigated the characteristics of fail-stop errors in Android apps (i.e., app crashes) in order to help to understand the cause of app crashes, avoid inducing such errors during app development, and improve the quality of Android apps. Wei et al. [37], [38] studied the characteristics of compatibility issues due to Android fragmentation, including the root causes and the common patches, to help to detect such fragmentation-induced compatibility issues in Android apps. However, existing work focuses on improving the quality, rather than understanding the characteristics of app itself. Our work focuses on this scope regarding the industrial unlabeled apps.

III. Workflow of Our Study

In this section, we first briefly introduce the study design, and then introduce the data preprocessing and sampling process on a large number of unlabeled apps from our industrial partner. After that, with the help of existing academic clustering algorithm [18], we take an in-depth investigation on the characteristics of these apps by employing manual analysis.

Fig. 1 shows our study workflow, which contains three main phases. (1) *Data preprocessing* starts with the "unlabeled Android apps", which fail to be categorized into either malware families or benign samples when being filtered by an ensemble industrial scanning engine; (2) *Data sampling* distillates the data from the majority of the unlabeled app clusters according to the clustering results, the sampled data is further used for manual analysis. (3) We further conduct *manual analysis* to investigate the common behaviors in different clustered categories with substantial efforts. To ensure the reliability of the analysis results, we cross-validate the results among three of the co-authors. We accept the categorization results only when all of us agree on it. Apart from the in-depth analysis, we demonstrate the characteristics of unlabeled apps in each category through a case study.

A. *Data Preprocessing*

Data preprocessing mainly contains two steps: Data collection and Data filtering. We collect the "unlabeled Android samples" from our industrial partner, Pwnzen Infotech Inc.. Our partner crawls from various Android markets including Google Play and other third-party markets, and also tracks the network traffic under a control environment from 2017 to 2020 and obtains the APK files. Before storing these apps on the server, an ensemble scanning engine including several industrial scanning engines (e.g., Qihoo 360 and Pwnzen Infotech Inc.) is used to classify these collected samples. The ensemble engine is based on the black-list (e.g., malicious code signatures) and white-list (e.g., official certificates) provided by these industrial companies. The black-list and white-list are timely updated and shared between these industrial companies. According to the scanning result of the ensemble engine, the malicious apps are identified and collected for further malware analysis [21], [22] to extract new malicious features. The benign apps are filtered and collected for app quality analysis. The remaining ones are "Unlabeled". Such "Unlabeled" samples fail to be categorized into neither malware families nor ordinary benign samples according to the black-list and white-list. The industrial companies collected thousands of such unlabeled apps, which would lead to potential security or privacy threats for app users. Negative impacts on mobile system performance and user experience also occur in these apps. Therefore, it is urgent and essential to understand the characteristics of these samples for industrial companies like our partner, which calls for further collaboration with us. To investigate and understand the characteristics of these apps, we finally collect 22,886 unlabeled apps in total to conduct an empirical investigation.

B. *Data Sampling*

To investigate the characteristics of unlabeled apps, we first employ the community detection technique based on the extracted features to cluster unlabeled apps into different clusters, and sample a statistically-significant number of apps

TABLE I: Our selected features

Category	Feature	#Original	#Used
Syntax Features (used by at least 0.1% apps)	Permissions	717	119
	API Calls	1,397	852
	Intent Action	845	34
	Intent Category	38	4
	Hardware	33	12
String Features	Source Code String	100	100
	XML Code String	768	768
Total	7 types	3,898	1,889

for further investigation (the confidence level is 95% and the confidence interval is 5%).

Feature Extraction. Following the widely-used general feature types in Android malware detection [19], [20], [14], [21], [22], we extract the following features such as permissions and API calls (Table I) by decompiling APKs into `Smali` code via Apktool [39]. Note that, we, here, do not take the malware-specific features (e.g., semantic, security-related, and context-aware features) into account, whose goal is to best represent the malicious behaviors in order to distinguish malware from samples.

- Permissions, which are declared in the AndroidManifest.xml file to indicate the necessary permission to access specific resources.
- API calls, which are invoked by the app to interact with the underlying Android system, such as sending SMS.
- Intent, which is regarded as the "medium" to transfer data between different components. We consider two types of data, i.e., action and category, as the features.
- Hardware, which is requested by the app to get access to components such as *NFC* and *GPS*.
- Strings, which indicate values of variables. We collect strings from two sources: declared in `Smali` files and defined in strings.xml files.

Specifically, for permissions, API calls, intent, and hardware features, we first extract them from AndroidManifest.xml files and `Smali` files, and obtain the union set of them. Since the size of the union set is too large, to avoid sparse data, we filter out the features that are used by less than 0.1% apps, only use the vectors of the remaining features for clustering. We then decode the features into one-hot vector, and set the value to "1", indicating the corresponding feature is requested by the app, otherwise the value is set to "0". For the strings extracted from the `Smali` files, we split them by predefined punctuations such as ".", "//", and "—", and also camel cases for method/class/API names [40], e.g., "AdActivity" to "Ad" and "Activity". To encode them with rich semantic representations, we resort to the popular GloVe pre-trained word vectors [41], a published repository containing 27 billion English vocabularies. For each app, we extract the 100-dimension vectors for the tokens from the `Smali` files and take the average as the representation for the strings from the source code of the app. With respect to the strings extracted from the strings.xml file, since they are generally mingled with multiple languages including English, Chinese, Hindi, etc., we turn to the multilingual word embeddings released by Google [42], and use the similar strategy as we encode strings from `Smali` files. The strings.xml file of each app is represented as 768-dimension real-value vectors.

Community Detection. Community detection algorithm has been widely adopted in malware detection for clustering malware into different families [43], [44]. In this paper, to effectively determine the categories of the numerous unlabeled apps, we follow the typical strategy [34] to preliminarily cluster these apps, from which we sample a statistically-significant number of apps for further manual analysis.

Specifically, we first build a bi-directional graph of the unlabeled apps. Each app is one node in the graph, and the edge weight between two nodes is computed based on the similarity of the corresponding two apps. The similarity score is determined by the cosine similarity [45] between their feature vectors. Note that not all the unlabeled apps are incorporated in the graph. According to Fan et al. [34], an app is removed from the graph if it presents loose relations with other apps (i.e., the similarity scores with all the other apps are lower than 0.75[1]). Such apps can be regarded as outliers and are not the interest of this work. After this step, 21,403 out of 22,886 apps are remained for subsequent analysis.

Then we use the popular community detection algorithm, infomap [46], to cluster the unlabeled apps in the graph. We choose the infomap algorithm since it presents superior performance on app clustering than other community detection approaches in prior studies [34]. After conducting the clustering step, we obtain 594 clusters with the count distributions of the apps shown in Fig. 2. Since we focus on identifying whether the unlabeled apps exhibit prominent categories, we only choose the top clusters for analysis. In this work, we select the top 15 clusters which include 10,918/21,403 apps, occupying 51.0% of the apps in the graph, for further analysis. On average, each of the top cluster has 727 apps.

Finally, we sample representative apps from each cluster. The representative score of each app is computed as the sum weight of all the adjacent nodes in the graph, based on the intuition that the apps presenting more similarities with surrounding apps tend to be more representative of the cluster [47]. We extract the top 25 apps from each cluster for manual analysis. In total, we manually analyze 375 apps (out of the 10,918 unlabeled apps) providing us with a confidence level of 95% and a confidence interval of 5%.

IV. Empirical Study

Based on the unlabeled apps we sampled from each cluster, we take an in-depth manual analysis on these apps from the following aspects: **1)** We summarize the taxonomy of these unlabeled apps by manually understanding the characteristics and common behaviors of each app; **2)** We present typical cases from the sampled unlabeled apps to illustrate the abnormal behaviors of apps; **3)** Finally, we also summarize the lessons learned from the study on such unlabeled apps, and propose useful insights for follow-up research.

[1]0.75 is determined following the prior study [34].

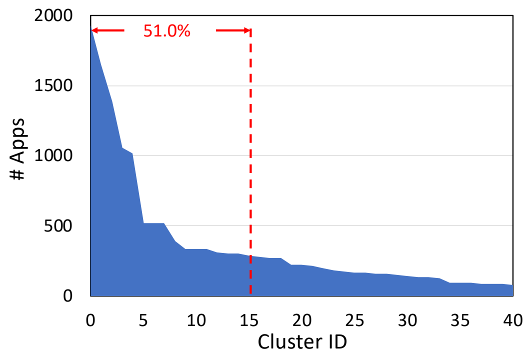


Fig. 2: Distribution of app numbers for the top 40 clusters. The top 15 clusters occupy 51.0% of all the clustered apps.

To conduct the manual analysis, we use 3 real mobile devices (i.e., Samsung Galaxy S10+, Samsung Galaxy S10, and HUAWEI Mate 20 with Android 9.0 OS) rather than Android emulators to ensure the real execution environment. We also use several reverse engineering tools to help analyze the unlabeled apps, such as Apktool [39] and jadx-GUI [48]. Apart from these tools, we use Android logcat [49] and instrumentation techniques to help observe the common behaviors of apps from different clusters. Meanwhile, all manual analysis is done under a control environment (in our research lab) to avoid spreading the threats to real app users.

A. *Taxonomy of Unlabeled Apps*

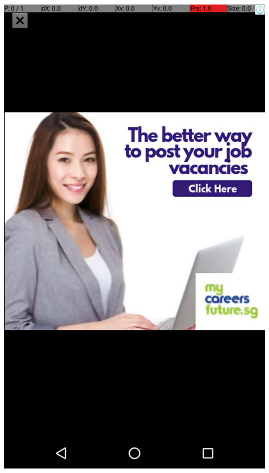
To summarize the categories of unlabeled apps and their common characteristics and behaviors, we select the top 15 clusters where each cluster provides us 25 apps for manual analysis. Based on the observed common behaviors from different unlabeled app clusters, the categories are summarized as follows.

Adware [50]. Unlabeled apps in this type usually pop up advertisements or notifications on the screen that disturb mobile users. For example, some of these apps pop up advertisements very frequently, which largely affects the normal usage of the apps and annoys users. Such advertisements either redirect users to the app market or websites to attract users to download apps or register on the websites, which may download malicious apps or leak private information. In addition, when analyzing this category, we find many redirected website links are forbidden by mobile systems due to security threats. Specifically, some advertisements provide a closing button, but the button will redirect to a website via mobile browser (the app is named *Lighting*). Some advertisements even have a full-screen image covering the whole screen of the device, however without a close or cancellation button, thus the users are forced to click on the advertisement if they want to continue using this app. For example, Fig. 3a shows a full-screen advertisement, inducing users to click on the link to register by showing a 30-second video, however without showing the close button on the top left. The close button only shows after the video is finished. Therefore users have no choice in the first 30 seconds but to click on the advertisement. Fig. 3b shows a frequent advertisement at the bottom. Such advertisements negatively affects user experience.

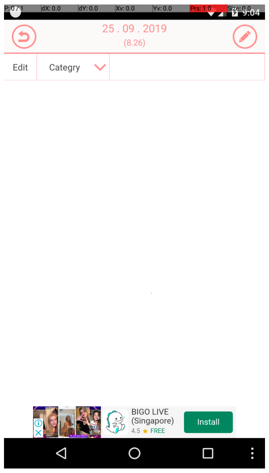
Fake Apps [5]. Following the definition in [5], fake apps are apps that imitate the corresponding official ones or look almost the same as their official correspondences, however without official certificates. The ultimate goal of fake apps is to elicit downloads or manifest malicious behaviors by plagiarizing the famous official ones. For example, as shown in Fig. 4, the fake Fruit Ninja app (i.e., Fig. 4b) mimics the similar functionalities as the original one (i.e., Fig. 4a) to attract app users. For most normal users, they lack rich experience to distinguish whether the current is a fake app. Many fake apps are uploaded to third-party markets and attract more downloads with the help of popularity of the official apps. From our observation, we find that such apps contain similar content and functionalities compared with the corresponding original one, however downgrade some functionalities. Meanwhile, there are dozens of fake apps that sometimes would target one official app. Fig. 3c shows some fake apps for an official music app, with similar icons or app names, we can see some of them are even with the same icon. Such fake apps negatively affect the use of the corresponding official apps.

Redirected Downloader Apps. Unlabeled apps in this type link to external downloads and installations of other apps or plugins. Such downloading and installation processes can be triggered when the app is launched, or triggered by user interaction, pretending as installing a necessary component providing the full functionalities, however it would also redirect to other downloading pages to downloading other unnecessary apps. Some apps (e.g., one app named *Beta WhatsApp* in Fig. 5a) do not actually provide the claimed service, instead, they contain a single page showing the redirect URL for users to download the corresponding apps that provide the claimed service. After users install the provided necessary component, the app may not provide the claimed functionalities, either. Moreover, sometimes the installation request dialog pops up again and again even if the users have already install the required plugin. Such downloading apps usually conduct the downloading and installation process by requesting permissions from users, not silently downloading in the background like what malware do. Fig. 3d shows a redirect downloading app named *Finance Pro*, when users open the app, it directly redirects to download a necessary component in order to provide the full functionalities.

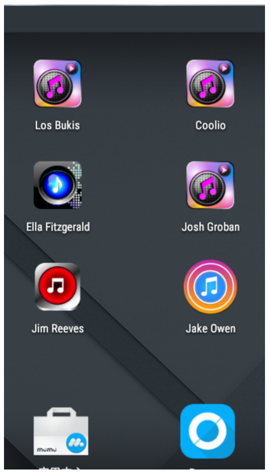
Redirected Promotion Apps. Different from adware, redirected promotion apps redirect users to external web content outside of the app, such as following accounts on social media (e.g., Facebook, Instagram (shown in Fig. 5b), or other sales apps), usually by opening a web browser page. Sometimes users are even required to actually click on the promotion links to do some special tasks (e.g., promoting third-party apps) in order to continue using the current app, otherwise the users are unable to use the services provided by the app. When the third-party apps are downloaded by users, the current app will check whether the downloaded app is installed on the current device to further unlock the app. Fig. 3e displays such an app, which is a *WI-FI hacker* app, however in order to use the functionality of hacking WI-FI, users are forced to click on



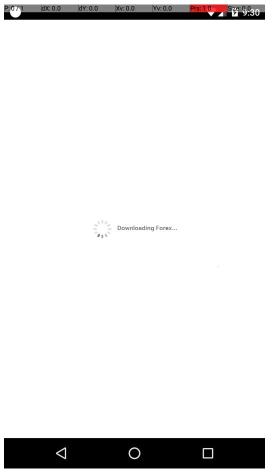
(a) Adware (Full screen)



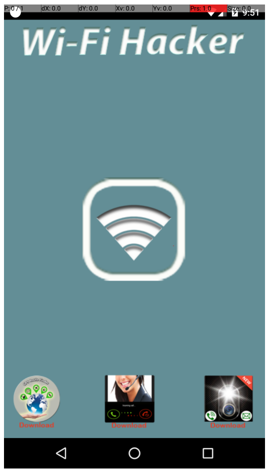
(b) Adware (Bottom)



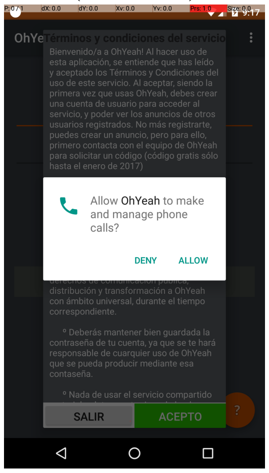
(c) Fake Apps



(d) Redirected Downloader



(e) Redirected Promotion


OhYe Términos y condiciones del servicio

Bienvenido/a a OhYeah! Al hacer uso de esta aplicación, se entiende que has leído y aceptado los Términos y Condiciones del uso de este servicio. Al aceptar, siendo la primera vez que usas OhYeah, debes crear una cuenta de usuario para acceder al servicio, y poder ver los anuncios de otros usuarios registrados. No más registrarte, puedes crear un anuncio, pero para ello, primero contacta con el equipo de OhYeah para solicitar un código (código gratis sólo hasta el enero de 2017)

Allow OhYeah to make and manage phone calls?

DENY ALLOW

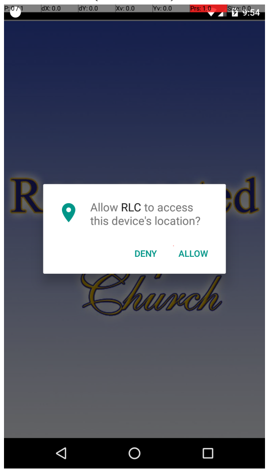
distribución y transformación a OhYeah con ámbito universal, durante el tiempo correspondiente.

º Deberás mantener bien guardada la contraseña de tu cuenta, ya que se te hará responsable de cuarquier uso de OhYeah que se pueda producir mediante esa contaseña.

º Nada de usar el servicio compartido

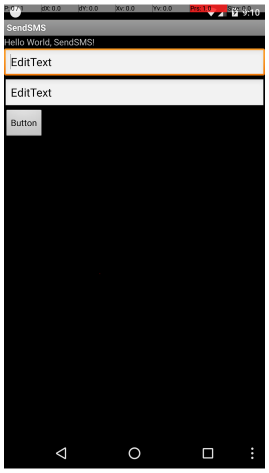
SALIR ACEPTO

(f) Dialing/SMS -Managing App

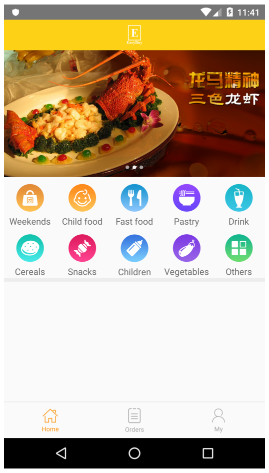

Allow RLC to access this device's location?

DENY ALLOW

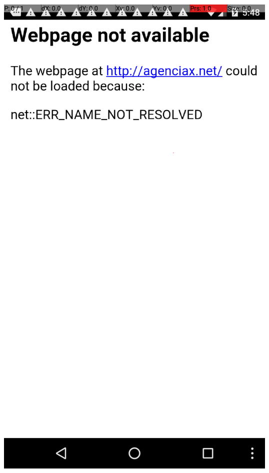
(g) Data Collection App



(h) Demo App



(i) Unfinished App

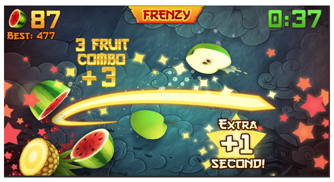

Webpage not available

The webpage at http://agenciax.net/ could not be loaded because:

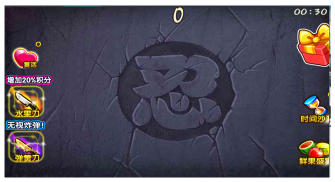
net::ERR_NAME_NOT_RESOLVED

(j) Broken App

Fig. 3: Examples of unlabeled app categories

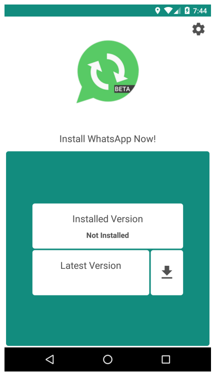


(a) Original

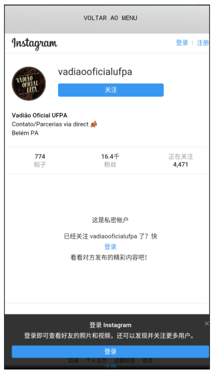


(b) Fake Fruit Ninja

Fig. 4: Original and Fake Fruit Ninja app



(a) Redirected Downloader (WhatsApp)



(b) Redirected Promotion (Instagram)

Fig. 5: Examples of Redirected Downloader/Promotion apps

one of the three promotions at the bottom of the page.

Dialing/SMS-Managing Apps. Dialing/SMS-Managing apps usually request permissions related to managing phone calls (e.g., making/receiving phone calls) or SMS (e.g., sending/reviewing SMS) from users, and conduct such behaviors when users do not attempt to do so. Such app may leak user personal information such as phone number and may also cause unexpected financial cost (e.g., calling charge, SMS charge). For example, Fig. 3f shows such an app named *OhYeah*, there is no functionality related to making phone calls. However it requests permissions of making phone calls, which is not what it requires to provide the claimed services. The behavior of SMS-Managing Apps is similar to Dialing-Managing Apps, they ask user to set the current app as the default app for controlling sending/reviewing SMS. Note that, it is difficult to identify the potential malicious behaviors within apps only by the declared permissions related to phone calls/SMS, which is one possible reason that they are not labeled to malicious and appear to be unlabeled apps.

Data Collection Apps. This type usually forces users to enter personal information (e.g., age, gender, photo number, address) or request sensitive permissions (e.g., access to location and contacts) to get access to the app's functionality. Sometimes the requested data is not what is indeed required to provide the claimed service. In addition, for Android platform, the Privacy Policy [51] is designed for app developers to declare what user data will be collected, why it will be collected, and how it will be used. The declaration is also consistent with GDPR [52]. Fig. 3g displays an app (named *RLC*) requesting access to location before launching the app. In fact, *RLC* is the abbreviation of "resurrected life church", which is not related to location information. In other words, accessing location is not a necessary behavior to provide the normal functionalities. However, users cannot use the app if they deny such permission requests, which definitely dissatisfies users.

Demo Apps. This type of unlabeled apps shares a common characteristic that the quality of such unlabeled apps is extremely low, only containing very limited functionalities for exercise or demonstration purpose. In detail, the code structure and Graphical User Interfaces (GUI) design are extremely simple with only one page. Some of them are only composed of default widgets which are automatically generated from Android Studio. For example, Fig. 3h is an app for demonstration or testing of some basic components, only containing two EditTexts and a Button with a single page. Such apps are not ready for release, which would dissatisfy users with such simple functionalities.

Unfinished Apps. This type of apps contains unfinished modules, i.e., some functionalities are not fully implemented, leaving some buttons unresponsive after users click on it, or leaving some blank contents in the pages. Fig. 3i shows an unfinished app, which is a suggestion app with no implementation, and the image buttons (e.g., "Weekends" button, "Child food", and "Fast food") are not responsive, either, thus users cannot use the corresponding functionalities. The only way to interact with it is to close it. Such apps fail to fulfill the expectation of users with the functionalities that they claim to provide, leading to bad user experience.

Broken Apps. From our observation, this type of unlabeled apps usually employs WebView [53], a kind of view in Android, to show web pages in the current app instead of a standard web browser like Chrome. These web pages usually contain important parts, if not all of the content and services that the app is trying to offer are ready to display, the app would fail to access these pages, making the app useless to users. Interestingly, the URL of the web pages are usually invalid, thus they cannot provide the corresponding service to users. For example, Fig. 3j shows a broken app with invalid URL, leading to a "Webpage not available" error. Obviously, such apps are useless for users with just an error page.

Porn Apps. This type contains porn contents and frequently pops up a dialog, attracting users to click on the links to externally download the provided "video player". The so-called video player is usually an advertising app or a malicious app collecting personal information. We find that some porn apps share the same icon and UI design, and the app names are also very similar, with only several different letters. Moreover, from our observation, these apps usually request a large number of permissions to conduct their abnormal behaviors. Such frequent pop-up windows have a negative impact on user experience, and pose potential security threats to users. Moreover, the content in porn apps is a public health risk for users, especially for the young mobile users. It is a great demand to distinguish the porn apps for app users in practice.

Background Service Apps. unlabeled apps in this category share a common characteristic that they keep consuming the resource (e.g., accessing location, refreshing periodically) in the background. Sometimes such background service cannot be stopped by killing the background tasks, which negatively affects user experience. We take a further investigate on the code of such apps and demonstrate two cases in Section IV-B to show the background behaviors.

In summary, we distill 11 categories of unlabeled apps based on the concrete app behaviors by an in-depth manual analysis.

B. Case Studies of Unlabeled Apps

In this section, we demonstrate a number of real cases found in the clustering results. These samples are observed with typical behaviors representing the unlabeled app categories.

```
public void mining(){
  // The mining process runs in a background service
  miner.getSettings().setJavaScriptEnabled(true);
  // mining.js contains Coinhive library
  miner.loadData("mining.js", null);
}
```

Listing 1: Simplified code snippet of *Urban Pulse live wallpaper*

1) Cases of Background Service Apps: Background service apps refer to apps that stay running in the background for their own purpose. One example is an app called *Urban Pulse live wallpaper*, which is used to configure live wallpaper for mobile devices. The corresponding code snippet is shown in Listing 1. As we can see, in addition to the basic functionality, it contains a special module using `JavaScript` to mine Bitcoin [54] in the background and keeps using the CPU resources of the mobile system. The mining behavior can hardly be noticed by mobile users or security analysts due to the attribute of background services. These characteristics definitely cause serious user annoyance. Another example is an app called *Geometry Dash*, which requests the *WAKE_LOCK* permission, which can be used to unlock the screen and show the content of the device. Once *Geometry Dash* starts running on the device, the ads will continuously pop up, even if the user closes the ads. After investigating the decompiled source code of the app, we find that it uses a timer to periodically pop up its continuous advertising service in the background. This kind of background service greatly annoys mobile users and compromises user experience.

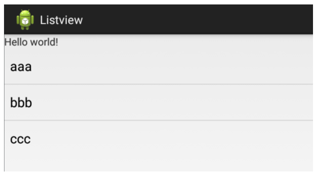


Fig. 6: Demo App (ListView)

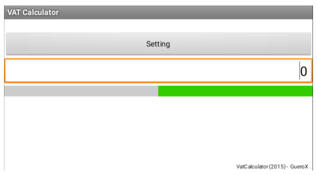


Fig. 7: Unfinished (Button)

2) Cases of Dialing/SMS-Managing Apps: Such apps usually make phone calls or send SMS unexpectedly without users' notice. An app called *OhYeah*, which is a shopping app. This app requires users to first register by entering with users' names, emails, telephone numbers in the registration page, then users can view and buy items. The simplified code snippet of the registration module is shown in Listing 2. After the user clicks the submit button, the app will first get the phone number from the input field, and make a call to the telephone number automatically and unexpectedly (lines 7-8 in Listing 2). Such behavior is abnormal for a shopping app.

```
 1 txtNumber = itemView.findViewById(R.id.pf_number);
 2 btn_Llamar = itemView.findViewById(R.id.btn_llamar);
 3 btn_Llamar.setOnClickListener({
 4     public void onClick(View view) {
 5       mNumber = "tel:" + RecView.txtNumber.getText();
 6       // Make phone calls unexpectedly
 7       Intent llamar = new Intent(
 8       "android.intent.action.CALL",Uri.parse(mNumber));
 9       Vendedor.startActivity(llamar);
10       }
11   });
12 }
```

Listing 2: Simplified decompiled code snippet of *OhYeah*

3) Cases of Demo Apps: Demo apps are too simple to provide useful services to users, usually containing code-demos or pure texts. Although no harm will be done to app users, such unlabeled apps are useless and should never be published as an item on the Android markets. As shown in Fig. 6, the app is called *ListView*, where there is only one page called `MainActivity` that creates and displays a `ListView`, without any other events attached to the list items. Listing 3 shows the corresponding code snippet. This app is only a basic demonstration of `ListView` for a beginner in Android development, but is totally useless to users on any Android markets.

```
public void onCreate(Bundle bundle){
  String[] values = new String[] {"aaa", "bbb", "ccc"};
  ArrayAdapter<String> adapter = new ArrayAdapter<
      String>(this,
    android.R.layout.list_view_item, values);
    setListAdapter(adapter);
}
```

Listing 3: Simplified code snippet of *Demo_ListView*

4) Cases of Unfinished Apps: Unfinished apps refer to apps that have obvious unfinished features, misleading users to expect more functionalities than what are actually implemented. Fig. 7 shows the UI page of an unfinished app called *VATCalculator*, which is used to compute the consumption tax people need to pay based on the value. The simplified decompiled code snippet of *VATCaculator* can be referenced by Listing 4. This app has a text-input field for users to enter values, and several buttons to perform calculations. Since the buttons are clearly visible to users, the users would naturally expect these clickable buttons to perform some calculation tasks. However, through an in-depth investigation of the decompiled source code of the app, we find that it does not implement the functionality of getting the value from the text-input fields (i.e., `onTouch`, `TouchDown`, and `TouchUp`), and it also fails to finish the functionality of performing calculation after clicking the "calculation" buttons (i.e., `onTouch` in the source code). As a result, the users can only observe the buttons changing colors after being clicked, however without any other reactions. It is obvious that the underlying functionalities for these buttons are not implemented, and the functionalities of this calculator are unfinished.

```
public boolean onTouch(View view, MotionEvent me){
/* Only changing the button's color, no calculation or
     value extraction from the text field */
if (me.getAction() == 0){
  if (ShowFeedback()){
    view.getBackground().setAlpha(70);
  }
  TouchDown();
}else if(...){
  ...
  TouchUp();
  }
}
// Only dispatching events, no calculation
public void TouchDown(){
  dispatchEvent(this, "TouchDown", new Object[0]);
}
// Only dispatching events, no calculation
public void TouchUp(){
  dispatchEvent(this, "TouchUp", new Object[0]);
}
```

Listing 4: Simplified code snippet of *VATCaculator*

5) Cases of Broken Apps: Broken apps are clustered as a category with malfunctioning features or failures identified during the usage. For example, *Jubaoyaojin* is an investment app that fails in accessing a web page (i.e., *http://www.jubaoyinjin.com*) immediately after it starts, possibly due to the shut-down server or the expired domain name.

```
1 public class MainActivity extends BaseActivity{
2 public void initView(){
3   String json = Utils.readJsonFile("property.json");
4   url = getUrl(json);
5   // Invalid url!
6   getCurrentFragment().getBrowser().loadUrl(url);
7   }
8 }
```

Listing 5: Simplified code snippet of *Jubaoyaojin*

After analyzing the corresponding decompiled code in Listing 5, we can see that whenever the app starts, it immediately posts a web request to load its official website (Line 6). The URL is actually stored in a JSON file (Line 3), and is preloaded into the main page when a fragment is created. However, the requested URL is invalid due to the shut-down server or the expired domain name. As a consequence, the user cannot even load the first page of this app and the app becomes useless and displays nothing but an error message.

C. *Comparison with Existing Grayware Study*

We first compare our identified categories of unlabeled apps with recent research results of grayware [4]. Grayware, defined by Andow et al. [4], represent the apps that contain annoying, undesirable, or undisclosed behaviors that cannot be classified as Android malware. Among our 11 identified unlabeled app categories, we have the following findings:

- 8 new categories are discovered based on our analysis, which are different from Android grayware studied in [4], including **Redirected Downloader Apps, Redirected Promotion Apps, Data Collection Apps, Demo Apps, Unfinished Apps, Broken Apps, Porn Apps,** and **Background Service Apps**.
- 2 categories (**Fake apps** and **Dialing/SMS-Managing Apps**) strictly subsume two known categories **Impostors** and **Dialers** [4], respectively. More specifically, apps in the category *Imposters* usually impersonate through repackage techniques, however, some apps impersonate functionalities of popular apps to attract installation and usage from app users. Due to this characteristic, we called this category as **Fake Apps** in this work. **Dialing/SMS-Managing Apps** not only contains the type of **Dialers** defined in [4] but also apps that have SMS behaviors.
- The remaining 1 category is completely matched (**Adware**) with the one in [4].

Apart from the 11 identified categories, we find 5 categories are unrevealed in our dataset compared with the recent research results [4], including Prankware, Scareware, Rooting Tools, Remote Access Tools, and Hijackers.

In fact, according to our empirical investigation of unlabeled apps, it seems that almost all the unlabeled apps have “gray” attributes instead of “malicious” attributes. They do not have obviously malicious behaviors as malware, so it is not easy to label them in industry and they are often easily ignored by analysts. However, according to the comparison results, we note that several categories of grayware are in the wild that lead to significant negative impacts on mobile users. The previous grayware study [4] only focused on the apps collected from Google Play Store, in this paper, we distill more categories and understanding of the gray area of the mobile world, which thanks to the real dataset from industry instead of only the official Android market.

V. Lessons Learned and Discussion

A. *The Boundary of Malware and Grayware*

According to our study, we distill many new grayware categories with negative attributes for users. In practice, the boundary of Android malware and benign apps might be blurry and subjective [15], [16], [21], it is also true for the two types of Android apps (i.e., malware and grayware). The definition of special types depends on specific scenarios and intentions of the apps. For example, for fake apps, they should be regarded as malware if they are repackaged to conduct malicious behaviors, however if they are designed to only receive more attention by imitating certain popular apps, they belong to the grayware category. Also, adware often wander between Android malware and grayware [50]. Similarly, some individual users choose to root their own devices (i.e., gain administrative or superuser permissions) by using third-party apps to get more fancy functionalities, which seems to be a normal operation. However, some analysis tools regard the rooting app as a malicious one since it gains the administrative permissions that can manipulate system applications (e.g., alter, replace) with privileges, giving highways to malware to conduct their malicious behaviors. Therefore, the boundary of them is unclear. In fact, manual analysis is an initial step to understand the characteristics of the grayware apps in the wild rather than only the official apps and further identify the boundary more clearly, especially for the unlabeled apps that we have analyzed in this paper.

B. *Industrial Profit Chain of Unlabeled Apps*

From our manual analysis, we find that some behaviors of unlabeled apps may unveil special industrial profit chain in the real world, which means that this study is a peephole to see the back-end profit mode of the unlabeled app ecosystem. For example, for the *Adware Apps*, we observe that the advertisements in different app categories redirect to the same URL to display the same content to maximizing profit of advertisement providers. Similarly, for the *Redirected Promotion Apps*, promotions in some apps redirect to the same promotion product such as browsers and social apps, such cases appear frequently in our dataset. In the future work, we aim to understand the back-end industrial profit chain to further help to identify the corresponding unlabeled apps, with the help of the dataset provided by the industry, which is a more challenging research for both academia and industry.

C. *Low Quality Unlabeled Apps*

Apart from the apps that are developed for making profit and have negative impact on user experience, we also find that some unlabeled apps are clustered by their own attributes including *Demo Apps, Unfinished Apps*, and *Broken Apps*, causing user dissatisfaction with extremely low quality. Such kinds of unlabeled apps are hardly a potential threat to the Android system or users’ personal data due to the limited complexity or malfunction. As a result, this type of low-quality unlabeled apps is underestimated and neglected in previous studies [4], [55]. Apart from such apps with limited functionalities, we also observed that a large number of apps suffer from crashes once the apps launch. User cannot use them at all. For the third-party Android markets, they should strengthen supervision of app quality to avoid spreading these low quality apps to users.

D. *Propagation of Unlabeled Apps*

Compared with malware, unlabeled apps can be propagated faster, as they are less likely to be identified by anti-virus detection systems. Before our study on unlabeled apps, there is no research focusing on this field, let alone the classification approaches for classifying them to each category. According to our analysis result, none of the unlabeled apps involves behaviors of attacking the Android system or gaining root access (i.e., privileged permissions) of the device. In contrast, Android malware usually focus on data leakage and vulnerabilities of the mobile system in order to breach system protection and gain root access of the device and further control it [56], [57]. Such nature of malware leads to different detection abilities of the Android anti-virus mechanism for malware and unlabeled apps (i.e., unlabeled apps and grayware are less likely to be identified). Once the system patches the vulnerability, substantial efforts are needed to find another vulnerability in order to develop a new malicious app. On the contrary, unlabeled apps can make profit form network-traffic theft or advertisements. Therefore, unlabeled apps only require relatively simpler development for a shell that carries components such as advertisement frameworks or network-traffic theft functionalities. As shown in Fig. 4b, a lot of fake apps are observed in unlabeled app categories, suggesting that only minor modification is needed for unlabeled app developers to produce a large number of unlabeled apps. Moreover, normal system updates do not affect unlabeled apps as their behaviors do not depend on the patched leakages, and unlabeled app developers can reuse most parts of the old versions. As a result, a lot of repeated icons, UI design, and even app names are observed in most of the unlabeled apps.

E. Feedback from Industrial Partners

According to the clustering results from both industry and academia, we find that the community-based/clustering techniques are not effective in characterizing unlabeled apps in practice. In fact, before the study, our partner leveraged the string features extracted from APK files and employed N-gram to cluster these apps. The result showed it was able to identify some *Porn Apps*, but ineffective to cluster other categories. Although it is a great demand for our industrial partner to use a multi-class classifier to distinguish the unlabeled apps, the classifier is still ineffective due to the multi-behaviors within each app. Thus, the labeled apps are especially more important for this task. To get useful feedback about the analysis on unlabeled apps from the industry, we had face-to-face meetings with them to discuss the characteristics of our analyzed results, they acknowledged our understanding and findings on unlabeled apps, meanwhile, they mentioned that compared to academic researchers, the industrial stakeholders are more interested in unlabeled apps. They aim to reduce the security risks and negative impacts as much as possible from a business perspective based on our analysis results.

Benefits to our industry partner. 1) They are constructing a dataset of all categories of unlabeled apps based on our empirical investigation, and further leverage learning-based approaches to classify these apps from others to mitigate the potential threats. **2)** It is difficult to identify all these categories through one model in the initial stage even based on our study due to the limited labeled dataset. With the labeled apps based on our study, the industrial analysts have defined and extracted special "features" for different categories in order to achieve the goal of characterizing different types of unlabeled apps. In fact, our partner has implemented several "feature" definitions on their app analysis platform, Anonymous platform, according to our analysis results. As for Data Collection Apps, Porn Apps, Background Service Apps, and Fake Apps, they leveraged different defined features to identify them with a great performance. For example, Fake apps [5] have been analyzed and studied by them in depth under our collaborations. They also encourage academic researchers to pay more attention on what are the real demand of app analysis in practice.

VI. Threats to Validity

The incompleteness of black/white-lists. This potential threat is from our initial data collection process. The unlabeled Android apps are acquired from an industrial company, i.e., Pwnzen Infotech Inc., and they use the black-list and white-list to filter out the known malicious and benign samples, respectively. It is hard to guarantee that the remaining "industrial unlabeled Android apps" do not contain any malicious or benign samples. However, manual analysis with cross-validation can eliminate such negative effect on the dataset as much as possible.

Manual analysis. Since we apply manual analysis to categorize these unlabeled apps, and investigate the characteristics of them, there may be bias regarding the analysis results. Thus we cross-validate the taxonomy and the characteristics across co-authors to make the result more reliable and convincing.

VII. Conclusion

In this paper, we conduct the first data-driven analysis of industrial unlabeled Android apps, leveraging the cluster algorithm to categorize unlabeled apps at scale from industry (i.e., data sampling), and performing an in-depth manual analysis. We systematically study the clustered unlabeled app categories by their common characteristics, and summarize 11 unlabeled app categories and their corresponding behaviors. Along with the categories, we also provide a few case studies of several categories. The other new categories can be published as a benchmark to foster further research on grayware, which thanks to the dataset of unlabeled apps from industry. Finally, we highlight our discoveries based on the comparison with Android malware and grayware, several categories are partial/completely match with the first Android grayware study. Meanwhile, the industrial partners acknowledged our understanding and findings in this paper, and keep in touch with us for further investigation and cooperation.

Acknowledgements

We sincerely thank Zhushou Tang, affiliated to PWNZEN InfoTech Co., LTD, for the valuable dataset of Android apps. This work was partially supported by the National Natural Science Foundation of China (Grant No. 62102284, 62102197, 62002084, 62072309, 61761136011), and Stable support plan for colleges and universities in Shenzhen under project No. GXWD20201230155427003-20200730101839009.

REFERENCES

[1] (2018) App download and usage statistics. [Online]. Available: http://www.businessofapps.com/data/app-statistics

[2] S. Chen, L. Fan, C. Chen, T. Su, W. Li, Y. Liu, and L. Xu, "Storydroid: Automated generation of storyboard for Android apps," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 596–607.

[3] X. Jiang and Y. Zhou, "Dissecting Android malware: Characterization and evolution," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 95–109.

[4] B. Andow, A. Nadkarni, B. Bassett, W. Enck, and T. Xie, "A study of grayware on Google Play," in *Proceedings of the 2016 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2016, pp. 224–233.

[5] C. Tang, S. Chen, L. Fan, L. Xu, Y. Liu, Z. Tang, and L. Dou, "A large-scale empirical study on industrial fake apps," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*. IEEE Press, 2019, pp. 183–192.

[6] L. Fan, T. Su, S. Chen, G. Meng, Y. Liu, L. Xu, G. Pu, and Z. Su, "Large-scale analysis of framework-specific exceptions in Android apps," in *Proceedings of the 40th International Conference on Software Engineering (ICSE)*, 2018, pp. 408–419.

[7] L. Fan, T. Su, S. Chen, G. Meng, Y. Liu, L. Xu, and G. Pu, "Efficiently manifesting asynchronous programming errors in android apps," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 486–497.

[8] S. Chen, T. Su, L. Fan, G. Meng, M. Xue, Y. Liu, and L. Xu, "Are mobile banking apps secure? what can be improved?" in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 797–802.

[9] C. Gao, W. Zheng, Y. Deng, D. Lo, J. Zeng, M. R. Lyu, and I. King, "Emerging app issue identification from user feedback: experience on WeChat," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*. IEEE Press, 2019, pp. 279–288.

[10] S. Chen, L. Fan, G. Meng, T. Su, M. Xue, Y. Xue, Y. Liu, and L. Xu, "An empirical assessment of security risks of global Android banking apps," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 1310–1322.

[11] H. Zheng, D. Li, B. Liang, X. Zeng, W. Zheng, Y. Deng, W. Lam, W. Yang, and T. Xie, "Automated test input generation for Android: Towards getting there in an industrial case," in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, 2017, pp. 253–262.

[12] (2019) Tencent. [Online]. Available: https://www.tencent.com/en-us

[13] (2019) Qihoo 360. [Online]. Available: https://www.360totalsecurity.com/en

[14] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket," in *Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS)*, vol. 14, 2014, pp. 23–26.

[15] V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, and E. Bodden, "Mining apps for abnormal usage of sensitive data," in *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering (ICSE)*, 2015, pp. 426–436.

[16] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "Appcontext: Differentiating malicious and benign mobile app behaviors using context," in *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering (ICSE)*, pp. 303–313.

[17] Z. Xu, K. Ren, and F. Song, "Android malware family classification and characterization using CFG and DFG," in *Proceedings of the 2019 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, 2019, pp. 49–56.

[18] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "Cypider: building community-based cyber-defense infrastructure for Android malware detection," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACM, 2016, pp. 348–362.

[19] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: Android malware detection through manifest and API calls tracing," in *2012 Seventh Asia Joint Conference on Information Security*. IEEE, 2012, pp. 62–69.

[20] Y. Aafer, W. Du, and H. Yin, "Droidapiminer: Mining API-level features for robust malware detection in android," in *International conference on security and privacy in communication systems*. Springer, 2013, pp. 86–103.

[21] S. Chen, M. Xue, Z. Tang, L. Xu, and H. Zhu, "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 (AsiaCCS)*, 2016, pp. 377–388.

[22] S. Chen, M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu, and B. Li, "Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach," *Computers & Security*, vol. 73, pp. 326–344, 2018.

[23] W. Zhou, Y. Zhou, M. Grace, X. Jiang, and S. Zou, "Fast, scalable detection of piggybacked mobile applications," in *Proceedings of the 3rd ACM conference on Data and application security and privacy*. ACM, 2013, pp. 185–196.

[24] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for Android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, 2011, pp. 15–26.

[25] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. L. Traon, D. Octeau, and P. D. McDaniel, "Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2014, pp. 259–269.

[26] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, p. 5, 2014.

[27] T. Chen, J. He, F. Song, G. Wang, Z. Wu, and J. Yan, "Android stack machine," in *Proceedings of the 30th International Conference on Computer Aided Verification (CAV)*, 2018, pp. 487–504.

[28] F. Song and T. Touili, "Model-checking for Android malware detection," in *Proceedings of the 12th Asian Symposium on Programming Languages and Systems (APLAS)*, 2014, pp. 216–235.

[29] R. Feng, S. Chen, X. Xie, G. Meng, S.-W. Lin, and Y. Liu, "A performance-sensitive malware detection system using deep learning on mobile devices," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1563–1578, 2020.

[30] R. Feng, J. Q. Lim, S. Chen, S.-W. Lin, and Y. Liu, "Seqmobile: A sequence based efficient Android malware detection system using rnn on mobile devices," *arXiv preprint arXiv:2011.05218*, 2020.

[31] R. Feng, S. Chen, X. Xie, L. Ma, G. Meng, Y. Liu, and S.-W. Lin, "Mobidroid: A performance-sensitive malware detection system on mobile platform," in *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 2019, pp. 61–70.

[32] B. Wu, S. Chen, C. Gao, L. Fan, Y. Liu, W. Wen, and M. R. Lyu, "Why an Android app is classified as malware: Toward malware classification interpretation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 2, pp. 1–29, 2021.

[33] A. A. A. Samra, K. Yim, and O. A. Ghanem, "Analysis of clustering technique in Android malware detection," in *Proceedings of the 7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE, 2013, pp. 729–733.

[34] M. Fan, X. Luo, J. Liu, M. Wang, C. Nong, Q. Zheng, and T. Liu, "Graph embedding based familial analysis of Android malware using unsupervised learning," in *Proceedings of the 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 771–782.

[35] S. M. Kolekar and P. N. Mahalle, "Malware prevention and detection system using smart phone," *International Journal of Computer Applications*, vol. 107, no. 21, pp. 31–35, 2014.

[36] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and A. Ribagorda, "Evolution, detection and analysis of malware for smart devices," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 961–987, 2014.

[37] L. Wei, Y. Liu, and S.-C. Cheung, "Taming Android fragmentation: Characterizing and detecting compatibility issues for Android apps," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 2016, pp. 226–237.

[38] L. Wei, Y. Liu, S.-C. Cheung, H. Huang, X. Lu, and X. Liu, "Understanding and detecting fragmentation-induced compatibility issues for Android apps," *IEEE Transactions on Software Engineering*, 2018.

[39] (2019) Android Apktool. [Online]. Available: https://ibotpeaches.github.io/Apktool

[40] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *Proceedings of the 40th International Conference on Software Engineering (ICSE)*, 2018, pp. 933–944.

[41] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

[42] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2019, pp. 4171–4186.

[43] H. M. Kim, H. M. Song, J. W. Seo, and H. K. Kim, "Andro-simnet: Android malware family classification using social network analysis," in *16th Annual Conference on Privacy, Security and Trust (PST)*, 2018, pp. 1–8.

[44] Y. Du, J. Wang, and Q. Li, "An android malware detection approach using community structures of weighted function call graphs," *IEEE Access*, vol. 5, pp. 17 478–17 486, 2017.

[45] A. Huang, "Similarity measures for text document clustering," in *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC)*, vol. 4, 2008, pp. 9–56.

[46] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proceedings of the National Academy of Sciences*, vol. 105, no. 4, pp. 1118–1123, 2008.

[47] C. Gao, H. Xu, J. Hu, and Y. Zhou, "Ar-tracker: Track the dynamics of mobile apps via user review mining," in *Proceedings of the 2015 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, 2015, pp. 284–290.

[48] (2019) Jadx-GUI. [Online]. Available: https://github.com/skylot/jadx

[49] (2019) Logcat. [Online]. Available: https://developer.android.com/studio/command-line/logcat

[50] J. Gao, L. Li, P. Kong, T. F. Bissyandé, and J. Klein, "Should you consider adware as malware in your study?" in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019, pp. 604–608.

[51] (2019) Privacy policy for Android apps. [Online]. Available: https://www.iubenda.com/en/help/11552-privacy-policy-for-android-apps

[52] (2019) Gdpr. [Online]. Available: https://gdpr-info.eu

[53] (2019) Android WebView. [Online]. Available: https://developer.android.com/reference/android/webkit/WebView

[54] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[55] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proceedings of the 8th symposium on usable privacy and security*, 2012.

[56] A. Stamminger, C. Kruegel, G. Vigna, and E. Kirda, "Automated spyware collection and analysis," in *Proceedings of the International Conference on Information Security*, 2009, pp. 202–217.

[57] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, 2011, pp. 3–14.