MAPS: A Dataset for Semantic Profiling and Analysis of Android Applications

AMIRMOHAMMAD PASDAR and YOUNG CHOON LEE, School of Computing, Macquarie University, Sydney, Australia
SEOK-HEE HONG and TONGLIANG LIU, School of Computer Science, University of Sydney, Australia

The vulnerability of smartphones to cyber attacks has been a serious concern for users emerging from the integrity of installed mobile applications (*apps*). These applications are fundamentally developed to provide legitimate and diversified on-the-go services; however, harmful and dangerous ones have found the perfect door to get into smartphones performing malicious behaviors. More effective and sophisticated application analysis is a practical strategy to reveal malicious activities and provide more insights into the application behavior. In this paper, we present the Malware Analysis and Profiling on Smartphones (MAPS) dataset for the research community that provides a very in-depth and detailed deep analysis of applications for extracting an enhanced set of features. The MAPS dataset relies on a diverse dataset collected from different repositories, containing more than 153000 applications (>2TB in size), and outputs more than 40000 features for analysis and training of deep neural network models.

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

Additional Key Words and Phrases: Malware profiling, Android malware analysis, Android malware dataset

ACM Reference Format:
Amirmohammad Pasdar, Young Choon Lee, Seok-Hee Hong, and Tongliang Liu. 2022. MAPS: A Dataset for Semantic Profiling and Analysis of Android Applications. In *17th ACM Workshop on Mobility in the Evolving Internet Architecture (MobiArch'22), October 21, 2022, Sydney, NSW, Australia.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3556548.3559629

1 INTRODUCTION

The growth of smartphone usage has exponentially increased in the past years due primarily to technological advancements. It has formed a promising ubiquitous platform for various private or enterprise services that developers or software vendors provide. However, the infiltration of smartphones for malicious behaviors by non-legitimate and malicious software has significantly risen, making smartphones vulnerable to threats and cyber attacks.

Approximately 80% of the worldwide market share belongs to the Android operating system (OS), the most frequently used OS on different mobile platforms and attracting developers for developing applications. In addition to the Google Play Store, third-party repositories facilitate searching and installing applications. Although regulations (i.e., Play Protect [6]) are in place for publishing applications on the official store, malware is still spread not only in the official market but also through non-official repositories as the potential gateway for malware distribution (e.g., [3]). Therefore, a thorough study of Android applications for profiling their design and logic characteristics are essential to reveal malicious application.

There are different types of malware datasets for Android OS, mainly collecting a set of applications and analyzing them through third-party tools (e.g., [2, 5, 7, 9–12, 14]). Although promising, these datasets barely extracted detailed information, leaving out essential features (e.g., programming practices seen at the code level) for profiling applications.

This paper presents Malware Analysis and Profiling on Smartphones (MAPS) datasets containing in-depth and detailed deep analysis of applications for profiling to extract an enhanced set of features. The malware analysis and profiling rely on a diverse dataset collected from different repositories, containing more than 153000 applications (>2TB in size), and outputs more than 40000 features available to the research community for training deep neural networks. The overall structure of MAPS is shown in Figure 1, and is equipped with a sophisticated Python parser for extracting the features and analyzing applications offline using real-world applications [1, 2, 11].

2 BACKGROUND AND RELATED WORK

2.1 Background

Android operating system is an open-source Linux-based OS that, since its emergence in 2008, many OS releases have become available. Applications for Android OS are written in

MobiArch'22, October 21, 2022, Sydney, NSW, Australia

ACM ISBN 978-1-4503-9518-2/22/10…$15.00
https://doi.org/10.1145/3556548.3559629

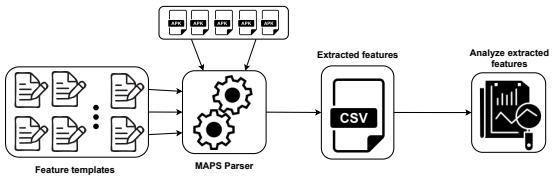


Fig. 1. MAPS employs an extensive parser to extract an extensive set of application features.

Java/Kotlin, are packaged in the form of ".apk", and are compiled into Dalvik bytecodes (.dex). Each application package consists of *AndroidManifest.xml* and *Dex* classes. The former provides essential information about the application and its components, including but not limited to activities, services, broadcast receivers, and content providers. The latter presents the application semantics meaning how the application logic is designed and converted to the code. Dex classes hold information about the list of identifiers (e.g., strings or prototypes), class definitions, and method handles. An extensive analysis of Dex classes leads to understanding the application's logical signature toward uncovering malware transformation techniques [13].

2.2 Related Work

There are different types of malware datasets for malware detection on Android, e.g., [2, 4, 5, 7–12, 14] to provide a representative but rich dataset in different aspects to the research community.

Arp et al. [2] presented DREBIN based on a collected dataset as a lightweight method for malware detection by employing static analysis of the applications. Chen et al. [4] studied potentially-harmful libraries across iOS and Android devices to find malicious behaviors over a large collected set of libraries since iOS libraries also have Android versions. Mahdavifar et al. [11] relied on a virtual machine introspection-based dynamic analysis system to extract a set of features, e.g., sequences of system calls, for collected datasets for malware category classification. Damshenas et al. [5] present M0Droid dataset for developing an Android application behavioral pattern recognition tool based on system call requests to identify and categorize malware. Kiss et al. [7] present the Kharon dataset for representing the diversity of malware types by manually dissecting each malware by reversing their code run in a monitored real smartphone. Lashkari et al. [9] employed real smartphones to provide the CI-CAndMal2017 dataset, and in [8] presented a labeled dataset of mobile malware traffic from different families of both adware and general malware. Zhou and Jian [14] aimed to systematize or characterize existing Android malware samples by collecting a dataset that covers the majority of existing Android malware families. Li et al. [10] presented the AMD dataset for the designed clustering system, investigating malicious payload in malware samples.

MAPS is the largest dataset that includes recent applications and is publicly available compared to the existing datasets, and provides the most extensive set of features, including some existing datasets in the literature.

3 DATASET PREPARATION

This section profiles Android applications to understand the semantic and security patterns practiced in the application code. MAPS employs a customized Python parser to analyze application components and Dex classes. This parser provides insights into the semantic and security patterns and produces detailed features extracted from the application package.

The parser employs real-world applications from a diverse range of applications in terms of size, application category, and release date (i.e., 2014 - 2021) in the form *malware* and *benign*. This dataset contains collected applications from [1, 2, 11] in which the number of samples in both classes is balanced to avoid biased analysis toward a specific class. In addition, the parser leverages the official Android Asset Packaging Tool (AAPT) and considers Android's official developer guide and Android SDK to decode and analyze the manifest file and the Dex class(es) for extracting the logical features.

3.1 MAPS APK Parser

MAPS employs a Python-based application parser to extract features from an initially collected but balanced dataset. The parser also utilizes 21 feature templates to collect the application package's essential features. These templates are crafted with respect to the official Android guide containing the *fundamental* information and attributes about the package's main parts: the Android manifest file and Dex class(es).

Due to the diversity of Android OS installed across devices, we initially analyze all the official builds of Android OS and consolidate the declared information in each of which OS for the following five main categories; permissions, intents, categories, services, and broadcasts. By consolidating the collected information, we create the ever-rich feature-related sets in the literature[1] that cover all Android APIs. The feature templates are heavily supported by the declared attributes in the official developer guide for creating the most *application-specific* feature set for benign and malware applications.

In the method-level application Dex class(es) parsing, we refer to the Android Developer Guide and look for syntax (or mnemonic) that implies calling "methods" and may result in storing the values. The used syntax for the developed parser directly or indirectly invokes methods (i.e., APIs), and

[1]The templates and datasets are published in the MAPS GitHub page https://github.com/amrmp/MAPS.

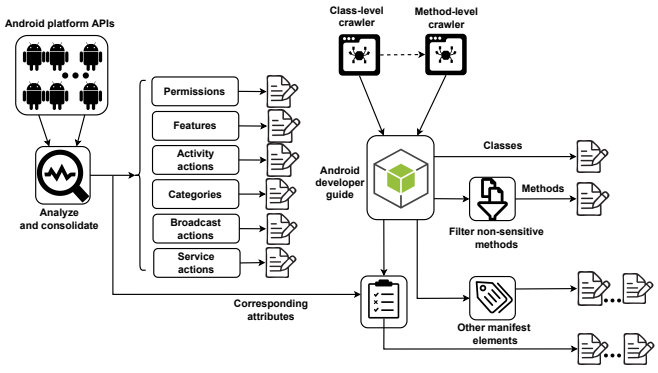


Fig. 2. MAPS feature template builder for the APK parser.

are *invoke-virtual*, *invoke-super*, *invoke-static*, *invoke-direct*, *invoke-interface*, *invoke-custom*, and *invoke-polymorphic*.

Information regarding invoking methods used in the application Dex class(es) is extensively collected and mapped against the sensitive template created by the method-oriented crawlers. For the class-level signature, the APK parser looks for the syntax "new-instance", which indicates constructing a new instance of the indicated type (i.e., *class*). Thus, not only are all the class-level calls captured, but it also assists the collection of highly sensitive methods within each class.

All the Android classes that also cover extra package classes, such as AndroidX, Google play APIs, and Google Ads, to name a few, are collected. This class-level index will be given to MAPS APK Parser to collect information regarding the classes used in each application. A method-specific crawler for the method-level template is also implemented that collects relevant information from the Android Class Index. Over 63000 methods were initially collected by the crawler based on the class-level template. Several rounds of semi-manual filtration are applied to focus on the essential methods, meaning in the initial rounds, all class methods related to the graphical interface (i.e., view) are removed. In the following rounds, the general classes (both in Java and Android) are removed to omit non-sensitive ones from the list. This filtration process eventually ended up having nearly 20000 sensitive features. Figure 2 presents the overall structure of the crawler and feature template builder.

MAPS APK Parser applies to the collected dataset for extraction of features with respect to prepared templates. The parser's output is a comma-separated values (CSV) file with the size of ~12GB that correlates an application SHA256 hash to the corresponding features.

4 MAPS DATASET ANALYSIS

4.1 Application Permissions

Declaring permissions limits access to sensitive user data (e.g., SMS) or certain system features (e.g., camera). Each permission has a declared protection level that falls into four different categories. (1) Normal protection level as lower-risk permissions to access isolated application-level features without explicit user approval. (2) Dangerous protection level as higher-risk permissions to access the private user data or control the device, being checked during runtime. (3) Signature protection level relies on the same certificate signs of the requesting application and the application that declared the permission. Finally, (4) signatureOrSystem protection level is only granted when they exist in a dedicated folder in the Android system image or are signed with the same certificate as the requesting application.

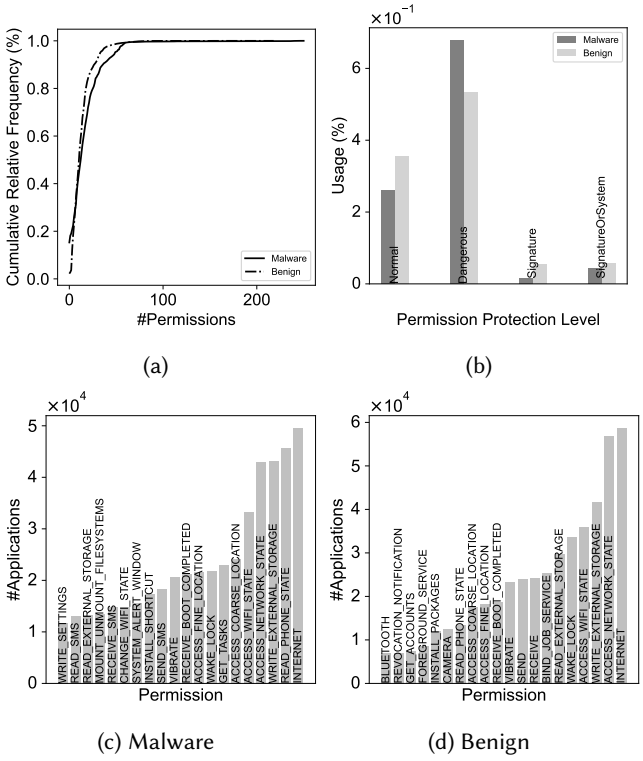


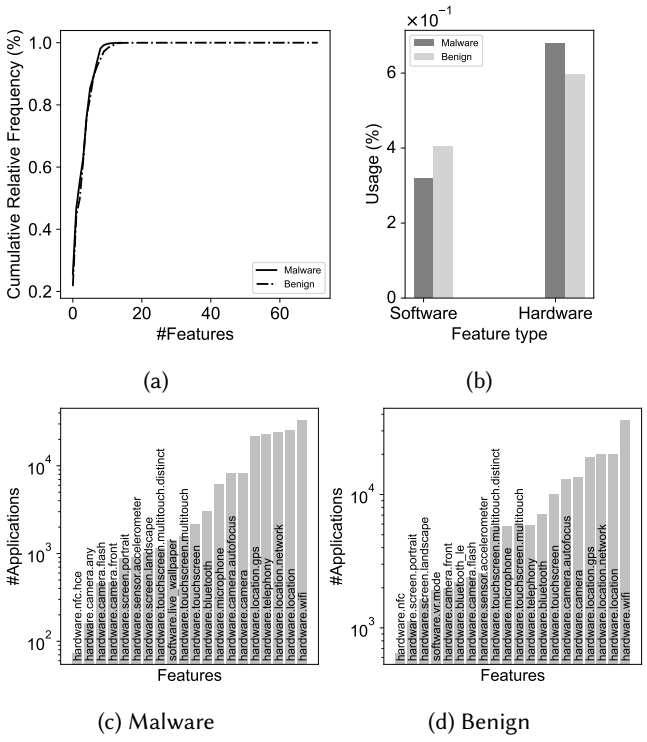
Fig. 3. Permission usage and permission protection level ratios; (a) CRF, (b) ratio of permissions used based on the protection level, the top 20 frequent requested permissions for (c) malware, and (d) benign applications.

Figure 3 presents the permissions' cumulative relative frequency (CRF) and the percentage of permission types in the dataset. Figure 3a highlights malware applications that request more permissions than benign applications, e.g., more than 90% of benign applications need at least ~20 permissions.

In comparison, it is around 70% for malware apps. More interestingly, up to 10% of malware or benign applications declared *zero* permissions which could not be matched against the official android guide; this can be due to different target SDK, deprecated permissions, and bad application development. In addition, Figure 3b illustrates benign applications request more normal-type permissions accounting for ~65% while malware applications claims more dangerous (or signature-OrSystem) permissions. Figures 3c and 3d also show the top frequent permissions used. They illustrate that the most used permissions differ in the number of requests and focus more on network connectivity and the phone state (e.g., malware applications tend to use more SMS-related permissions).

4.2 Application Features

The feature declaration in the manifest file notifies any external entity that an application relies on hardware or software features, stating that an application's functionality depends on the manifest file.



(a) (b)

(c) Malware (d) Benign

Fig. 4. The frequency of declared features. (a) CRF, (b) ratio of feature type, the top 20 frequent requested features for (c) malware, and (d) benign applications.

Figure 4 illustrates that more than 90% of malware applications did not declare features in the manifest file, and the type of feature declaration was mostly hardware ones (i.e., more than 80%). Compared to benign applications, approximately 90% applications declare at least five features, and the feature type does not weigh on a particular type; hardware or software. Moreover, Figures 4c and 4d present the most frequent features used by malware and benign applications. Benign applications tend to employ more diverse features than malware applications, while the top features for malware applications are relatively higher for connectivity, location, camera, and microphone.

4.3 Intents and Categories

Intent filters are designed to assist activities, services, or broadcast receivers with facilitating communication between application components and are assigned to categories based on the intent types. Categories are used for implicit intents, declaring a general action to perform and allowing another app component to handle it.

Figure 5 illustrates that benign applications declare more intents and categories in the manifest file. Conversely, malware applications barely specify intents and categories, accounting for ~80% declaring less than two intents and categories. Also, comparing Figure 5c with 5d reveals malware tendency in Telephony and communications (e.g., making phone calls), while the declared categories are slightly similar and are tied to the application behaviors in the code level.

4.4 Broadcast Receivers and Content Providers

Broadcast messages are intended to communicate between applications and the Android system (e.g., charging a smartphone), similar to the "publish-subscribe" pattern. Applications can register to receive specific-intended broadcast messages, and the system automatically routes them to the subscribed applications; however, excess use of broadcast messages causes slowing the system's performance. Broadcast messages are wrapped in "intent" objects whose action strings identify the occurred events. In contrast, content providers are part of Android applications that may provide their user interface with data and control access to a central data repository. Content providers are mainly designed to be used by the other applications through a provider-client object that offers a standard and consistent interface to data, handling inter-process communication and securing data access.

Figure 6a emphasizes that malware applications barely declare broadcast messages and rely less on the service providers shown in Figure 6b. In contrast, benign applications take advantage of broadcasts, accounting for more than 90% of applications declared 25 receivers and twice the number of services compared to the malware ones. Also, most of the broadcast receivers in Figures 6c and 6d used by malware applications are about 'changes' to OS or application states, and nearly 1.5 to 2 times more than benign applications. Interestingly, both malware and benign applications tend to

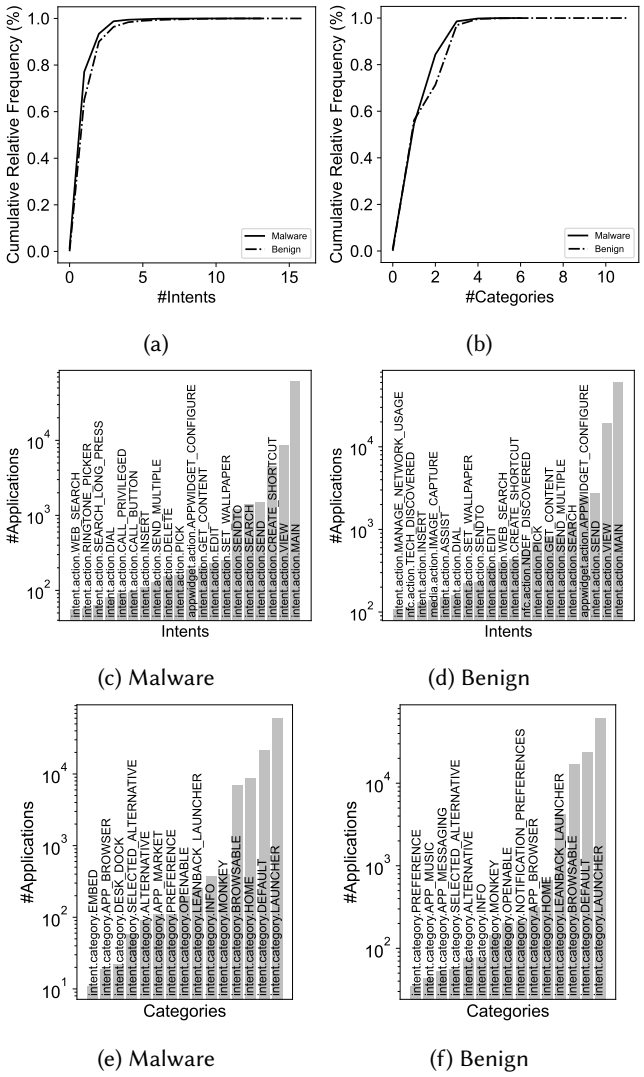


Fig. 5. The frequency of declared (a) intents and (b) categories, (c,d) the top frequently used intents, and (e,f) categories for malware and benign applications.

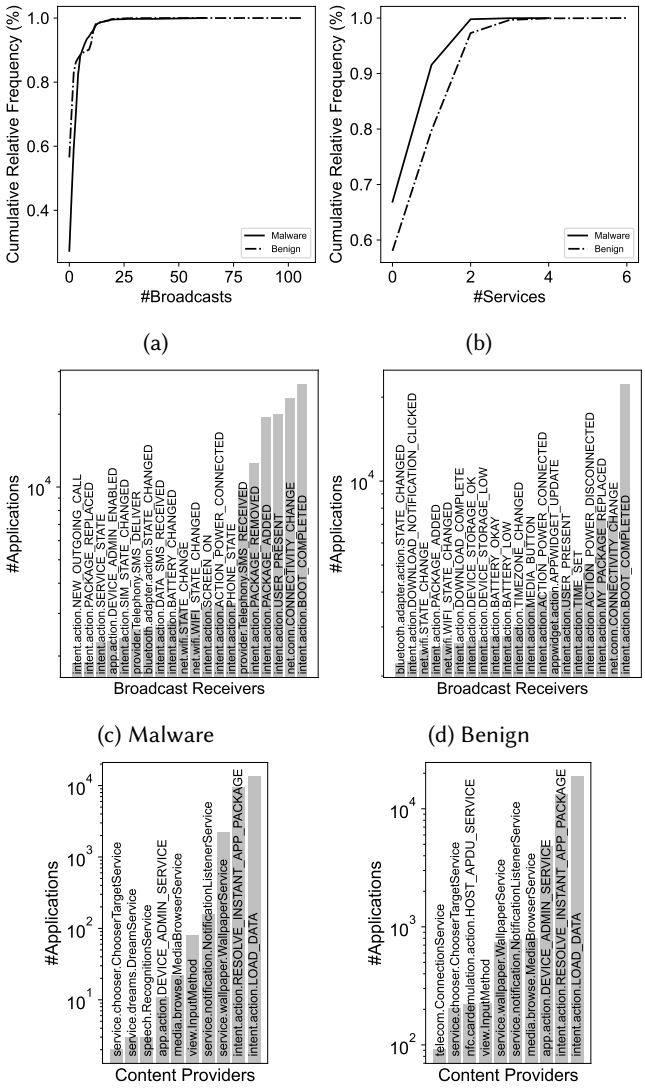


Fig. 6. The frequency of declared (a) broadcast receivers and (b) content providers, (c,d) the top frequently used broadcast receivers, and (e,f) content providers for malware and benign applications.

use a set of diverse content providers (Figures 6e and Figures 6f), having similar frequencies for the top two ones.

4.5 Android Platform APIs and Method

Android APIs are the building blocks of application logic that present how the application will *mostly* behave during runtime, providing a better and more detailed view of the logic and the significant differences in implementing malware and benign applications. Therefore, the frequency of official Android platform APIs used for developing applications in the malware and benign datasets are investigated. Figure 7 presents the cumulative relative frequency of the number of APIs and sensitive methods used in the collected dataset for malware and benign applications. This figure illustrates that malware applications use fewer APIs, focusing on more sensitive and essential ones for malicious activities. In contrast, benign applications employ more APIs, providing more legit logic behind their applications. For example, while nearly 10% of benign applications use around 500 or fewer APIs, the ratio is almost more than 90% for the malware apps. Moreover, Figure 7c and Figure 7d represent the top sensitive API classes and (corresponding) methods used in the benign and malware applications. Malware applications

obviously rely on the network state for their malicious behaviors shown in Figure 7c. Conversely, benign applications use more frequent API classes for application development. In addition, the frequency of (corresponding) sensitive methods (Figure 7d) are higher for benign applications as, based on our investigations, more Dex classes could be found for benign applications, emphasizing the better development and application logic behind apps.

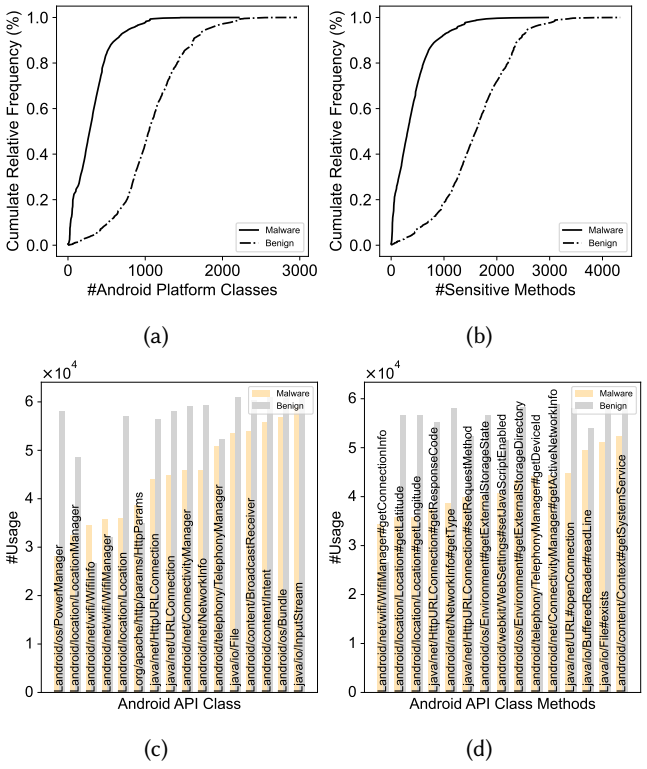


Fig. 7. The relative frequency of declared classes and methods. (a) API classes, (b) sensitive methods, (c,d) the top frequently used API classes, and (corresponding) methods.

5 CONCLUSION

This paper presented the MAPS, Malware Analysis and Profiling on Smartphones dataset, that profiled applications through deep analysis of application components for extracting an enhanced set of features using a crafted Python parser. The malware analysis relied on a diverse range of applications in terms of size, category, and release date (i.e., 2014 - 2021), collected from different repositories containing more than 153000 applications and outputted more than 40000 features. An in-depth analysis of extracted features revealed how malware applications took advantage of the principal application components to perform malicious activities.

REFERENCES

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

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

[3] Alex Chapman. 2021. Google Play Store removes seven apps downloaded thousands of times over Joker malware fears. https://7news.com.au/.

[4] Kai Chen, Xueqiang Wang, Yi Chen, Peng Wang, Yeonjoon Lee, XiaoFeng Wang, Bin Ma, Aohui Wang, Yingjun Zhang, and Wei Zou. 2016. Following devil's footprints: Cross-platform analysis of potentially harmful libraries on android and ios. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 357–376.

[5] Mohsen Damshenas, Ali Dehghantanha, Kim-Kwang Raymond Choo, and Ramlan Mahmud. 2015. M0droid: An android behavioral-based malware detection model. *Journal of Information Privacy and Security* 11, 3 (2015), 141–157.

[6] Google. 2021. Google Play Protect. https://developers.google.com/android/play-protect.

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

[8] Arash Habibi Lashkari, Andi Fitriah A Kadir, Hugo Gonzalez, Kenneth Fon Mbah, and Ali A Ghorbani. 2017. Towards a network-based framework for android malware detection and characterization. In *2017 15th Annual conference on privacy, security and trust (PST)*. IEEE, 233–23309.

[9] Arash Habibi Lashkari, Andi Fitriah A. Kadir, Laya Taheri, and Ali A. Ghorbani. 2018. Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification. In *2018 International Carnahan Conference on Security Technology (ICCST)*. 1–7.

[10] Yuping Li, Jiyong Jang, Xin Hu, and Xinming Ou. 2017. Android malware clustering through malicious payload mining. In *International symposium on research in attacks, intrusions, and defenses*. Springer, 192–214.

[11] Samaneh Mahdavifar, Andi Fitriah Abdul Kadir, Rasool Fatemi, Dima Alhadidi, and Ali A Ghorbani. 2020. Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. IEEE, 515–522.

[12] Davide Maiorca, Davide Ariu, Igino Corona, Marco Aresu, and Giorgio Giacinto. 2015. Stealth attacks: An extended insight into the obfuscation effects on android malware. *Computers & Security* 51 (2015), 16–31.

[13] Vaibhav Rastogi, Yan Chen, and Xuxian Jiang. 2013. Droidchameleon: evaluating android anti-malware against transformation attacks. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. 329–334.

[14] Yajin Zhou and Xuxian Jiang. 2012. Dissecting Android Malware: Characterization and Evolution. In *2012 IEEE Symposium on Security and Privacy*. 95–109.