How do Apps Evolve in Their Permission Requests? A Preliminary Study

Paolo Calciati♠,♢ Alessandra Gorla♠
{paolo.calciati,alessandra.gorla}@imdea.org
♠IMDEA Software Institute, Madrid, Spain ♢Universidad Politécnica de Madrid, Spain

Abstract—**We present a preliminary study to understand how apps evolve in their permission requests across different releases. We analyze over 14K releases of 227 Android apps, and we see how permission requests change and how they are used. We find that apps tend to request more permissions in their evolution, and many of the newly requested permissions are initially overprivileged. Our qualitative analysis, however, shows that the results that popular tools report on overprivileged apps may be biased by incomplete information or by other factors. Finally, we observe that when apps no longer request a permission, it does not necessarily mean that the new release offers less in terms of functionalities.**

I. Introduction

With auto-updates enabled by default on Android devices, it is easy for developers to produce and distribute new releases of their apps. Users can immediately use the new releases without even noticing the update process, unless there are changes in dangerous permission requests that require their approval. The most recent releases of the Android framework, though, no longer notify users about changes in the dangerous permissions list, unless the app requires a permission that belongs to a new permission group. For instance, an app declaring the RECEIVE_SMS permission, and thus capable of receiving SMS, could suddenly start sending SMS without further user approval by adding the SEND_SMS permission, since these two permissions belong to the same group. Changes in the permissions list are however worth notice, since they most likely represent a major change in the behavior of the app.

In this paper we perform a preliminary study on the evolution of permission requests of Android apps. We analyze over 14K different releases of 227 apps, with a minimum of 50 releases per app, and we study how developers request legitimate permissions (i.e., requested and used in the code) and overprivileged permissions (i.e., declared in the manifest file but not used by the app).

This is not the first paper aiming to study the evolution of permission requests in Android apps. Tailor and Martinovic [8] studied over 1,6M apps, taking quarterly snapshots of the Google Play store for almost two years. Their analysis highlights trends in the store in terms of permission requests, but does not look at single applications specifically. Wei et al. [9] conducted a study similar to the one we present in this paper, but used a much smaller dataset with an average of 4 releases per app. Moreover, they focused their interest in comparing pre-installed apps and third-party apps, while we are mainly interested in the latter. To the best of our knowledge, ours is the first study to analyze such a big number of releases for single applications.

Our empirical study highlights several interesting findings: We see a common trend of apps requiring more permissions over time, confirming what [9], [8] observed; Some dangerous permissions that are added in following releases do not need further approval from the user; There is a mild correlation between changes in target SDK and permission requests; A qualitative analysis of the results highlights that permission removal does not imply the loss of a functionality; Last but not least, we identify several problems with Androguard, the state of the art tool to report overprivileged apps, showing that using it blindly can lead to biased results.

The remainder of the paper is structured as follows: Section II introduces the necessary notions of the Android platform; Section III describes the dataset we used and the process we followed during our study; Section IV presents the main findings of our quantitative and qualitative studies. Finally, Section V briefly compares this to the related work.

II. The Android Platform

Android applications are distributed as APK archive files. Among other things, the AndroidManifest.xml, which every APK must contain, specifies the minimum and target Android API that the app supports, and declares the permissions that the application needs in order to access protected features of the API. Android permissions fall into four categories, depending on the access they grant: *normal*, *dangerous*, *signature* and *signatureOrSystem*. The CAMERA permission, for example, belongs to the dangerous category. If declared in the manifest, the user has to grant it to give the application access to the device's camera. In our analysis we consider only dangerous permissions, since they represent high-risk accesses to sensitive features.

Android 6 introduced two major changes in the permission model: 1) Apps targeting SDK 23 or higher request permissions to the user at run-time. Previously, requests occurred upon installation; 2) if an app requests a dangerous permission, but there is already another permission granted from the same group, the system immediately grants it without any further interaction with the user. The manifest file may contain *overprivileged* permissions. A permission is considered as such if it appears in the manifest, but the app does not actually need it, and removing it would not hinder the app's

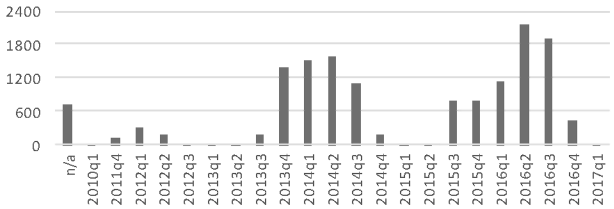

Fig. 1: APK date distribution

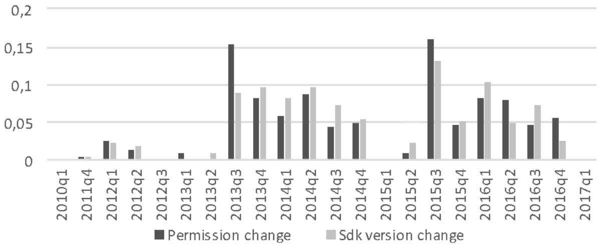

Fig. 2: Permission change vs. Android SDK version change.

functionalities. Overprivileged permissions are considered bad practice, and raise a security concern, since they leave space to possible malicious exploits.

III. Dataset and Evaluation Process

To run our preliminary empirical evaluation, we resorted to the Androzoo dataset [2], which comprises a collection of over 5 million Android apps retrieved from different sources. Since this study does not focus on malware, we selected the Google Play store as the only source to consider, given that several studies have shown that the apps distributed through this channel are largely trustworthy [1], [7], [10]. From this selection, we considered only apps with at least 50 releases, to have enough data for a study on apps evolution. The resulting set contained 14,880 releases of 235 different applications.

We analyzed each release with Androguard, a tool that can extract declared permissions and report the overprivileged ones[1]. In this study we focused only on Android official permissions, leaving out app-specific permissions. Androguard crashed while analyzing the Dalvik bytecode of several releases of 10 different applications. We decided to keep in our analysis only the apps for which Androguard managed to analyze at least 50 releases: with this additional pruning, our final dataset comprises 227 applications with a total of 14,450 releases. The distribution of releases per app goes from a minimum of 50 up to 171, with a mean value of 64. The APKs in the dataset have been released within the time-frame of August 2008 – January 2017, with the distribution shown in Figure 1. Out of the APKs in our dataset, 727 (5.03%) do not have a valid release date, as reported by the Androzoo developers[2], and are represented in the plot as n/a.

To further understand the nature of the apps in our dataset, we looked at the category, number of downloads, and rating as reported in the Google Play store. We conclude that the dataset is quite varied, since it covers all the 32 Android categories, but it focuses mainly on high quality and popular applications, with an average of over 44M downloads per app, and an average star rating of 4.29 out of 5. All applications we considered are still active and available on the Google Play store, except for 8, which either changed package name or have have been discontinued.

IV. Empirical Results on Permissions Evolution

We first focus on a quantitative study to evaluate how permission requests evolve. We later conduct a qualitative study to have a better understanding of the phenomena observed in the first study.

A. Quantitative Study

The quantitative study aims to look at how permission requests change across the lifetime of an APK. More precisely, we look at the following research questions:

- *RQ1: How do permission requests evolve?* To answer this question we analyze how the permissions list changes between a release and its following one. We also look for specific patterns in the evolution, such as whether a permission is first requested and actually used only later.
- *RQ2: Which are the most frequently added permissions?* We check which permissions developers tend to add over time, and whether they are legitimate or overprivileged. Given the new group permission model in Android 6 (explained in Section II) we also check which are the most commonly added permissions that would not need any further approval of the user.
- *RQ3: Do permission changes concentrate within specific time periods, and in particular correlate with changes in the Android SDK?* In this final study we try to understand if there are relevant facts related to permission changes.

RQ1: How do permission requests evolve?: We first look at how the permissions list changes between a release and its following one. We observe that for 13,637 out of 14,223 release transitions, which accounts to 95.88% of the total number, there are no changes in the list of requested permissions.

Figure 3 shows the distribution of added and removed permissions on each version change after taking the versions without any change out of the equation. On average we observe 0.64 permission additions per release, showing a clear trend of apps increasing the number of asked permissions over time. As depicted in the plot, most changes involve only adding a single permission (287 occurrences). We observe only 78 occurrences of adding two permissions at the same time, while only 42 involving three or more permissions added. Permissions are rarely removed (151 occurrences in total), and most of the times this involves a single permission.

Similarly to what Wei et al. did in their work [9], we look for specific patterns in our dataset: Table I compares results obtained by Wei et al. with ours. We use the same notation,

[1] https://github.com/androguard/androguard

[2] https://androzoo.uni.lu/lists

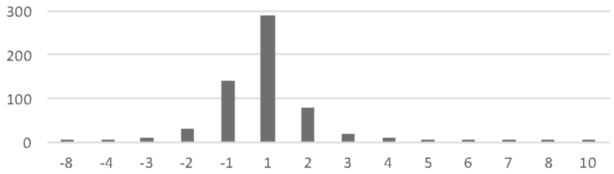

Fig. 3: Permission change

TABLE I: Patterns compared to [9].

Pattern	Our results (%)	Results in [9] (%)
0 → 1	80.18	90.46
1 → 0	15.27	8.59
1 → 0 → 1	3.27	0.84
1 → 0 → 1 → 0	1.27	0.11

where ‘0’ represents a non requested permission, ‘1’ represents a requested permission, and ‘→’ represents a state transition. Considering the fact that we have a much larger set of releases for each application (more than 50 versus an average of less than 4), and thus it is more likely for us to find longer patterns, the distribution found in [9] finds confirmation in our results despite the slight difference in the percentages. The patterns we consider in this comparison account for 83% of patterns found in our study, with the remaining 17% containing the 0→1→0 pattern, and longer patterns in negligible percentages.

Our analysis extends to cover also overprivileged permissions. For the rest of the section, we refer with ‘1’ to a legitimate permission, and with ‘2’ to an overprivileged one: Table II shows the most frequent patterns we discovered. Surprisingly, the most frequent pattern is to add a permission that remains overprivileged for the whole lifetime of the app. We further analyze these trends in Section IV-B.

RQ2: Which are the most frequently added permissions?: Given that RQ1 showed that the most common trend is to add permissions, whether legitimate or overprivileged, we look at which ones are most frequently added. Table III shows the most frequent among the 646 permission addition found.

Taking into account the recent changes in Android 6 explained in Section II, we further analyze each newly added permission, and verify if it belongs to a permission group that was already granted in the previous release: in 35.75% of the cases (231 permission addition out of 646 in total) the newly added permission indeed falls in the same group of one of the already granted permission. READ_EXTERNAL_STORAGE is the permission for which this behavior happened the most: many apps add this permission while they already declare WRITE_EXTERNAL_STORAGE. What is apparently unclear to developers is that apps granted with WRITE_EXTERNAL_STORAGE implicitly have read access as well, so there is no need to additionally ask for the READ_EXTERNAL_STORAGE permission. A different story regards accessing the user's location through either the ACCESS_COARSE_LOCATION or ACCESS_FINE_LOCATION permissions, which belong to the same permission group but have information from different sources (from the network provider and from the GPS respectively). We observe that many apps initially use just coarse location, and request the ACCESS_FINE_LOCATION afterwards. Thus, without any further consent of the user, they can obtain more precise location information.

TABLE II: Observed Patterns in permission evolution.

Pattern	Count	Frequency (%)
0 → 2	219	24.39
0 → 1	141	15.70
2 → 1	90	10.02
0 → 1 → 0	50	5.57
1 → 0	48	5.35

TABLE III: Most Frequently added permission

Permission (legitimate)	Usage
ACCESS_COARSE_LOCATION	58 (19.33%)
READ_PHONE_STATE	49 (16.33%)
ACCESS_FINE_LOCATION	49 (16.33%)
CAMERA	32 (10.67%)
READ_CONTACTS	31 (10.33%)
Permission (overprivileged)	**Usage**
READ_EXTERNAL_STORAGE	112 (32.37%)
WRITE_EXTERNAL_STORAGE	25 (7.23%)
READ_SMS	25 (7.23%)
CAMERA	25 (7.23%)
RECEIVE_SMS	24 (6.94%)

RQ3: Do permission changes concentrate in specific time periods?: The last step of our quantitative analysis looks at the distribution of releases over time. More precisely we want to see if releases are more frequent in specific time frames. We thus divide the time frame in year quarters (x axis in Figure 2), and we report the percentage of permission changes that we observe in that quarter (blue bars in the plot). We can observe that there are peaks in the number of releases in q3 2013 and q3 2015, in our opinion due to the low amount of releases in the preceding quarters. To further understand this trend, we list changes occurring in two subsequent releases that involve either the target or the minimum Android SDK. We plot, as yellow bars, the percentage distribution of such changes on the same time frame in Figure 2. We see that the two distributions have similar trends, suggesting that a change in the minimum or target SDK might trigger permission changes. We checked whether high change picks would correspond to release dates of new Android SDKs, but we could not find a positive correlation.

B. Qualitative Analysis

The results of the quantitative analysis reported in Section IV-A show that several apps have interesting patterns in their permission requests. Some apps, for instance, add and remove the same permission multiple times in their lifetime, as if some functionalities were added and removed from the app. In many other cases, instead, developers seem to ask for permissions they never use. To further analyze the permission evolution of the apps in our dataset, we plot legitimate and overprivileged permission requests as in Figure 4. On the y-axis we report the permissions that the app requires at

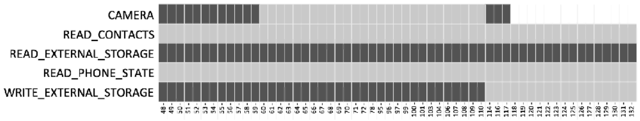

Fig. 4: Permission evolution plot of the TabShop app.

least once in its lifetime, and on the x-axis we report all the releases for that app. We use white, light blue and dark blue to represent not requested, legitimate and overprivileged permissions, respectively.

Figure 4 is the permission evolution of TabShop, a free shop keeping, cashier and point of sale (POS) app. Among the functionalities it offers, it can scan barcodes. CAMERA thus seems an essential permission for the whole lifetime of this app. The plot, instead, reports that the CAMERA permission is initially requested but not used up to release 60, it is later legitimately used for several releases, and finally it disappears after a few more releases that see the CAMERA permission as overprivileged. We installed and ran different APKs of this app in the emulator, and found out that in the first releases it uses an intent to launch "Barcode Scanner", an alternative app to scan barcodes. Between release 60 and 110 the app instead implements the feature itself, thus using the device's camera directly. Within this timeframe (i.e., between release 60 and 110) the CAMERA permission appears twice in the manifest. From version 114 to version 117 the app switches back to using the intent to "Barcode Scanner", but removes only one CAMERA permission from manifest, thus leaving the duplicate and making it overprivileged. Developers finally removed the unnecessary permission in version 118. The qualitative analysis on this app shows us that if an app removes a permission it does not necessarily mean that it no longer offers the same functionality, but rather may implement it in an alternative way. This app is not the only one in our dataset that declares *duplicate permissions*. More precisely, 52 apps of the 227 we analyzed show this problem in at least one of their releases, with Firefox Beta topping the list with duplicated permissions in over 100 releases. Similarly, we observe that many apps include misspelled permissions (e.g. VIBRATION instead of VIBRATE). Although these cases are harmless, since the app asks for a permission that either does not exist or it already uses, they represent a bad practice, and may falsely report alarms. Furthermore, we notice that Androguard incorrectly reports other permissions as overprivileged. This happens for example for WRITE_EXTERNAL_STORAGE, which is also one of the most frequently reported permissions as overprivileged. We install the app com.popularapp.periodcalendar, v9, released in 2013-02 on an emulator, removing WRITE_ and READ_ EXTERNAL_STORAGE permissions, which were both labeled as overprivileged. Running the repackaged app leads to a crash when it tries to read or write some information on the SD card, showing that the WRITE_EXTERNAL_STORAGE is legitimate. The incorrect information reported by Androguard may be due to different causes: 1) it is known that a complete and correct mapping between API calls and permissions does not exist yet [3], [5], [4]; 2) I/O operations may be implemented in native code, preventing Androguard from seeing them; 3) finally, we also identified a bug in Androguard, which causes the analysis to always use the API mappings relative to API 19, even when the app targets another release. Thus, the lesson we learned is that trusting tools such as Androguard blindly may bias significantly the results.

V. Related Work

Krutz et al. [6] used different static analysis tools to analyze 4416 releases belonging to 1179 apps, creating a dataset containing information about applications' development and maintenance. Taylor and Martinovic [8] performed a longitudinal analysis over 1,6M apps, taking quarterly snapshots of the Google Play store for almost two years and analyzing the evolution of permission, with a focus on the *dangerous* category. Their analysis highlights different trends in app permission usage, such as free and popular apps being more likely to add new permissions. Wei et al. [9] conducted a study on the evolution of Android permission, focusing on the differences between pre-installed and third party apps. They analyzed patterns and permission distributions, and reported that applications tend to be overprivileged and to request more permission over time. With respect to the related work, our analysis focused on studying a small set of apps but with a high number of releases, rather than simply taking in consideration the biggest possible APK set. Moreover, we present findings and qualitative analyses that other works do not cover.

VI. Conclusions

In this paper we presented an overview on the evolution of Android permission over a dataset of application with a large number of versions available. In our quantitative analysis we see a common trend of apps requiring more permissions over time, confirming similar studies in the literature. We found new evolution patterns, and a mild correlation between changes in target SDK and in permission requests. The qualitative analysis highlighted some confusion between developers regarding the use of permissions, as suggested by our finding of mislabeled, duplicated and overprivileged permissions. Finally, we discovered that the removal of a permission does not imply the loss of a functionality, as shown with the CAMERA permission in Section IV-A. Last but not least, we identified several problems with Androguard, the state of the art tool to report overprivileged apps, showing that using it blindly can lead to biased results.

Acknowledgment

This work was supported by the EU FP7-PEOPLE-COFUND project AMAROUT II (n. 291803), by the Spanish project DEDETIS, and by the Madrid Regional project N-Greens Software (n. S2013/ICE-2731).

References

[1] K. Allix, T. F. Bissyandé, Q. Jérome, J. Klein, R. State, and Y. L. Traon. Empirical assessment of machine learning-based malware detectors for android - measuring the gap between in-the-lab and in-the-wild validation scenarios. *Empirical Software Engineering*, 21(1):183–211, 2016.

[2] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon. AndroZoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 468–471, New York, NY, USA, 2016. ACM.

[3] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie. PScout: analyzing the Android permission specification. In *Proceedings of the 19th Conference on Computer and Communications Security (CCS)*, pages 217–228, New York, NY, USA, 2012. ACM.

[4] A. Bartel, J. Klein, M. Monperrus, and Y. Le Traon. Automatically securing permission-based software by reducing the attack surface: An application to Android. pages 274–277, 2012.

[5] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *Proceedings of the 18th Conference on Computer and Communications Security (CCS)*, pages 627–638, New York, NY, USA, 2011. ACM.

[6] D. E. Krutz, M. Mirakhorli, S. A. Malachowsky, A. Ruiz, J. Peterson, A. Filipski, and J. Smith. A dataset of open-source android applications. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, pages 522–525, Piscataway, NJ, USA, 2015. IEEE Press.

[7] Y. Y. Ng, H. Zhou, Z. Ji, H. Luo, and Y. Dong. Which android app store can be trusted in china? In *Proceedings of the 2014 IEEE 38th Annual Computer Software and Applications Conference*, COMPSAC '14, pages 509–518, Washington, DC, USA, 2014. IEEE Computer Society.

[8] V. F. Taylor and I. Martinovic. A longitudinal study of app permission usage across the google play store. *CoRR*, abs/1606.01708, 2016.

[9] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. Permission evolution in the android ecosystem. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 31–40, New York, NY, USA, 2012. ACM.

[10] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In *NDSS*. The Internet Society, 2012.