There are Many Apps for That: Quantifying the Availability of Privacy-Preserving Apps

Vincent F. Taylor
Department of Computer Science
University of Oxford
Oxford, United Kingdom
vincent.taylor@cs.ox.ac.uk

Alastair R. Beresford
Computer Laboratory
University of Cambridge
Cambridge, United Kingdom
alastair.beresford@cl.cam.ac.uk

Ivan Martinovic
Department of Computer Science
University of Oxford
Oxford, United Kingdom
ivan.martinovic@cs.ox.ac.uk

ABSTRACT

The adage "there's an app for that" holds true in modern app stores. Indeed, app stores usually go further and provide multiple apps with very similar functionality; examples range from flashlight apps to alarm clocks. We call these *functionally-similar* apps. When searching for these apps, users are often presented with a vast array of choices, but no distinction is made in the user interface to highlight the relative privacy risks inherent in choosing one app over another. Yet the availability of many functionally-similar apps raises the question of whether some apps are significantly less invasive than others. In this paper, we take several steps toward answering this question. We begin by enumerating 2 500 groups of functionally-similar apps in the Google Play Store. Within groups of apps, we use static analysis to understand the real-world risks coming from apps with aggressive permission usage. By leveraging an established ranking system, and combining it with real-world data from over 28 000 Android devices, we quantify the improvements that can be made if users installed apps with privacy in mind. We observe that at least 25.6% of apps contain libraries that gratuitously exploit available permissions and find that 43.5% of apps could be swapped for comparable alternatives that require fewer permissions. Permissions saved may deliver important privacy and security improvements, including preventing access to the calendar (in 24% of cases), sending text messages (12%) and recording audio (8%). This is particularly important for apps which embed third-party libraries, since library code executes with the same permissions as the app itself.

ACM Reference format:
Vincent F. Taylor, Alastair R. Beresford, and Ivan Martinovic. 2017. There are Many Apps for That: Quantifying the Availability of Privacy-Preserving Apps. In *Proceedings of WiSec '17 , Boston, MA, USA, July 18-20, 2017,* 6 pages.
DOI: 10.1145/3098243.3098266

1 INTRODUCTION

App stores are littered with groups of functionally-similar apps competing for market share. The functionally-similar apps we focus on are general-purpose in nature, such as *flashlight* or *alarm clock* apps, meaning they can feasibly be replaced with alternatives if the need arises. When a user searches for general-purpose apps, they are presented with a (usually large) list of apps matching their search criteria. However, the app store fails to indicate to users the relative risk to their privacy that may come from choosing one app over another. As a result, users are left to grapple with app metadata, with some blindly trusting the search ranking algorithms and favouring more highly-ranked results [13]. None of these are good approaches to obtain privacy-preserving apps.

App store ranking algorithms are a closely guarded secret, but cursory analysis suggests that the popularity, age and user rating of an app plays a key role in ranking. It is unclear, however, whether the privacy-preserving nature of an app (or lack thereof) factors into ranking. Taylor and Martinovic developed an alternate app ranking system [21] that penalises apps with aggressive permission usage, after analysing relative permission usage within functionally-similar apps. Permission usage is defined as *aggressive* if an app uses one or more dangerous permissions that are rarely used by functionally-similar apps. For example, a flashlight app that reads contact information uses the `READ_CONTACTS` permission aggressively, since typical flashlight apps do not read contact information. In this paper, we build on the aforementioned ranking system by measuring the additional risk coming from apps that use permissions aggressively and understanding the extent to which this risk can be mitigated. We exclusively focus on Android dangerous permissions, i.e., any of the 24 system permissions denoted as guarding sensitive user data [1]. For this reason, we refer to *dangerous permissions* as *permissions* for the rest of the paper.

Android 6.0 introduced run-time permissions to empower users to selectively reject permissions that are undesirable [1]. Unfortunately, many users *blindly accept run-time permissions* because of conditioning or lack of understanding. Recently, Eling et al. showed that this was the case for approximately 40% of users [6]. Moreover, users are known to ignore permission warnings altogether [5]. Additionally, while run-time permissions are a step in the right direction, they are not triggered on compatible Android versions unless the apps themselves target API level 23 or higher; this is not always the case.[1] A user may also approve permissions to an aggressive app to extract (perceived or actual) added functionality. Thus, for myriad reasons, permissions continue to be granted both consciously and unconsciously by users. We aim to quantify the (potential) additional harm that comes as a result of granting these permissions.

Aggressive permission usage does not necessarily indicate malicious intent by the app developer. Indeed, general-purpose apps may separate themselves from functionally-similar alternatives by

WiSec '17 , Boston, MA, USA

[1] It also remains unclear whether app developers have incentive to implement run-time permissions at all.

using permissions in novel ways to provide attractive new functionality. This is both a strength and weakness of the app ecosystem. Users are able to find many highly functional apps to choose from, but the use of additional permissions by some apps presents an increased risk to the privacy and security of users since such apps may steal sensitive data or be vulnerable to an exploit which does so. To make matters worse, third-party libraries that are embedded within apps enjoy the permissions granted to the host app, and many libraries gratuitously leverage available permissions (see Section 3.1). Thus, what started out ostensibly as app developers using permissions in novel ways, has devolved into providing third-parties with the means to capture and exfiltrate additional data.

We build on the aforementioned alternative app ranking system along several dimensions. We use contextual permission analysis (as summarised in Section 2 and detailed in [21]) to identify apps using permissions aggressively. We then compile a dataset of 1 400 apps with aggressive permission usage and subject them to static analysis to assess their real-world danger. Finally, we leverage a database of alternative apps that are functionally-similar, but use fewer permissions, to understand the improvements that can be made to 28 000 real-world devices.

Concretely, our contributions are as follows:

- We identify 1 400 apps that use permissions aggressively and perform static analysis on them to understand increased threats to privacy coming as a result of their libraries leveraging aggressive permissions.
- We analyse a dataset of over 28 000 real-world devices and their lists of installed apps to quantify the tangible benefits of favouring privacy-preserving apps over those that use permissions aggressively.

2 APPROACH

Generating the dataset for our study was a three-part process. First, we enumerated groups of functionally-similar apps in the Google Play Store (Section 2.1). Next, we analysed permission usage within groups of functionally-similar apps to identify apps using permissions aggressively (Section 2.2). Finally, we downloaded a sample of apps with aggressive permission usage and subjected them to static analysis to understand the (potential) negative impact of their aggressive permission usage (Section 2.3).

2.1 Finding Functionally-Similar Apps

To understand the choices users are given when they search for functionally-similar general-purpose apps, we first need to derive popular general-purpose app search queries made on the Google Play Store. To compile this corpus, we first crawl the descriptions of all apps in the app store and extract the most frequently occurring (non-stop) words. We then run these words against the Google Play Store autosuggest API to determine its suggestions. These suggestions represent popular search queries but these queries may not necessarily return groups of functionally-similar general-purpose apps. To filter unsuitable search queries, we run each search query on the app store and collect the Top 20 apps. Using app descriptions as a proxy for functionality, we use a text similarity measure to ensure that the app descriptions of apps in each set of search results are within a threshold of similarity to each other. We manually identified a suitable similarity threshold. App descriptions are a good proxy for functionality since developers typically describe the functionality of their apps carefully in order to attract users. Further detail on finding functionally-similar apps can be found in [21].

2.2 Aggressive Permission Usage

We arbitrarily limited our search queries to those that returned at least 20 functionally-similar apps and chose the Top 20 from each group to analyse. Our next step was to analyse permission usage by apps within search results to identify those apps that were using permissions aggressively. We define permission usage as aggressive if 10% or fewer functionally-similar apps in a group use a particular permission. Aggressive permission usage may be for the purposes of providing additional functionality, but this functionality can be considered tertiary, since other functionally-similar apps provide the required functionality without needing to leverage such a permission. For this reason, we refer to a permission being used aggressively as an *extraneous* permission for that app.

2.3 Dataset Generation

Combining the concepts of finding functionally-similar apps and detecting aggressive permission usage, we aim to uncover unnecessary privacy risks present in real-world apps, especially those risks coming from embedded libraries. To this end, we randomly generated a list of 1 400 apps that used extraneous permissions as defined above. Additionally, we downloaded the `.apk` files for these apps from the Google Play Store and subjected them to static analysis.

We performed static analysis along two dimensions. First, we measured the extent to which third-party libraries could gratuitously leverage and benefit from extraneous permissions. Second, we examined the context that extraneous permissions were used in, to uncover potentially concerning behaviour from apps using such permissions. The details of our static analysis tools and techniques are explained in Section 3.

3 STATIC ANALYSIS

3.1 Measuring Library Empowerment

To measure the benefit of extraneous permissions to embedded libraries, we determined whether the libraries embedded within each app called permission-protected Android API methods that were guarded by the extraneous permissions used by the app. To this end, we first decompiled the `.apk` files of the apps in question using `apktool` [11] to obtain the `.smali` code for each app. Android API calls were extracted from the `.smali` code and PScout [2] permission mappings were used to derive the permissions needed to make the relevant API call. Third-party libraries were detected using library signatures provided by the authors of FlexDroid [20], but could also have been done using techniques such as those presented in [3]. If library code contained one or more API calls that are guarded by extraneous permissions used by an app, we consider that library to be empowered by the use of these extraneous permissions.

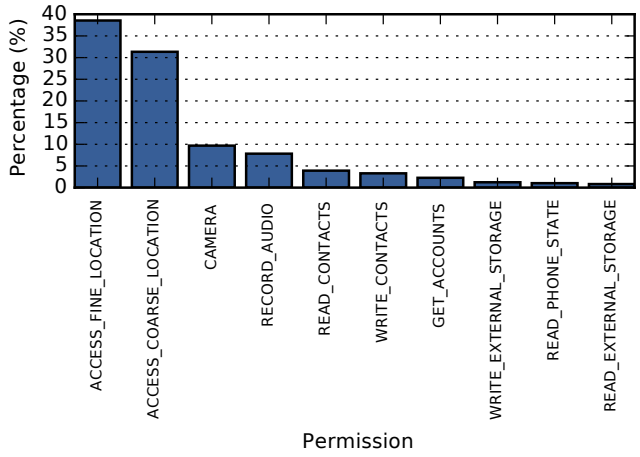

Figure 1: Breakdown of which rare permissions libraries were able to gratuitously leverage.

Across our dataset of 1 400 apps, 358 (25.6%) apps used one or more extraneous permissions that could be gratuitously exploited by libraries embedded within them. Of these 358 apps, approximately 72% had libraries that were able to access one permission, while 28% had libraries that were able to access two or more permissions. Library ability to access multiple extraneous permissions monotonically decreased with an increase in the number of permissions. Interestingly, one app in our dataset used at least five extraneous permissions that its embedded libraries were also able to access.

Fig. 1 shows a breakdown of which extraneous permissions embedded libraries were able to leverage. Libraries were able to leverage location permissions in most cases, and were able to access the device's camera, microphone, contacts, user accounts and external storage in fewer cases. This is worrying, since these permissions are typically not used by apps that are functionally-similar to the offending app, yet as a result of their use, embedded libraries may now enjoy additional privileges unnecessarily.

3.2 Analysing Contextual Integrity of Extraneous Permission Usage

Wijesekera et al. propose the notion of contextual integrity to ensure "information flows are appropriate, as determined by the user" [24]. By instrumenting the Android platform and collecting data on the context under which permissions are used, the authors showed that many apps use permissions without user interaction, i.e., in the background. The authors learnt from their user studies that 80% of participants would have preferred to block one or more permission requests and 35% of permission requests overall. In general, study participants wanted to block permission requests because they did not seem to be required to support app functionality or they were not comfortable sharing such information. Along similar lines, we are interested in understanding whether extraneous permissions are leveraged in the background without user interaction. The idea is that extraneous permission usage is already worrying, and it should be even greater cause for concern if such permission usage happens in the background. This is considering the fact that one might expect the extraneous permission to only be necessary in the first place if it enabled the app to provide tangible (i.e., not happening in the background) functionality to the user.

Table 1: Percentage of apps using extraneous permissions without user interaction.

Permission	Percentage
ACCESS_FINE_LOCATION	22%
CAMERA	4%
READ_CALENDAR	0%
READ_CONTACTS	16%
RECORD_AUDIO	2%
SEND_SMS	26%

To measure how well apps using extraneous permissions performed with regard to "contextual integrity", we leveraged the EviCheck tool [19] and modified it for our purpose. EviCheck is a static analysis tool that certifies and verifies fine-grained permission usage policies for Android apps. With EviCheck, we are able to statically analyse apps to determine the context under which APIs guarded by permissions are used. EviCheck understands permissions being used in contexts such as inside services, when activities are started, and when buttons are clicked.

We constructed an EviCheck policy to warn when an app used any of ACCESS_FINE_LOCATION, CAMERA, READ_CALENDAR, READ_CONTACTS, RECORD_AUDIO, or SEND_SMS when the app is not visible to the user, i.e., in a service/receiver used by the app or when the app is otherwise in the background. We then calculated the percentage of apps with a policy violation from the group of those apps that used the extraneous permission in the first place. This result is shown in Table 1. From the table, the most likely extraneous permissions to be used in the background are SEND_SMS (26%), ACCESS_FINE_LOCATION (22%) and READ_CONTACTS (16%). Using the camera or microphone happened in fewer cases, at 4% and 2% respectively. There were no observable incidences of background access to the device calendar.

4 IMPROVEMENT ON REAL-WORLD SMARTPHONES

Device Analyzer is a research project which collects usage statistics from smartphones [22] in which participants install a data gathering app on their smartphone to support research. The Device Analyzer dataset currently contains over 100 billion records of data from over 28 000 users across the world. Among this data is the list of installed apps on devices and usage patterns for these apps.

We leverage lists of installed apps on devices from the Device Analyzer dataset and combine them with the output of the alternative ranking system [21], to understand the real-world improvement that can be gained if less aggressive apps were chosen by users.

4.1 App Usage Statistics

From the Device Analyzer dataset, users were seen to have a mean of 201 apps installed per device. However, this figure includes

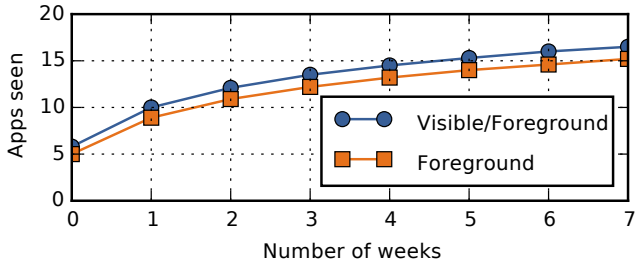

Figure 2: Number of distinct third-party apps with visible/foreground or foreground priority on devices over time.

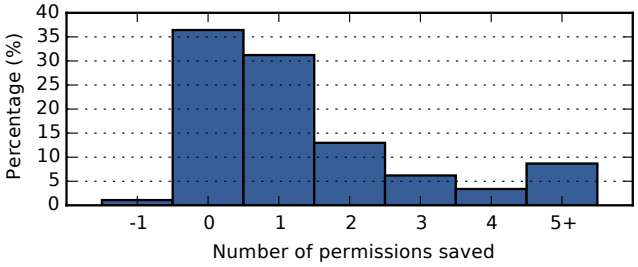

Figure 3: Change in the number of permissions no longer used on devices if all recommendations are followed.

OEM/Android system apps and thus are not (third-party) apps that will have suitable replacements. By taking the intersection of apps installed on devices and third-party apps currently available in the Google Play Store, we determined that each device in the Device Analyzer dataset had a mean of 56 relevant apps installed. Directly performing analysis on this set of apps, however, may give an over-estimate of the utility of the alternative ranking system, since although apps are installed, there is no guarantee that they are still used.

Therefore we defined third-party apps as "used" if they had *foreground* or *visible* priority set at any point, according to Android's `RunningAppProcessInfo.importance` level. This provides a conservative estimate of the improvement that can be derived since we ignore unused apps and apps that run exclusively in the background.

Fig. 2 shows the mean number of apps that were used across all Device Analyzer devices plotted against time. In general, the number of apps with either visible or foreground priority was slightly higher than those with foreground priority only. We considered apps with visible/foreground priority for our calculations. Additionally, we arbitrarily chose a threshold of four weeks to decide if an app meets the criteria of being used. That is, any app not seen within four weeks of the most recently seen app is considered to be unused. This is important because many devices have contributed months or years of data to Device Analyzer. By setting a four week threshold, we ensure that old (possibly uninstalled or no longer used) apps are discarded.

4.2 Replacing Apps

Using app lists from 28,476 devices in Device Analyzer, we observed that, on average, 43.5% of apps could be replaced with a functionally-similar app with less aggressive permission usage. We next analysed the ratings of the apps (out of five stars) to understand whether replacement apps were considered to be high-quality by those users that used them. Approximately 81.5% of recommended alternative apps were +/-0.5 stars of the rating of the app they were intended to replace. This suggests that recommended alternative apps may be well-received by users.

Next, we wanted to understand the overall improvement in permission usage by devices that can be obtained if all recommendations are adopted. Note that this would represent an upper limit on the permissions that could be saved. To measure the permissions saved, we started with generating the union of the sets of permissions used by each app in the original list of third-party apps on a device; this is the *Old Permission List*. We then simulated what the app list on a device would look like if all recommendations were adopted, being careful to include those apps that did not have suitable replacements. We then calculated the union of the sets of permissions used by each app in the simulated app list; this is the *New Permission List*. By taking the set difference of *New Permission List* and *Old Permission List*, we determined the number of devices that had changes in the overall number of permissions used by third-party apps and what the permissions were.

A histogram of permission usage change for our dataset is shown in Fig. 3. From the figure, approximately 37% of devices retained their overall permission usage (i.e., they had a saving of 0 permissions) after replacing apps with more preferable functionally-similar alternatives. This is understandable, since eliminating a permission altogether requires the replacement of all third-party apps that use that particular permission. It may be the case that some apps that use a particular permission simply do not have a replacement, or that the permission in question is vital to delivering required functionality. It is encouraging, however, to see that approximately 31% of devices reduced permission usage by one permission and 13% of devices had a saving of two permissions. This is while at the same time fulfilling the original required functionality of apps on the device.

Note that a small proportion of devices actually used more permissions (-1 in Fig. 3) when all app recommendations were adopted. This is because an app (in rare cases) may be considered to be less aggressive in its permission usage although it uses a greater number of distinct permissions, if its permission usage more closely matches the permission usage for apps of that functionality. As a simplified example, a phonebook app that uses only the camera permission may be considered more aggressive than another phonebook app that uses two permissions to read and write contacts. This is because camera usage in the set of phonebook apps in general would be very rare and thus the former would be more heavily penalised.

Finally, we examined which permissions were saved when app recommendations were adopted. This result is shown in Fig. 4. For brevity we omit permissions that account for less than 1% of occurrences. The most commonly saved permission is `WRITE_CALENDAR`. Because of permission groups, the `WRITE_CALENDAR` permission also allows an app to read calendars on a device. Calendars are typically

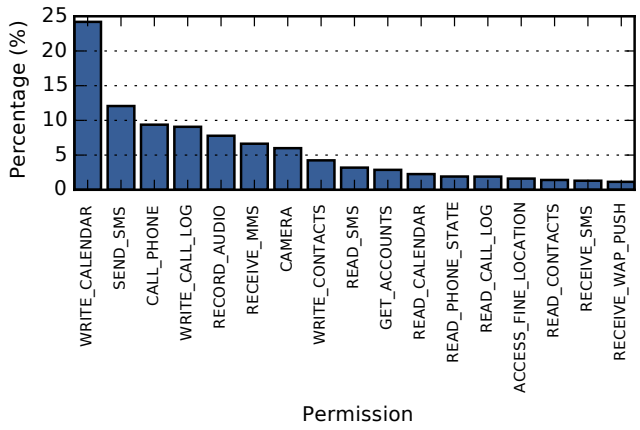

Figure 4: Details of which permissions were saved.

used to store reminders for events a user is interested in, and thus can be a rich avenue for additional profiling data. Some other notable commonly saved permissions include `SEND_SMS`, `CALL_PHONE`, `RECORD_AUDIO`, and `CAMERA`. These permissions can cost a user financially or record from the device's microphone/camera. For devices with permission savings, third-party apps can no longer access the sensitive resources in question.

A privacy-preserving alternative app ranking system can thus serve to complement run-time permissions: by penalising aggressive apps that are incompatible with run-time permissions, and penalising compatible apps that have libraries that gratuitously leverage permissions that are granted.

5 DISCUSSION

A critical pillar underpinning the validity of our results relates to the accuracy with which groups of functionally-similar apps can be obtained. Prior work on the alternate ranking system [21] validated that groups of functionally-similar apps were found with high accuracy. However, some false positives are inevitable. To improve the precision of finding functionally-similar apps, natural language processing could be applied to app descriptions. Additionally, app components such as `Activities` could be analysed to consider the text they contain or methods they call to obtain greater confidence that apps are similar. Code similarity could also be leveraged for a similar effect.

We used static code analysis to understand the extent to which libraries benefited from aggressive permission usage by apps. Roughly one quarter of apps had libraries that were able to leverage permissions that were used aggressively and there was an edge case where an app used five permissions aggressively and contained libraries that were able to utilise all five. This highlights the fact that while adding features (necessitating new permissions outside of the norm) to apps may be a good thing, third-parties may also obtain more user data simply because they inherit the permissions granted to the host app. It is worth exploring approaches such as FlexDroid [20], that seek to separate privileges between apps and their bundled libraries. Additionally, it is great cause for concern that some of the permissions that are used aggressively are being used in the background, i.e., not as a direct response to user interaction with the app, such as being used in the click handler for a button.

Static analysis tools suffer from their own limitations. One major limitation is that they do not properly handle native code or dynamic code that is determined at runtime. Programming techniques, such as reflection, can also be used to obfuscate app behaviour. Thus, the results we present may actually underestimate the gravity of the actual situation. For a more complete picture, dynamic code can be tagged for manual analysis, or apps can be run dynamically on instrumented platforms.

6 RELATED WORK

Felt et al. [8] measured whether users paid attention to or understood permission information as presented to them during app installation. They observed that only 17% of study participants paid attention to install-time permissions and a lower 3% correctly answered comprehension questions about the permissions. Since then, Android has introduced run-time permissions, but it remains unclear what the effect on user comprehension has been as a result of this. Very recently however, Eling et al. [6] showed that approximately 40% of users will accept run-time permissions for minimal reward. Kelley et al. [12] did a related study and found that users are generally unaware of the risks to security and privacy that come from mobile apps. Complementary to this, we study the additional risks to user privacy coming from extraneous permissions and measure the extent to which the negative effects can be mitigated. Felt et al. [7] further looked at permission usage by Android apps and designed a tool called Stowaway that could measure app over-privilege. In a similar vein, we aim to replace such over-privileged apps with functionally-similar alternatives.

In this work, we identify apps using permissions aggressively using a combination of contextual analysis of app functionality and static analysis on the apps in question. Taylor and Martinovic [21] provide the groundwork for scalable identification of groups of functionally-similar apps using text mining of app descriptions. Sarma et al. [18] address the problem from a similar direction, but use the (more coarse-grained) category that an app belongs to in addition to its permission usage to identify anomalies. Peng et al. [16] propose a risk scoring scheme using similar analyses and show that their models outperform related work. Along similar lines, Gorla et al. [10] cluster by description topic before analysing permission usage, in an attempt to identify outliers. The main difference between these pieces of work and our work, is that we gain greater contextual understanding by getting more fine-grained through analysing permission usage in functionally-similar apps.

Other authors also use text mining and natural language processing to understand app functionality and permission usage. An early example is Pandita et al. [14], who developed WHYPER, a system that aims to validate the need for a particular permission from the natural language description of an app. WHYPER was able to flag apps that did not justify the usage of one or more permissions. Watanabe et al. [23] proposed a framework called ACODE, and found that it gave similar performance to WHYPER. WHYPER was later outperformed by Qu et al. [17].

More generally, several authors leverage permission usage patterns to identify malfeasance. We briefly describe them for brevity. Wijesekera et al. [24] assessed the extent to which apps used permission-protected resources when users were unaware. We extend on this by understanding whether apps using extraneous permissions did so in ways that users are typically uncomfortable with. Chia et al. [4] show that some apps attempt to mislead users into granting additional permissions. Frank et al. [9] proposed a clustering approach and showed that low-reputation apps typically deviated from the typical permission request patterns of high-reputation apps. Finally, Papamartzivanos et al. [15] demonstrate the feasibility of a crowdsourcing solution to detect app privacy violations. Complementary to these pieces of work, we measure the potential harm caused by permission-hungry apps and quantify the improvements that can be made.

7 CONCLUSION

In this paper, we examined the real-world privacy impact of apps using apparently extraneous permissions. Using a sample of 1 400 apps, we determined that approximately one quarter of them contained libraries that were able to leverage permissions that appear to be used extraneously. In most cases, this allowed libraries to access the location of a device, and in other cases allowed access to the camera, microphone or contact address book. Worryingly, we found that many extraneous permissions were sometimes used in the background, i.e., not as a result of user interaction with the app. Finally, by using real-world data from over 28 000 users, we showed that up to 43.5% of apps can be replaced with a preferable functionally-similar alternative. As smartphones become more ingrained in our daily lives, it is becoming increasingly important to understand the consequences of using the apps we choose. By highlighting these concerns and showing that alternative choices are available, we aim to transfer power to preserve privacy to the hands of users, where it should always be.

ACKNOWLEDGEMENT

Vincent F. Taylor is supported by a Rhodes Scholarship and EPSRC. Alastair R. Beresford is partly supported by EPSRC [grant number EP/M020320/1] and The Boeing Company. The Device Analyzer project was partly funded by a Google Focused Research Award.

REFERENCES

[1] Android. 2017. Requesting Permissions. (2017). https://developer.android.com/guide/topics/permissions/requesting.html

[2] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. 2012. PScout: Analyzing the Android Permission Specification. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12)*. ACM, 217–228. DOI: https://doi.org/10.1145/2382196.2382222

[3] Michael Backes, Sven Bugiel, and Erik Derr. 2016. Reliable Third-Party Library Detection in Android and Its Security Applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, 12. DOI: https://doi.org/10.1145/2976749.2978333

[4] Pern Hui Chia, Yusuke Yamamoto, and N. Asokan. 2012. Is This App Safe?: A Large Scale Study on Application Permissions and Risk Signals. In *Proceedings of the 21st International Conference on World Wide Web (WWW '12)*. ACM, 311–320. DOI: https://doi.org/10.1145/2187836.2187879

[5] Erika Chin, Adrienne Porter Felt, Vyas Sekar, and David Wagner. 2012. Measuring User Confidence in Smartphone Security and Privacy. In *Proceedings of the 8th Symposium on Usable Privacy and Security (SOUPS '12)*. ACM, Article 1, 16 pages. DOI: https://doi.org/10.1145/2335356.2335358

[6] N. Eling, S. Rasthofer, M. Kolhagen, E. Bodden, and P. Buxmann. 2016. Investigating Users' Reaction to Fine-Grained Data Requests: A Market Experiment. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*. 3666–3675. DOI: https://doi.org/10.1109/HICSS.2016.458

[7] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. 2011. Android Permissions Demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11)*. ACM, 12. DOI: https://doi.org/10.1145/2046707.2046779

[8] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. 2012. Android Permissions: User Attention, Comprehension, and Behavior. In *Proceedings of the 8th Symposium on Usable Privacy and Security (SOUPS 2012) (SOUPS '12)*. ACM, Article 3, 14 pages. DOI: https://doi.org/10.1145/2335356.2335360

[9] M. Frank, Ben Dong, A.P. Felt, and D. Song. 2012. Mining Permission Request Patterns from Android and Facebook Applications. In *IEEE 12th International Conference on Data Mining (ICDM)*. DOI: https://doi.org/10.1109/ICDM.2012.86

[10] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. 2014. Checking App Behavior Against App Descriptions. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM, 11. DOI: https://doi.org/10.1145/2568225.2568276

[11] iBotPeaches. 2017. Apktool - A tool for reverse engineering Android APK files. (2017). https://ibotpeaches.github.io/Apktool/

[12] PatrickGage Kelley, Sunny Consolvo, LorrieFaith Cranor, Jaeyeon Jung, Norman Sadeh, and David Wetherall. 2012. A Conundrum of Permissions: Installing Applications on an Android Smartphone. In *Financial Cryptography and Data Security*, Jim Blyth, Sven Dietrich, and L.Jean Camp (Eds.). Lecture Notes in Computer Science, Vol. 7398. 68–79. DOI: https://doi.org/10.1007/978-3-642-34638-5_6

[13] Jessica Lee. 2013. No. 1 Position in Google Gets 33% of Search Traffic. (2013). http://searchenginewatch.com/sew/study/2276184/no-1-position-in-google-gets-33-of-search-traffic-study

[14] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. 2013. WHYPER: Towards Automating Risk Assessment of Mobile Applications. In *Proceedings of the 22nd USENIX Conference on Security*. 16. http://dl.acm.org/citation.cfm?id=2534766.2534812

[15] Dimitrios Papamartzivanos, Dimitrios Damopoulos, and Georgios Kambourakis. 2014. A Cloud-based Architecture to Crowdsource Mobile App Privacy Leaks. In *Proceedings of the 18th Panhellenic Conference on Informatics (PCI '14)*. ACM, Article 59, 6 pages. DOI: https://doi.org/10.1145/2645791.2645799

[16] Hao Peng, Chris Gates, Bhaskar Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. 2012. Using Probabilistic Generative Models for Ranking Risks of Android Apps. In *Proceedings of the 19th ACM Conference on Computer and Communications Security (CCS '12)*. ACM, 241–252. DOI: https://doi.org/10.1145/2382196.2382224

[17] Zhengyang Qu, Vaibhav Rastogi, Xinyi Zhang, Yan Chen, Tiantian Zhu, and Zhong Chen. 2014. AutoCog: Measuring the Description-to-permission Fidelity in Android Applications. In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS '14)*. ACM, 1354–1365. DOI: https://doi.org/10.1145/2660267.2660287

[18] Bhaskar Pratim Sarma, Ninghui Li, Chris Gates, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. 2012. Android Permissions: A Perspective Combining Risks and Benefits. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*. ACM, 10. DOI: https://doi.org/10.1145/2295136.2295141

[19] Mohamed Nassim Seghir and David Aspinall. 2015. *EviCheck: Digital Evidence for Android* . Springer International Publishing, Cham, 221–227. DOI: https://doi.org/10.1007/978-3-319-24953-7_17

[20] Jaebaek Seo, Daehyeok Kim, Donghyun Cho, Taesoo Kim, and Insik Shin. 2016. FLEXDROID: Enforcing In-App Privilege Separation in Android, In Proceedings of the 2016 Network and Distributed System Security Symposium. *NDSS* (2016).

[21] Vincent F. Taylor and Ivan Martinovic. 2016. SecuRank: Starving Permission-Hungry Apps Using Contextual Permission Analysis. In *Proceedings of the 6th ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM '16)*. ACM, 10. DOI: https://doi.org/10.1145/2994459.2994474

[22] Daniel T Wagner, Andrew Rice, and Alastair R Beresford. 2014. Device analyzer: Understanding smartphone usage. In *Mobile and Ubiquitous Systems: Computing, Networking, and Services*. Springer, 195–208.

[23] Takuya Watanabe, Mitsuaki Akiyama, Tetsuya Sakai, and Tatsuya Mori. 2015. Understanding the Inconsistencies between Text Descriptions and the Use of Privacy-sensitive Resources of Mobile Apps. In *Proceedings of the 11th Symposium On Usable Privacy and Security*. https://www.usenix.org/conference/soups2015/proceedings/presentation/watanabe

[24] Primal Wijesekera, Arjun Baokar, Ashkan Hosseini, Serge Egelman, David Wagner, and Konstantin Beznosov. 2015. Android Permissions Remystified: A Field Study on Contextual Integrity. In *Proceedings of the 24th USENIX Security Symposium*.