How much Privilege does an App Need? Investigating Resource Usage of Android Apps

Nurul Momen Tobias Pulls Lothar Fritsch Stefan Lindskog

PriSec, Karlstad University, Sweden
Email: name.surname@kau.se

***Abstract*—Arguably, one of the default solutions to many of today's everyday errands is to install an app. In order to deliver a variety of convenient and user-centric services, apps need to access different types of information stored in mobile devices, much of which is personal information. In principle, access to such privacy sensitive data should be kept to a minimum. In this study, we focus on privilege utilization patterns by apps installed on Android devices. Though explicit consent is required prior to first time access to the resource, the unavailability of usage information makes it unclear when trying to reassess the users initial decision. On the other hand, if granted privilege with little or no usage, it would suggest the likely violation of the principle of least privilege. Our findings illustrate a plausible requirement for visualising resource usage to aid the user in their decision-making and finer access control mechanisms.**

I. Introduction

Nowadays, the user-driven marketplace possesses an incredible interest in the word 'smart'. Having an interface on a mobile device for an arbitrary system, operating within the surrounding environment, is becoming the prerequisite to be considered as a smart system. As a result, application (app) markets are being populated with astounding speed (over 2.8 million and 2.2 million apps are available in Google play store[1] and Apple app store[2] respectively). Additionally, the rapid growth of pervasive computing has boosted the number of mobile devices. Planet Earth currently has more hand-held devices than the human population[3]. Consequently, smartphones and other mobile devices, including wearables and tablets, have become an integral part of today's lifestyle. These devices are always turned on and capable of monitoring, collecting, storing, processing and transmitting sensitive data. In order to perform various tasks and solve complicated problems, apps require access to sensitive user information. A permission-based structure is used to guard such private data and to protect privacy, and research efforts have been invested to enhance and improve these structures. Contributing to this ongoing effort, we report an empirical study on privilege usage of apps running on Android devices.

Android users are asked to grant or deny privileges for access to resources available on their devices prior to app installation (Android 5.0 and previous versions) or, during runtime (Android 6.0 and later versions). Users are expected to understand and make informed decisions in this regard, which is unlikely to be true, is considered ineffective for privacy preservation and is found to mislead users to unintentional disclosure of sensitive information [1]–[4]. In October 2015, Android 6.0 (Marshmallow) introduced the runtime permission architecture with a view to offer finer control over installed apps[4]. Throughout this paper, we address the permission structure of Marshmallow and later versions (API 23 and onward)[5]. Since the introduction of the new permission model, users can grant permission within a better context instead of blindly accepting everything during installation. However, such consent from a user allows an app to gain access to resources for eternity unless the permission is explicitly revoked. The user has an option to change her mind and withdraw consent. On the other hand, there is no option to be informed about *how many times* the resource is accessed or, *how much* of it is being consumed. Thus, it cannot provide additional information to reassess the situation and to reconsider initial decisions made by the user. We argue that the lack of usage information nullifies the effectiveness of the permission revoking option to some extent. We also argue that the potential to gain uncontrolled access to sensitive information for infinite time leaves the permission model prone to abuse of the principle of least privilege.

The main objectives of this work are to: a) Identifying the dicey resource usage patterns for apps running on Android; b) Identifying how often Android apps access granted resources, and c) Addressing how app's resource usage information can be utilised. The research focus is kept on the privacy sensitive resource utilisation of installed apps. Thus instead of user behaviour, we monitor app behaviour in different combinations, situations and time frames. Our results show that there is room for improvement, and point to the absence of quantifying parameters for accessing sensitive information available on mobile devices.

This paper is organised as follows: Section II describes related work. Section III illustrates relevant necessary background and two motivational scenarios. Our experimental methodology is elaborated in Section IV. We present results, findings and observations in Section V. Our discussion with

[1]https://www.statista.com/statistics/263795/number-of-available-apps-in-the-apple-app-store/; Accessed: 2017-05-13

[2]https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/; Accessed: 2017-05-13

[3]http://www.independent.co.uk/life-style/gadgets-and-tech/news/there-are-officially-more-mobile-devices-than-people-in-the-world-9780518.html; Accessed: 2017-05-12

[4]https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html; Accessed: 2017-05-12

[5]https://developer.android.com/about/versions/marshmallow/android-6.0.html; Accessed: 2017-05-13

limitations and forecast for future work is in Section VI, and our concluding remarks in Section VII.

II. Related Work

A handful of previous research work has focused on the area of Android's permissions, or privileges, and introduced solutions or recommendations to aid the scenario [5]–[8]. Felt et al. [5] contributed in this quest by investigating app development practice with least privilege and their findings show significant infringement: one third of the apps were found to be over-privileged. Au et al. [6] developed PScout to perform static analysis on Android permissions. They examined four versions of Android and 22% of the non-system permissions were found unnecessary. Peng et al. [7] emphasize the importance of developers' awareness during app development, and recommend the notion of risk score development to diminish exposure of sensitive information. Wei et al. [8] highlight the evolution of permissions, and their investigation indicates the growth of over-privileged apps.

Similar questions concerning the frequency of resource access by mobile apps are also asked by several researchers in independent studies [9]–[11]. Almuhimedi et al. [9] show that data on the frequency of access can encourage users towards privacy preserving behaviour. They introduced a method to warn the user, in the form of nudging, about the potential implication in order to encourage privacy preserving behaviour. Franzen and Aspinal [10] introduced a policy based resource usage mechanism for JavaScript apps developed within PhoneGap framework. The 'how often' question, is partially addressed by Kleet et al. [11] who introduced a prototype with data controller indicators against apps' data sharing practices, in order to help the users to make informed decisions.

With similar intentions, we concentrate on finding *how often* and *to what extent* available resources are accessed on mobile devices. This study also addresses, and tries to quantify, the infraction of the least privilege principle. Though there are many aspects of app privilege (requests, relation to advertisement libraries, usability etc.) that have been studied, our effort to quantify apps' behaviour concerning resource utilisation will help to design and develop better user interfaces to aid in decision-making processes. Quantifying an apps' behaviour will also help in designing temporal constraints to support finer access control requirements.

III. Android and Scenarios

As this paper illustrates the behaviour of apps with respect to resource utilisation of a device, a brief summary of Android's permission architecture provides a necessary background. We also discuss two motivational scenarios where potential privacy risks for users are demonstrated.

A. OS Architecture and Permissions

Android runs on a customised Linux kernel which is responsible for running the basic drivers and components (display, audio, binder, etc.). Collectively, these perform the role of a foundation for the Dalvik virtual machine, providing runtime and other native libraries. A tertiary layer, named the application framework, is responsible for accommodating the apps. Each of the installed apps remains virtually isolated in their own sandboxes. In order to perform tasks, apps can request permission to access system resources through privileges. Depending on the resource types, consent from the user is required.

1) Permission Types: There are four types of permissions[6]: *normal*, *dangerous*, *signature* and *signatureOrSystem*. *Normal* level permissions allow access to resources that are considered low-risk, and they are granted during installation of any package requesting them. The *dangerous* level permissions are required to access resources that are considered to be high-risk. In this case, the user must grant permission. This paper confines its focus on the first two permission types (normal and dangerous), as listed by Android.

2) The Principle of Least Privilege: A system can be used by various entities: users, modules, programs, another system etc. According to the principle of least privilege, these parties should require the least amount of privilege and possess legitimate reasons to access operations associated with the corresponding system [12]. This principle is considered to be indispensable for the Android operating system. *"The Android system implements the principle of least privilege. That is, each app, by default, has access only to the components that it requires to do its work and no more. This creates a very secure environment in which an app cannot access parts of the system for which it is not given permission"*[7].

B. Scenario One: The Forgotten Decisions

Consider Alice, a pseudonymous user, who installed a popular social app on her Android device two weeks ago. As she switched on the app and explored new functionalities, she was required to grant access to following permissions:

```
ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION,
    READ_CONTACTS, WRITE_CONTACTS,
    GET_ACCOUNTS, CAMERA,
    READ_EXTERNAL_STORAGE,
    WRITE_EXTERNAL_STORAGE, READ_CALENDAR,
    WRITE_CALENDAR, RECORD_AUDIO
```

Assuming these circumstances, there are a few probable privacy violating situations which could emerge. First, Alice may not attempt to perform any action after first use which would require her location, or information, from external storage. So the reason, for such resource usage is a surprise. Second, perhaps granting these permissions seemed plausible to Alice at that time. Though Alice has the choice to revoke all, or a subset, of those permissions from her settings, no information is available to support her reevaluation of the decisions she took earlier. There could be an alternative app offering a similar service that required less private information that Alice would have preferred, because of better privacy protection. We argue that statistics about an app's resource usage patterns would help the user make decisions with respect to preserving privacy.

[6]https://developer.android.com/guide/topics/permissions/requesting.html; Accessed: 2017-05-23

[7]https://developer.android.com/guide/components/fundamentals.html; Accessed:2017-05-10

C. Scenario Two: The Consistent Inconsistency

Our second pseudonymous user, Bob, likes to work out at the gym and he is passionate about music. Bob has installed fitness and music apps to monitor his workout statistics, and to listen to music while working out. The following permissions were requested by both the apps and granted by Bob:

```
ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION,
    READ_CONTACTS, WRITE_CONTACTS, CAMERA,
    READ_EXTERNAL_STORAGE,
    WRITE_EXTERNAL_STORAGE, RECORD_AUDIO
```

However, Bob is not always at the gym, or running on a trail and listening to music. Usually, he leaves his mobile device on his desk while he focuses on his daily chores. Though his device is lying idle on his desk, it is connected to the Internet through WiFi. If resource usage is observed through the aforementioned dangerous permissions by the music, or fitness, apps, unnecessary resource access events are indicated. Rather than saving his music on his device's storage, Bob prefers to use an online music library, and the music app does not need to access the device storage. Should there be confirmation of no resource usage by a granted permission, violation of the principle of least privilege will be evident due to the futile consumption of the available resources. Furthermore, permission to access an idle resource could lead to exploitation of individual privacy. For example, RECORD_AUDIO permission is subject to ultrasonic beacon side channel attack [13].

IV. Methodology

In this section, the study procedure is explained. First, each component of this experimental setup is outlined: the device, platform and tools. Then the app behaviour monitoring procedure is illustrated. We also specify the few changes between the two observation periods because this study was performed in two phases: a) a stationary phase and b) a user interaction phase.

A. Device and Experimental Platform

We used ten Nexus 7 (2012) tablets with no cellular connectivity. For communication, the device has WiFi, bluetooth, NFC and GPS. Four sensors are available on the device: accelerometer, gyroscope, proximity sensor and compass. During the study, the devices ran on Android 6.0 - Marshmallow. As there is no stock ROM available for this model, we relied on a custom one named AOSP Grouper[8]. Each of the devices was flashed with the AOSP Grouper ROM. Basic apps were downloaded from the Open Gaaps project[9], and installed. The prototype app required root access for all the devices in order to accumulate logs of the system events that related to the resource usage.

B. Prototype: Monitoring App

We developed a monitoring app (runs as a service) which is able to log each resource access event. It plays the role of a listener on the device while focusing only on documenting the resource usage by the installed apps. First, it gathers the list of all the installed apps, (also called packages). We used AppOpsCommand to log the events[10]. Periodically, the app checks for the last resource access event by each of the installed apps and writes respective events in a pre-defined format. The app stores the log records in a JSON file which contains three fields: 1) the name of the package/app, 2) the name of the accessed resource and 3) the time of the resource access event. The following example shows a sample command and the log registered from it:

```
#root: adb shell appops get
    com.google.android.youtube
{"Package":"com.google.android.youtube",
"Permission":"READ_EXTERNAL
_STORAGE","Timestamp":"Fri Mar 03 09:56:
35 GMT+01:00 2017"}
```

It should be noted that we have tried to minimize the logged data. Though such system-generated data is subject to extraction of partial identity, effort is given to achieve anonymity throughout this setup. The prototype is transparent to itself, and the log documents resource usage activities of the prototype app.

C. Stationary Phase

During the stationary phase, each of the devices was assigned to monitor a particular category of apps. The summary of devices and apps is given in Table I. An exception was made for the first category by choosing vendor specific apps. Due to the fact that Google is the largest vendor to offer many apps of different categories, we choose to compare their behaviour with the others. Throughout this phase, the devices remained idle (kept on a shelf). The only notable interaction was to use the prototype app for collecting logs, or to plug in the recharging

TABLE I. Mapping of categories and apps.

App Category	Apps
Google	Youtube, Chrome, Google music, Google maps, Weather, Google+, Photos, Hangout, News, Drive, Docs, Slides, Sheets, Fit, Playgames, Play books, Play movies, Android TV remote, Gmail, Calendar, Earth, Google news
Communication	Messenger, Kiko, Viber, Skype, Imo, Telegram, WeChat, Line, Tango, Slack
Social	Facebook, Instagram, Twitter, Musically, LinkedIn, Snapchat, Tumblr, Pinterest, Foursquare, Yelp
Fitness	Lifesum, Endomodo, 30dayFit, LifeLong, RunKeeper, Pedometer, CalorieCounter, Runtastic, 7minWorkout, Fitbit
Music	Spotify, SoundCloud, Shazam, Tidal, FreeMusic, Sonos, Deezer, JangoRadio, SoundHound, iHeartRadio
Weather	weather.com, WeatherApp, AccuWeather, YahooWeather, WeatherBug, PalmaryWeather, WeatherAndClock, GoWeather, WeatherAndRadar, WeatherXLpro
Games	TempleRun 2, 8ballPool, FruitNinza, TalkingTom, Pou, AsphaltAirborn, ClashOfClans, Farmville, CandyCrush, SubwaySurfers
Shopping	eBay, Amazon, Wish, Zalando, AliExpress, Zara, Lidl, Asos, shpock, H&M
Travel	booking.com, trivago, TripAdvisor, Uber, hotels.com, airBnB, TomTom, Kayak, Expedia, Here
Misc.	Tinder, Badoo, OkCupid, Duolingo, Babel, Netflix, Dropbox, AVG, Firefox, NewsRepublic

[8]https://androidcommunity.com/android-marshmallow-ported-to-nexus-7-2012-everything-working-20151019/; Accessed: 2016-12-27

[9]http://opengapps.org/; Accessed: 2017-05-23

[10]https://android.googlesource.com/platform/frameworks/base/+/android-6.0.1_r25/cmds/appops/src/com/android/commands/appops/AppOpsCommand.java;Accessed:2017-05-23

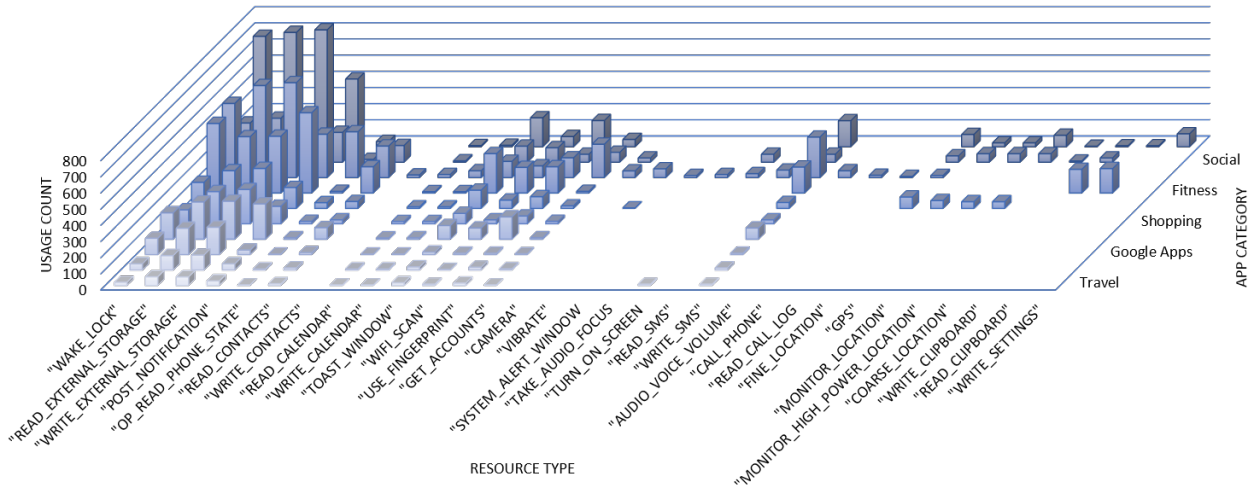

Figure 1. Resource usage summary by installed app groups during stationary phase.

cable once a day. This phase was carried out between 4–10 March, 2017.

1) Test Accounts: Pseudo user accounts were created in order to download and install apps from Google Play Store. These were new, had no history, and did not reflect a real person's everyday behaviour. We named them as 'pseudonymous test accounts'. However, a few default apps from Google had to be installed in each device which resulted in common resource utilisation by those apps and services. This is another reason for isolating Google apps in a separate category.

2) App Selection: As the precise download count is not obtainable for apps, they were chosen based on their visible popularity in Google Play Store. As depicted in Table I, the category of the apps was also considered as another factor for the selection process. Apps from communication and social category possess the most popularity based on visible download counts. Music, fitness and weather apps are the second most probable apps to be found on a device. Favoured apps from games, shopping and travel category seem to be immensely variable depending on user's geographical location. Due to limited number of available devices, ten categories were selected for this phase. Other popular apps (dating, language learning etc.) which could not be put in the aforementioned categories, were kept in the miscellaneous category.

D. User Interaction Phase

The second phase of this study was carried out with more than one variation compared to the previous experimental setup. Instead of leaving the devices idle, they were given to users along with instructions to use the installed apps, to simulate user interaction. Due to the shortage of available human resources, nine devices were operational for this phase. Some study parameters remained unchanged: devices, OS, pseudonymous test accounts and monitoring app.'Factory Reset' was performed before handing the devices to the participants. This phase was carried out between 4–10 May, 2017.

1) Pseudonymous Users: The participants were recruited from a master-level course at Karlstad University. They were all computer science students and were expected to possess relatively good understanding of permission structures and the Android operating system. The students voluntarily participated in the experiment through pseudonymous test accounts and credentials. Two out of the nine participants decided not to donate their data for research after the data collection period. Thus, we managed to analyse accumulated logs from only seven devices upon completion of this phase.

2) Constraints: There were some constraints due to device compatibility and the absence of mobile telephony. As a result, pseudonymous users were restricted on several occasions while choosing apps for this study. Though the student participants were recommended to install at least one app from each category mentioned in Table I, they had the freedom to choose different apps from the Google Play Store. They were required to install and use the prototype app and store the log in a JSON file once a day. This process was repeated each day, for 7 days.

V. Results

How hungry are the apps when accessing resources on a mobile device? The answer is that the amount of resources consumed depends on the app. Due to the vast pool of apps, it seems almost impossible to examine each one of them and expect to yield an accurate answer. So, we bundled them into categories and calculated an estimation for each category. In this section, we quantify the different app group's resource utilisation patterns.

A. Idle-time Usage

Figure 1 summarises the resource utilisation scenario by the installed apps during the Stationary Phase of this study. Though the devices were left idle, apps continued to access the resources containing sensitive user information. The most accessed resource is the READ-/WRITE_EXTERNAL_STORAGE. Understandably, communication and social apps have access to the highest amount of resources. Surprisingly, dating apps from the miscellaneous category are found to be responsible for the biggest consumption of idle-time resource utilisation. The vast idle-time resource usage suggests that the requirement for constraints should be based on the reason for allowing access to the

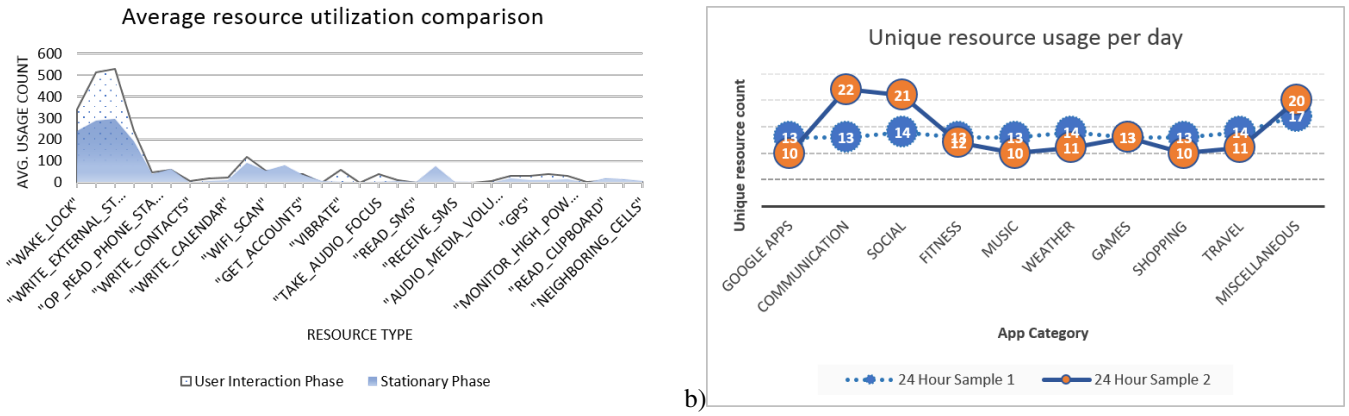

Figure 2. a) Average resource usage comparison between stationary and user interaction phases; b) Unique resource usage comparison between two log samples.

resource. The following log snippet shows sensitive resource usage during idle-time by a dating app called *Badoo*:

```
{"Package":"com.badoo.mobile","Permission":
"FINE_LOCATION","Timestamp":"Mon Mar 06
    19:33:16 GMT+01:00 2017"},
{"Package":"com.badoo.mobile","Permission":
"MONITOR_HIGH_POWER_LOCATION","Timestamp":"Mon
    Mar 06 19:33:17 GMT+01:00 2017"}
```

We argue that access to such sensitive information in a poorly defined context should be categorised as excessive resource usage and controlled by introducing purpose-based permission access.

B. Bundled Permission Usage

As the apps perform different tasks, they require access to variable resources. So, the resource access events are supposed to be task specific. We found the opposite. We observed bundled API calls, which means that a collection of unique resource types were accessed when a subset of that collection would be sufficient to serve the purpose. This result goes against the philosophy of run time permission access. Though permission usage is supposed to be 'activity' specific, the collected log hints this is not the case. For example, the log of a messaging app named *Imo* was recorded on event of launching and closing without further interaction and the following permission usage took place:

```
VIBRATE, READ_CONTACTS, READ_CALL_LOG,
    POST_NOTIFICATION, READ_SMS, WRITE_SMS,
    READ_ICC_SMS, WAKE_LOCK,
    OP_READ_PHONE_STATE,
    READ_EXTERNAL_STORAGE,
    WRITE_EXTERNAL_STORAGE, GET_ACCOUNTS.
```

There was no operation which would require to use WRITE_SMS or, WRITE_EXTERNAL_STORAGE. Introducing a context based constraint to this scenario would help to reduce the amount of the resources accessed.

C. Comparison Between Phases

How much difference did participants make compared to the 'idle' scenario? In principle, it should increase the resource usage which is exactly what happened. Figure 2a shows a relative comparison between average permission usage by a device in two phases. Due to different setups and the number of operational devices, average values of resource accessing events were taken into consideration. Though occasional discrepancies exists, the two graphs have almost similar shapes. Presumably, user involvement caused the spikes in the second phase. We argue that there is room to reduce a portion of the idle-time usage. Should there be no user interaction with the device, sensitive resource usage should be maintained at a minimum. Figure 2a visualises an unequivocal scope to curtail resource utilisation during idle-time.

D. Temporal Usage Discrepancy

Consider two separate log samples from the stationary phase of this study that were registered over a 24-hour time span, and examine only the number of unique resource accesses. Figure 2b shows the comparison between the two sample logs, each accumulated during a 24-hour time span. If we concentrate on the discrepancy between the two samples, we observe that the same group of apps, running on identical platforms, accessed fewer types of resources over different time periods. This example indicates inconsistent app behaviour as long as permission usage, or resource consumption, is a concern. Here we would like to draw attention to two particular observations. First, it is plausible to infer that there is room for limiting resource usage, because the requirement of fewer privileges is evident for similar time frames. Second, it is possible for the apps to carry on their operations with fewer permissions while they have more granted permissions. Our observation demonstrates a clear violation of the principle of least privilege for a certain period of time.

VI. Discussions and Limitations

This study concentrates on documenting the patterns of excessive resource utilisation on mobile devices. However, this study did not cover all aspects, and several issues remained unexplored. First, the devices did not have functionality to use mobile telephony which can be considered as a shortcoming.

Only Nexus 7 tablets were used which did not allow to document app behaviour in different mobile phones and other hand held devices. So, a generalised conclusion may not be credible due to the lack of multi-vendor and multi-model device diversity. Instead of a stock ROM, our selection of the custom AOSP Grouper is another limiting factor. Second, involving pseudonymous users may not provide the acumen of a real user. Though participants voluntarily participated and donated the logs, their activities and interaction pattern may lack the intensity of a real user. Certainly all of the participants had their own devices to use, which left the test device exposed to a limited user interaction. Third, we cannot distinguish between the app's information needs from those of the included advertising libraries. Book et al. analysed Android ad library permissions and found that advertising libraries are increasingly gaining access to more permissions [14]. Furthermore, we did not consider apps servicing each other, or going through a server to trade data, which would allow them to circumvent permissions to some extent [15].

The runtime permission model enables Android to constrain the usage of different 'resource types'[11]. However, the 'frequency of permission use' and 'quantity of resource consumption' were not taken into account. As our study documents questionable resource usage, we argue that the app's behaviour goes against the philosophy of least privilege and supplementary parameters are required to add control over resource usage. Despite its limitations, the overall outcome of this study is encouraging because of the evidence for excessive resource usage. We plan to address the idle-time usage and potential violation of the principle of least privilege in our future work by extending the temporal permission granting method based on context and user interaction. We also plan to test our approaches through user involvement, and overcome the majority of the limitations identified here.

VII. Concluding Remarks

In this study, quite a few cases were found documenting unreasonable access to sensitive information for unlimited time which is indeed alarming for individual privacy. Certainly, there are exceptions which requires continuous permission access in order to deliver real-time services. Time-frame and context based access control could be an alternative constraint for the corresponding scenarios. Analysis and results of this study demonstrate the need for a granular permission control that ensures the principle of least privilege in practice.

Throughout this study, we concentrated on answering the questions of our initial objectives: a) Is there indecisive resource usage patterns for apps running on Android? Brief answer: *Though experimental setup had significant limitations, questionable and inconsistent resource usage patterns were observed.* b) How often do the Android apps access granted resources? Brief answer: *Apps were found accessing granted resources more frequently than an ordinary user would perceive from the interface.* The third question (*How could apps' resource usage information be utilized?*) is tricky to answer due to variable reasons and it will require further investigation and analysis to present a conclusive answer. However, we can forecast a partial answer. We believe that the gathered information is useful to determine the apps' behavior which would help the user to reassess her initial decisions. We need to incorporate this data within mobile interface and observe effectiveness. We would like to carry on our quest toward this direction.

[11] https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html; Accessed: 2017-05-12

Acknowledgment

Authors would like to thank Professor Simone Fischer Hübner for insightful suggestions and comments.

References

[1] M. Alohaly and H. Takabi, "Better privacy indicators: A new approach to quantification of privacy policies," in Twelfth Symposium on Usable Privacy and Security (SOUPS 2016), 2016.

[2] P. G. Kelley, L. F. Cranor, and N. Sadeh, "Privacy as part of the app decision-making process," in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 2013, pp. 3393–3402.

[3] S. Rosen, Z. Qian, and Z. M. Mao, "Appprofiler: a flexible method of exposing privacy-related behavior in android applications to end users," in Proceedings of the third ACM conference on Data and application security and privacy. ACM, 2013, pp. 221–232.

[4] P. G. Kelley, S. Consolvo, L. F. Cranor, J. Jung, N. Sadeh, and D. Wetherall, "A conundrum of permissions: installing applications on an android smartphone," in International Conference on Financial Cryptography and Data Security. Springer, 2012, pp. 68–79.

[5] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in Proceedings of the 18th ACM conference on Computer and communications security. ACM, 2011, pp. 627–638.

[6] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "Pscout: analyzing the android permission specification," in Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 2012, pp. 217–228.

[7] H. Peng, C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Using probabilistic generative models for ranking risks of android apps," in Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 2012, pp. 241–252.

[8] 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. ACM, 2012, pp. 31–40.

[9] H. Almuhimedi, F. Schaub, N. Sadeh, I. Adjerid, A. Acquisti, J. Gluck, L. F. Cranor, and Y. Agarwal, "Your location has been shared 5,398 times!: A field study on mobile app privacy nudging," in Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems. ACM, 2015, pp. 787–796.

[10] D. Franzen and D. Aspinall, "Phonewrap-injecting the how often into mobile apps," in Proceedings of the 1st International Workshop on Innovations in Mobile Privacy and Security co-located with the International Symposium on Engineering Secure Software and Systems (ESSoS 2016). CEUR-WS.org, 2016, pp. 11–19.

[11] M. Van Kleek, I. Liccardi, R. Binns, J. Zhao, D. J. Weitzner, and N. Shadbolt, "Better the devil you know: Exposing the data sharing practices of smartphone apps," in Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. ACM, 2017, pp. 5208–5220.

[12] J. H. Saltzer, "Protection and the control of information sharing in multics," Communications of the ACM, vol. 17, no. 7, 1974, pp. 388–402.

[13] D. Arp, E. Quiring, C. Wressnegger, and K. Rieck, "Privacy threats through ultrasonic side channels on mobile devices," in Security and Privacy (EuroS&P), 2017 IEEE European Symposium on. IEEE, 2017, pp. 35–47.

[14] T. Book, A. Pridgen, and D. S. Wallach, "Longitudinal analysis of android ad library permissions," arXiv preprint arXiv:1303.0857, 2013.

[15] C. Marforio, H. Ritzdorf, A. Francillon, and S. Capkun, "Analysis of the communication between colluding applications on modern smartphones," in Proceedings of the 28th Annual Computer Security Applications Conference. ACM, 2012, pp. 51–60.