Privacy Panel: Usable and Quantifiable Mobile Privacy

Debmalya Biswas
Iprova, Switzerland
dbiswas@iprova.com

Imad Aad
University of Bern, Switzerland
aad@iam.unibe.ch

Gian Paolo Perrucci
Nespresso, Switzerland
gp.perrucci@gmail.com

***Abstract*—The ever increasing popularity of apps stems from their ability to provide highly customized services to the user. The flip side is that in order to provide such services, apps need access to very sensitive private information about the user. This leads to malicious apps that collect personal user information in the background and exploit it in various ways. Studies have shown that current app vetting processes which are mainly restricted to install time verification mechanisms are incapable of detecting and preventing such attacks. We argue that the missing fundamental aspect here is a comprehensive and usable mobile privacy solution, one that not only protects the user's location information, but also other equally sensitive user data such as the user's contacts and documents. A solution that is usable by the average user who does not understand or care about the low level technical details. To bridge this gap, we propose privacy metrics that quantify low-level app accesses in terms of privacy impact and transforms them to high-level user understandable ratings. We also provide the design and architecture of our Privacy Panel app that represents the computed ratings in a graphical user-friendly format and allows the user to define policies based on them. Finally, experimental results are given to validate the scalability of the proposed solution.**

I. Introduction

Apps (application for smartphones) are the lifeline of today's smartphones. Their immense popularity is evident from the thousands of apps available in Apple's AppStore, Google's Android Marketplace, Nokia's Ovi Store, Windows Marketplace Hub, etc. The reason apps are so popular and useful is because they provide services highly customized to different aspects of our life, from recommending location based services of interest to monitoring our health. The downside is that, in order to do so, apps need access to very real-time and contextual information about the user. This information is provided by the many input devices (Camera, Microphone, etc.) and sensors (GPS, Accelerometer, etc.) embedded in mobile devices nowadays which are then provided to the apps. Clearly, such information is very sensitive and has grave privacy implications if it falls in the wrong hands.

To avoid such misuse, most mobile platforms including Meego [1], Android [2], Windows Phone (WP7) [3], provide a 'needs'-based access control model where access to device resources is only given to an app after explicit authorization by the user. At a high level, the model works as follows:

- Apps declare the list of resources to which they need access to provide their functionality in their XML-based Manifest files.
- During installation, the Manifest file is read and the list of required resources' access is presented to the user in a user-friendly format.
- The app is allowed to be installed only if the user 'Accepts'. After installation, the mobile OS provides the needed access control to ensure that the app is only allowed access to those resources as declared in its Manifest file.

While this acts as a deterrent, studies [4], [5], [6], [7] have shown that such a model is not sufficient by itself. Many apps, including some of the most popular ones, have been observed to misuse the install-time access given to them at run-time. For instance, while a weather app requires legitimate access to the user's location, install-time verification as above would be unable to prevent the app from retrieving the user's location every few seconds and feeding it to an external server.

Recent efforts in this direction include fine grained run-time access control models [8], expressive mobile privacy policies [9], [10], smart anonymization techniques especially with respect to protecting the user's location [11], among others. While they are helpful and solve parts of the problem, the fundamental aspect missing there is a *comprehensive and usable mobile privacy solution*. By comprehensive, we mean a solution that not only protects the user's location data (as most of literature currently focuses on) but also other equally sensitive user data types such as the user's contacts and content (pictures, music, etc.). By usable, we refer to solutions that can be used and understood by the "average" user. Defining sophisticated policies restricting access to the device's Accelerometer by apps does not help if the user does not understand what an Accelerometer is, or what personal information can be easily inferred from it. Our goal is thus to provide usable privacy controls that both provide privacy related information and allow the user to set policies at a user understandable level. To this end, we propose the Privacy Panel (PP) with the following main contributions:

- Identify and abstract the user data categories to be protected on the mobile device: Location, Contacts and Content.
- Privacy metrics that quantify low-level app accesses in

This work was performed while the authors were at Nokia Research Center

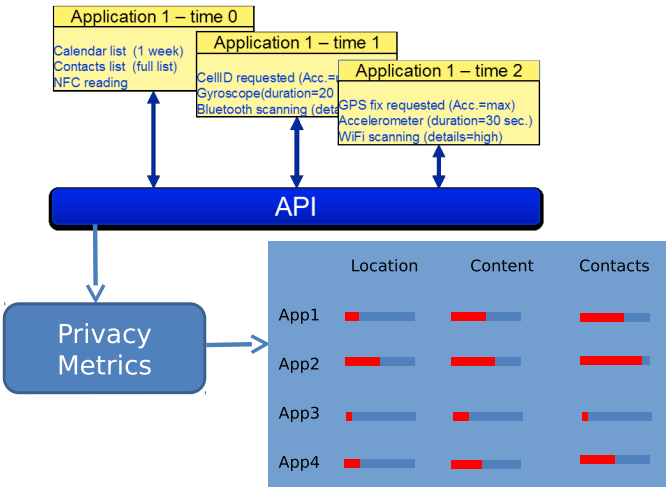


Fig. 1. Privacy Panel Architecture

terms of privacy impact and transforms them to user understandable ratings (Section II-C)
- User friendly app design representing privacy ratings and allowing to define policies based on them (Sections II-B and II-D)
- Implementation and experimental results showing the performance scalability of the proposed approach (Section III).

II. Privacy Panel

A. Architecture

The PP architecture as illustrated in Fig. 1 consists of the following four stages:

1) Intercepting the data accessed by apps installed on the mobile device.
2) Quantifying their privacy impact.
3) Displaying the ratings in a user-friendly format.
4) Providing the user effective access control mechanisms.

The behavior of an app on the device corresponds to how it interacts with the device resources e.g.

- Accessing sensors to read their data values. Common smartphones nowadays include many of the following sensors: GPS, Accelerometer, Ambient Light, Compass, Magnetometer, Orientation, Proximity, Rotation.
- Accessing the network, file system, contacts.
- Intercepting the actions on the touch-screen.
- Accessing the camera, microphone.
- Intercepting the more traditional actions on the device such as making phone calls, sending SMS, etc.

The app usually performs the above actions by invoking (developer friendly) APIs of mobile platform dependent middleware frameworks such as Qt, Windows Phone SDK, Android SDK, etc. Given this, the first PP stage consists of monitoring these API invocations, intercepting and logging their details[2]. It is important to note that in a real-life implementation, mobile apps will have multiple interfaces to the underlying sensors, e.g. HTML5 apps running within the browser, or Android apps with native code/JNI. The PP app will thus need to ensure that all possible entry/exit points are covered, such that all accesses are intercepted. The second stage consists of quantifying the privacy impact of the actions performed by an app. Quantification is based on privacy metrics designed and presented for three user data categories: Location, Contacts and Content (in Section II-C). In the third stage, the computed specific ratings are presented to the user in a user-friendly format, presented in Section II-B. Performance-wise, it is necessary to minimize the execution overhead of computing ratings as much as possible so that their effect on simultaneously running apps is minimal. We consider two possible strategies here in terms of when to (re-)compute the privacy ratings:

- Upon invocation: Compute the ratings only when the PP is invoked. This assumes computing the ratings for a specified time period (possibly user specified), thus delaying the start of the PP application.
- Continuous updating: A background process is continuously running, updating the ratings of relevant app categories for each interception or a group of interceptions (configurable) performed by the app. This basically requires the metrics to be computable in an incremental fashion. Performance wise, this option has more overhead as both interception/re-computation need to be performed in parallel with normal apps' execution at run-time. However this is absolutely essential if the PP app is also expected to provide the access control feature.

The fourth and final PP feature is providing access control capabilities to the user. Once the user becomes aware of how his personal data is collected by an app and what its privacy implications are, he has the option of setting policies to regulate the app's behavior in future (detailed in Section II-D).

B. User Interface

In this section, we describe the Privacy Panel user interface flow. Screenshots are given in Fig. 2.

Screen (1) is the start screen, shown to the user upon invoking the PP. It lists the apps installed on the mobile device with their corresponding privacy ratings. The privacy ratings are displayed with respect to three categories: Location, Contacts, and Content. Clicking (tapping) on one of the app rows leads to the corresponding app's detailed privacy report - Screen (2). We consider the Weather App here for illustration. Apart from the privacy related information, Screen (2) has two text fields: "From the Developer" and "About the Privacy Panel". The latter is self-explanatory and gives high

[2]Apps running on mobile devices without communicating with external entities do not pose any privacy threat. However, a malicious app has various potential means of 'stealthy' communications, e.g. encrypting data, deferring the communication, or communicating via other apps etc., making the interception of these communications another stand-alone problem that we keep out of scope of this paper.

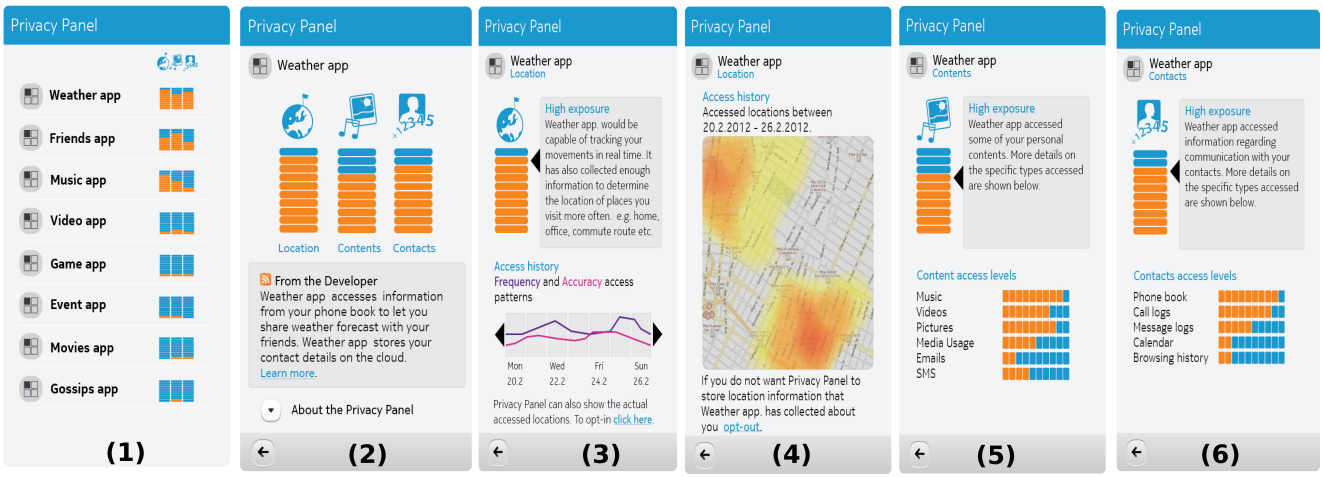


Fig. 2. Privacy Panel User Interface

level information regarding the app software, underlying logic, installed version, etc. "From the Developer" provides a RSS feed type scrolling message box. Messages can come from the app developer explaining to the user why a certain level of access (leading to a higher privacy violation rating) with respect to a data category is needed.

Tapping on one of the category icons (location, contacts, content) leads to the privacy details corresponding to that category. Let us first consider the location category - Screen (3). The screen shows the location privacy rating computed based on the app's location accesses on the device till date. The ratings are abstracted into three levels with a higher privacy rating corresponding to a higher intrusion level, i.e. an app with low privacy rating is safer, i.e. less intrusive.

The lower part of Screen (3) shows the frequency and accuracy access patterns based on the user location data access by the app for the past week. The user can scroll backwards and forwards to see the respective patterns for the previous weeks. The daily frequency pattern is computed based on the number of times user location data was requested on that day. As will be explained later in Section II-C, the user's location can be requested with different levels of accuracy having different implications on the user's privacy and the device battery. The daily average of requested location accuracy levels by the app is plotted in Screen (3).

The final aspect with respect to app location privacy is to display the actual locations where user location data was accessed by the app. This helps the user gain an understanding of the locations where the app has been more active, e.g. if a malicious app has been performing selective profiling targeting specific sensitive user locations such as home, office, etc. To display this information, the PP app also needs access to user location data (to log the actual locations accessed by the app), and hence this feature is enabled only after an explicit 'opt-in' by the user. Once enabled, the user is shown this information in the form of a heat map- Screen (4).

On the lines of location, tapping on the Content or Contacts icons in Screen (2) leads to Screen (5) or (6) respectively. Screens (5) and (6) display app access information related to Content and Contacts categories. Content privacy ratings are computed based on app accesses to physical files, media, messages on the phone. Contacts ratings are computed based on app accesses to the address book entries, and metadata related to communication with contacts, e.g. which contact was called/emailed, when, and how often.

C. Metrics

In this section, we present privacy metrics for the three user data categories: Location, Contacts and Content. The metrics allow us to quantify the privacy impact of an app accessing specific user data. It is worth noting that the proposed privacy metrics are by no means foolproof. They are a first attempt to quantify the privacy impact of mobile apps in general. However, we are convinced that this is still the right path to follow. We draw an analogy to the password strength checkers widely used on the web. Based on heuristics, they are a practical and generically applicable estimate of password strengths, not necessarily matching the cryptanalytical results. They successfully have come to be trusted by average users who are taking now the extra effort in setting "Strong" passwords. Similarly, our PP aims at providing average users rough estimates of their privacy exposure to apps running on their mobiles, without necessarily offering strict security guarantees.

1) Location: Location data in very simple terms corresponds to the (longitude, latitude) geo-coordinates of the user's location. Our objective here is to quantify the privacy impact of user location data accessed by an app. The quantification aims to capture how much location data has been collected by the app, how precise it is, and how privacy sensitive it is from the user's point of view. The purely linear measure of "how much" can be computed by recording when (how

frequently) was user location data accessed by the app in question. With respect to the precision of location data, it is important to recall here that user location information can be obtained via different positioning methods, e.g. satellite based (GPS), non-satellite based (WiFi, Cell tower), etc. with different energy and accuracy trade-offs. We thus compute the location privacy metric $\mathcal{L}_P$ of N samples, considered during the periods when the application is active, as a function of *frequency* and *accuracy* factors as follows:

$$\mathcal{L}_P = F \times A$$

where F is the frequency factor, and A is the accuracy factor defined as:

$$F = N/(T^{ON} \times f_{max})$$

where T^{ON} is the sum of periods when the application is active, and f_{max} is the maximum sampling frequency supported by the location sensors ($= 1/sec$ by default). The accuracy factor A is formulated as:

$$A = 1/N \times \sum_{i=1}^{N} \frac{P_{max}^2}{H_i^2}$$

where P_{max} is the maximum precision provided by the sensors (i.e. $2.2m$). H_i is the horizontal accuracy of the i^{th} sample. It is computed as the average returned precision when an app requests a location sample with *high*, *low*, *any* precision. Default values to be considered are $2.2m, 30m, 15m$ respectively.

Let $\mathcal{L}_P(t)$ denote the privacy metric computed till time t. Then, the incremental version of the metric only needs to take into account the variable parameters:

$$\mathcal{L}_P(t+1) = \mathcal{L}_P(t) + \frac{P_{max}^2}{H_{t+1}^2 \times T_{[t,t+1]}^{ON}}$$

where H_{t+1} denotes the horizontal accuracy of the location data read at time $t+1$ and $T_{[t,t+1]}^{ON}$ denotes the run-time of the app between times t and $t+1$. It is worth mentioning that there might be a difference between the type of data requested by an app and the data it actually receives. The former hints at the "curiosity level" (maliciousness) of the app, for instance an app that always requests for user location with the highest precision. However, owing to a variety of technical and system issues, the location data that is finally returned to it may have a lower precision than requested due to the GPS not being available (indoors), battery conservation, etc. The actual privacy leakage in terms of location data obtained by an app clearly depends on the latter aspect. We briefly discuss below the aspects that influence the actual privacy impact of location data obtained by an app. Some of these aspects are already taken into account in previously presented location privacy metric $\mathcal{L}_P$. A comprehensive location metric taking into account all the below aspects is beyond the scope of this work.

- App usage characteristics: We currently consider usage characteristics of the user as the time the app was actually running ('ON') within a specified time period. This allows us to discount factors that the app was not requesting user location updates only because it was not running. This also requires the PP to log the app start and terminate timings.
- User movement pattern: The actual location co-ordinates (latitude, longitude) can be used to re-create the user movement pattern, including the speed at which the user is moving. The intuition is that the accuracy with which an app (attacker) knows the user's current position decreases as time elapses after the last location update, till the next one.
- User surroundings: Environmental factors such as if the user lives in a densely or sparsely populated area influence the impact of user location data obtained by an app.
- User sensitivity to different locations vary at different times.

2) Contacts: We consider Contacts to be mainly the user's Address book on the mobile device. Each entry in the address book has a unique identifier (e.g. URI, Name, Date of Birth) and associated fields e.g. Address, Email, Phone numbers, etc., also referred to as 'details'. The Contacts privacy rating $\mathcal{C}_P$ of an app is intuitively based on two factors:

- The number of unique Contacts accessed by the app, and
- The number of Details of each contact accessed

More precisely,

$$\mathcal{C}_P = 1/N_t \sum_{C=1}^{N_a} D_a(C)/D_t(C)$$

where N_t is the total number of contacts, N_a is the number of contacts accessed by the application, $D_a(C)$ is the number of details of contact C that were accessed, and $D_t(C)$ is the total number of his details.

The uniqueness of contacts accessed here is significant as accessing the same contact details twice does not give the attacker any additional information. This model leaves room for extending the metric with weights assigned based on the user's relationship with the accessed contact, e.g. spouse, close friend, office colleague, etc. Similarly, weights can also be assigned based on the sensitivity of the accessed contact detail fields, e.g. Address, Email, Phone Number, etc. The Contacts privacy rating $\mathcal{C}_P$ can be computed incrementally as follows. For each new interception of a Contacts entry C:

1) Check if C is unique, i.e. has been accessed for the first time based on its ID. If so, proceed; else exit.
2) Determine the number of contacts n in the Address Book.
3) $\mathcal{C}_P = \frac{1}{2} \times (\mathcal{C}_P + 1/n)$.

For each new interception of a Contacts detail field D of contact C:

1) Check if field D has been accessed for the first time based on uniqueness of the pair (C's ID, D). If so,

proceed; else exit.

2) Determine the number of detail fields m of C in the Address Book.
3) $\mathcal{C}_P = \frac{1}{2} \times (\mathcal{C}_P + 1/m)$.

3) Content: We consider Content to be one of the following types of files: Audio (Music), Video, Picture (Image), and Documents. Apart from capturing "how many" files and of "what type" have been accessed by an app, we also take into consideration the file's properties and tags to quantify the privacy sensitiveness of that file for the user e.g. user-generated content is considered more privacy sensitive than a non-user-generated one. Similarly, from a privacy impact perspective, content containing geo-tagged information are rated higher than non-tagged ones. We give below an incremental algorithm to compute the Contents privacy metric $\mathcal{D}_P$ taking into account the above aspects. We consider three configurable scaling factors y, p, t defined as follows:

- y: File type factor
- p: Property factor
- t: Tag factor

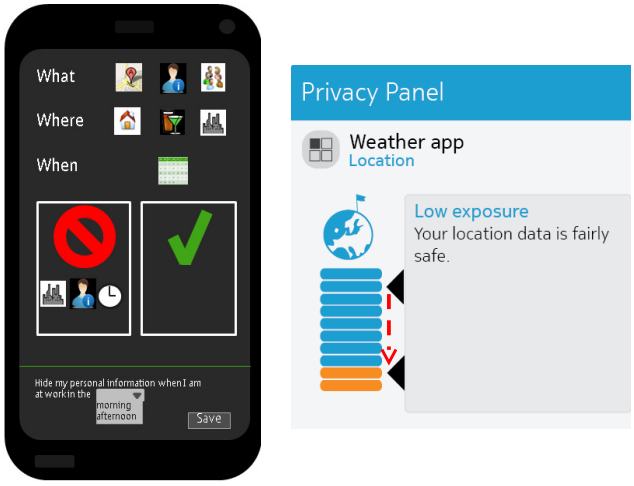
Let the default values for all the above scaling factors be 0.5.

For each file f accessed by the app, $\mathcal{D}_P$ is updated as follows:

1) Check if f belongs to the user: Only consider files residing in user document directories. This is to discount app file accesses to retrieve app owned media e.g. icons. The assumption here is that there exists a "Documents" folders on the device where all user generated files are stored by default. If yes, continue; else exit.
2) Check if f is unique, i.e. accessed for the first time, based on the file's modification timestamp. Here we consider a file having a different modification timestamp as a "unique" file access. This is based on the observation that a modified file has the potential of providing additional information to the attacker. If unique, continue; else exit.
3) Compute the scaling factor s: Let $s = 1$ initially.
   - If file type is 'Image' or 'Video', then $s = s \times y$. The file type is determined based on the file extension.
   - If f is not user-owned, then $s = s \times p$. The file's Creator/Owner property is used to determine this aspect.
   - If f is not geo-tagged, then $s = s \times t$. The file's EXIF tags are inspected to determine if it is geo-tagged or not.
4) Depending on the file type, scan the corresponding user directory 'Music (Audio)', 'Videos', 'Pictures', 'Documents' directories to compute $n-$ the number of files in that directory.
5) Finally, update $\mathcal{D}_P = \frac{1}{2}(\mathcal{D}_P + s/n)$.

D. Policies

Once the user becomes aware of how his personal data is collected by apps and what its privacy implications are, it is natural for him to want to set policies regulating future app behaviors. Note that setting policies does not necessarily equate to preventing the app from performing certain actions. The protective measure may be in terms of a simple notification message when an app performs a certain action or a certain pre-defined privacy rating threshold is reached.



(a) Low-level policy generator

(b) Setting privacy thresholds

Fig. 3. User control Interfaces

Fig. 3(a) gives the PP user interface to generate low-level privacy policies controlling accesses of an app to the device resource level. The interface is primarily intended for advanced technical users who understand the device resources/internals, and is in line with controls offered by related apps such as Privacy Guard [12] and Lookout [13].

Following the general philosophy of this work to provide user-friendly privacy control to "average" users, we allow such users to also regulate apps by specifying threshold privacy ratings for each data category. With reference to Screen (3) of Fig. 2 discussed earlier, Fig. 3(b) shows the user setting of the allowed location access of Weather App to 'Low'. Enforcing such policies clearly requires following the "continuous updating" execution model with the relevant privacy category rating re-computed corresponding to each intercepted access performed by the app.

III. Evaluation

We implemented the PP app on the Nokia N900 mobile. The relevant middleware on the N900 is Qt. Qt provides a platform neutral API framework for apps to interact with the mobile. The Qt API framework is categorized into modules to perform different tasks on the device. We thus *modified the Qt module libraries to intercept and log details whenever an app invokes a Qt API to access device resources*. The intercepted data logs as well as computed ratings are stored in a SQLite database on the mobile. The experiments were performed with an energy profiling app running recording the CPU load. In the

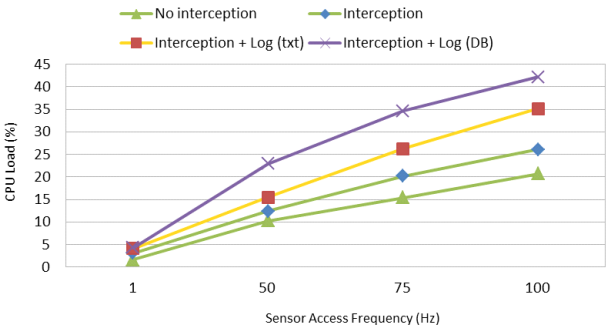


Fig. 4. Privacy Panel performance

experiments, sensor accesses correspond to apps requesting for user location information. Fig. 4 gives the performance results. 'No interception' denotes the CPU load when only the app is running, without PP monitoring. 'Interception' refers to the scenario when PP intercepts any sensor accesses performed by the app without logging their details. The scenario is relevant when only low-level policies need to be enforced without computing the PP metrics. 'Interception + Log (DB)' gives the CPU load for PP when all sensor accesses are intercepted and details stored in the SQLite DB. As the overhead in this case is significant, we also explored the scenario where intercepted details are stored in text files: 'Interception + Log (txt)'. The alternative seems much more scalable with only a 10% rise is execution overhead for even very high sensor access frequencies of 100 Hz.

IV. Related Works

Mobile privacy is an active research area. The research focus till now has clearly been on protecting user location information [11]. Interesting approaches include techniques [14], [15] that anonymize location information to a certain level of abstraction before returning it to the apps. The challenge here is in anonymizing to the "right" level so that the user's information is protected and the app is still able to function. Clearly, it does not help to abstract the user's current location to the 'City' level before returning it to an app that is supposed to recommend nearby restaurants.

The other commonly used privacy protection mechanism is based on policies. This includes specifying policies restricting the apps from performing certain actions, mainly with respect to accessing resources/sensors on the device. Research in this direction has focused on accommodating fine grained [8] and expressive [9], [10], [16] policies. Commercial apps such as Privacy Guard [12] and Lookout [13] are also available that give the user quite detailed control of the app accesses, sorted by app or by sensor. However, as discussed earlier, an average user is unlikely to understand the magnitude of the threats when citing simply the sensor being accessed by the intrusive application. To bridge this gap, we present privacy metrics in this work that map low level sensor accesses to high level privacy ratings.

From a quantification perspective, statistical measures have long been used to quantify leakage of security protocols. The commonly used statistical measure for quantification is the notion of Shannon entropy [17]. However, Shannon entropy based measures have in the recent past been shown to be not so suitable for leakage quantification [18]. To overcome this, [19] has proposed leakage quantification measures based on conditional entropy. Our work is a first attempt towards defining quantification measures for different user data categories relevant for mobile privacy.

V. Conclusion

In this work we consider the problem of quantifying app accesses with respect to mobile user data in terms of their privacy impact. We presented privacy metrics for three user data categories, namely Location, Contacts and Content. The metrics allow mapping app accesses with respect to those categories to high level privacy ratings. We then outlined the design and architecture of our proposed PP that both displays privacy related information to the user and allows them to specify policies in terms of the computed ratings. We finally presented performance results that show the implementation feasibility of our concept.

We believe that the proposed approach lays the foundation towards the long-term objective of making mobile privacy safeguards more usable and accessible to the average user.

References

[1] E. Reshetova and C. Schaufler, "Mobile Simplified Security Framework Overview," 2010.
[2] C. Nachenberg, "A Window Into Mobile Device Security," 2011.
[3] M. Corp., "Windows Phone 7 Security Model."
[4] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," in *OSDI*, 2010.
[5] A. Chansanchai, "iPhone Flaw allows Apps access to your Contacts."
[6] J.-L. Boyles, A. Smith, and M. Madden, "Privacy and Data Management on Mobile Devices."
[7] P. I. Project, "Survey: Mobile users wary of privacy invasion by apps."
[8] Y. Zhou, X. Zhang, X. Jiang, and V. Freeh, "Taming Information-Stealing Smartphone Applications (on Android)," in *TRUST*, 2011.
[9] D. Biswas, "Privacy Policies Change Management for Smartphones," in *MUCS*, 2012.
[10] D. Biswas, N. Nefedov, and V. Niemi, "Distributed and Minimal Usage Control," *Service Oriented Computing and Applications*, vol. 6, no. 4, 2012.
[11] L. Liu, "Privacy and Location Anonymization in Location-based Services," *SIGSPATIAL Special*, vol. 1, no. 2, 2009.
[12] "LBE Privacy Guard." [Online]. Available: {http://www.lbesec.com/lite/en/}
[13] "Lookout Mobile Security." [Online]. Available: {https://www.lookout.com/features/security}
[14] B. Gedik and L. Liu, "Location Privacy in Mobile Systems: A Personalized Anonymization Model," in *ICDCS*, 2005.
[15] G. T. R. Shokri, J.-Y. L. Boudec, and J.-P. Hubaux, "Quantifying Location Privacy," in *S&P*, 2011.
[16] B. Gedik and L. Liu, "MONPOLY: Monitoring Usage-Control Policies," in *RV*, 2011.
[17] C. Shannon, "Communication Theory of Secrecy Systems," Bell System Technical Journal, Tech. Rep. 27: 379423, 1948.
[18] G. Smith, "Adversaries and Information Leaks (Tutorial)," in *TGC*, 2007.
[19] M. Wibmer, D. Biswas, and F. Kerschbaum, "Leakage Quantification of Cryptographic Operations," in *IS*, 2010.