2013 IEEE 14th International Conference on Mobile Data Management

Privacy Management for Mobile Platforms — A Review of Concepts and Approaches

Christoph Stach† and Bernhard Mitschang
University of Stuttgart
Institute for Parallel and Distributed Systems
Universitätsstraße 38
70569 Stuttgart, Germany
Email: Christoph.Stach@ipvs.uni-stuttgart.de, Bernhard.Mitschang@ipvs.uni-stuttgart.de

***Abstract*—The still rising popularity of modern mobile phones results in an increased demand for manifold applications for these devices. As Android OS supports the development and usage of third-party software greatly, there are more and more developers for this platform. However, many of those applications handle private data grossly negligent which immediately leads to serious privacy concerns. To make matters worse, the current Android permission rules are much too coarse and incomprehensible from the average user's perspective. But even if s/he understands the meaning of the permissions, s/he must either accept all of them or waive the application.**

Therefore we review concepts and approaches towards effective privacy management for mobile platforms. All this is discussed based on the prevailing key players in the mobile market, namely Apple, RIM, Microsoft and Google. As this work has been initiated by Google we mainly concentrated on Android-based concepts towards customizable privacy management approaches. As a result of our review and taking into account current initiatives and trends in the market, we come up with a novel approach, an implementation architecture and a prototype.

***Keywords*-privacy; profound overview; permission model.**

I. Introduction

Smartphones are established in our society and constitute an inherent part in our everyday life. These devices are no longer used as mobile phones only, but as mobile computers for your pant's pocket. IDC [1] forecasts a high demand for smartphones despite a general decline in sales of mobile phones worldwide: 686 million units will be sold by the end of 2012 and the target audience is still growing.

One of the many reasons for this trend is that there are a lot of third-party applications for these devices. For instance Apple and Google currently offer all in all over one million applications in their marketplaces [2]. In order to keep this developer community alive, the smartphone vendors seek to publish all kinds of tools and infrastructure to support the whole development process—starting with SDKs or IDEs through to application stores. As these devices are on the one hand often operated by inexperienced and careless users and on the other hand carry a large number of sensitive data, they become more and more interesting for data thieves.

Moreover, sophisticated security mechanisms are missing. Mechanisms that are urgently needed to provide reasonable protection of the large amount of private data (e.g. pictures, contact information or messages) stored on the smartphone. Examples for data abuse can be found with increasing frequency across all mobile platforms (e.g. [3] or [4]).

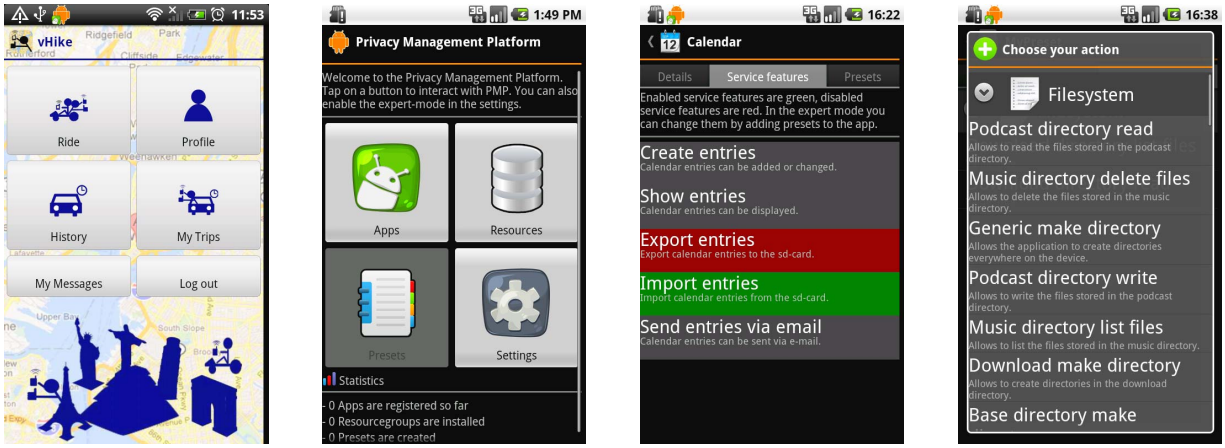
Alarmed by these reports, users get more conscious about protecting their privacy. Smartphone operating system vendors are aware of these hazards and provide several different approaches to prevent privacy compromises. However, despite all of these efforts, none of these approaches informs the user comprehensively, which rights s/he has granted the application at installation time or what they are doing with his or her data. Not to mention the fact that s/he cannot restrict an application's permissions—either one grants all permissions or the application cannot be installed at all.

Consider the example of a mobile application providing various services to enable and support dynamic ride-sharing such as *vHike* [5] [6]. Both drivers and hitchhikers receive benefit throughout the entire process from vHike: Starting in finding offered or requested rides, providing information about potential passengers, getting in touch with fellow travellers to clarify some details before the ride, guiding the driver to the negotiated pick up locations and last but not least guaranteeing that all parties involved reach their destination safely. Additionally, third parties can track the ride in real-time via Internet to ensure a trouble-free hitchhike.

Such an application will require inter alia the following permissions: *manage private data* (e.g. for identifying a user to the community), *access current location* (e.g. for navigation), *use phone functions* (e.g. for making contact before the ride), *access the Internet* (e.g. for searching distant ride providers), *use Bluetooth features* (e.g. for searching nearby ride providers), and so forth. Albeit, vHike (depicted in Figure 1a) provides some useful features, such an application could cause a great harm due to the large amount of personal data it has access to. That fact would probably scare off

† This work was supported by a Google Research Award.

978-0-7695-4973-6/13 $26.00 © 2013 IEEE
DOI 10.1109/MDM.2013.45



(a) vHike Main Menu (b) PMP Management (c) Service Features Menu (d) Privacy Settings Menu

Figure 1. Captures of the vHike Sample Application and the PMP GUI

many users giving such an application a try. Keep in mind that users would often be satisfied with a reduced service scope if they do not have to sacrifice that much privacy.

Wouldn't it be nice to be able to manage the permissions of an application in a more fine-grained manner than it is possible in the currently available mobile operating systems? Shouldn't the privacy settings be modifiable at run-time without causing an application to crash? Why isn't it possible to provide an application with fake / blurred data instead of the privacy compromising original data? Wherefore isn't the user informed in more detail what happens to critical data? Wouldn't it be reasonable to add contextual constraints to the privacy rules?

The remainder of the paper is structured as follows: In Section II we reflect upon the strategies the key system vendors come up with. We consider the major third-party approaches from research as well as the smartphone community in Section III. Subsequently, we introduce an approach to customizable privacy management and define in Section IV the so-called *Privacy Management Platform (PMP)*, a fine-grained, context-based and extendable approach for securing privacy on Android-based devices. Therefore we give an overview of its features firstly (Section IV-A) and deal with our model for privacy policies, secondly (Section IV-B). Section V highlights the main components of the approach. Afterwards we discuss different implementation strategies which might be relevant for the approach in Section VI. Finally, Section VII evaluates the presented approach under qualitative aspects before Section VIII comes up with a brief summary of this paper and an outlook on future work.

II. Prevailing Mobile Platforms

Smartphones combine three critical factors, when concerning privacy: *private information* (which should not be shared with anyone), *context data* (which could reveal a lot about the user's current situation) and *network access* (which is a permanent threat for data exposure). In addition, users generally don't have an adequate sense of how easily an application could misuse their data, anyway. According to Wetherall et al. [7] users can manage their privacy settings if and only if a reliable source (e.g. the OS) informs them how their information is used. Due to constantly emerging reports in the news about vulnerabilities in almost every current smartphone OS, the users call for fine granular adjustment options that support application independency, transparency in the use of the data and the possibility to import configuration presets suggested by reputable authorities.

However, not only the smartphone itself entails various risks, often the applications pose a threat, also. There are several reasons for this: Firstly, the smartphone vendors need an active developers community in order to stay attractive for the customers by constantly offering new applications. Hence, they lower the entry barriers for the developers so much, that even non-programmers can build their own applications. Even distribution channels are provided so that these homemade applications can be propagated easily. Thus, they also attract script kiddies who, driven either by inability or evil intentions, compromise the smartphones. Added to this is that between 30 percent and 50 percent of all Android applications are overprivileged regarding their requested permissions. Felt et al. [8] discovered this fact, when they examined 940 randomly selected Android applications. Finally, there are also planned attacks on the system (and thus on the private data) of course. Enck et al. [9] introduced a tool called TaintDroid, which monitors an application's access to and use of private data in real time without interfering. They observed a misuse of sensitive data in two out of three applications. All analyzed applications were (and still are) available in the Google Play Store.

Table I
OS COMPARISON

	Apple iOS	BlackBerry	Windows Phone 7	Android OS
Application Isolation	system-based sandbox	Java virtual machines	extended sandbox	Manifest isolation
Reason for Code Signing	control mechanism	system protection	fingerprint	application protection
Root Access	no	no	no	no
Kill Switch	yes	?	yes	yes
Application Installation Model	Walled-Garden Model	Guardian Model	Guardian Model	User Control Model
Controlled Marketplaces	iTunes App Store	multiple sources	Zune Marketplace	multiple sources

The privacy violations reach from sending the device ID or other private information (e.g. the phone number) to the application's context server to reporting the geographical coordinates to third party servers (e.g. advertisement servers).

Thus, all smartphone vendors are aware that they have to provide strategies to face these threats. In the following we present the different approaches of the current key players in the market, namely Apple, RIM, Microsoft and Google.

We compare the four systems regarding the following six properties: *Application isolation*, *code signing*, *root access*, *kill switches*, *application installation model* and the *controlled marketplaces*.

Application Isolation. Principally, each of the four considered OSs ensures that every application runs in its own sandbox isolated from the rest of the system, albeit they differ in some details: iOS provides isolation at system level while BlackBerry provides process isolation via Java virtual machines as BlackBerry applications are plain JavaME code. Windows Phone 7 extends the sandbox concept by introducing four chambers. Each application is assigned to one certain chamber wherein each chamber has only a limited amount of permissions. Depending on how trustworthy an application is, a corresponding chamber is selected. Basically Android also isolates each process, albeit applications of the same developers can run in a common process.

Code Signing. In general, applications are signed in order to validate their origin and guarantee that an application has not been modified furtively. Admittedly, this technique can be applied in various ways. Apple signs every application which is available in the App Store—i.e. any application that can be run on an iOS device. Thereby Apple confirms the application a good behaviour after an intransparent and time consuming vetting process. RIM also demands that each application has to be signed, though they differentiate between two signatures: The *Carrier Code Signature* which is available for every developer and the *RIM-Signature* which has to be requested directly by RIM. While the former one authorizes the execution of functions regarded as harmless, only, the latter one enables to use also the restricted ones. Developers of applications for Windows Phone 7 have to apply for a digital certificate by Microsoft and sign their applications with it. In this way Microsoft is able to backtrack who can be made responsible for any malware. And finally, Android applications are mere self-signed, i.e. the developer autonomously generates a key pair without any involvement on behalf of Google. This is just to ensure that any update for an application originates from the developer of the original application.

Root Access. All of today's smartphones unite the fact that they have a protected memory area. Only the vendor is allowed to modify code in that area. This ensures that the OS cannot be compromised.

Kill Switch. In iOS, Windows Phone 7 and Android the smartphone vendors have the ability to delete applications from the phone remotely, when they show a bad behavior. It can be assumed that also BlackBerry has such a feature, even if it is not officially confirmed by RIM.

Application Installation Model. Barrera et al. [10] distinguish three different application installation models: *The walled-garden model*, *the guardian model* and *the user control model*. The walled-garden model gives the smartphone vendor full control over any application on the device. The Vendor alone decides whether an app is allowed to run or not. Even though the user is released of a lot of responsibility thereby, this approach is heavily discussed because of the omnipotence of the vendor. Apples iOS is a perfect example for this model. In the user control model the vendor passes all the control over the software on the system (and therefore all the responsibility) to the user. The user is not restricted in choosing his or her software. However, it is crucial that the user is aware of possible consequences. Google relies with Android on this model. The so called guardian model—which is the model of choice of both Windows Phone 7 as well as BlackBerry—is in between the walled-garden and the user control model, sharing the control over the applications evenly between the user and the smartphone vendor.

Controlled Marketplace. Even though almost every smartphone vendor provides a marketplace for third-party applications, the handling with it is extremely divergent: Before an application is available in the App Store Apple checks the code of it. If a violation of the user policies or malicious code is detected, the application will be rejected. Since the App Store is the only source for new applications, this also means a prohibition of it. Microsoft pursues with their marketplace called Zune a similar strategy. RIM and Google indeed provide their own shops—the BlackBerry

Table II
ANDROID-BASED PRIVACY APPROACHES IN COMPARISON

	Focus	Granularity	Extendable	Contextual Constraints	Configuration Time	Crash Safe	Data Privacy	Feed-back	Realization
Android Permissions	hardware	coarse	yes	no	installation time	yes	no	no	kernel
Apex	hardware	coarse	no	yes	run-time	no	no	no	kernel
AppFence	hardware	fine	no	no	run-time	yes	mock	no	kernel
Dr. Android & Mr. Hide	application	coarse	no	no	convert time	yes	no	no	application
MockDroid	hardware	coarse	no	no	run-time	yes	mock	yes	kernel
YAASE	application	fine	no	no	run-time	no	no	no	kernel
CyanogenMod	hardware	coarse	no	no	run-time	no	no	no	kernel
PDroid	hardware	fine	no	no	run-time	yes	mock & custom	no	kernel
Privacy Protector	application	fine	no	no	run-time	no	no	no	application
PMP	application & hardware	coarse & fine	yes	yes	run-time	yes	mock & blur	yes	application

App World respectively the Google Play Store—but applications for BlackBerry or Android devices can be purchased from any source. The applications are not reviewed before their release. However, they will be removed from the store when a malicious behavior is reported.

Table I summarizes the most prominent characteristics concerning privacy issues of the smartphone vendors considered in this paper.

III. CONCEPTS AND APPROACHES TO PRIVACY MANAGEMENT FOR ANDROID

Looking at the sales figures, it is remarkable that with iOS and Android, both the most monitored, and the most open OS lead the popularity charts [11]. Even though developers like Android devices because of their high degree of freedom [2], the user acceptance decreases slowly inter alia because of the uncertain privacy concept [12]. The initial goal of our Google cooperation is to come up with a review of existing approaches to privacy management in a mobile setting with a special focus on Android. In the following we take a closer look at existing approaches:

Especially for Android, there are a variety of research approaches to protect privacy. Hereinafter, we reflect on *Apex*, *Dr. Android & Mr. Hide*, *MockDroid* and *YAASE*. However, not only in research attention has been paid to the issue of privacy on mobile devices. Also the Android community regularly introduces new solutions on how to improve security of sensitive data. Hence, three representatives of this type will also be presented in this work: *CyanogenMod*, *PDroid Privacy Protection* and *Privacy Protector*.

Apex. Nauman et al. [13] analysed the current permissions an Android user can grant an application in order to identify the most critical ones. Based on these observations they introduced a powerful policy model and extended the Android OS by a mechanism to apply these policies. This model has three major advantages: Firstly, the user even can install an application, although if only a subset of the requested permissions is granted. Secondly, however, at run-time permissions can be added, modified, or removed. Certainly the biggest improvement is undeniably that the user can define various constraints under which certain functions may not be executed. However, the biggest drawback of this approach—apart from the fact that a user needs a device with root rights—is that applications might crash if they try to access data though the user has withdrawn the therefore required permissions.

AppFence. Hornyack et al. [14] address two problems with AppFence: Uncontrolled data access and information sharing. Therefore, they enable the user to shadow private data for certain applications by providing mocked data and furthermore restrict network transmissions of classified information. However, the information flow tracking system used for the exfiltration blocking feature can be bypassed by an app.

Dr. Android and Mr. Hide. Jeon et al. [15] introduce with Dr. Android and Mr. Hide a unique approach to increase privacy without modifying the OS. They developed an Android background service (Mr. Hide) which has full access to any data and provides several interfaces to share this data with authorized applications. To authorize an application a fine granular application-centric permission system is available. For any application there is a converter (Dr. Android—an acronym for Dalvic Rewriter for Android) which replaces all critical API calls in an application with requests to Mr. Hide. However, the user gets no feedback to which extent the functionality of the application is affected by his or her permission settings.

MockDroid. Beresford et al. [16] came to the conclusion that the mere access to critical data is not too bad, as long as the user can control which applications get the correct data back. To this end, they modified the OS so that a

user can select at run-time some applications that henceforth receive mock data by certain APIs—e.g. faked location data. Thereby some applications become useless, of course (e.g. a navigation application with mocked location data).

YAASE. Russello et al. [17] pursue two main targets with YAASE: On the one hand they provide a way to formulate extremely fine-grained access control policies whose generation does not overburden the user since s/he would not use it otherwise. On the other hand YAASE addresses the problem of permission spreading between several applications—e.g. when they run in the same process. Even if this indeed can stop the direct exchange of permissions, it does not prevent any application from sharing the critical data itself.

CyanogenMod. extends the original Android OS by some features. Our interest in this paper rests on the application permissions management, solely. CyanogenMod allows an user to revoke any permission s/he deems unnecessary for any application even a posteriori after installation. Then the affected application has no longer this certain permission. If it still tries to accesses a now prohibited function, it crashes in nine out of ten times. **PDroid Privacy Protection** deals with this problem: Instead of blocking an application from accessing vulnerable data PDroid supplies the application with custom or random values. While both approaches require root rights, **Privacy Protector** does not need them. Privacy Protector simply turns off locating and disconnects any network connections when certain applications are running. Thereupon all applications are affected by these restrictions.

Table II summarizes the most prominent features of these approaches. They include, whether the privacy settings focus on applications or hardware features, how precisely the rules can be expressed, whether the policy model is expandable, whether context constraints are possible, when the rules can be modified, whether the method is crash safe, how private data can be disguised, whether the user is informed about the impact of the settings on an application and finally how the approach is implemented. As a result of this review (cf. Table II, bottom row) we extended the goal of our Google cooperation to come up with a more comprehensive and effective approach. This will be described in the subsequent sections.

IV. An Approach to Customizable Privacy Management

Unfortunately, none of these proposals convinced us entirely, as each of them reveals more or less grave shortcomings. Therefore, we introduce an approach to customizable privacy management that can be characterized as an extendable, fine-grained, interactive, context sensitive and fail safe mechanism to enable the user to become master of his or her private data.

Even though many developers fear the restrictions on their applications (e.g. disabling in-app advertising [18]), it also cannot be in their interest that they have to publish the same

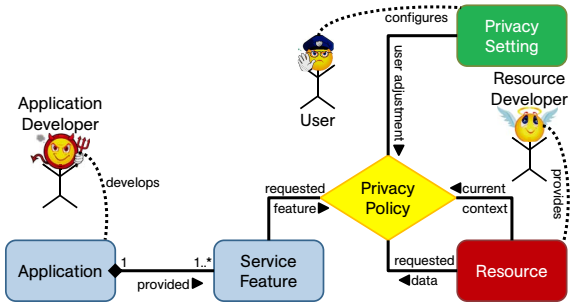


Figure 2. PMP's Privacy Policy Model

applications several times with different permissions as it is already practiced sometimes. For example an application in an ad-free and two ad-supported versions—once location-based and once general advertising.

The basic idea of the approach that we call PMP (**P**rivacy **M**anagement **P**latform) will be described in Section IV-A and its privacy model in Section IV-B in particular.

A. Overall Picture

Obviously, current mobile applications such as the introductory example vHike require highly sensitive personal data, such as position, contact data, etc. Naturally, the user does not want to share this data with everybody in every context. Albeit, without any private data at all it is often impossible for an application to provide any reasonable service. It is essential to find a compromise between privacy and service level. Thus, a central aspect of the PMP is to enable applications offering various Service Features that build upon user acceptable permission sets.

To achieve this, an application has to specify various Service Features with mandatory private data requirements. PMP informs the user about the available Privacy Settings and their consequences for the scope of services.

The most outstanding features of PMP are its *extendibility*—in terms of adding user generated new Resources in arbitrary granularity—its *context sensitivity*—as permissions can be granted or revoked according to the current situation—and its *user interaction*—as a user is always informed about the impact of any decision both on privacy as well as on application's service.

B. PMP's Privacy Policy Model

The approach of the PMP is based on a model that comprises applications (respectively their Service Features), private data, and Privacy Settings. Private data can originate from various data providers referred to as Resources. As shown in Figure 2, Service Features, Resources, and Privacy Settings are highly interrelated and must be able to interoperate with each other. The Privacy Policy holds all needed information for granting access to private data: It basically interrelates Resources (private data sources)—provided or

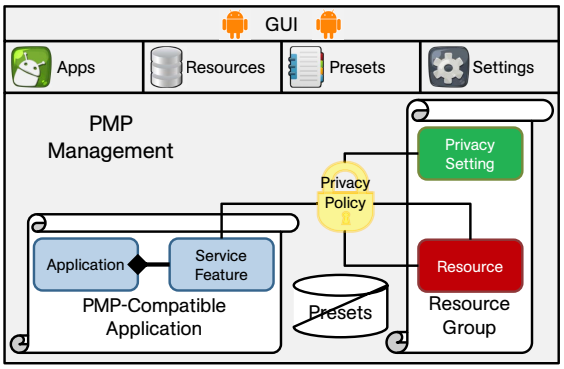


Figure 3. PMP's Components Overview

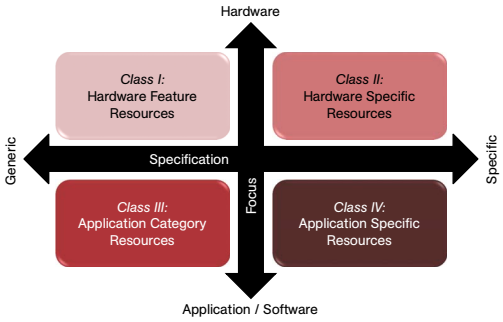


Figure 4. Resource Classes

audited by trustworthy organizations—required for a certain Service Feature (as a part of an application)—distributed by potentially malicious developers—with Privacy Settings adjusted by the attentive user. In accordance to the closed world assumption, an access is denied if a certain interrelation is not represented. Furthermore, the PMP hereby is able to consider the user's current context, also.

To pick up the vHike example introduced in Section I, the navigation feature that directs the driver to a hiker requires position data from both of them. However, it provides two different Service Features: GPS navigation or static map. The user can adjust the Privacy Settings so that access to the exact position is granted if the hiker is within a few kilometers (which enables navigation). Otherwise, an approximation of the current position is shared and vHike shows a map of the hiker's current surrounding, only.

A privacy management system is of no use if it is too complicated for the user to handle. Thus, the Privacy Settings are designed such that despite expressive power an intuitive user interface is feasible. Moreover, as PMP supports Presets as a kind of default, the user is not bothered with excessive access permission requests.

V. PMP Architecture

In this section we introduce the six main components of the PMP: *Resources*, *Privacy Settings*, *PMP-compatible Applications*, *Service Features*, *PMP Management* and *Presets*. Figure 3 illustrates the correlation and interaction between these components. The components implement the Privacy Policy model introduced in the previous chapter and visualized in Figure 2. Resources, pooled in Resource Groups, provide access to private and system data of the device. A user controls and restricts its usage via the Privacy Settings. A PMP-compatible Application asks the Resource Groups for their data and the PMP Management component decides whether this request is legitimate. Depending on the number of Resources an application gets access to, it provides a variety of Service Features. As not every user is able or willing to configure the Privacy Policy appropriately, Presets can be established, exported, shared and imported.

Every depicted component is described in more detail in the subsequent paragraphs. The major implementation aspects of these components are discussed in Section VI.

Resources. Resources can be seen as information broker between the applications and the Android system: Each Resource obtains a certain kind of data from the system and provides interfaces for the applications to access some of these data. Resources are registered at the PMP. Thus, PMP can manage the access to the data by managing access to its Resources.

One big advantage that comes with the Resource model compared to the original Android content provider respectively data manager concept, is its extensibility. This means that Resources can be added or updated whenever it is necessary. Moreover, s/he can pool content-related Resources within a so called *Resource Group*.

Whenever the PMP Management detects that an application uses a Resource which is currently not installed on the device or if an installed Resource is outdated it acquires them from a central Resource repository. Those new Resources are automatically integrated into the PMP system.

Since this Resource approach is a very generic one, a Resource can either be modeled hardware-centric or application-centric. Furthermore, the Resource developer decides whether a Resource should be generic or rather specific. Based on these two dimensions we divide four different Resource types as shown in Figure 4. We illustrate this based on vHike: A class I Resource might be *Location*, which can be used by any application to locate the user—however, it is masked which technology is applied. On the contrary a class II Resource—let us assume a Resource such as *GPS_Location*—constitutes a certain technology explicitly. This Resource is also available to all applications, but it uses the GPS for locating solely. A third Resource *GeoNavigation* (class III) provides location data to any application, but restricted to the Service Feature *Navigation* exclusively for navigation services. The final Resource type (class IV) cuts down the number of applications which are

supported by this Resource—e.g. *vHikeLocation* makes the geolocation available for vHike, only.

It is striking that both class I and class II are indeed covered by the Android permissions up to a certain point—*ACCESS_COARSE_LOCATION* is a textbook example for a class I Resource, as it encapsulates multiple location techniques, while *CAMERA* represents a class II type, because it explicitly states the used technology. Dr. Android and Mr. Hide is able to deal with class III Resources. Its *AdsGeo* permission enables any application to display ads and forwards the current location to the advertisers. Rules to control class IV Resources can be defined by the very fine-grained policy language which is used in YAASE. However, it is remarkable, that PMP is the only system supporting the full spectrum of Resource types, which is a prerequisite for comprehensive privacy control.

Privacy Settings. Each Resource may define a certain number of so-called Privacy Settings. A Privacy Setting is permanently linked to a Resource Group and consists mainly of an unique identifier, a significant name, its description for the user and a value (i.e. the actual setting). A Privacy Setting for the Resource `Location` could be `useFineLocation`. Possible values for this setting are `true` and `false`. Therefore, the Privacy Settings can be most likely compared to the Android permissions. However, this approach provides two crucial advantages:

On the one hand, as the Privacy Settings are linked to individual Resources and not to an application, more precise configurations are possible. E.g. you could grant the use of location data for the *Navigation* Resource in vHike and prohibit its use for the *Observation* Resource. This example illustrates that it makes sense to decouple the Privacy Settings from physical hardware features and relate them to the logical Resource Groups.

On the other hand the Privacy Settings are more powerful than the Android permissions. While the latter can be granted or removed, only, a Privacy Setting accepts any data type as a value. Besides Boolean values, which corresponds to the behavior of the original permissions, also numerical or textual values are accepted. This makes it feasible to define e.g. a rule that vHike is allowed to use location data via the Resource `Location` up to an accuracy of `100` meters.

PMP-compatible Applications. The development of a PMP-compatible Application differs only marginally from the conventional application development process. The implementation of the application logic is not altered, except that the developer should use the PMP Resources to access needed data, instead of the Android content providers. Additionally, if the permission entries have been removed from the Android Manifest, the PMP cannot be bypassed.

However, as the PMP attempts to inform the user profound, some additional details about the application are required. For this purpose we introduced the so called *App Information Set*. Alongside with the application's name a multilingual description of the usage of the application is specified therein. In an analogous manner to the presentation in the Google Play Store this information is displayed when an application is installed. Thus, this overhead for a developer keeps within reasonable limits.

Service Features. The last component a developer has to worry about are the Service Features. Since PMP-compatible Applications can provide a variable range of functions, depending on the Privacy Policy, a developer has to determine which feature requires access to which Resources respectively how the Privacy Settings have to be set. These statements are stored in the App Information Set as well. Apart from the references to the used Resources an unique identifier for the Service Feature, its name and a description what functions are provided by it can be specified there.

A component like the Service Features is not provided in the original Android concept. In order to imitate such a behavior, a developer will have to implement many different versions of the same application. However, an application with n Service Features leads to 2^n variants already. The maintenance for this sheer volume of applications cannot be handled efficiently anymore. Not to mention the resulting irritations of the user when the search for a certain application yields a high number of similar findings.

Presets. The configuration of a PMP-compatible Application (or more precisely the selection of activated respectively deactivated Service Features) can be stored within a so called Preset. A Preset contains—beneath the accumulation of application identifiers and their appropriate Service Features—a name and a short description. As a matter of fact it is up to the user which settings should be included in a Preset. Apart from the possibility to store these settings locally, the Presets can also be deposited on a web server and thus made available for other users. One could imagine, that trusted organizations provide certified Presets as a recommendation of how to ensure a maximum of privacy—e.g. the German Federal Office for Information Security Technology (BSI) offers something similar for the configuration of web browsers in terms of Internet security. Another use-case for Presets could be to ensure compliance with company policies—for instance, the deactivation of certain safety-critical functions during working hours.

Since several different Presets can be imported, the PMP might have to deal with conflicting Presets. A conflict may occur, when two or more different Presets exist for the same application. If the settings do not contradict each other, then all Presets are combined additively. However, if there is a conflict, the afflicted Presets are highlighted immediately after the import. Then, based on this information, the user can decide which Service Features he wants to retain in order to resolve the conflict.

PMP Management. The PMP Management enforces the Privacy Policy and represents the interface towards the user:

Table III
EXCERPT OF A PRIVACY POLICY FOR VHIKE

Service Feature	Resource	Privacy Setting	Constraint
vHikeContacts	Contacts	usePhone = true	—
		sendSMS = true	—
vHikeFineLoc	Location	useFineLoc = true	$cond_1$
vHikeAprxLoc	Location	locPrecision = 100	$cond_2$

On the one hand, from system startup the PMP Management component monitors whether a PMP-compatible Application is either installed or running. Thus, new PMP-compatible Application or Resources will be automatically registered. When a PMP-compatible Application is started, the PMP verifies at run-time that all activated Service Features are covered by the Privacy Policy. For this, the PMP Management checks whether for the given Service Feature and the requested Resource the Privacy Settings (and optional constraints) are valid (as indicated in Table III). If that's not the case, the user is immediately informed.

On the other hand, the PMP Management component provides a graphical user interface for administrating the PMP-compatible Applications, the Resources and the Presets. The main menu is shown in Figure 1b. The four submenus are detailed in the following. The tab labeled with *Apps* provides an overview of all PMP-compatible Applications currently installed on the device. Thereby these applications can be started directly via the PMP Management. Additionally, the user gets further information about the applications (e.g. a brief description) and a summary of all Service Features offered by the application. Here, s/he sees immediately which of them are activated, respectively deactivated and s/he can modify the settings as needed. Figure 1c shows some exemplary Service Features for a calendar application—e.g. for the creation, export / import, or distribution of calendar entries. When a Service Feature is disabled because of missing Resources, they can be downloaded. The submenu *Resources* provides an overview of all installed respectively all currently available Resources. Here one can also install, update, or remove them independently of any application. As the name already suggests, from the menu *Presets* all Presets can be managed—e.g. create, import or export. Besides mere administration tasks the user can also modify all permission settings at run-time, in order to make more private data available, feed some applications with mocked data or cut down existing permissions. For a Resource *Filesystem* a Privacy Setting could be read or write permissions for a certain directory, as shown in Figure 1d. Finally, in the *Settings* menu, one can switch between simple and expert mode or activate some logging functionalities (e.g. a summary of all Resource requests).

VI. PMP REALIZATION STRATEGIES

Essentially, there are two distinct implementation strategies (Figure 5): Either PMP is implemented as an ordinary application (left side of Figure 5) or it is integrated into the OS itself (right side of Figure 5). We chose the former option, however, also the latter would have been easily possible, as the Resource Groups are very similar to the Android Managers in the Application Framework layer whereby a merger would be feasible. Both approaches have pros and cons:

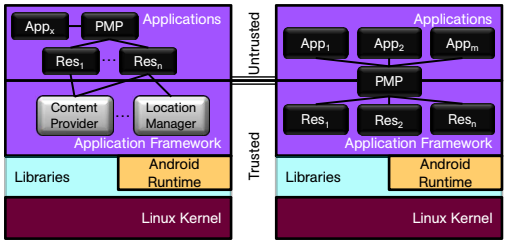


Figure 5. Realization Strategies within the Android Stack

An implementation on top of Android provides two significant benefits: On the one hand this implementation strategy does not require any modification to the Android OS making PMP usable on every Android device even without root rights. On the other hand all current applications continue to work properly as they are not affected by the new privacy mechanism. This, however, represents the biggest disadvantage at the same time. Since an application is not forced to request all private data via the PMP, it can also get access to this data directly using the Android Application Framework. Though, the user has to grant the application the required permissions initially. Implementing the PMP within the Android OS—so it is the only interface towards all applications—closes this, albeit small, vulnerability concerning privacy and such an implementation is more efficient. However, since a PMP-compatible Application requires some additional code, conventional Android applications have to be adapted in order to work with the PMP. Here an APK repackaging comparable to the one used in Aurasium [19] might help.

VII. ASSESSMENT

Looking back at the requirements postulated in Table II, PMP satisfies in all categories: Thanks to the flexible Resource model, PMP supports all four Resource classes and therewith is able to cope with either application-centric or hardware-centric Resources of an arbitrary granularity. Moreover, further Resources can be added when necessary. Besides, in PMP a user can define in the Service Feature settings, that a Resource should only provide mocked data for a certain application. The Service Feature concept ensures, that an application does not crash if some required permissions are missing, as each feature is activated respective deactivated accordingly. The PMP Management takes care of this and gives feedback to the user how s/he can limit or extend the functionality of the application by

adjusting the Privacy Settings. Finally, the Privacy Policy model enables to add (contextual) constraints to each policy rule (cf. Table III). A rule can be generated, removed, or modified at run-time.

We have implemented three PMP-compatible Applications: In addition to vHike, we implemented a privacy-aware calender application (CalendarApp) and an application called InfoApp, which lists various stats about your device. Thus we were able to gain early experiences with the PMP. Initial tests showed that the concept of Service Features is very useful, intuitive, and means almost no overhead for a **user**. An **application developer** has to define Service Features, which leads to an overhead compared to a plain Android application. The **Resource developer's** effort depends on the capacity of the Resource—e.g. pure forwarding of the content providers' data generates only a slight overhead. Moreover, the PMP Eclipse plugin assists developers with these duties a lot.

Admittedly, there is an overhead concerning the run-time of an application. Though it became apparent in benchmarks of similar approaches that the performance overhead of mere policy verification can be ignored (e.g. [20]). Of course, this applies only if a Resource simply passes values provided by an Android Manager. This changes significantly when values need to be accumulated or calculated within the Resource—e.g. to approximate a geolocation—depending on the complexity of these calculations.

Since PMP as well as all the sample applications are hosted on Google Code do not hesitate to test it yourself:

http://code.google.com/p/pmp-android.

VIII. Conclusion and Future Work

The review performed in our Google cooperation project and presented here clearly shows that there are various and diverse approaches to privacy management for mobile platforms available. However, a truly comprehensive and satisfactory one is currently not provided. Therefore we chose to develop the PMP as a customizable, fine-grained, context-based and extendable privacy management platform for mobile devices. PMP can be characterized to apply and augment useful features from existing approaches and to add functionalities for missing features.

Acknowledgment

This approach resulted from a close collaboration with Google Munich office. Not only in the planing phase we received useful advices and ideas, but we were given support with the implementation, also. Hence, the authors would like to thank Google for their support of our work and their helping suggestions for improvements.

References

[1] International Data Corporation (IDC), "Android Expected to Reach Its Peak This Year," Press Release, June 2012. [Online]. Available: http://goo.gl/H1uKy

[2] G. J. Spriensma, "The Need for Cross App Store Publishing and the Best Strategies to Pursue," Distimo, Tech. Rep., 2012.

[3] N. Bilton, "Apple Loophole Gives Developers Access to Photos," The New York Times, February 2012. [Online]. Available: http://goo.gl/qnVnC

[4] B. X. Chen and N. Bilton, "Et Tu, Google? Android Apps Can Also Secretly Copy Photos," The New York Times, March 2012. [Online]. Available: http://goo.gl/5e7fq

[5] C. Stach, "Saving time, money and the environment - vHike a dynamic ride-sharing service for mobile devices," in *PERCOM '11*, 2011.

[6] C. Stach and A. Brodt, "vHike - A Dynamic Ride-Sharing Service for Smartphones," in *MDM '11*, 2011.

[7] D. Wetherall *et al.*, "Privacy Revelations for Web and Mobile Apps," in *HotOS '11*, 2011.

[8] A. P. Felt *et al.*, "Android Permissions Demystified," in *CCS '11*, 2011.

[9] W. Enck *et al.*, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," in *OSDI '10*, 2010.

[10] D. Barrera and P. van Oorschot, "Secure Software Installation on Smartphones," *IEEE Security and Privacy*, vol. 9, pp. 42–48, 2011.

[11] Gartner, "Gartner Says Worldwide Sales of Mobile Phones Declined 2 Percent in First Quarter of 2012," Press Release, May 2012. [Online]. Available: http://goo.gl/0FnBQ

[12] M. King and S. Ellison, "Q2 2012 Mobile Developer Report," Appcelerator and IDC, Tech. Rep., 2012.

[13] M. Nauman *et al.*, "Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints," in *ASIACCS '10*, 2010.

[14] P. Hornyack *et al.*, ""These Aren't the Droids You're Looking For": Retrofitting Android to Protect Data from Imperious Applications," in *CCS '11*, 2011.

[15] J. Jeon *et al.*, "Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications," in *SPSM '12*, 2012.

[16] A. R. Beresford *et al.*, "MockDroid: trading privacy for application functionality on smartphones," in *HotMobile '11*, 2011.

[17] G. Russello *et al.*, "YAASE: Yet Another Android Security Extension," in *PASSAT '11*, 2011.

[18] I. Leontiadis *et al.*, "Don't kill my ads! Balancing Privacy in an Ad-Supported Mobile Application Market," in *HotMobile '12*, 2012.

[19] R. Xu *et al.*, "Aurasium: Practical Policy Enforcement for Android Applications," in *USENIX Security '12*, 2012.

[20] P. Kodeswaran *et al.*, "Securing Enterprise Data on Smartphones using Run Time Information Flow Control," in *MDM '12*, 2012.