Design and Implementation of the Privacy Management Platform

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*—Nowadays, mobile platform vendors have to concern themselves increasingly about how to protect their users' privacy. As Google is less restrictive than their competitors regarding their terms of use for app developers, it is hardly surprising that malware spreads even in Google Play. To make matters worse, in Android every user is responsible for his or her private data and s/he is frequently overwhelmed with this burden because of the fragile Android permission mechanism. Thus, the calls for a customizable, fine-grained, context-based, crash-proof, and intuitive privacy management system are growing louder. To cope with these requests, we introduce the Privacy Management Platform (*PMP*) and we discuss three alternative implementation strategies for such a system.**

***Index Terms*—Android; policy model; implementation strategies.**

I. Introduction

Mark Weiser's vision of the next generation of computers [1] came true with the introduction of the first smartphones in the late nineties. Whereas the user has hitherto been happy to be *"always-on-call"*—i. e. s/he could be contacted by phone anywhere at anytime—smartphones caused a demand to be *"always-on-line"* in terms of having a constant connection to other people via the Internet. Additionally, these devices possess enhanced sensor techniques (e. g., GPS). With a smart device, a user manages his or her contacts in an address book, arranges appointments via a synchronized calendar, writes (and stores) private as well as business mails, takes pictures, and so forth. To facilitate these activities, users avail themselves of the extensive offerings in *App Stores*. However, the ability of unlimited data transmission, the access to situation information as well as private data, and the possibility to use apps of unknown third-parties is a severely dangerous combination. There is a lot of evidence for inadequate usage of private user data throughout various mobile platforms [2]. Therefore, each major platform vendor pursues a specific strategy in order to preserve privacy: Apple and other vendors give their users absolutely no control over their data, but determine which apps are benign and thus can be installed on an iOS device. Google's Android OS chose an entirely different strategy and put the end-user in total control over his or her private data—and transferred all responsibility over this data to the user.

† This work was supported by a Google Research Award.

However, also Google's privacy mechanism does not meet with the users' needs, since it lacks of a customizable, fine-grained, context-based, crash-proof, and intuitive privacy policy model. Therefore, we introduced the **P**rivacy **M**anagement **P**latform (*PMP*) in our earlier work [3]. This paper builds upon our prior approach, but clearly extends it w. r. t. details on design decisions and implementation alternatives.

The remainder of the paper is structured as follows: In Section II, we consider systems enhancing the Android privacy system. Then, we introduce our privacy policy model in Section III: We give a brief overview of some design issues and highlight its main components. Then, we discuss three alternative implementation strategies for such a privacy policy model for Android OS in Section IV. Finally, Section V comes up with a short summary of this paper.

II. Related Work

A lot of research work is done in the area of privacy threats on Android devices. Basically these approaches can be divided into three categories: App-based solutions (e. g., Privacy Protector), approaches integrated into the OS (e. g., CRêPE), and systems using an app converter (e. g., AppGuard). *Privacy Protector* considers location data and network access as privacy-critical factors, only. The user can control a specific app's usage of these two functions. Once Privacy Protector detects that this very app is running, GPS or network access is disabled system-wide—so any app is affected by this. *CRêPE* [4] adds context references to its privacy policy: A user can describe situations in which a certain policy rule should be applied, respectively when they are invalid. Each policy rule can be defined in a very fine-grained manner and they can be changed even at run-time. However, CRêPE does neither support faked data nor does it provide the user with any feedback on his or her settings. Moreover, an app may crash if a required permission has been withdrawn. *AppGuard* [5] introduces a bytecode rewriting system that integrates a monitor component as well as a set of built-in privacy policies into an app. Hence, it puts the user back in charge of controlling the permissions of an app. S/he can decide both at installation time and at run-time what data an app is allowed to access. However, AppGuard's modus operandi is legally questionable (see Section IV-C). Unfortunately, non of the currently existing privacy management systems complies

with all the users' requests. For this reason, we introduced in our previous work the **P**rivacy **M**anagement **P**latform (*PMP*) [3]—a context-aware and crash-proof privacy management system where a user can define fine-grained permission rules at run-time and obfuscate his or her private data on demand. The user is always informed about what his or her settings mean for the affected app's scope of service.

III. The Privacy Policy Model

In this paper, we focus on refining the PMP's privacy policy model, its realization (Section III-B), and sound implementation strategies (Section IV).

A. The Privacy Policy Model of the PMP

Figure 1 shows the relationships among the key components of the privacy policy model in an UML-like notation:

Service Feature. A PMP-compatible app encapsulates its provided services in so-called Service Features. Permissions are not directly assigned to an app, but to its Service Features. The user can enable or disable any of these features—with the corresponding impact on the scope of service—and thus expand or respectively limit the permissions this app requires.

Resource. Resources represent interfaces to protected data. Depending on the granted permissions, a Resource provides an app either with unaltered data, aggregated data, randomized data, or no data at all. Resources can be added, updated, or removed at any time. The Resources are provided by a protected on-line repository—hereby, we can ensure that all Resources are free of any malware. This approach is pretty similar to Apple's modus operandi in their App Store. However, as the number of new Resources that have to be checked is vanishingly low compared to the number of new apps, also the overhead caused by these checks keeps within reasonable limits. Resources with related content are encapsulated in **Resource Groups**.

Privacy Setting. The Privacy Settings are configured by the user. For that, the Resources specify a valid range of values for each Privacy Setting. Accordingly, a Resource developer defines with the Privacy Settings how fine-grained a user can assign permissions to a Service Feature. Thus, e. g., for the *Location*-Resource, a user is able to prohibit the use of location data entirely, restrict it's accuracy up to x meters, randomize his or her location data, provide predefined mock data, or grant the unrestricted use of location data.

Privacy Rule. A Privacy Rule combines these three components: Such a rule consists of an actor (Service Feature) requesting access to private data (Resource) and the permissions that a user has granted for that actor (Privacy Setting). By additional contextual constraints—the context is also provided by the appropriate Resource—a rule can be further detailed e. g., if a rule should be applied in a certain place or at a certain time, only. In accordance to the closed world assumption, every request is denied when there is no rule for that particular action. The set of all Privacy Rules forms the so-called **Privacy Policy**.

An arbitrary set of Privacy Rules can be exported to a file, the so-called **Preset**. A Preset can be shared easily with any PMP user. Subsequently, they can be added to a Privacy Policy, replace an existing policy or create a new policy. These Presets significantly facilitate the configuration of the PMP as unexperienced users can fall back on initial settings recommended by trustworthy third-parties.

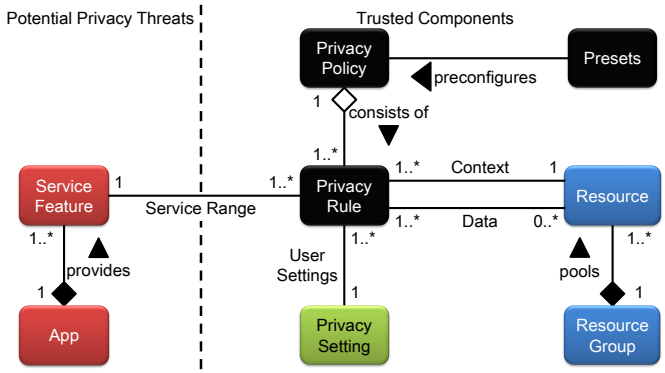


Figure 1. PMP's Privacy Policy Model (cf. [3])

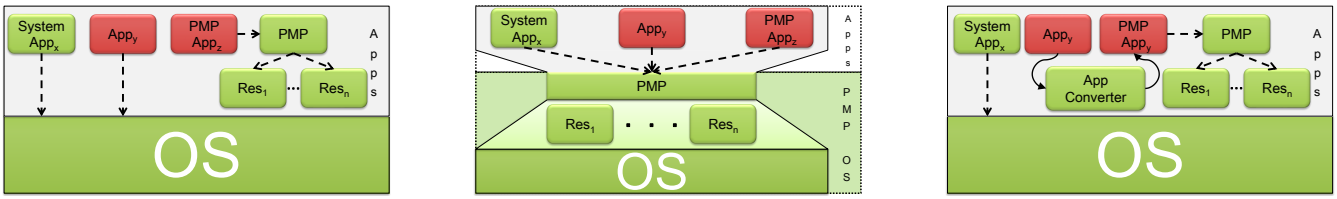
B. Realization Concept of the Privacy Policy Model

For the realization of the privacy policy model its components are partitioned into three separated software units: The *apps* including their Service Features, all Resources pooled in *Resource Groups*, and finally the *PMP* managing the user's Privacy Settings within the Privacy Policy.

When a user installs new apps on the system, a check is performed whether the PMP is up and running. Then, the app registers to the PMP and the user can specify an initial set of Privacy Rules; the user is able to reconfigure each Privacy Setting at any time. If additional Resources are required, the PMP automatically downloads and mounts them. Whenever a Service Feature requires access to protected data, the PMP checks whether there is already a Privacy Rule for this action. If that is not the case, the user is informed by the PMP and s/he can add rules to the policy. Elsewise, the PMP grants or denies the access to the corresponding Resource depending on the Privacy Policy. The Resource has access to protected data. It transfers this data (possibly altered according to the Privacy Settings) to the enquiring app.

A privacy management system is reasonable only if it is comprehensible and ergonomically designed. Therefore, the PMP is topped off with an intuitive management interface and an import function for preconfigured rules (referred to as *Presets*). Furthermore, the PMP has two different user modes: In the *simple mode*, laymen are not overwhelmed, whereas experts still can use the PMP's full range of services in the *expert mode*. For further information on the PMP please refer to our previous work [3].

As the privacy policy model's components are completely abstracted from its underlying system, it can easily be applied to any OS. For the PMP, we focused on Android, however. The most reasonable implementation strategies are discussed in the following section.



(a) On Top of the OS (b) Integrated into the OS (c) Usage of an App Converter

Figure 2. Implementation Strategies for a Privacy Management System

IV. Implementation Strategies

After presenting how the privacy policy model is realized in an OS-independent manner, it is necessary to clarify how these software fragments can be embedded into a specific OS, here the Android OS. Basically, there are three different approaches, as shown in Figure 2: The implementation on top of the OS (Section IV-A), the integration into the OS (Section IV-B), or the usage of an app converter in order to manipulate the bytecode of an app such that critical API calls are diverted to secured ones (Section IV-C). In Figure 2 trustworthy components are colored green whereas potentially dangerous apps are portrayed in red.

A. Implementation on Top of the OS

As a proof of concept, we implemented all components of the PMP as plain Android apps initially [6]. So, neither root privileges nor modifications to the Android platform are necessary—any Android user is able to give the PMP a trial by installing it from the project's website (http://goo.gl/1pwt4). The PMP runs as an Android service in the background and listens whether any new PMP Resource or any app compatible to the PMP is installed. Legacy apps—i. e. apps using the Android permission mechanism—are not affected by the PMP and still run without any problems.

However, this is the central issue of this implementation strategy: As the PMP is a plain Android app without any further privileges, it can only control apps which cooperate with the PMP. Whenever an app requests data directly from the OS (see Figure 2a), the PMP cannot prevent this access. To make matters worse, even apps seemingly cooperating with the PMP can access the private data via this backdoor! Thus, this approach is applicable as a first prototypical implementation, only. For a real privacy system, a different strategy has to be chosen. Looking at the various approaches discussed in Section II, apparently only two different strategies are appropriate for a privacy management system for Android: Either by enhancing the Android permission mechanism or by using an app converter to intercept any privacy-critical data access.

B. Integration into the OS

If the Android permission mechanism is replaced by an alternative privacy management system, it has to be ensured that every app does no longer access data via the old mechanism. Therefore, the new privacy management system is used as a bouncer, regulating the communication between the apps and the OS (see Figure 2b). This works only if an app collaborates with the new system. Hereto, adaptions to the app are necessary. This can either be done automatically, as discussed in Section IV-C, or by the app developers themselves. As it is hardly likely that all developers do so, it has to be clarified, how to deal with legacy apps.

A very basic approach is to manipulate the app's Manifest at installation time and remove any permission entry from it. Thus, it is ensured that an app no longer can use the old permission mechanism. In this way, many crashes will occur since required permissions are no longer available. Moreover, there are also laws prohibiting such an approach (see Section IV-C).

So, it is advisable to manipulate the authorization process instead: The `enforce`-method checks for any data request whether an app has all required permissions. Then, the data access is either granted or denied. A simplistic approach is to manipulate this method so that it denies any request. Thus, an app has to support the new privacy management system in order to get access to private data—please note that this affects system apps (i. e., apps shipped with the OS), also.

Therefore, we differentiate between system apps and third-party software. To achieve this differentiation, it is sufficient to consider the app's install location: System apps are located in the directory `/system/app`, while third-party software is installed in `/data/app`. So, the `enforce`-method has only to query an app's package information, including the file path of the apk-file. Depending on this path, `enforce` passes requests from system apps to the *ActivityManagerService* as usual. Requests from third-party software can be rejected per se. So, all system apps still work unrestrained, while third-party apps can access their data safeguarded via the new privacy management system, only.

However, this approach has also a disadvantage concerning the system performance. The additional checking of the install location creates a performance overhead, which seems at a first glance to be negligible. However, the Android system calls the `enforce`-method in the background with very high frequency. Nevertheless, early evaluations show that this performance overhead keeps within reasonable limits.

C. App Converter

So, the major problem for a new privacy management system is how to deal with legacy apps. Depending on the implementation strategy, this means that those apps either continue to use the old (insufficient) permission mechanism

or are no longer executable. From a user perspective, both alternatives are unacceptable. For this very reason some related work approaches apply a third implementation strategy: the so-called app converter (see Figure 2c).

An app converter proceeds as follows: First, it scans an app's bytecode for privacy-relevant system calls that require special permissions. Then it replaces the corresponding code fragments with equivalent requests towards the new privacy management system. The converter has to adapt the Manifest according to the new privacy management system, also. Thus no longer needed entries can be removed, additional application metadata can be added, or new permissions, activities, services, or receivers can be registered. Subsequently, the app has to be repacked and re-signed.

This strategy has many benefits: It provides an inviolable privacy protection, since it guarantees that any data access has to be made through the new privacy management system. Moreover, no modifications of the mobile platform itself are necessary. Finally, a privacy management system following this implementation strategy is fully compatible to any legacy app. Thereby, it is not only ensured that all legacy apps are still executable without any crashes, but all apps use the new privacy management system right after conversion.

However, all bytecode rewriting systems are immature [7]: Any app has to be converted before its first-time usage and after every update or else there is no additional protection. Besides, due to the bytecode manipulations and the associated re-signing process, an automatic update process is virtually impossible. So, severe security problems in an app cannot be fixed as updates do not reach the user. Also, bytecode manipulation is partly based on heuristics. However, an incorrect assumption could have serious security consequences.

Furthermore, this implementation strategy extends the range of permissions of an app inevitably due to its signature alternations. Android apps with an equal signature are allowed to share data and even code fragments among each other. Since after converting all apps are signed by the converter with the identical signature, this leads to security risks that are hardly comprehensible to the user.

Besides the massive security risks inevitably associated to this implementation strategy, there are also copyright laws that exclude such an approach: For example in Germany, by manipulating the bytecode of an app, the converter violates the copyrights of the respective developer. Even though a limited manipulation is allowed for private use, such serious interferences in the program flow concern also other laws.

Whereas from a user perspective the usage of an app converter is the allrounder among the implementation strategies, arising security threats, applicable law, and Google's terms and conditions of use prohibit us from applying it to realize our privacy policy model.

Hence, the optimal implementation strategy for the privacy policy model is the "*integration into the OS* strategy" with some crucial adaptations (e. g., an enhanced selection mechanism for trusted legacy apps or optimizations towards the runtime performance). Early evaluations show that such an implementation of the PMP produces an acceptable overhead concerning CPU and memory usage while still satisfy the users' expectations.

V. Conclusion

Mobile platform vendors have to deal with attacks against their customers' privacy. As Google chose to give their users the highest degree of autonomy, they delegated the responsibility for the protection of private data to the users—an almost unsolvable task with the current Android permission mechanism—a fact which even Google is aware of by now. Yet, the so-called *App Ops* feature introduced in Android 4.3, where the user has the opportunity to selectively grant and withdraw a specific permission, has been removed in Android 4.4.2 as it is incomprehensible for the user and a permission withdrawal results in a crash quite likely.

Since no currently existing privacy management system fulfills all user requirements, i. e. customizability, context-aware privacy rules, crash safety, or fine-grained operability, we introduce our approach to a **P**rivacy **M**anagement **P**latform (*PMP*) as described in [3]. The complementing design and implementation of the PMP is provided by this paper. Therefore, we characterize the design as well as realization decisions and assess various implementation alternatives. The validity of our approach is confirmed by early evaluation results.

References

[1] M. Weiser, "How computers will be used differently in the next twenty years," in *IEEE Symposium on Security and Privacy*, 1999.

[2] W. R. O'connor, *Mobile Device Security: Threats and Controls*. Nova Science Publishers, Inc., 2013.

[3] C. Stach and B. Mitschang, "Privacy Management for Mobile Platforms - A Review of Concepts and Approaches," in *MDM'13*, 2013.

[4] M. Conti, B. Crispo, E. Fernandes, and Y. Zhauniarovich, "CRêPE: A System for Enforcing Fine-Grained Context-Related Policies on Android," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 5, pp. 1426–1438, 2012.

[5] M. Backes, S. Gerling, C. Hammer, M. Maffei, and P. von Styp-Rekowsky, "AppGuard - Enforcing User Requirements on Android Apps," in *TACAS'13*, 2013.

[6] C. Stach, "How to Assure Privacy on Android Phones and Devices?" In *MDM'13*, 2013.

[7] H. Hao, V. Singh, and W. Du, "On the Effectiveness of API-Level Access Control Using Bytecode Rewriting in Android," in *ASIA CCS'13*, 2013.