Collaborative Privacy Management: Mobile Privacy Beyond Your Own Devices

Yao Guo, Lin Zhang, Xiangqun Chen
Key Laboratory of High-Confidence Software Technologies (Ministry of Education)
School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China
{yaoguo, zhanglin08, cherry}@sei.pku.edu.cn

ABSTRACT

As the development of mobile devices and applications, mobile privacy has become a very important issue. Current researches on mobile privacy mainly focus on potential leakages on a particular device. However, leakage of sensitive data on a mobile device not only violates the privacy of the phone (or data) owner, but also violates the privacy of many other people whose information are contained in the data directly or indirectly (they are called *data involvers*). To address such problems, we introduce a collaborative privacy management framework, which aims to provide fine-grained data privacy protection for both data owners and data involvers in a distributed manner. Based on individual privacy policies specified by each user, a collaborative privacy policy is generated and enforced on different devices automatically. As a proof-of-concept prototype, we implement the proposed framework on Android and demonstrate its applicability with two case studies.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*

Keywords

Privacy protection; mobile systems; Android; collaborative

1. INTRODUCTION

Mobile devices such as smartphones have become ubiquitous, while mobile applications (*apps* for short) are developed at a dramatic speed. The number of mobile apps from Apple App Store and Google Play (Market) have both exceeded one million from 2013.

Mobile apps bring us attractive functionalities, while introducing various new privacy threats [7]. Smartphones often carry sensitive data or private information, such as contacts, personal messages, locations, or even financial data. Most mobile apps request to access a subset of these sensitive information, in order to function correctly and provide various functionalities to smartphone users.

Mobile systems such as Android protect private information using the permission label mechanism. Every app applies for a list of permissions (during installation) to access the data in the system. For example, if an app needs to access the "fine" location of a smartphone device, it will request the ACCESS_FINE_LOCATION permission. The user will have to allow the permission while the app is installed. In newer versions of Android, the permissions of each app can also be re-assigned after installation.

However, the permission label system on Android is not adequate to prevent information leakage. Many researches have been focused on studying and improving data privacy on smartphones, including both static and dynamic techniques. Static techniques such as Kirin [3] report the existence of dangerous permission combinations by analyzing the Android Manifest files of Android apps. Similarly, another static technique PiOS [1] analyzes data flows in iOS applications to detect possible leaks of sensitive information from a certain mobile device to untrusted third parties.

Dynamic analysis, especially for the purpose of information flow analysis, has been extensively studied in recent years, such as TaintDroid[2], AppFence[4], TISSA[9]. These techniques first mark various sensitive information as taint sources, and then study their propagations among mobile apps and system services in order to identify potential leaks.

1.1 Privacy Beyond Your Own Devices

Current privacy protection approaches are mainly focused on the phone itself, including analyzing and protecting mobile apps, the Android OS and its middleware framework. However, in the mobile era, *protecting one's privacy is beyond protecting his/her own device*, because many private information about one person could be stored and accessed on many other devices owned by different people.

One of our observations on smartphones is that it carries sensitive data not only from the phone owners, but also data from many other people. For example, in your contact list, you can find phone numbers, email addresses of many other people. In your photo album, there are many photos of other people you took with your phone. The potential leakage of these information might not only violate your own privacy, but also the privacy of many other people (here we call them *data involvers*, in comparison to *data owners*).

1.2 A Motivating Example

As shown in Figure 1, Alice's phone has applied strong security protection techniques so that the phone number

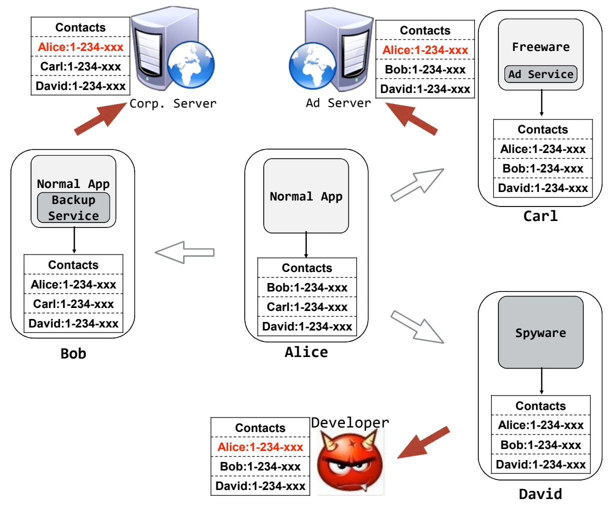


Figure 1: A motivating example: privacy leakages through Friends

information on her phone can not be acquired by third parties, such as malicious developers, Ad-lib providers and so on. Unfortunately, her phone number is not only located within Alice's phone, but also distributed among many phones of her friends. As a result, potential privacy risks could take place due to various reasons:

- Bob backs up his messages to a (insecure) cloud for archiving purpose;
- Carl grants an ad-supported app permissions to access messages;
- David installs a spyware (unintentionally), which collects all kinds of personal information, including phone numbers in the contact list.

We use *leakages through friends* to describe privacy leakages similar to the above scenario.

1.3 Key Ideas

We argue that in order to enhance the privacy of mobile applications on smartphones, all people involved should work collaboratively. Everyone has a responsibility to apply security measures to protect your phone and also honor other people's privacy requirements if their data reside on your phone, because your carelessness might involuntarily violate other people's privacy. Only if you reach out to protect other people's information on your phone, it will be possible to protect your information on other people's phones.

In order to solve the problem, we introduce the concept of *Collaborative Privacy Management*, which was first proposed and studied in on-line social networks [6, 5]. Our key idea is moving from a *phone-centric mechanism* (i.e., everyone protecting their own phones) to a *collaborative data-centric mechanism* (i.e., protecting specific data, no matter it is on which phone). As in the earlier example, besides Alice, her friends (Bob, Carl, David, etc.) should act collaboratively to (1) protect their own phones from data leakage; (2) honor all other data involvers' specified privacy requirements if it is different from one's own settings.

Our *assumption* behind the collaborative technique is that all smartphone users are good-will users who are willing to help to satisfy the privacy requirements of all involvers of the data on their phones. We believe this assumption is reasonable since every user has the incentive to honor other users' privacy requirements, because otherwise his/her own privacy cannot be protected. The technique is intended to prevent unintended privacy leakage, instead of malicious leakages, which belongs to a different research area.

In the proposed collaborative method, every user could specify their own privacy policy on their data. The policies will be enforced distributively on every phone where the data is located. All phone owners must enforce the privacy policies specified on the data on their phone. In this way, the privacy of all users involved will be protected.

1.4 Solution Overview

In this paper, we design an XML-style policy template for mobile users to describe their privacy requirements: what kinds of sensitive data are accessible or inaccessible for a given app or a specific app category. The policies of each user will be collected and analyzed, in order to generate the corresponding collaborative policy, which is to be enforced on each device, for each user respectively.

We modify the Framework layer of the Android platform to provide a fine-grained mechanism for sensitive data access, which augments the Android permission system. The collaborative policies are then enforced to block the sensitive information accesses from specific apps, so that privacy requirements coming from data involvers can be satisfied on all devices containing the data.

We believe it is an overlooked problem on how to protect the privacy and security of other people's data residing on one's phone. This paper makes the following contribution:

- We identify the *data involver privacy* issue, in which we are concerned about the privacy of not only phone (data) owners, but also data involvers.
- We propose a fine-grained mechanism called *collaborative privacy protection*, in which every user can specify their own privacy policy, while the policies will be enforced collaboratively in a distributed manner.
- We design and implement CoDroid, a proof-of-concept prototype system for collaborative privacy management on Android, and demonstrated the effectiveness and applicability of the proposed techniques on two case studies: contact access and photo publishing.

2. CODROID DESIGN

This section describes the design of CoDroid, a collaborative privacy management mechanism for the Android operating system. The components in CoDroid include policy specification for privacy requirement specification, policy merging for collaborative privacy policy generation, and policy enforcement for fine-grained sensitive data access.

2.1 Overview

Figure 2 shows the system architecture of CoDroid, including a *Policy Server* and the modifications we made to the Android operating system.

The *Policy Server* is a centralized server connecting all mobile users together, storing necessary information of all

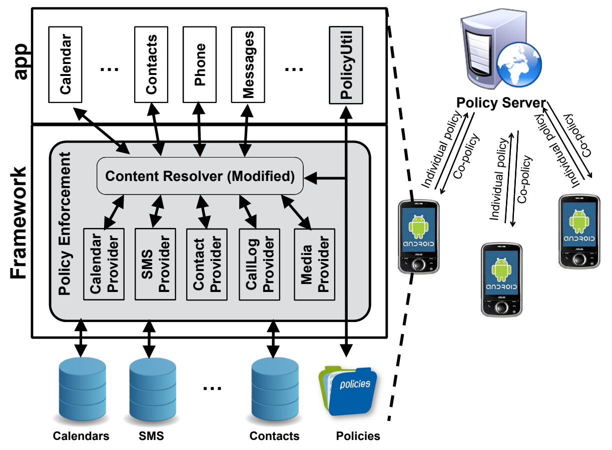
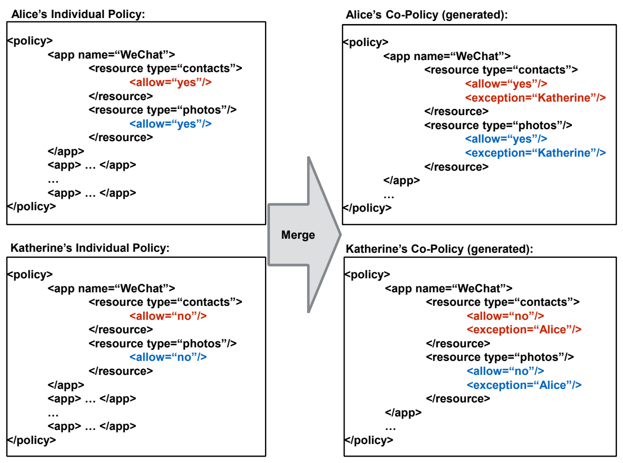


Figure 2: The Architecture of CoDroid.



Alice's Individual Policy:

```
<policy>
        <app name="WeChat">
                <resource type="contacts">
                        <allow="yes"/>
                </resource>
                <resource type="photos"/>
                        <allow="yes"/>
                </resource>
        </app>
        <app> … </app>
        …
        <app> … </app>
</policy>
```

Katherine's Individual Policy:

```
<policy>
        <app name="WeChat">
                <resource type="contacts">
                        <allow="no"/>
                </resource>
                <resource type="photos"/>
                        <allow="no"/>
                </resource>
        </app>
        <app> … </app>
        …
        <app> … </app>
</policy>
```

Alice's Co-Policy (generated):

```
<policy>
        <app name="WeChat">
                <resource type="contacts">
                        <allow="yes"/>
                        <exception="Katherine"/>
                </resource>
                <resource type="photos"/>
                        <allow="yes"/>
                        <exception="Katherine"/>
                </resource>
        </app>
        …
</policy>
```

Katherine's Co-Policy (generated):

```
<policy>
        <app name="WeChat">
                <resource type="contacts">
                        <allow="no"/>
                        <exception="Alice"/>
                </resource>
                <resource type="photos"/>
                        <allow="no"/>
                        <exception="Alice"/>
                </resource>
        </app>
        …
</policy>
```

Figure 3: An example of individual privacy policies and co-policies after merging.

devices and individual privacy policies specified by each user. Each mobile user registers an account on the Policy Server, and establishes social relationships with others through it. The server also includes the information describing what apps or what kinds of apps are installed on each mobile device. According to these information, the Policy Server could generate a policy template for each registered user.

Mobile users express their privacy requirements through *individual privacy policies*, which specify what kinds of sensitive data can be accessed by a given app or a specific app category. Each policy can be created based on policy templates generated by the policy server. The template can be edited using text editors or specialized visual editors [1].

All individual policies provided by users are then uploaded to the Policy Server. Based on the individual policies of his/her friends, the Policy Server applies a policy merging process to generate a *collaborative policy (co-policy)* for each registered user. The corresponding co-policy is then downloaded to the respective device and enforced through a policy enforcement component, which is implemented with modifications to the Android Content Resolver.

Please note that an individual policy specifies the privacy requirements of each user, while a co-policy, which is generated based on individual policies, specifies the privacy requirements of all data involvers on one's device. For example, if Alice and Bob ask their phone numbers be protected from the Facebook app, then as their friends, Carl should honor such requirements, which will be represented by Carl's co-policy.

2.2 Policy Specification

In order to provide fine-grained access control capabilities, each user can specify his/her own individual policy requirement in an XML-style format shown in Figure 3. The policy is similar to a list of permission labels for all apps installed on one's phone, however, it specifies not only the privacy requirements for the individual's device, but also for all his/her data on other people's devices.

[1]We have implemented a policy editor PolicyUtil, which provides visualized support for policy editing, as well as functions such as policy configuration, policy uploading and downloading, as well as account registration/login and adding friends, etc.

Within the policy specification, each user lists his/her privacy requirements for each app. Each app can be specified with an attribute of "name" and/or a specific app category with an attribute of "category". For each app, all permitted resource types are declared within the XML entry of this app.

2.3 Co-policy Generation

A co-policy represents the privacy specification of all data on a device, which satisfies all data involvers who have provided their individual privacy policy regarding their data on this device.

Figure 3 shows how to generate co-policies based on each user's individual policies. In order to generate one's co-policy for a particular app, we check all individual policy settings for the app from all his/her friends (determined by the policy server). All settings from different users are merged into a new format which describes a "default" policy (allow = "yes/no") and a list of exceptions. Each exception represent one data involver whose permission settings are different from the default setting.

For the example in Figure 3, after merging, we can see that in Alice's co-policy, the two resources (contacts and photos) are permitted by the app (WeChat[2]) as default, while Katherine is an exception. The situation is opposite in Katherine's generated co-policy.

While generating co-policies based on individual policies, we follow the following principles:

- The default co-policy of each user is defined as his/her individual privacy policy, which means that if your friends do not have any special requirements, you can use the original Android permission system.

- Each policy item can be defined for each individual app or a category (group) of apps. Approximate algorithms are needed to match the app names in cases where the same app might use different names in different versions.

[2]or WeiXin in Chinese, which is the most popular online chatting tool in China, whose function is similar to WhatsApp.

- For each app (or app category) in the co-policy, we calculate its co-policy as follows: the default policy is defined as oneself's individual policy, while other people who require a different setting was specified as a list of exceptions.
- We notice that some co-policies could be generated automatically based on each user's settings. We will investigate the possibility of automatic co-policy generation in future work.

2.4 Policy Enforcement

In order to enforce the co-policy specification, we modify the Content Resolver in the Android Framework to monitor public APIs for sensitive data access. The process is similar to the original Android permission system as describe earlier. However, we make modifications to accommodate the fine-grained privacy specifications in each user's co-policy. In our current implementation of CoDroid, we consider five popular data types on Android phones: calendar, SMS, contact, callLog and media (photos in particular).

First of all, all data accesses are allowed only if the corresponding app declares the relative permission label in its manifest file. This indicates that we are not allowing extra access privileges beyond what the current Android system allows for each app.

As we mentioned earlier, a permission allows the user to access the whole database. In CoDroid, we check the returned results to see whether specific data entry is against the privacy specification in the co-policy. We then filter the results to exclude those data from the returned results.

For example, in the sample policy we show earlier, in Alice's co-policy, although she allows the WeChat app to access her contacts, it specifies that Katherine's contact information should not be accessed by WeChat. Thus, when WeChat requests for a list of Alice's contacts, Katherine's information will be excluded from the results WeChat received from the Content Resolver.

3. IMPLEMENTATION

Sensitive data are retrieved using the corresponding content providers. In Android, five standard interfaces are defined and implemented in each content provider, including *getType, insert, update, delete and query*, such that other components or applications can manage the underlying sensitive data conveniently. Among these five interfaces, *insert, update and delete* create information flows from mobile apps to underlying databases, while the other two trigger information flows in the other direction.

Content providers are not invoked by mobile apps directly. Instead, they are called with content resolvers as the mediation. Consequently, we hook and tweak the corresponding APIs using the Cydia Substrate[3]:

- ContentResovler $\rightarrow$ query
- ContentResovler $\rightarrow$ openInputStream

The interface of openInputStream is also monitored because photos are accessed in a different way from the other types of sensitive data supported.

Take the query interface as an example, there are five SQL-style parameters passed in the query interface:

[3]http://www.cydiasubstrate.com

- *URI* represents the type of the content which is requested. For example, "content: //com.android.contacts/ contacts", and "content://call_log/calls".
- *Projection* represents the table columns to be selected from the underlying database. Unfortunately, it is often set to "null" by mobile apps, which means that all available columns should be returned.
- *Selection and SelectionArgs* are similar to the "where" clause in SQL. Each question mark in Selection represents an argument, which is located in the array of SelectionArgs. These two parameters are often set to "null" by mobile apps, which means that all available rows should be returned.
- *Order Claims* specifies the order in which records should be returned. In general, there is a default order pre-defined for each content provider in Android.

We parse the URI parameter to determine whether or not the current request is relevant with the sensitive data supported. If it does, according to the co-policy specification, we filter the projection parameters, and append the *selection* and *selectionArgs*. After that, we invoke the original query interface with the modified parameters. In this way, mobile apps will receive abridged "views" of the requested sensitive data, from which certain items are hidden to satisfy the collaborative privacy requirements.

4. CASE STUDIES

In order to evaluate CoDroid, we construct two case studies with a popular social app - WeChat.

The Android phones used in experiments are rooted and installed with the Cydia Substrate at first. Then, we install our modified content resolver with API hooking, link the substrate files, and soft-restart them.

4.1 Case Study 1: Contact Access

In Android, many apps may access contact information through the Contact Provider. We tweak the corresponding query interfaces in CoDroid, so that passed-in parameters can be modified according to the downloaded co-policy. In this way, certain contact entries can be protected without showing up in the returned results.

We employed the policies shown in Figure 3 and test the function of importing contacts to WeChat on Alice's phone. The screenshots are shown in Figure 4. The left screenshot shows the contact list showing up without applying CoDroid. In this case, the privacy policy used is identical to Alice's individual policy in Figure 3. Because Alice allows WeChat to access all her contacts, all entries in her contacts (include Katherine) show up in the picture.

We then apply CoDroid with Alice's co-policy, which specifies that Katherine does not want WeChat to access her contact info. In the right screenshot, we can see that CoDroid successfully hides Katherine's contact info.

4.2 Case Study 2: Photo Publishing

Many mobile users take photos with their phones and publish them through all kinds of social media such as Facebook and Twitter. One thing they normally forget is that these photos might contain some images of other people

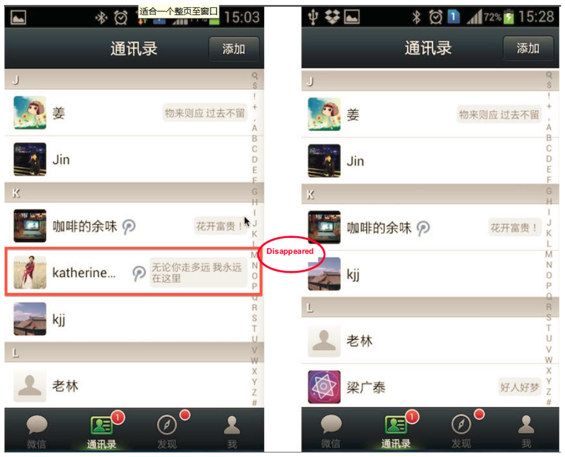


Figure 4: Case Study 1 - Contact Access in WeChat. *The screenshots show the contact list when the Alice tries to import all friends in her contact list to WeChat. Left: without CoDroid (using Alice's original policy in Figure 3). Right: with CoDroid (using Alice's co-policy specified in Figure 3.)*

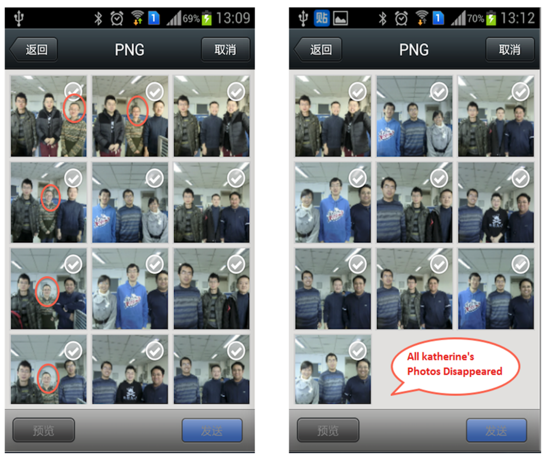


Figure 5: Case Study 2 - Photo publishing in WeChat. *The screenshots show the list of photos when Alice tries to select a photo to publish in WeChat. Left: without CoDroid (using Alice's original policy in Figure 3). Right: with CoDroid (using Alice's co-policy specified in Figure 3.)*

(normally friends). Some of the people in these photos might not want to show them to the world due to various reasons.

In order to protect the photos according to each user's privacy requirements, we need to identify the people in the photos. This could be achieved using face recognition techniques with the profile pictures of each contact stored on the local phone. More information could be referred to our earlier work [8].

Now we assume that each photo has been successfully tagged with names (IDs) of all people in it. Suppose Alice is trying to upload one of the pictures from her photo album. Figure 5 shows the list of photos showing up in the picture selection window. Similar to the last case study, the left screenshot shows the case with Alice's original personal policy, while the right screenshot shows the results with CoDroid, where Alice's co-policy specifies that Katherine does not want WeChat to access her photos. As a result, all Katherine's photos are hidden from WeChat.

5. CONCLUDING REMARKS

Although many studies on privacy in the mobile systems have been proposed, they mainly focus on detecting potential leakages on a particular phone. Based on our observation that sensitive information such as personal contact information are not only located in one's own phone, but also distributed among the phones of his/her friends, we describe a new privacy threat of *leakages through friends*.

To mitigate this threat, this paper introduces the concept of collaborative privacy management to protect the privacy of not only data owners but also data involvers on mobile devices. We designed and implemented the proposed mechanism on Android to support fine-grained access control on sensitive data. It allows each user to specify their privacy requirements and enforce them collaboratively in a distributed manner.

Defeating privacy threats such as leakage through friends is not an easy task. Our current solution represents only a very small step towards solving the problem, while a more complete and practical solution requires a much larger effort in further investigation. Nonetheless, we believe *privacy protection beyond one's own device* is an important issue and collaborative privacy management could become a viable solution in the future.

Acknowledgment

This work is supported in part by the NSFC (No. 61103026), the High-Tech Research and Development Program of China (No. 2013AA01A605), and the National Basic Research Program of China (973) (No. 2011CB302604).

6. REFERENCES

[1] M. Egele, C. Kruegel, E. Kirda, and G. Vigna. PiOS: Detecting privacy leaks in iOS applications. In *NDSS*, 2011.

[2] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *OSDI 10*, pages 1–6, 2010.

[3] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. In *CCS 09*, pages 235–245, 2009.

[4] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These aren't the droids you're looking for: retrofitting Android to protect data from imperious applications. In *CCS 11*, pages 639–652, 2011.

[5] H. Hu, G.-J. Ahn, and J. Jorgensen. Detecting and resolving privacy conflicts for collaborative data sharing in online social networks. In *ACSAC '11*, pages 103–112, 2011.

[6] A. C. Squicciarini, M. Shehab, and F. Paci. Collective privacy management in social networks. In *WWW'09*, pages 521–530, 2009.

[7] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen. Investigating user privacy in Android ad libraries. In *IEEE Symposium on Security and Privacy 2012 Workshops*, 2012.

[8] L. Zhang, Y. Guo, and X. Chen. Patronus: Augmented privacy protection for resource publication in online social networks. In *MobileCloud 2013*, pages 578–583, 2013.

[9] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh. Taming information-stealing smartphone applications (on Android). In *TRUST'11*, pages 93–107, 2011.