PrimAndroid: Privacy Policy Modelling and Analysis for Android Applications

Guillaume Benats[1,2], Arosha Bandara[1], Yijun Yu[1], Jean-Noël Colin[2], and Bashar Nuseibeh[1,3]

[1]Centre for Research in Computing, Department of Computing, The Open University, Milton Keynes, MK10 9FG, UK
{a.k.bandara,y.yu,b.nuseibeh}@open.ac.uk
[2]Faculty of Computer Sciences, FUNDP, Namur, Belgium
gbenats@student.fundp.ac.be,jean-noel.colin@fundp.ac.be
[3]Lero, Irish Software Engineering Centre, Ireland
nuseibeh@lero.ie

***Abstract*—The rapid growth of mobile applications has imposed new threats to privacy: users often find it challenging to ensure that their privacy policies are consistent with the requirements of a diverse range of of mobile applications that access personal information under different contexts. This problem exacerbates when applications depend on each other and therefore share permissions to access resources in ways that are opaque to an end-user. To meet the needs of representing privacy requirements and of resolving dependencies issues in privacy policies, we propose an extension to the P-RBAC model for reasoning about plausible scenarios that can exploit such weaknesses of mobile systems. This work has been evaluated using the case studies on several Android mobile applications.**

***Index Terms*—Privacy policy, Mobile applications, Policy conflicts, Android security, Role-based access control.**

I. Introduction

GOOGLE'S Android has become a leading mobile platform, supported by a wide community of developers. Its popularity is reflected by the availability of over 100K mobile applications on the Android Marketplace. A common feature of many of these applications is the ability to access, store, modify and share users' personal data in one way or another. In fact, a study by Enck et al. [1] showed that Android applications often misuse users' data. Such findings confirm our view that mobile applications raise issues of privacy and trust, which need to be better understood. Android is an attractive platform to study in this regard because, being an open source system, it is possible to get a better understanding of its security and privacy mechanisms when compared with rival platforms.

Conventional approaches to protecting users' privacy have been based on guidelines such as UK Information Commissioner's Fair Information Principles or the OECD guidelines on cross border information flows, both of which contain the notion of informed consent. This requires that any collection of personal data is preceded by users giving their explicit consent for the data to be collected. This is also the approach adopted by the way Android mobile applications are installed: a user is first presented with a list of required resources (i.e. data sources); then the user gives permissions for the resources to be accessed by choosing to proceed explicitly. External privacy management tools such as *Apex* [2] extend the basic Android permission system to allow users to specify conditions under which an application is given permission to access the specified resources. For example, using *Apex* a user can specify that *"this application can only access GPS location when country is United Kingdom"* at installation time. Without *Apex*, users can only grant the access to GPS Locations without differentiate the contexts. Our premise for all scenarios cited in this paper is that Apex is already used to manage permissions over resources.

In this paper we identify two significant shortcomings in Android's approach to privacy management. First, Android's permissions system allows applications to call each other and under certain conditions one application can make use of any permission that was granted to another application. This raises significant privacy concerns because users are not made aware that an application that has not request any access to personal data at the installation time can gain access to this data by calling another application that was given the permission. Secondly, although the *Apex* extension allows users to specify constraints on policies, there is currently no support for checking conflicts inside the resulting privacy permissions. To address these two problems, we extend the P-RBAC [3] privacy policy model to represent dependencies between applications together with the permissions granted to each application. Our extension can thus support automated analysis to identify permissions available to a particular application, and the extended privacy policy model can also be used to detect inconsistencies between the permissions granted to applications.

The remainder of the paper is organised as follows. Section II gives an overview of Android's permission subsystem; Section III presents a concrete scenario to lead our discussions; Section IV presents our approach to analysing Android permissions; Section V goes over the state-of-the-art of practical privacy tools for Android; and Section VI discusses our work and presents some conclusions.

II. Android and Apex Permissions Framework

Android [4] is a multi-process system based on a version of Linux kernel. By default, Android assigns each application its own unique `UID`. However, if explicitly specified in the file *"AndroidManifest.xml"*, this UID can be set to the same value for two or more applications. They will then all share the same ID - provided that they are also signed by the same certificate. Applications with the same

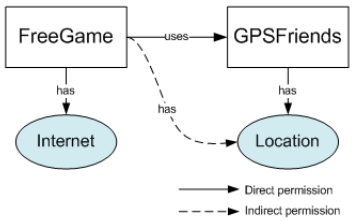

Fig. 1. Application dependency resulting from a shared `UID`

user ID can access each other's data and, if desired, run in the same process. Android applications do not contain a single entrance such as the *main* method where everything starts. Instead, all android applications are indeed a set of components that are named *activities* and each activity can be called by other applications. Each application, by default, has no permission granted that allow it to perform specific operations on phone's resources or invoke services from other applications.

The only way for an application to get a permission in Android is to include it in the file *"AndroidManifest.xml"*. All permissions requested in this file are then presented to the user who installs the application, either all of them are granted or nothing is granted if he refuses. This was one of the limitations of the current Android system before using a tool like *Apex* [2]. About a hundred permissions are provided by the Android system, for more details see [2] or the Android developers webpage [4]. Examples of Android permissions are: `write_sms, internet, call_phone`, which are respectively used to grant access to the "send SMS", "internet", "calling" features of the mobile phone.

One application using activities (i.e.: components) of another application can only do that if it is granted the same permissions as the ones required by the callee. The only way to override this is when two applications share the same `UID`.

III. Android Scenario

This section presents a concrete case study to express and deal with privacy issues mentioned in the introduction:

Alice installs three applications on her Android phone:

- *`GPSFriends`: An application using* Google Maps *to locate friends who are using the same application.*
- *`FreeGame`: A free game application that Alice installed by clicking at an advertising link.*
- *`GPSWifi`: An application to share free Wi-Fi locations with friends.*

Each application requests and obtains the following permissions at the installation time:

- *`GPSFriends` and `GPSWifi`: access to location data.*
- *`FreeGame`: access to the internet.*

Although Alice is not aware of it, these applications are interrelated in the following way:

- *`FreeGame` and `GPSFriends` are sharing the same `UID`*
- *`GPSWifi` both use `GPSFriends` to get the user's location.*

The dependency graph in Figure 1 shows the relationship between permissions (shown as ovals) and applications (shown as boxes) for the GameApp and GPSFriends applications. An arrow from an application to a permission indicates those permissions that are specified in the application's manifest file and are thus granted by the user at the installation time. However, in this example the existence of a path from GameApp to Location indicates that this application also has permission to access location data, despite not explicitly requesting this permission from the user. Figure 2 shows the dependencies between the GPSFriends and GPSWifi applications. The plain arrows show "direct links", i.e.: created by user granting permissions to applications. Dashed arrows show "Indirect links", a way for an application to get permissions of another application and this way accessing data it should normally not. Indirect links created in this way are the privacy issues we want to address.

Finally, Alice uses the Apex tool (see section II) to specify constraints that limit the conditions under which each permission applies. These constraints are:

- *`GPSFriends`: can only access location data if location is in London and the time is between 8am and 4pm.*
- *`GPSWifi`: can only access location data if the location is in the UK.*

The above scenario illustrates how information in the *"AndroidManifest.xml"* file only is insufficient for a user to grant informed consent for an application to access personal data. This is because the installation process does not expose how dependencies between applications can affect the actual permissions available to the applications. In order to address this issue, we propose an extension of the P-RBAC specification [3] to support automated reasoning about the actual permissions available to an application. Our modifications make it possible to take into account both dependencies and user specified constraints when reasoning about Android permissions.

IV. P-RBAC for Mobile Applications

The core of P-RBAC [3] is adapted to fulfill mobile permissions requirements, and in our case, Android permissions requirements in the following manner.

Since Android devices are typically for a single user, there is only one role to consider. Therefore we can let the concept of *"role"* aside for now. It will be replaced with the concept of *"dependency groups"* later on. In addition, we propose to remove the *"purpose"* concept from the basic P-RBAC since the purpose description associated with Android permissions is simply a text label without well-defined semantics to analyse privacy policies. We keep the *"obligation"* entity even though it is not currently supported by the Android permissions system, because it is useful for logging the accesses to resources made by applications and for giving users feedback on potential privacy violations.

We can now express simple Android permissions together with constraints using this model. Consider the permissions and constraints defined by Alice in our case study (III) :

The permission on `GPSFriends`: *"`GPSFriends` can*

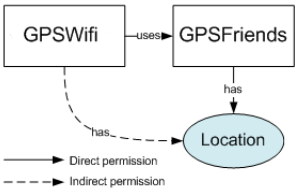

Fig. 2. Dependencies due to usage of a component

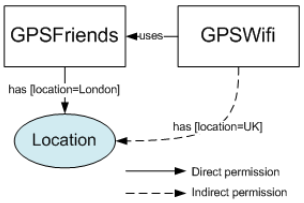

Fig. 3. Privacy issue due to usage of another component

only access location between 8 a.m and 4 p.m" can be expressed as follows (`ACCESS_FINE_LOCATION` denotes the Android permission to read the GPS sensor location):

$permission$(GPSFriends, ((has, ACCESS_FINE_LOCATION), $(time > 8 \wedge time < 16)$, Log_Access())

However, the above permission expressions do not take into account the dependencies that can arise between applications. To address this limitation we need to represent dependencies between applications and to this end we introduce a notion of *"dependency groups"* into the P-RBAC model. This new entity can be seen as the equivalence of the *"role"* entity as an application belongs to a group depending of its purpose. The details of this are discussed in the next section.

A. Dependency-aware privacy

In order to include the notion of dependencies between applications in the P-RBAC model, we introduce the *"Dependency Groups"* entity in-between applications and privacy data permissions. This makes a dependency group analogous to a *"role"* in the original RBAC formulation.

This new *"Dependency Groups"* entity is referring to the environment of an application in terms of usage dependencies. Each application that invokes operations on other applications, and applications sharing the same `UID`, should be assigned to the same dependency group. The permissions are then granted to the group. The user can thus constrain those permissions with conditions like before, using Apex, but those conditions would apply to all applications associated with the concerned dependency group. Existing tools would assign the same permissions on applications that are used together. If two or more applications depend on each other, then they belong to the same group.

Let A be a set of applications, G be a set of dependency groups, R be a set of resources, X be a set of actions, and O be a set of obligations, respectively, our P-RBAC model is defined using a condition language LC, which includes the following sets:

- Resources Permissions $RP = \{(x,r)|x \in X, r \in R\}$;
- Privacy-sensitive Resources Permission $PRP = \{(rp,c,o)|rp \in RP, o \in O,$ c is an expression in $LC\}$;
- Application Assignment $AA \subseteq A \times G$ is a set of many-to-many mappings from applications to dependency groups;
- Privacy-sensitive Resources Permission Assignment $PRPA \subseteq G \times PRP$ is a set of many-to-many mappings from dependenccy groups to privacy-sensitive resource permission assignments.

Under this new formulation, `GameApp` would also be constrained by privacy rules over location, since both `GameApp` and `GPSFriends` would be in the same dependency group, sharing the same privacy rules. This makes it easier to prevent unauthorised access to Alice's location.

A potential complication of using dependency groups is in dealing with situations where large number of applications are in the same group due to complicated dependencies between all of them. Alice's scenario is an extreme case where there is a lot of relations between applications but in a system with hundreds of applications, a lot of them would run as standalone entities or be coupled with, at most, one or two other applications. Therefore, we expect dependency groups to be of a manageable size.

An application can also belong to several groups, as long as the application does not share a `UID` with an application of a different group. So, if application a_1 is in group g_1 and g_2, and it shares a `UID` with application a_2, which is in group g_2, then groups g_1 and g_2 should be merged. In our scenario, for example, `GPSWifi` could be in another dependency group meant for internet settings applications, as well as being in a dependency group with `GPSFriends`.

Now let's have a look at another interesting scenario which is avoided with our dependency groups entities: Imagine that Alice has added a new condition on the permission of `GPSFriends` to access location, constraining access to location information when she is in London:

$Permission_1$: (GPSFriends, ((use, ACCESS_FINE_LOCATION), $Location = London$, Log_Access()))

Alice also constrains `GPSWifi` to be granted access to location information only when her location is in the UK:

$Permission_2$: (GPSWifi, ((use, ACCESS_FINE_LOCATION), $Location = UK$, Log_Access()))

However, the problem that arises is that `GPSWifi` has not been given a direct permission to use the GPS location data. Instead it accesses location data via `GPSFriends`, which now has a constraint that limits its ability to access location information to those times when Alice is in London. This means that any attempt by `GPSWifi` to access location information when Alice is in the UK, but outside of London will fail, because `GPSFriends` does not have the required permission in this situation.

There are two possible solutions to this problem, the

first one being to grant `GPSWifi` a direct permission to location data which can then be constrained to be applicable whenever Alice is in the UK. Alternatively, we could define a dependency group and assign `GPSWifi` and `GPSFriends` to it. This would allow Alice to define the conditions she wanted on permissions for the whole group and not for each application separately as if they were used in completely different groups.

In addition to dependency-aware permissions management, we also have to deal with the issue of constraint conflicts that could arise when users are allowed to specify the conditions under which particular permissions arise. Indeed, Alice can now define privacy rules on a *"dependency group"* and, this way, constraining permissions of all applications inside this group. However, if the constraints she defines inside a group conflict with each other, all the applications belonging to the same group could be prevented from accessing required resources; otherwise they will be allowed access them under the wrong circumstances. We deal with the issue of constraint conflicts in the next section.

B. Constraint Conflicts

As `GPSWifi` uses `GPSFriends` to get information about location, they will end up in the same dependency group, thus the two constraints added by Alice on each application would lead to conflicting constraints (i.e. conditions in our P-RBAC model) in our dependency group since it is not possible to enforce both permissions together. This becomes apparent when we combine the permissions by taking the conjunction of the conditions specified:

$Permission_1 \wedge Permission_2$: ($Group_1$, ((use, ACCESS_FINE_LOCATION), $time > 16 \wedge time < 20 \wedge time > 8 \wedge time < 16$, Log_Access()))

We have a conditions conflict due to conflicting constraints between both privacy rules. The algorithm used in P-RBAC [3], based on a SAT solver, is the one we implement for detection of those conflicts. If one applies this algorithm to Alice's scenario, the result would be `true` as she defined two conflicting constraints on applications inside the same dependency group.

V. Related Work

This section briefly presents some of the research literature on privacy management in mobile applications, focusing on the investigations of the Android system.

Research on extending the Android permission framework has concentrated on implementing fine-grained, context-aware policy mechanisms [5], [6]. However, these approaches do not include mechanisms for detecting privacy problems that arise due to application dependencies. Other work has explored ways in which the Android permission system can be attacked [7] using application dependencies, but does not suggest ways of detecting or resolving these problems.

There is also work focusing on monitoring Android applications to detect information leakage [1], [8]. The scope of our work is thus limited mainly on the idea of dependency and its impact on privacy.

VI. Discussion and Conclusion

In this paper we present a modified P-RBAC model to represent scenarios of Android applications, and to reason about dependencies between applications and the impact on user privacy when those applications use each other as services. In addition to expressing permissions scenarios, our model allows resolution of some current issues of privacy in Android applications. Along with existing tools such as *Apex*, it addresses some privacy weaknesses that were still present in Android and allows expression of more advanced policies to rule out those issues. Our approach of user-defined rules can be an overhead to users, it is why our upcoming prototype focuses on an easy-to-use policy management application. Although other mobile platforms such as the iOS are as important, they are not as open as the Android platform. In future we will study whether it is possible to make such extensions applicable to the iOS platforms.

VII. Acknowledgments

This work was undertaken as part of an internship at the Open University for a master thesis in Computer Sciences at the University of Namur. The work was in part funded by the EPSRC PRiMMA project and the EU FP7 SecureChange Project.

References

[1] W. Enck, P. Gilbert, and B.-G. Chun, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," 2010.

[2] M. Nauman, S. Khan, and X. Zhang, "Apex : Extending Android Permission Model and Enforcement with User-defined Runtime Constraints," Apr. 2010.

[3] Q. Ni, A. Trompette, E. Bertino, and J. Lobo, "Privacy-aware role based access control," ACM Press, June 2007.

[4] Google, "Security and permissions," 2010. `http://developer.android.com/guide/topics/security/`.

[5] M. Conti, V. Nguyen, and B. Crispo, "Crepe: Context-related policy enforcement for android," in *Information Security* (M. Burmester, G. Tsudik, S. Magliveras, and I. Ilic, eds.), vol. 6531 of *Lecture Notes in Computer Science*, pp. 331–345, Springer Berlin / Heidelberg, 2011.

[6] G. Bai, L. Gu, T. Feng, Y. Guo, and X. Chen, "Context-aware usage control for android," in *Security and Privacy in Communication Networks*, vol. 50 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 326–343, Springer Berlin Heidelberg, 2010.

[7] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy, "Privilege escalation attacks on android," in *Information Security* (M. Burmester, G. Tsudik, S. Magliveras, and I. Ilic, eds.), vol. 6531 of *Lecture Notes in Computer Science*, pp. 346–360, Springer Berlin / Heidelberg, 2011.

[8] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, "SCanDroid : Automated Security Certification of Android Applications." Technical Report.