Oglaza *et al. EURASIP Journal on Information Security* (2017) 2017:13
DOI 10.1186/s13635-017-0065-4

EURASIP Journal on Information Security

RESEARCH Open Access

# A new approach for managing Android permissions: learning users' preferences

Arnaud Oglaza* , Romain Laborde, Pascale Zaraté, Abdelmalek Benzekri and François Barrère

**Abstract**

Today, permissions management solutions on mobile devices employ Identity Based Access Control (IBAC) models. If this approach was suitable when people had only a few games (like Snake or Tetris) installed on their mobile phones, the current situation is different. A survey from Google in 2013 showed that, on average, french users have installed 32 applications on their Android smartphones. As a result, these users must manage hundreds of permissions to protect their privacy. Scalability of IBAC is a well-known issue and many more advanced access control models have introduced abstractions to cope with this problem. However, such models are more complex to handle by non-technical users. Thus, we present a permission management system for Android devices that (1) learns users' privacy preferences with a novel learning algorithm, (2) proposes them abstract authorization rules, and (3) provides advanced features to manage these high-level rules. Our learning algorithm is compared to two other well-known approaches to show its efficiency. Finally, we prove this whole approach is more efficient than current permission management system by comparing it to Privacy Guard Manager.

**Keywords:** Android permission, Access control model, Recommender System

## 1 Introduction

Defining privacy and thus protection of privacy is difficult. If at the end of the nineteenth century privacy was "The right to be let alone" [1], it is hard to isolate yourself in our digitalized world that has been created to facilitate the flow of information [2]. Some people even wonder if privacy still exists arguing that our digital life is either shared or public [3]. Many researchers have taken a more balanced view and propose solutions to control the collection, analysis, and dissemination of personal information, as well as solution to avoid intrusion/decisional interference [4].

Smartphones have a predominant place in this digital world. According to Gartner[1], "the smartphone market has reached 90 percent penetration in the mature markets of North America, Western Europe, Japan and Mature Asia/Pacific" in 2016. These devices being more and more powerful, they include more and more applications. A survey from Google and TNS/SOFRES [5] in 2013 shows that French people have installed an average of 32.7 applications on their smartphone[2]; most of which are free. We performed an analysis of the 50 most downloaded free applications on the Google Play Store. This study showed that an Android application requests an average of 11.4 permissions, 5.72 of which can directly harm privacy. Multiplying it by the number of applications per device results in a total of 372.7 permissions and 184 of which are highly dangerous to manage on each device. Each of these dangerous permissions shall be carefully chosen since some applications, not considered as trojan, collect and sell users' data under the guise of providing some services. For instance, Yo[3] is a free application whose unique feature is sending a notification with the word "Yo" and the current location to user's friends. Yo requests a lot of permissions. Access to the contact list and the location seems coherent. However, Yo also asks to have access to the identity information, the files, the pictures, and the camera. The privacy policy of Yo states that the company collects personal and activity data, shares them with companies they trust, and keeps all these data indefinitely.

On Android, controlling access to applications is complex. Until version 5.x (included), installing an application is equivalent of granting all the permissions requested by the application (more than 50% of Android devices are still running version 4.x or 5.x in early 2017[4]). Permission

*Correspondence: oglaza@irit.fr
University of Toulouse, IRIT UMR 5505, Toulouse, France

management is very limited: either you authorize everything or you cancel the installation. Additional permission management systems (such as Privacy Guard Manager, Permission Master, XPrivacy, or DonkeyGuard) can be installed to enhance the basic native Android system by allowing users to modify permissions after the installation of applications. All these permission management applications follow an Identity-Based Access Control model, i.e., the user has to control every permission for every installed application. Although IBAC allows fine-grained access control, it is not suitable for managing hundreds of permissions. Scalability issue has been studied by the access control models research community that proposed the use of abstractions leading to high-level authorization rules making global access control policies more understandable. Nonetheless, applying these models requires to understand the associated abstractions and thus suffer accessibility for non-technical users. More recently, Google has enhanced the native Android permissions management system in version 6.x by (1) allowing users to modify permissions at run time and (2) reducing the number of permissions. However, it is no more possible to have a fine-grained control on applications (more details are given in Section 2.2).

The ideal permission management should allow users to write high-level permissions without narrowing the capability of access control. High-level permissions will reduce the number of permissions and make the global policy more understandable. However, writing high-level policies being more complex, non-technical users would not be able to do it without help. In this article, we present a recommender-based system, called Kapuer (KAPUER is an Assistant for Protection of Users pErsonal infoRmation), that assists people in managing permissions on their Android device. Kapuer includes a novel machine learning algorithm, based on an aggregation operator called Kagop (Kapuer AGgregation OPerator), to capture users' preferences in terms of privacy. These preferences, when validated by users, are transformed into XACML V3 policies and then enforced by our XACML authorization management system. We have implemented Kapuer in an Android application based on the Xposed[5] framework. It has been tested on Android 4.4 and is freely available to download at the following address: http://www.kapuer.org. Kapuer also includes additional permission management features to enhance its efficiency in understanding and visualizing abstract authorization rules.

This article summarizes all our previous works on Kapuer. In [6], we proved the benefit of recommender-based systems for writing policies. In [7], we introduced a first version of our problem-solving model as well as the initial Android prototype. In [8], we described the final Android implementation. This article extends [8] and presents the whole learning process. For the first time, we detail the latest version of our aggregation operator Kagop, which is the cornerstone of our learning algorithm. We also describe the integration of Kagop in Kapuer. And finally, we prove Kagop has better results than two well-known aggregation operators for learning privacy preferences on Android.

The rest of the article is structured as follows: In Section 2, we review and discuss access control management approaches applied to Android. In Section 3, we give an overview of decision support system and we present the generic architecture of Kapuer. Section 4 introduces Kapuer's problem-solving model and its instantiation to the Android permission management context. In Section 5, we detail the learning algorithm Kagop. In Section 6, we introduce some features that complement the learning process of privacy preferences. We evaluate, in Section 7, Kapuer and Kagop on a real life scenario. Finally, we conclude in Section 8.

## 2 Related works

In this section, we summarize different works related to Android permission management.

### 2.1 High-level policies in Android

Access control models can be seen as design patterns to help the specification of policies. They all consider three main entities: the subject, the action, and the resource. In addition, some access control models propose abstractions. For Barker [9], these abstractions (he calls them categories) represent "any of several fundamental and distinct classes or groups to which entities may be assigned." One of the advantages of using abstractions is simplifying policies. For instance, Role-Based Access Control (RBAC) [10] uses the concept of role to group subjects according to their function in an organization. Other access control models propose abstractions for other elements. Organization-Based Access Control (OrBAC) [11] uses abstractions on all three main elements: roles abstract subjects, activities are for actions, and views for resources. Some access control models are designed for privacy like PBAC [12] that introduces the intent of the subject or P-RBAC [13] that extends RBAC with concepts like purpose, condition, and obligation. An exception is Attribute-Based Access Control (ABAC). ABAC does not introduce abstraction but is a specification pattern to express authorization policies using any abstraction.

Each model offers concepts and abstractions to guide security experts in the writing of policies. Even if analyzing and implementing abstractions is time-consuming, resulting authorization policies are more powerful and easier to manage. But it requires security expertise for writing authorization policies and most of mobile device users lack this skill.

Some systems already use access control models to enforce Android permissions. For instance, CRêPE [14] uses context-related policies to control how applications can use their permissions. Detection of contexts triggers the activation of the policy to use.

MOSES [15] is also a system enforcing policy-based security on Android. MOSES relies on system compartmenting to create isolated areas. Each area can be used to separate, for example, data and applications used for work and those for personal use. Although both CRêPE and MOSES allow Android to enforce high-level policies, they did not address the usability issue. Only skilled people can write those policies or define contexts and security profiles.

### 2.2 Writing policies with a graphical editor

Until version 6.x, official Android releases did not provide any efficient tools to manage permissions. Many custom releases or applications give users a way to better control permissions (such as Privacy Guard Manager, Permission Master, XPrivacy, or DonkeyGuard). Our study will concentrate only on Privacy Guard Manager (PGM) from CyanogenMod[6] since all other applications have the same drawback. PGM is available in the settings menu of CyanogenMod and presents all installed applications. For each application, there is the list of its permissions. Users have the choice to select ON or OFF for each application to allow or deny the permission. An option also exists to ask users to decide at the first time the permission is requested. Thus, the interface is very easy to master and requires no technical skills: only a simple action on the device for each permission.

Although the process is easy to learn, checking all permissions for every applications is painful. As a matter of fact, we have shown in introduction that French people have to handle an average of 372.7 permissions. Few users will browse the whole list of permissions to protect their privacy. Although PGM is interesting when dealing with few applications, current smartphone environment is much more complex and using no abstraction has difficulties facing scalability.

Google has improved its native permission management approach in Android version 6.0. First, permissions are requested at runtime (the first time applications require them) and users can change allowed/refused permissions at any time. Now, Android application developers shall manage the fact that their application may not have access to all the permissions listed in the *manifest file*. Google has also addressed the large number of permissions issue by introducing protection levels and groups of permissions. Each permission is associated to one of four protection levels. Permissions with level *normal* are considered as low-risk permissions and are automatically granted without any user approval. Permissions with level *signature* are related to communication between applications developed by the same organization. The requesting application needs to be signed by the same certificate as the application providing the service and declaring the permission. In that case, the system automatically grants the permission without any action required from the user. Protection level *signatureOrSystem* works like signature but concerns also the applications that are in the Android system image. Finally, the last level, *dangerous*, contains permissions with high risk for the user. There are 24 dangerous permissions. To reduce this number, every dangerous permission is attached to one permission group. Nine different groups exist, and each one of them represents a resource or a set of resources like *CALENDAR*, *CONTACTS* or *PHONE*. For example, the group CALENDAR consists in two permissions *read calendar* and *write calendar*. To reduce the number of interactions with the user, the new permission management works as follows. When an application requests a dangerous permission, Android does not ask the user to accept or deny that particular permission. It asks the user to accept or deny the whole permission group.

Thus, even if there exist more than 130 permissions in Android, a user will be asked to grant or refuse permissions to a specific application only nine times at most (one per group). Although this approach seems more user-friendly, it has significant drawbacks in terms of security. All permissions to access any network services (3G/4G, NFC, Bluetooth) are associated to protection level normal. As consequences, users cannot control network access and any application can communicate anywhere. In addition, this loss of control is increased by the use of groups of permission. Indeed, users' control regresses even with dangerous permissions. For instance, group PHONE includes seven permissions (*use SIP*, *call phone*, *read phone state*, *process outgoing calls*, *read call log*, *write call log* and *add voicemail*). Thus, when a Voice over IP application requests permission *use SIP* which seems relevant, the user can only grant the group of seven permissions. These permissions are not all relevant to a VoIP application. Thus, to limit the number of interactions with the users and cope with scalability, Android 6.0 has decreased the privacy protection capability of the system. These IBAC coarse-grained permissions only give the user an illusion of control.

### 2.3 Writing policies with a text editor

Textual editors use specific languages to write authorization policies. Textual editors are less accessible than graphical ones because the language must be learned and understood before writing anything. However, these languages provide much more flexibility and the possibility to create very powerful rules.

XACML V3 [16] (eXtensible Access Control Markup Language) is a language standardized by OASIS, for writing authorization policies. XACML uses attributes to build policies thereby works great with ABAC. Every security element can be represented as an attribute. Then it is possible to create any kind of abstractions such as roles in RBAC, activities in OrBAC, etc. Genericity and flexibility are the main advantages of XACML but also its main flaw. Technical skills are required: understanding access control models to select suitable abstractions and write policies according to them. In addition, XACML is an XML language which is not known to be user-friendly for non-technical people. Thus, the whole process demands lots of technical skills and cannot be performed by owners of smartphones.

Arena et al. [17] have proposed an XACML-based extension of the Android's security framework called SecureDroid. This tool allows users to define situations and specify which permissions are accepted or denied in these situations. Users can also be prompted when a permission is requested. This approach do not use abstraction so scaling up is still a problem. Stepien et al. [18] have worked on a graphical editor to help non-technical users write XACML rules. With this editor, it is possible to choose an attribute, an operator, and a value to compare to. It makes writing rules possible without using an XML format. Nonetheless, understanding how abstractions work is still required.

## 3 A decision support system for writing high-level policies

Allowing non-technical users to write policies by themselves is not a simple task. A graphical editor like PGM is easy to use but lacks efficiency. With hundreds of permissions to handle, using abstractions seems mandatory. These abstractions can be specified with textual editors, but it requires a lot of technical skills so it is not accessible to the public. Non-technical users should be helped by a security expert to write authorization policies to protect their privacy but, of course, it is not possible to have an expert behind every smartphone user. Since none of these approaches is satisfying, we present our work which aims at (i) *requiring no skill before being used like PGM* and (ii) *allowing non-technical users to write policies with abstractions*.

We have chosen to create a Decision Support System (DSS) to help users write their complex policies [19]. DSS are a set of methods and techniques used to help someone facing a problem to make a decision [20]. We use a DSS to interact with users and understand how they want to protect their data. We present in this section our system, named Kapuer, applied to Android permission management. It informs users when applications request permissions, it learns how users react to requests, and it uses these preferences to propose abstract authorization rules. Kapuer consists in an architecture to interact with users and control applications, and a problem-solving model and an aggregation operator to learn users' preferences.

### 3.1 Introduction to Decision Support System

The main goal of a Decision Support System is *not to make the decision on behalf of the user but instead to give him precious information to understand the situation, to give parts of solutions, or possible alternatives to allow him make the final decision* [20]. Among the different approaches of DSS, we have focused on recommender systems. A recommender system works with a profile of the user. It filters and analyzes information, extracts the most useful to build knowledge about users, their preferences. By learning these preferences, the system is able to propose solutions to the user by analyzing new information each time new preferences are acquired. Thereby, the system is always learning and adapting itself to the user. Three types of recommender systems exist [21]:

1) Content-based recommendations rely only on objects characteristics to make propositions. All available information on the object can be used to describe it. For example, a book can be described by a title, an author, a release date, etc. To make recommendations, the system compares objects to find those that seems to be the closest to the user's preferences. Content-based recommendations are very interesting with detailed objects. Because they are seen as a set of characteristics, a new object can be immediately proposed to users if its characteristics fit their preferences. If users have always the same behavior, the system will always propose relevant objects. The drawback is the starting: when the system has no information about user's preferences, a learning period is required before propositions can be relevant. Similarly, if users suddenly change their behavior, there will be a certain latency before the system learns those changes.

2) Recommendations by collaborative filtering work with the preferences of all people using the system. The idea is if one user has similar preferences with other users, then he should like objects chosen by such users. Thereby, they can be relevant recommendations for him. Unlike content-based, the system does not need much information to start. It will quickly find other people with a close profile. Collaborative filtering also works well with objects that are hard to describe like emotions. However, this approach also has some drawbacks. When few people are using the system, finding a similar profile might fail. In this case, recommendations will not be relevant. In the same way, if a new object is added to the system, as long as it is not chosen by some users, it will not be recommended.

3) Hybrid systems use both content-based and collaborative filtering. It allows getting rid of some flaws of each approach. Content-based recommendations for new

objects in the system, collaborative filtering for users with few information to work with. A well-known hybrid recommender system is used by Amazon to create lists of similar items when a customer visits the page of an object or adds one in his basket.

Despite the advantages of hybrid recommender systems, we chose a pure content-based approach in Kapuer for two reasons. Firstly, using collaborative filtering requires to store every user's privacy preferences somewhere on a server and protection of users' preferences is complex [22, 23]. With a content-based recommender system, user's preferences are stored locally and are not shared at all. Secondly, privacy recommendation can also be provided by a set of experts like in [24]. We think that privacy is by nature personal and these solutions do not allow experts to customize their recommendations to specific users. For example, the five authors of this article do not agree on what access should be granted to an application like Facebook.

### 3.2 Architecture of Kapuer

Figure 1 shows the global architecture of Kapuer with a clear distinction between the XACML access control part and the decision support part. The process begins when an application requests the access to one of the user's private information. This request is intercepted by the Policy Enforcement Point (PEP) (step 1) that creates a request in the XACML V3 format and transmits it to the Policy Decision Point (PDP) (step 2). The PDP is the decision unit; it compares the request with the access control policy. If one of the rules matches, the PDP sends back the associated decision (Permit or Deny) to the PEP which applies it on the Android system. If there is no matching, the PDP returns decision "Not Applicable" to the PEP. No matching rule means that our system needs to learn more information about the user's preferences regarding this request. As a consequence, the PEP transfers the request to the DSS. To gather information about the user's preferences, the DSS interacts with him. Kapuer has two sorts of interactions with the user:

1) When a request is sent by the PEP to the DSS (step 3), Kapuer informs the user that an application wants to access one of his private data. If possible, details about the request are also given such as similar applications asking for the same permission or indications about the permission asked. The user just needs to accept or decline that request. Making a decision in the appropriate context (i.e., when the user runs the application) is easier than doing it out of the context of use; it limits the cognitive load. Once the user has made his decision, the DSS creates the corresponding XACML V3 rule for that specific couple (application, permission) and puts it in the policy database (step 4). Then, all the preferences regarding the attributes used in the request are updated (more information on the preferences update can be found in [7]). When the update process is completed, Kapuer calculates a score reflecting the user's preferences knowledge level.

2) When the score of a request reaches a predetermined threshold (found by experimentations), the system has acquired enough information to propose a new abstracted rule that covers a broader range of access requests. In this interaction, Kapuer presents this rule to the user and explains all the abstract elements (step 5). For example, if a rule is proposed for all game applications, Kapuer lists all games. If a rule is proposed for all resources

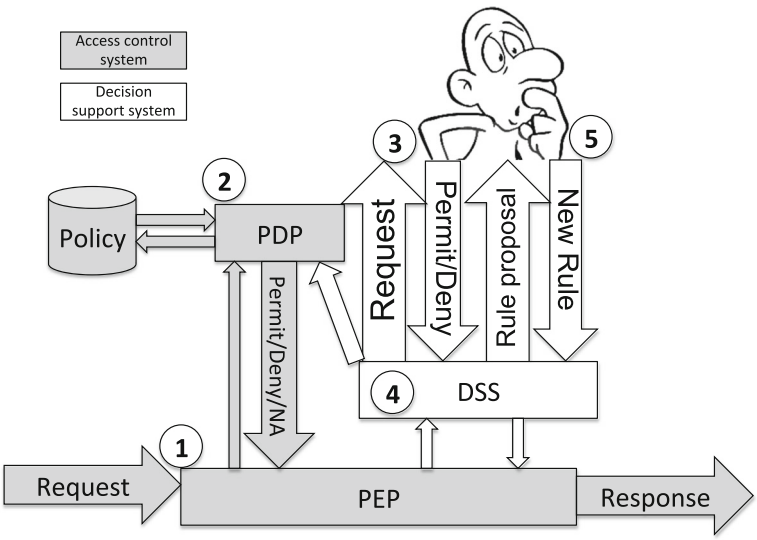


**Fig. 1** The architecture of Kapuer

linked to networks, it details all the associated permissions. This way, user are informed and even without any technical skills, they are able to understand what the rule means. Then they can accept or reject the proposed rule. If accepted, the DSS transforms the rule in XACML V3 and adds it to the policy database (step 4 again).

## 4 Kapuer problem-solving model and its instantiation to Android

In this section, we present our problem-solving model that represents the users' privacy preferences. This model is used by our learning calculus and is independent from any access control model. Thereafter, we instantiate it to the context of Android permission management where we define a specific access control model.

### 4.1 The problem-solving model

Our problem-solving model consists in four elements:

- **Criteria**—A criterion is the basic and main element of our problem-solving model. It represents an attribute of an access request like the name of an application, a resource, or an action. We have defined criteria to be very similar to attributes in ABAC. An attribute-based request can be easily converted into a list of criteria. The set of criteria is noted $CR$. Criteria are composed of an identifier and two values. The first value, $g^t : CR \rightarrow [0, \infty[$, increments each time the user accepts a request including this criterion. The second value, $f^t : CR \rightarrow [0, \infty[$, increments each time the user refuses a request including this criterion. This dual unipolar scale gives much more information than a unique bipolar scale. We can easily differentiate if a criterion has a low preferences value because we have few information on user's preferences or because users do not always behave in the same way (sometimes they accept requests with this criterion and sometimes they deny them). The preferences score regarding criterion $c$ can be found by subtracting $f^t(c)$ and $g^t(c)$. We define two preferences scores. The disclosure preference score $s^t_D(c) = f^t(c) - g^t(c)$ of criterion $c$ at time $t$ and the dual preference score about non disclosure $s^t_{nD}(c) = g^t(c) - f^t(c)$.
- **Classes of criteria**— We introduce the notion of class of criteria to express security objects introduced in the access control models like visibility, temporal and spatial aspects, retention, or purpose. Each criterion is part of a class with relation Association Criterion Class: $ACC \subseteq CR \times C$ where the set of class of criteria is noted $C$. It is possible to create any class depending on the type of data used by the system. The set of criteria of a class is defined by the function $class$ :

$$class : C \rightarrow \mathcal{P}^{CR}$$

$$x \mapsto \{y \in CR | (y, C) \in ACC\} \quad (1)$$

- **Meta-criteria**—Access control models propose abstractions of security objects. We define the notion of meta-criterion to represent these abstractions. For instance, "Games" is a meta-criterion for applications describing this kind of application. A meta-criterion is a criterion, with the same structure but with a higher level of abstraction. The set of meta-criteria is noted $MCR$ where $MCR \subset CR$. A meta-criterion is also part of a class. Each criterion is linked to one meta-criterion of the same class. A meta-criterion can be linked to another meta-criterion of a higher level. Then, we can create a hierarchy of criteria and meta-criteria for each class, noted by transitive relation $H_c$ such that $(x, y) \in H_c$ means that criterion $y$ is a meta-criterion of $x$ and $class(x) = class(y) = c$. Values of meta-criteria are updated each time a criterion or a meta-criterion linked to it is updated. Then if a criterion is updated, all meta-criteria in the same branch of the hierarchy are also updated.
- **Groups of criteria**—We have defined groups of criteria to represent the interactions (links or conflicts) between criteria or meta-criteria from different classes. A group of criteria is formed by at least two criteria and at most by the number of classes. A group has its own preferences values $f^t$ and $g^t$ (and therefore its two preferences scores $s^t_D(x)$ and $s^t_{nD}(x)$). The preferences scores are not calculated based the values of the criteria or meta-criteria present in the group. Values of groups are updated each time all the criteria or meta-criteria of the group are present in a request. The larger a group is, the more important it is because it conveys more detailed information about users preferences. The set of groups of criteria $G$ is defined by:

  $$G \subseteq \mathcal{P}^{CR}.$$

  A group of criteria is composed by at least two criteria.

  $$\forall g \in G, |g| \geq 2$$

  Two criteria of a same group cannot belong to the same class.

$$\forall g \in G, \forall (c_1, c_2) \in g \times g, c_1 \neq c_2 \Rightarrow class(c_1) \neq class(c_2)$$

### 4.2 The problem-solving model instantiation

When Kapuer intercepts a permission request, information about the application and the permission (i.e., what action on what service) is collected. Thus, we have defined

three classes to implement our problem-solving model: *Applications*, *Actions*, and *Resources*. Criteria of class Applications are represented by the name of the applications that are installed on the device. Kapuer retrieves the first level of meta-criteria from the application categories provided by the Google Play Store (e.g., games, entertainment, work, etc.). We have created a meta-criterion called *no category* for the applications that are not included in the Google Play Store. Finally, we added meta-criterion *all applications* to group all the meta-criteria of this class (Fig. 2). Criteria of classes Actions and Resources are extracted from the permissions (e.g., "android.permission.READ_CALENDAR" contains criteria READ and CALENDAR). We chose for class Resources to use meta-criteria *media*, *network*, *service*, *user data*, and *system data*. We added meta-criteria *hardware* (superior to *media*, *network*, *service*) and *data* (superior to *user data* and *system data*). Finally, the root of this hierarchy is meta-criterion *all resources* (Fig. 3). With the same reasoning, we have extracted criteria from the permission list for class Actions. We have defined three meta-criteria : *local access* (regrouping criteria *execute*, *control*, *read*, and *write*), *external access* (regrouping criteria *send* and *receive*), and the root of the hierarchy called *all actions* (Fig. 4).

Classes Resources and Actions are fixed; their criteria and meta-criteria remain always the same regardless the user's device. Meta-criteria of class Applications are also fixed. However, criteria of class Applications will depend on which applications are installed on the target device.

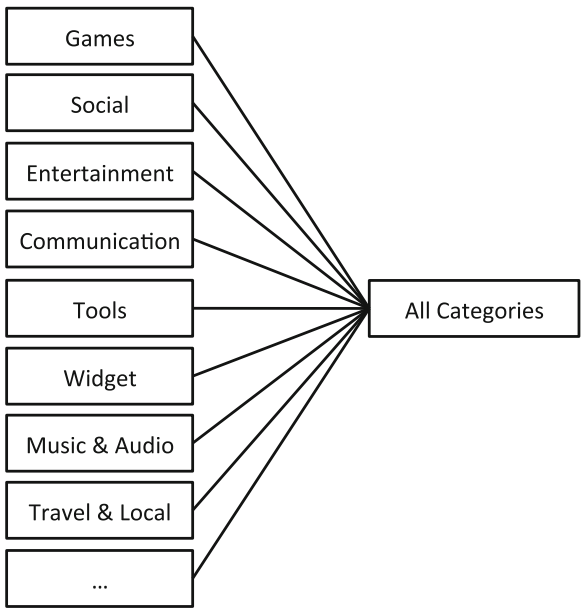


**Fig. 2** Hierarchy of criteria for class Applications

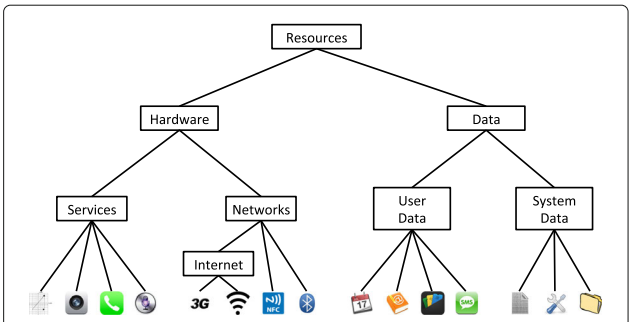


**Fig. 3** Hierarchy of criteria for class Resources

## 5 Our privacy preferences learning algorithm

Our privacy learning algorithm relies on Multi-Criteria Decision Analysis (MCDA), which is a set of methods used to solve complex problems where a mono-criteria approach cannot. Zeleny [25] said "It is only when facing multiple attributes, objectives, criteria, functions, etc., that we can talk about decision making and its theory. As alternatives of choice become more complex and are characterized by multiple attributes as well as multiple objectives, the problem of combining these various aspects into a simple measure of utility becomes more difficult and less practical." A method frequently used in MCDA is to decompose alternatives of a problem into criteria and aggregate their values to calculate an utility score. Then, alternatives can be ranked by score and recommended to users. In the following section, we present our own aggregation operator called Kapuer aggregation operator (Kagop).

### 5.1 Kapuer aggregation operator

When applications request permissions to access some services, the request is decomposed into a set of criteria. For instance, "Facebook tries to read the contact list" is seen as criteria *Facebook* of class Applications, *Read* of class Actions, and *Contact list* of class Resources. Kagop also computes all the possible groups based on the criteria included in the request (for instance, *{Facebook,Read}* or *{Social apps, Read, Contact list}*). Figure 5 shows the decomposition of the Facebook request with our problem-solving model. The result is 3 criteria, 7 meta-criteria,

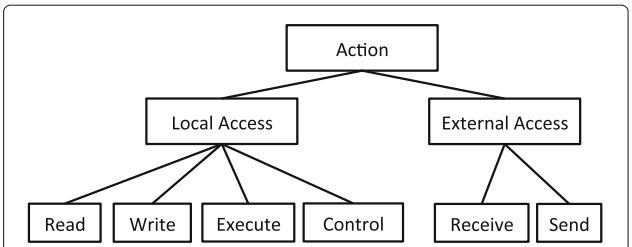


**Fig. 4** Hierarchy of criteria for class Actions

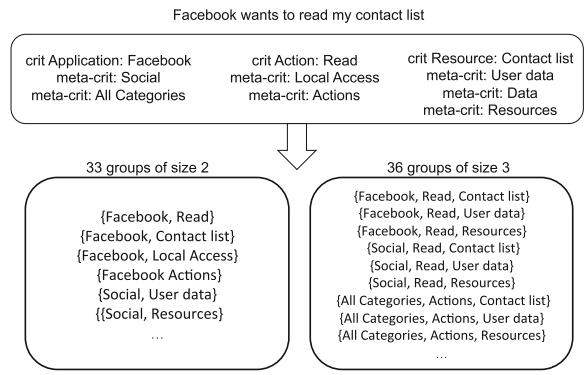


**Fig. 5** Decomposition of a request in criteria, meta-criteria, and groups of criteria

33 groups of criteria of size two, and 36 groups of size three. Groups of size three are actually the possible alternatives of abstract rules that Kapuer can propose to the user. Groups of size two helps to better evaluate interactions between criteria. At the same time, Kapuer asks the user to make a decision (Accept or Deny the request) and records it (see Fig. 6). This interaction is displayed only if Kapuer has no authorization rule to matches the request (corresponds to the step 3 in Fig. 1). This decision allows Kagop to update $f^t$ and $g^t$ for every criteria, meta-criteria, and groups.

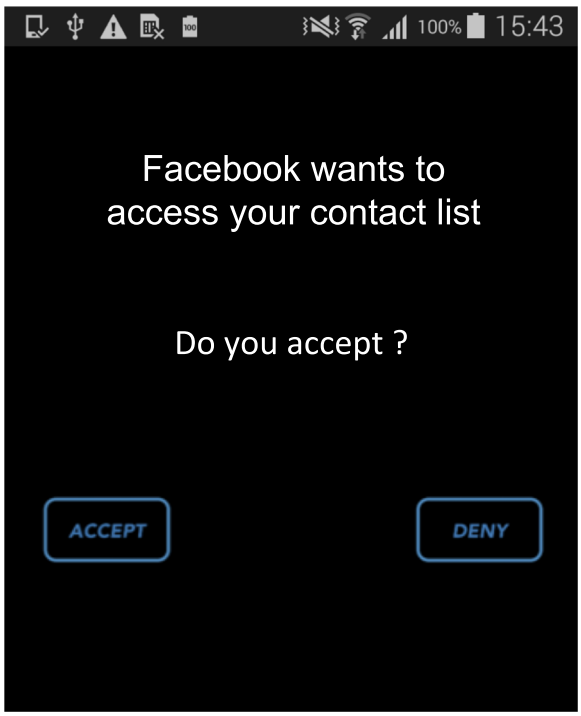


**Fig. 6** Interactions of Kapuer during the learning phase

If later, the user accepts to disclose the contact list to Twitter and to Instagram, the value $g^t$ of alternative *{Social apps, Read, Contact list}* will increase each time whereas low-level alternatives *{Facebook, Read, Contact list}*, *{Twitter, Read, Contact list}*, and *{Instagram, Read, Contact list}*) will increase once only. Thus, the preference score $s_D^t$ of alternative *{Social apps, Read, Contact list}* will be higher.

By learning preferences on criteria, meta-criteria, and groups of criteria, Kapuer is able to understand this kind of relations and to propose to the user a rule that permits the disclosure of the contact list to all social applications. Kagop calculates the utility score of an alternative as follows.

Let $A_R$ be an alternative of request $R$ and $C_{A_R}$ the set criteria of alternative $A_R$. The set of possible criteria and meta-criteria for building alternatives is:

$$PCA = C_{A_R} \cup (\cup_{x \in C_{A_R}} \{y | (x, y) \in H_{class(x)}\}) \quad (2)$$

Finally, the set $PGA$ of possible valid groups built on alternative $A_R$ is formed by the intersection between the set of all valid groups $G$ and the power set of $PCA$ such as:

**Algorithm 1** Score $S_{A_R}$ calculus

**if** the request was accepted **then**

$$S_{A_R} = \frac{\sum_{\forall c \in PCA} s_D^t(c) + \sum_{\forall g \in PGA} s_D^t(g)}{m}$$

**else**

$$S_{A_R} = \frac{\sum_{\forall c \in PCA} s_{nD}^t(c) + \sum_{\forall g \in PGA} s_{nD}^t(g)}{m}$$

**end if**

where $m$ is the number of groups.

Once the system has calculated the scores of all alternatives, it selects the best one to update the user's preferences (i.e., $f^t$ and $g^t$ of criteria and meta-criteria in $PCA$ and groups in $PGA$).

### 5.2 Preferences update

The best alternative is considered by our system as the most representative of the user's behavior. As consequence, we calculate the weighting factor used to update the preferences values of all criteria, meta-criteria, and groups of criteria based on this alternative score, noted $S_{A_R}^{best}$.

Updating the preferences values allows Kapuer to refine its representation of the user's preferences. When the request has been accepted, only $g^t(x)$ for all $x \in PCA \cup PGA$ is updated. Conversely, when the request is denied, only $f^t(x)$ is modified. Then, if a user behaves consistently,

only $g^t(x)$ or $f^t(x)$ will have high values resulting in high values of $s^t_D(x)$ or $s^t_{nD}(x)$. High values of preferences scores for some criteria, meta-criteria, and/or groups $x$ (either $s^t_D(x)$ or $s^t_{nD}(x)$) denote a good representation of the user's preferences on them.

The update calculus is a very important step of the preferences learning. Indeed, it determines the learning speed and accuracy of the algorithm. Choosing good update algorithm is complex and requires the appropriate balance between maximizing the accuracy of the preferences and minimizing interactions with the users. If the system learns users' preferences slowly, it needs to interact more often with the user but preferences values are more representative. On the contrary, if the system learns preferences quickly, interactions with the user are less frequent, but there is a risk of less accurate preferences, which might lead to irrelevant propositions. We have developed a simulator [26] that provides metrics to evaluate the accuracy and the number of interactions during a learning process. The simulator allows to specify a given user model through a set of predefined privacy policies. Then, the simulator can create random requests, automate the interactions with users by using the predefined user model, and execute the learning algorithm. This simulator allowed us to determine the update calculus by experimentation.

The value $M_c$ to update criteria and groups, and the value $M_{mc}$ to update meta-criteria are calculated as follows:

$$\begin{aligned} M_c &= log(S^{best}_{A_R}) \\ M_{mc} &= \frac{log(S^{best}_{A_R})}{n_c \times n_l} \end{aligned} \tag{3}$$

where $n_c$ is the number of criteria depending on the meta-criterion and $n_l$ the number of level between the criteria level and the one of the meta-criterion.

Based on these update values, we calculate the new values of criteria, meta-criteria an groups as follows:

The update affects all the criteria, meta-criteria, and groups of criteria involved in the request. Once the update is finished, the system verifies if the best alternative can be proposed to the user.

### 5.3 High-level proposition

Proposing abstract rules to users is the last step of our process. After the update, a proposition score is calculated and, if it is high enough, Kapuer interacts with the user to propose him a new high-level rule. These rules must have an interest for the user otherwise he does not accept them. An irrelevant abstract rule is no more than a disturbance and can lead the user to stop using the system.

Interaction in Fig. 7 is an example of a high-level proposition. It offers information about all abstractions used in the rule. Here, there are only meta-criteria so Kapuer details all criteria contained in each meta-criteria to help the user understand exactly what this new rule will do if he accepts it.

**Algorithm 2** Update calculus

```
for all x ∈ PCA ∪ PGA do
  if x is a criterion or a group then
    if the request was accepted then
      g^{t+1}(x) = g^t(x) + M_c
    else
      f^{t+1}(x) = f^t(x) + M_c
    end if
  else if x is a meta-criterion then
    if the request was accepted then
      g^{t+1}(x) = g^t(x) + M_mc
    else
      f^{t+1}(x) = f^t(x) + M_mc
    end if
  end if
end for
```

Making a proposal implies that the action associated to it is strictly preferred as its contrary. In other words, either the user agrees to disclose the resource either he does not. If the system cannot make a proposal, it comes from no real preferences between the two actions. Two situations can lead to this non-preference:

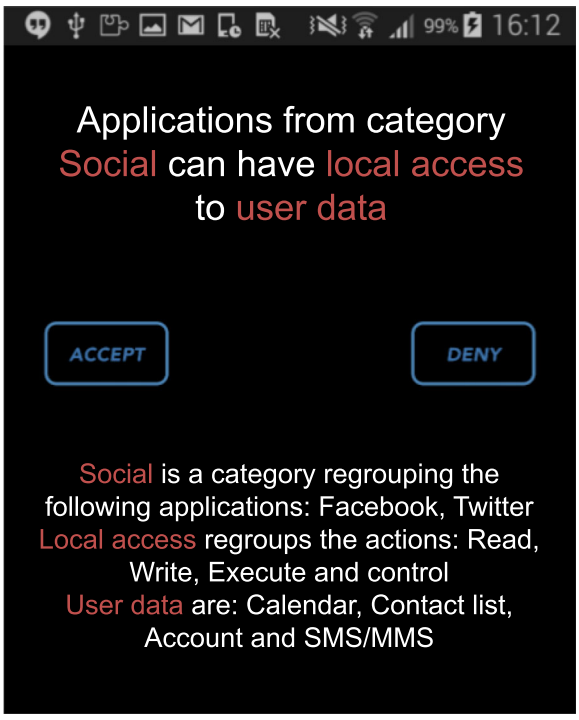


**Fig. 7** Proposition of a new high-level rule

– The system does not have an accurate representation of the user behavior, therefore lacks data on his privacy preferences;
– The user does not behave consistently. In that case, the system cannot infer his behavior and make a proposition.

This situation of non-preference is captured in the relational system of preference [27] that consists in the following two fundamental situations:

– Indifference $\sim$ or non-preference where there are no clear reasons to choose one action on the other.

$$\sim: a \sim a' \Leftrightarrow aIa' \tag{4}$$

$I$ is a binary relation, symmetric, and reflexive.

– Strict preference $\succ$ where there exist clear reasons to justify that one action is significantly preferred to the other.

$$\succ: a \succ a' \Leftrightarrow aPa' \tag{5}$$

$P$ is a binary relation, asymmetric, and irreflexive.

Kapuer recalculates with Algorithm 1 the score $S_{A_R}^{t+1}$ of last best alternative with the new values $f^{t+1}(x)$ and $g^{t+1}(x)$. Then this score is compared to $\lambda$, a parameter corresponding to the threshold between the relation of indifference and of strict preference. If $S_{A_R}^{t+1}$ is lower than $\lambda$, there is indifference and no proposition can be made. If $S_{A_R}^{t+1}$ is greater than $\lambda$, there is strict preference and the rule composed by the criteria and meta-criteria in the alternative can be proposed to the user. $\lambda$ is a parameter that affect the speed of proposition making. A small value leads to make propositions faster with less time to learn preferences. A high value leads to more time to learn preferences and also more time before propositions are made. We used our simulator to run experimentation, and we conclude that the most satisfying value for $\lambda$ is 3.5.

When $S_{A_R}^{t+1}$ is greater than $\lambda$, a new proposition is made to the user during a new interaction. The system proposes to add in the policy base a new rule where the attributes are the criteria and meta-criteria of the chosen alternative combined with the right decision. The user can accept or refuse the rule. Because the rule is abstract and can contain meta-criteria, details about them must be present. Then, the user can take an informed decision about his privacy. If he accepts the abstract rule, the DSS transforms it into XACMLv3 and adds it to the access control policy database to be considered for next requests by the authorization engine.

## 6 Post learning permission management

Finally, Kapuer also offers a more classic interface for viewing the privacy policy and features to modify it. Figures 8, 9 and 10 shows three screenshots of the application. Users can access the list of all rules and have a description for each one (Fig. 8). They can also edit each rule and modify it on four parts: the application, the resource, the action, or the decision (Fig. 9). Except for the decision, they can modify the level of abstraction if needed. We added a summary of all permissions granted to an application (Figure 10). This view provides detailed information about permissions handled by Kapuer.

## 7 Evaluation of Kapuer for managing Android permissions

We evaluate in this section Kapuer's and Kagop's efficiency against a real life scenario. Firstly, we compare Kagop to two other well-known aggregation operators. Secondly, we compare Kapuer to Privacy Guard Manager. Finally, we evaluate the speed of the learning process of our approach.

We setup our experimentation as follows. The same scenario is used in the three evaluations. First, we installed the 50 most downloaded free applications in the Google Play Store. It resulted in 28 games, 4 social apps, 4 communication apps, 4 widgets apps, 3 tools, 3 entertainment apps, 2 apps about music, and 2 apps about travel. Then, we defined the following arbitrary user privacy preferences model expressed through the following high-level authorization policy:

– *Rule 1*: Games can access the Internet.
– *Rule 2*: Social applications can access network and system data.

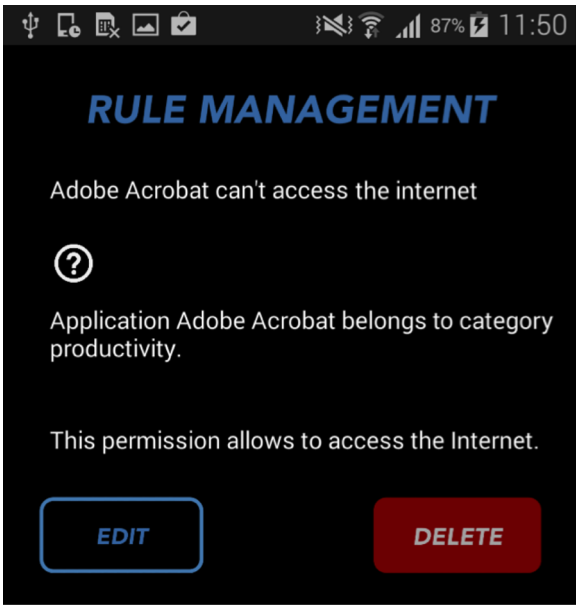


**Fig. 8** Rule information screen

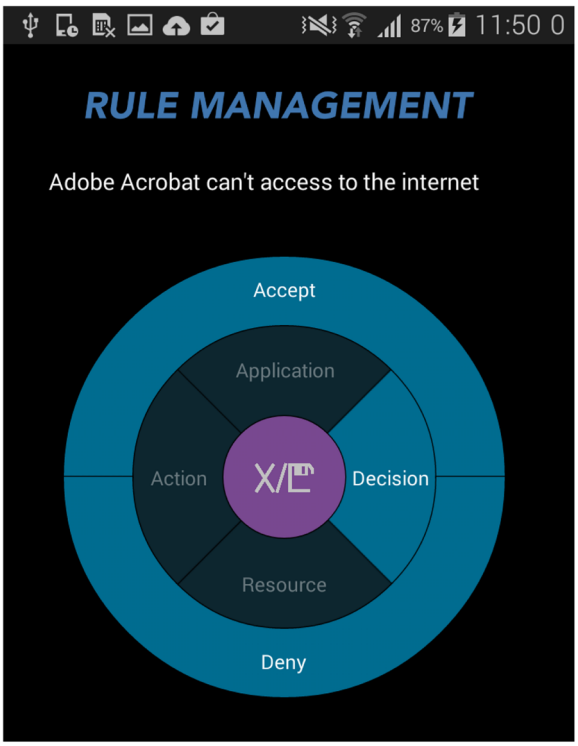


**Fig. 9** Rule modification screen

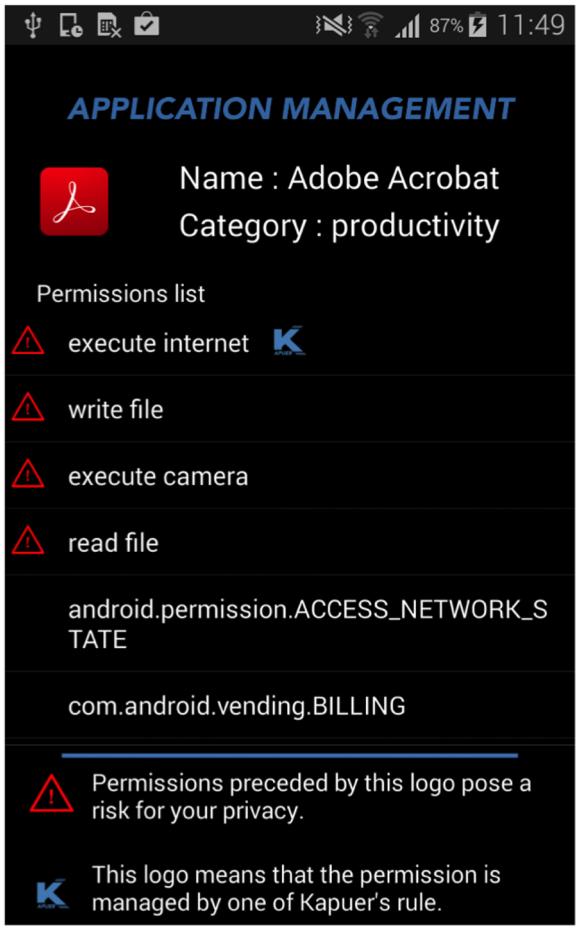


**Fig. 10** Application information screen

- *Rule 3*: Communication applications can access network and services.
- *Rule 4*: Widget applications can access the Internet.
- *Rule 5*: Music&Audio applications can access network and audio.
- *Rule 6*: Tools applications can access everything.
- *Rule 7*: Travel&Local applications can access network and GPS.
- *Rule 8*: Entertainment applications can access the Internet.

### 7.1 Kagop vs other aggregation operators

A multicriteria analysis requires the use of an aggregation operator to reach the goal of a decision support system: suggest one or many solution(s) to the decision maker to help him to make a decision. We have presented in this article *Kagop* our own aggregation operator, built to work with Kapuer's problem-solving model. In this section, we compare Kagop with two other well-known aggregation operators: the *weighted mean* and the *Choquet integral*.

(1) The weighted mean is an aggregation operator often used for its simplicity and good result. It does not take into account interactions between criteria. The weighted mean is defined by:

$$\psi(a_1, \ldots, a_n) = \sum_{i=1}^{n} w_i a_i \tag{6}$$

with $a_i$ the value of a criterion and $w_i \in [0, 1]$ its weight such as

$$\sum_{i=1}^{n} w_i = 1 \tag{7}$$

(2) The Choquet integral is an aggregation operator introduced by the french mathematician Gustave Choquet [28]. It takes into account the importance of class of criteria and interaction between criteria. Conversely to the weighted mean that uses a weight on each criteria to aggregate their scores and gives a global score to an entity, the Choquet integral employs a capacity (also known as fuzzy measure) to calculate weights for all groups of criteria. A capacity is defined as follows:

Let $N = 1, \ldots, n$ a set of criteria. A capacity on $N$ is a function $\mu$: $2^N \rightarrow [0, 1]$ with $\mu(\emptyset) = 0, \mu(N) = 1$ and $\mu(A) \leq \mu(B)$ if $A \subseteq B$.

Given $\mu$ belong to the set of all capacity on $N$, the Choquet integral of $x \in N$ with respect to $\mu$ is defined by:

$$\mathcal{C}_\mu(x) := \sum_{i=1}^{n} x_{\sigma(i)} \left[\mu\left(A_{\sigma(i)}\right) - \mu\left(A_{\sigma(i+1)}\right)\right] \quad (8)$$

where $\sigma$ indicates a permutation of $N$ such that $x_{\sigma(1)} \leq \ldots \leq x_{\sigma(n)}$. Also $A_{\sigma(i)} = \sigma(i), \ldots, \sigma(n)$ and $A_{\sigma(n+1)} = \emptyset$. The fuzzy measures in the Choquet integral allows to understand the dependance between criteria and also the importance of each criterion. Let have an example with 3 criteria and the following values of $\mu$:

| A | $c_1$ | $c_2$ | $c_3$ |
|---|---|---|---|
| $\mu(A)$ | 0 | 0.2 | 0.2 |
| A | $c_{12}$ | $c_{13}$ | $c_{23}$ |
| $\mu(A)$ | 0.8 | 0.8 | 0.4 |

When looking at the first line of the table only, it seems that $c_1$ is not useful because its capacity is equal to zero. However, we can see that when it is included in a group with another criterion, it affects the group value. Thus, $c_{12} > c_1 + c_2$ and $c_{13} > c_1 + c_3$. We used the plug-in Kappalab [29] available for R to implement the Choquet integral in our tests.

We implemented the three aggregation operators in our simulator [26]. In this way, we could evaluate each operator on the same set of requests and the same user privacy preferences model. We performed 10 simulations of 5000 requests each. The number of requests might seem very high. Nevertheless, it ensures the whole policy is enforced at the end of each simulation. In fact, requests being randomly generated, the learning phase could be incomplete with a lower number of requests. Aggregation operators weighted mean and Kagop have been used without specific tuning. However, the performance of Choquet integral can be erratic depending on the scenario [7]. Thus, we have configured it to have good result with this specific experiment.

The Fig. 11 shows how many authorization rules were created for each operator and each simulation. We can see that the results of Choquet integral and Kagop are close with an average of respectively 35 and 42 rules created. The weighted mean results are higher with an average of 74 rules created. There are a lot of interactions between criteria in our test; therefore, the weighted mean learns the user's preferences slowly. It generates more rules to complete the same user privacy preferences model because the rules created by the weighted mean have a lower level of abstraction and depends more on criteria than on meta-criteria. Figure 12 displays the number of interactions needed to fully cover each rule of the user privacy preferences model. Figure 13 shows the average of all interactions for each operator. Generally, we can see that the weighted mean needs much more interactions than the two other operators. Predictably, not taking into account interactions between criteria decreases significantly the efficiency of the learning phase.

Except for rule n1, Choquet integral and Kagop results are pretty similar. Rule n1, which states that games can access the Internet, is special. Indeed, in our test, meta-criterion *Games* relates to 28 applications (out of 50). This proves Kagop has good results when meta-criteria are linked to lot of criteria.

These comparative results should be balanced by the fact that the Choquet integral has required an initial configuration dependant on the user's preferences and a finite set of criteria. This is a serious issue to deploy this operator in practice. We cannot configure the operator each time the user installs or removes an application. We need a generic configuration working with everybody. Kagop does not need any initial configuration to work. It learns privacy preferences on criteria, meta-criteria, and groups of criteria to best fit on the user's behavior. But this advantage is also its main drawback. Groups of criteria are function of the set of criteria and meta-criteria and also of the number of classes of criteria. The size of a group is equal to the number of classes of criteria. There is a risk of combinatorial explosion. Nevertheless, with three classes of criteria, the number of groups created is

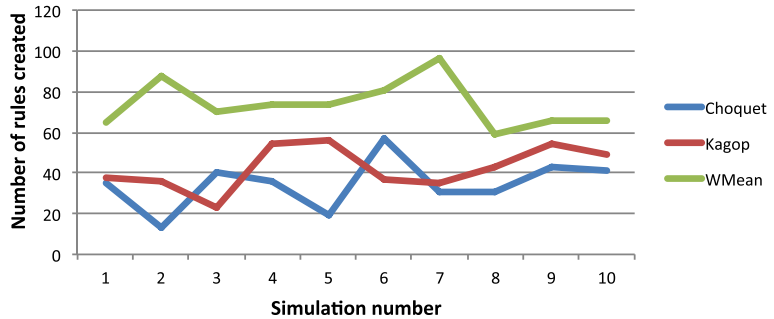


**Fig. 11** Number of created autorization rules

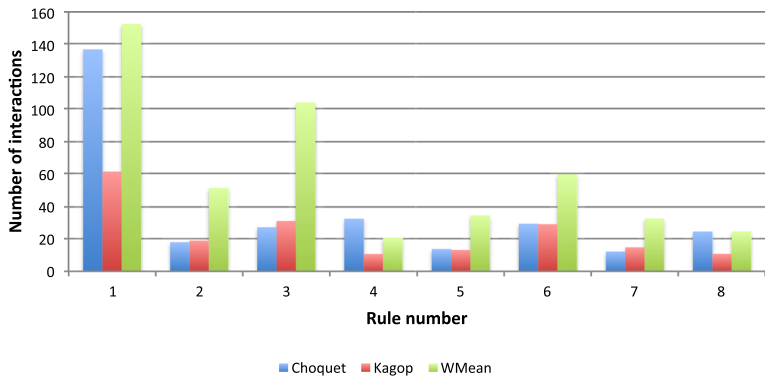


**Fig. 12** Number of interactions needed to cover the user privacy preferences model

reasonable. This evaluation proves Kagop is suitable for learning android high-level permissions.

### 7.2 Kapuer vs Privacy Guard Manager

We also evaluated the cost of writing this high-level policy using a classic system such as Privacy Guard Manager (PGM) and Kapuer. This cost is calculated by the number of actions the user has to perform. For PGM, an action consists in all pressures (screen navigation and on/off flipswitches selection). For Kapuer, any interaction as explained in Figs. 6 and 7 is an action. Since Kapuer learning process is not predetermined and depends on the received requests, a large number of tests must be executed to get its average behavior. Thus, we also used our simulator that can automate this task. We ran 10 simulations with the same high-level policy but with different requests each time since they are generated randomly. After each simulation, we checked the rules recommended by Kapuer.

Figure 14 illustrates the number of actions needed to write each rule of our privacy policy. It also shows that Kapuer needs fewer actions than Privacy Guard Manager for each rule. There is even a huge difference on the first rule. It concerns all the applications with the category "Games" and they represent 28 applications out of 50 so more than the half. For this rule, Kapuer needs 61 actions and PGM 476, so nearly eight times more. For rules number 2 and 3, the difference between Kapuer and PGM is also important when it is closer for all the others. This is due to the few number of applications involved in the last rules. If we compare the whole policy, Kapuer required 190 actions only when PGM has needed 848 actions. The abstractions in the high-level rules proposed by Kapuer are really providing a faster process for the user to fulfill his privacy policy. We have also looked at these high-level rules in details to see if some of them do not fit the user's preferences. None of the created rules proposed the opposite of what the high-level policy stated. Nevertheless, some of them did not have the right level of abstraction. It happens that a rule is proposed to the user with a higher abstraction than needed so it does not totally fit the user's behavior.

### 7.3 Evaluation of the speed at which Kapuer learns privacy preferences

The number of interactions needed to recreate one rule or all the policy provides information about the effort needed by users. It is also interesting to see how fast Kapuer is

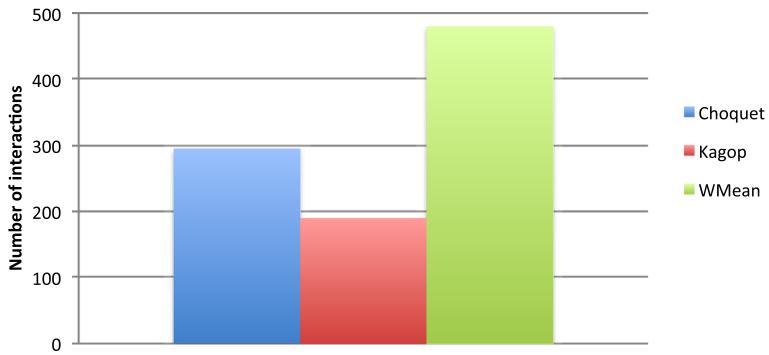


**Fig. 13** Average number of interactions

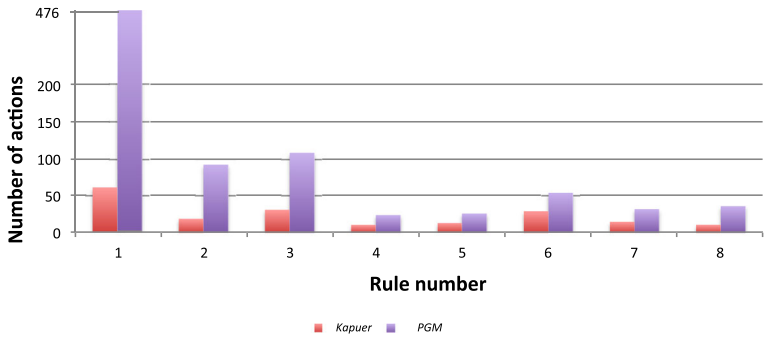


**Fig. 14** Comparison of Privacy Guard Manager and Kapuer

learning and how the level of completeness progresses. We compared, for each simulation, how many requests were needed to reach, 20, 50, 80, and 100% of completeness. The results are shown in Fig. 15. At the beginning, Kapuer does not know anything about the user's preferences. It needs requests to learn his behavior and to start proposing high-level rules. The first rule is proposed on average after 50 requests. Then, the average number of requests to reach the first threshold, 20% of completeness, is 75. For the second threshold, 50%, the average number of requests is 104. For the 80% threshold, the average number of requests is 123, and finally 100% of completeness needs an average of 190 requests. From these simulations, we calculated the global average completeness. Figure 16 shows the average of completeness at 10, 20, 30%, etc. It confirms that after learning from the first requests, Kapuer proposes regularly rules until the policy is nearly complete. Then, the process to achieve 100% of completeness is slower. This learning speed could still be improved. Today, Kapuer starts without any initialization and has to learn users preferences from scratch. With an initialization on these preferences, the number of requests needed to propose the first rule could be reduced.

## 8 Conclusions

We have presented in this article a custom permission management system for Android 4.4. Unlike other approaches, Kapuer does not only provide a way to modify what permissions an application can use. It learns from users' behavior to help them and advise them by proposing rules with different levels of abstractions. This way, they can protect their privacy more easily, without needing knowledge about access control models or policy's structure. We have also presented a detailed description of our learning engine. Kagop exploits all the concepts of our problem-solving model to take into account interactions and conflicts between criteria. Compared to two other methods, Kagop achieved good results. Evaluations on Kapuer show that hundreds of permissions can be handled with a limited number of actions by using abstractions. When Android 6.0 sacrifices control of privacy for the sake of simplicity with concepts of protection levels

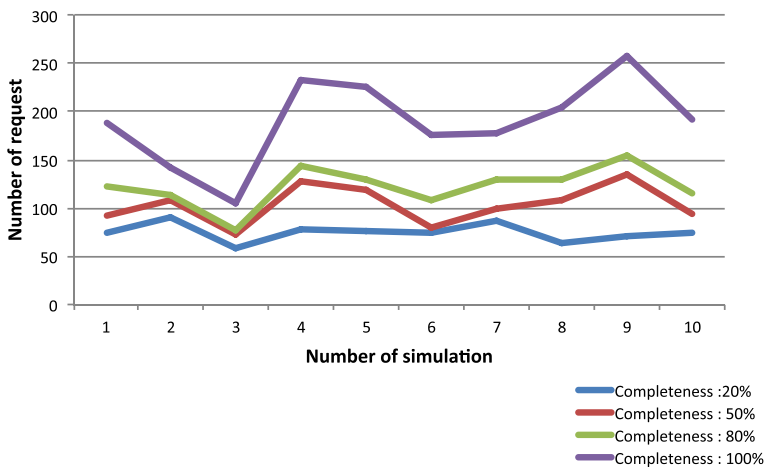


**Fig. 15** Completeness comparison

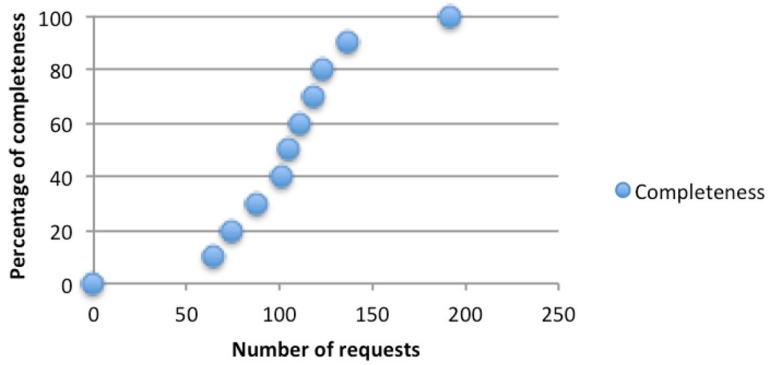


**Fig. 16** Average of completeness

and groups of permissions, Kapuer learns and proposes both fine-grained and abstract privacy rules. In addition, Kapuer supplies users with features to control the level of abstraction of privacy rules.

The current version of Kapuer runs on Android 4.4. For short-term future works, we will upgrade it to Android version 6.x or later in order to benefit from the new permission management system introduced in Android 6.0. For the moment, each time a request is denied, Kapuer makes Android act as if the application does not have the permission. Since developers of Android 4.4 applications do not manage this case, some applications crashed. This issue will be resolved on Android 6.0 because now developers shall handle permission verification before trying to use it. We will also take advantage of the new Android permission request interception to implement our interactions with users. Finally, we will integrate new Android 6.0 information (protection levels and groups of permission) as new meta-criteria. As a consequence, Kapuer will be easier to maintain.

One of the initial goals when we designed Kapuer was to inform people about privacy risks. For longer term research, we want to go further in that direction and not only inform people but also educate them about privacy issues. As an example, we need to explain them the consequences of granting some permissions to an application using approaches like privacy mirrors [30]. The more people understand these risks, the better their privacy decisions will be.

Finally, Kapuer learns users preferences from scratch. A significant number of requests is needed before any proposition can be made to the user. It is possible to improve the beginning of the learning phase by initializing the system. Making surveys with different kind of users can help to find the best way to initialize these users' preferences.

#### Abbreviations

DSS: Decision support system; IBAC: Identity-based access control; KAPUER: Kapuer is an assistant for protection of users personal information; KAGOP: Kapuer aggregation operator; MCDA: Multi-criteria decision analysis; OrBAC: Organization-based access control; PBAC: Purpose-based access controle; P-RBAC: Privacy-aware role based access control; PGM: Privacy guard manager; PEP: Policy enforcement point; PDP: Policy decision point; RBAC: Role based access control

#### Acknowledgements

Not applicable.

#### Funding

Not applicable.

#### Availability of data and materials

Not applicable.

#### Authors' contributions

AO participated to the conception and the design, implemented the system on Android, and helped to draft the manuscript. RL participated to the conception and the design and helped to draft the manuscript. PZ participated to the conception and the design. AB participated in the evaluation and helped to draft the manuscript. FB participated in the evaluation and helped to draft the manuscript. All authors read and approved the final manuscript.

#### Authors' information

Not applicable.

#### Ethics approval and consent to participate

Not applicable.

#### Consent for publication

Not applicable.

#### Competing interests

The authors declare that they have no competing interests.

## Publisher's Note



Received: 14 February 2017 Accepted: 25 June 2017
Published online: 12 July 2017

#### References

1. SD Warren, LD Brandeis, The right to privacy. Harvard law review, 193–220 (1890). JSTOR
2. M Langheinrich, *Privacy in ubiquitous computing*. (Chapman & Hall Crc, 2012)
3. E Cauvin, La vie privée sur internet expliquée par secret story (2015). http://tempsreel.nouvelobs.com/rue89/rue89-nouveau-monde/20150922.RUE2054/la-vie-privee-surinternet-expliquee-par-secret-story.html. Accessed 7 July 2017

4. DJ Solove, A taxonomy of privacy. U. Pa. L. Rev. **154**, 477 (2005). HeinOnline
5. OECD, *OECD Science, Technology and Industry Scoreboard 2015*. (OECD Publishing, 2015). /content/book/sti_scoreboard-2015-en. doi:http://dx.doi.org/10.1787/sti_scoreboard-2015-en
6. A Oglaza, P Zaraté, R Laborde, in *Joint International Conference on Group Decision and Negotiation*. Kapuer: A decision Support System for protecting privacy (Springer, 2014), pp. 100–107
7. A Oglaza, P Zarate, R Laborde, Annals of Data Science. **1**(3-4), 369 (2014). doi:10.1007/s40745-014-0027-3, http://dx.doi.org/10.1007/s40745-014-0027-3
8. A Oglaza, R Laborde, A Benzekri, F Barrère, in *Availability, Reliability and Security (ARES) 2016 11th International Conference on*. A Recommender-Based System for Assisting Non-technical Users in Managing Android Permissions (IEEE, 2016), pp. 1–9
9. S Barker, in *Proceedings of the 14th ACM symposium on Access control models and technologies*. The next 700 access control models or a unifying meta-model? (ACM, 2009), pp. 187–196
10. R Sandhu, D Ferraiolo, R Kuhn, in *ACM workshop on Role-based access control,* vol. 2000. The NIST model for role-based access control: towards a unified standard, (2000), pp. 1–11
11. F Cuppens, A Miège, in *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*. Modelling context in the Or-BAC model (IEEE, 2003), pp. 416–425
12. JW Byun, E Bertino, N Li, in *Proceedings of the tenth ACM symposium on Access control models and technologies*. Purpose based access control of complex data for privacy protection (ACM, 2005), pp. 102–110
13. Q Ni, E Bertino, J Lobo, C Brodie, CM Karat, J Karat, A Trombeta, ACM Trans-actions on Information and System Security. (TISSEC). **13**(3), 24 (2010)
14. M Conti, B Crispo, E Fernandes, Y Zhauniarovich, Inf. Forensic. Secur. IEEE Trans. **7**(5) (1426). doi:10.1109/TIFS.2012.2204249
15. Y Zhauniarovich, G Russello, M Conti, B Crispo, E Fernandes, Dependable and Secure Comput. IEEE Trans. **11**(3), 211 (2014). doi:10.1109/TDSC.2014.2300482
16. Oasis Xacml Committee, extensible access control markup language (xacml) version 3.0. https://www.oasis-open.org/committees/xacml/. Accessed 7 July 2017
17. V Arena, V Catania, G La Torre, S Monteleone, F Ricciato, in *Privacy and Security in Mobile Systems (PRISMS), 2013. International Conference on*. Securedroid: An android security framework extension for context-aware policy enforcement (IEEE, 2013), pp. 1–8
18. B Stepien, A Felty, S Matwin, in *Collaboration Technologies and Systems (CTS), 2014. International Conference on*. A non-technical XACML target editor for dynamic access control systems (IEEE, 2014), pp. 150–157
19. A Oglaza, R Laborde, P Zarate, in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013. 12th IEEE International Conference on*. Authorization policies: Using decision support system for context-aware protection of user's private data (IEEE, 2013), pp. 1639–1644
20. PG Keen, MSS Morton, et al., Decision support systems: an organizational perspective (1978)
21. G Adomavicius, A Tuzhilin, Knowledge and data engineering, IEEE Transactions on. **17**(6), 734 (2005)
22. AJ Jeckmans, M Beye, Z Erkin, P Hartel, RL Lagendijk, Q Tang, in *Social media retrieval*. Privacy in recommender system (Springer, 2013), pp. 263–281
23. A Friedman, BP Knijnenburg, K Vanhecke, L Martens, S Berkovsky, in *Recommender systems handbook*. Privacy aspects of recommender systems (Springer, 2015), pp. 649–688
24. B Rashidi, C Fung, T Vu, in *Integrated network management (IM), 2015. IFIP/IEEE International Symposium on*. Dude, ask the expert!: Android resource access permission recommendation with recdroid (IEEE, 2015), pp. 296–304
25. M Zeleny, JL Cochrane, *Multiple criteria decision making*, vol. 25. (McGraw-Hill, New York, 1982)
26. A Oglaza, R Laborde, P Zaraté, in *Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual*. Difficulties to enforce your privacy preferences on Android? Kapuer will help you (IEEE, 2016), pp. 315–316
27. E Jacquet-Lagrèze, B Roy, et al., *Aide à la dècision multicritère et systèmes relationnels de prèfèrences*. (LAMSADE, 1980)
28. M Grabisch, M Roubens, Application of the Choquet integral in multicriteria decision making. Fuzzy Measures and Integrals-Theory and Applications, 348–374 (2000). Berlin: Springer
29. M Grabisch, I Kojadinovic, SP Nantes, P Meyer, in *Proceedings of the 11th international conference on information processing and management of uncertainty in knowledge-based systems*. Using the Kappalab R package for capacity identification in Choquet integral based MAUT, (2006), pp. 1702–1709
30. DH Nguyen, ED Mynatt, Privacy mirrors: understanding and shaping socio-technical ubiquitous computing systems. Tech. rep., Georgia Institute of Technology (2002)