KAPUER: A Decision Support System for Protecting Privacy

Arnaud Oglaza, Pascale Zaraté, Romain Laborde

Kapuer: A Decision Support System for Protecting Privacy

Arnaud Oglaza, Pascale Zaraté, and Romain Laborde

IRIT, University of Toulouse,
118 Route de Narbonne, F-31062 TOULOUSE CEDEX 9, France
{oglaza,zarate,laborde}@irit.fr

Abstract. Pervasive computing allows a world full of electronic devices connected to each other, autonomous, context aware and with a certain level of intelligence. They are deployed in our environment to ease our life. However today users don't control the traffic around their data. The use of mobile devices might increase this problem because the system is more complex and requests of personal data is transparent to users in order to reduce their cognitive load. The goal of Kapuer is to inform the user on requests about his privacy and help him protect his privacy by assisting him in the writing of authorization policies. But a risk remains, informing too much the user can drown him into a huge amount of information and could quit using the system. The idea around Kapuer is to make the user conscious of the situation without bothering him too much.

Keywords: privacy, decision support system, access control.

1 Introduction

With the rising of mobile computing and the increasing amount of devices connected to the Internet, exchanges of data between those devices have exploded. Disclosure of these data can put in danger privacy. The user of a device has to know about those exchanges of personal data and there must be a way for him to control them. Access-control systems allow to write authorization policies and then to control personal data. However, one needs some skills to write such policies. So it's not accessible to all users. In addition, because users have a different way to see their privacy and then a different idea on how to protect their personal data, it's not possible to have an administrator behind all users to write policies instead of users. In order to make accessible control of personal data by the user, we have developed a system, Kapuer, combining an access control system to write authorization policies and a decision support system (DSS) to understand the behavior of a user regarding privacy. In order to do that, the system needs to learn user's preferences in terms of privacy. The goal of our work is to develop a DSS, which proposes authorization policies to the user after having learned his preferences. The system uses interactions with the user to learn his preferences through a continuous learning during all the system execution . This preferences learning is based on multi-criteria decision making

(MCDM) [1]. We present in this article how we have designed Kapuer and also notions like meta-criteria, which help the system having better results. The rest of the article will be organized as follow. We first present access-control system. Then we present our way to model preferences. After, we introduce Kapuer and its architecture. Finally we conclude and discuss about future work.

2 Access Control System

Since their beginning, access control systems have evolved a lot, but their goal is still the same: using authorization policies to control access to resources. Some of these systems have interesting features. Role Based Access Control (RBAC) [7] systems are, as indicated in their names, using the notion of roles to build their authorization policies. These systems have emerged in the nineties at the same time than multi-users applications. Roles are useful as they allow administrators to group users according to their functions. Then permissions can be granted directly to roles instead of users. The number of authorization policies is lower than if administrators had to write one for each permission for each user. But RBAC is not accessible to ordinary people since grouping access authorization is a complex task. Moreover user's preferences about privacy being different for each one, an administrator behind each user to write their own authorization policies is mandatory.

Roles are efficient with lot of users but each one of them has to have well-defined roles. Every time a new user enters the system, the administrator has to give him one or many roles or this new user won't have any access. If we are working in an open and dynamic environment where users can come and go, RBAC is no more helpful. To deal with all these new situations, another approach has been proposed: Attribute Based Access Control (ABAC). Attributes are used to characterize all elements present in a policy. That way, contextual situations [2] can be described by using attributes for the user requiring access, the action he wants to perform, the resource he wants to access but also for all kind of environmental or contextual attributes like when is the resource requested or why is it requested. This allows more flexibility for administrators who no longer write policies based on users but on situations. This way, users can be added to the system and granted permissions without any additional tasks, they will be automatically recognised by their attributes. Once again, even if ABAC is designed for open environment and attributes allows a lot of flexibility, it doesn't fit our needs for the same problem as RBAC, an administrator has to be present to write the policies due to the complexity of the task.

3 Preferences Modelling

Our approach is based on a Multi-Criteria Decision Making model. In general situations, a criterion is the association of an identifier and a value. The value in the MDCA approach is associated to the weight of the criterion. The higher this value is, the more important this criterion is for the user. Here Kapuer is

used to learn user's preferences in terms of privacy. Then if a criterion has a low value and a request is received with this involved criterion, does it mean that the user considers that the data doesn't have to be disclosed or is it just that the system hasn't had the time to learn enough about that criterion? It is not possible to answer this question with criterion build in this way. To resolve this problem, for each criterion c_i, we don't use one value but two values. The first, g_{c_i} represent the user's preference for disclosure and the second f_{c_i} represent the user's preference for non-disclosure. These two values are correlated. During their update, g_{c_i} and f_{c_i} can only increase. The learning of preferences is continuous and infinite, a high value of g_{c_i} doesn't always mean the user agrees to disclose his data in the presence of this criterion. It can also mean that this criterion has shown pretty often among all requests and that is has been updated many times. To identify the signification of the two values of a criterion, we have to calculate the score s_{c_i} of the criterion c_i. This score is used by the DSS after an interaction with the user. During this interaction, the user has decided if he agrees or not to disclose the requested data. The calculation is $s_{c_i} = |f_{c_i} - g_{c_i}|$. This way, the score s_{c_i} determines the signification of c_i among the user's preferences. A low score reflects that there are no clear reasons for the user to choose between the two actions based on this criterion. On the contrary, a high score tells that this criterion is important for the user's decision. The technical part of the system (equations) will be considered as confidential for commercial purpose.

Each criterion is part of a class of criteria. We have defined six different classes, corresponding in the six types of attributes used for the access-control :

- **What** data to protect (phone number, contact list, etc.)
- **Who** wants to have access to data (John, Lisa, an unknown person, etc.)
- **When** is the access requested (the different days of the week, the different hours of the day, etc.)
- **Where** is the user at the time of the request (at home, at work, etc.)
- **How** data will be stocked (how long, who will have access, etc.)
- **Why** data will be used, what is the purpose. (charitable, to be sold, etc.)

A criterion can be seen as an instance of a class of criteria. For example the criterion ("John", 2.0, 3.0) is an instance of the class "Who", the name of the criterion is "John" and his values are "2.0" for f_{c_i} and "3.0" for g_{c_i}.

We introduce the notion of meta-criterion. A meta-criterion is a criterion defined by other criteria of the same class. Each criterion refers to a meta-criterion. A meta-criterion allows to group many criteria sharing a same property like roles group users. For example, we have two criteria ("Father", 1.0, 1.0) and ("Mother", 1.0, 1.0). The meta-criterion ("Parents", 3.0, 3.0) gathers the two criteria "Father" and "Mother". So a criterion is defined by a 4-uplet (identifier, associated meta-criterion, disclosure value, non-disclosure value). Then, we can redefined criteria "Father" and "Mother" as follow: ("Father", "Parents", 1.0, 1.0), ("Mother", "Parents", 1.0, 1.0).

A meta-criterion being also a criterion, he is defined similarly. Thereby it is possible for each class of criteria to create a hierarchy between its criteria

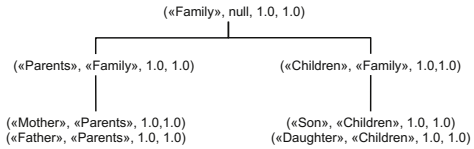

Fig. 1. Criteria's hierarchy

(Figure 1). There are two cases when a criterion doesn't have a meta-criterion to define his place in the hierarchy. When the criterion is on top of the hierarchy (for example, the criterion "Family" in Figure 1) or when the criterion doesn't belong to any group and can't be associated to a meta-criterion. This hierarchy of criteria is defined for the six class of criteria.

4 System Architecture

Kapuer is a combination of an access control system and a decision support system. Its goal is to provide assistance to a user who wants to protect his privacy. Writing authorization policies is a hard task, even for administrators so for a user without any skills in security, it is impossible. To design Kapuer, we have used a classic XACML [5] architecture build around our main component, the DSS (Figure 2).

Kapuer starts to work when a request is received. A request is a demand from a person or an application to access some personal data. Whenever a request is received,the Policy Enforcement Point (PEP) intercepts it. The PEP is a component used to translate the request into attributes. When the request is successfully translated, it's send to the Policy Decision Point (PDP), which pursue the process. The goal of the PDP is to verify if there is a match between existing authorization policies in the policy base and the request and if there is to give the associated decision. If there is a match, the decision of the rule is sent back to the PEP. The decision can have two different value, "Permit"

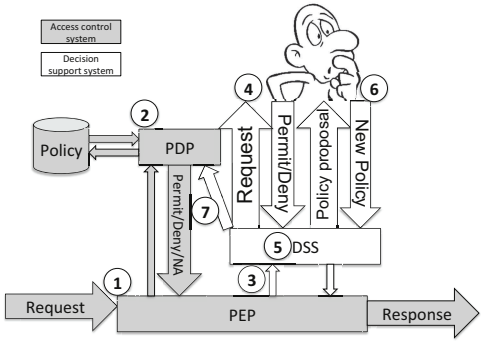

Fig. 2. Architecture of our approach

or "Deny". If there is no match between policies and the request, the decision "Not Applicable" is sent back to the PEP. This last case is when there is no rule to handle the request. Normally in that case, the designer of the access-control system implements a logic to still give an answer to the requester. In most of the case, the decision "Not Applicable" is seen as a "Deny" to prevent disclosure of data not managed by an authorization policy. In Kapuer, the decision "Not Applicable" only means that the actual knowledge of user's preferences is not enough to propose a new authorization policy and handle the request. Then Kapuer has to improve its learning and for this, it informs the user of the current request and ask him what is the decision to take. The user can choose between the two available actions, "accept" or "decline" the disclosure. When the user has taken his decision, the DSS analyses it in order to understand it and improve the user's preferences.

It is required, for a better understanding of the user's behaviour, to analyse why this decision was taken. In order to do that and to follow the MCDA approach, the request is decomposed into criteria. Meta-criteria intervenes here. Kapuer doesn't use the criteria of the request only but also all the combination of the criteria and their meta-criteria. Then we don't have a unique list of criteria but a set of lists. Once Kapuer has all those lists, it uses them and the actual user's preferences to calculate the score S_R of each list. To calculate these score, we aggregate the criteria using a aggregation operator. The result of S_R allows Kapuer to know if the user has a strict preference for the choice he just made or not. If there isn't a strict preference, Kapuer just updates all criteria and meta-criteria involved in the request. Otherwise it interacts again with the user to propose him to write a new authorization policy. Kapuer explains what the rule is and let the user decides if it really suits him. If that's the case, a new policy is written and placed in the policy base. On the contrary, if the policy doesn't suit the user, it is because there is an error of understanding in the preferences so criteria are updated with the new user's decision. At this time, Kapuer always have a decision to transmit to the PEP, which translates it into the language of the requester and then sends the final decision.

5 Evaluation

Evaluating our work is necessary in order to show its feasibility in real life. Without interaction, our system isn't able to learn preferences, thus assistance isn't possible. In order to obtain enough data to compare, we must have lots of users involved for tests and as many devices. To avoid those constraints, we have developed a simulator. We have implemented in this simulator a set of criteria and a hierarchy for each class. That way, we are able to generate easily a lot of requests by taking randomly one criterion of each class. Then the system takes these requests one by one and analyses them as explained in the previous section. The remaining problem is the lack of users to take decision during all interactions. To go beyond this issue, we have implemented a set of aurhotization rules to imitate an user's behaviour. Each time the system needs an interaction,

it will simply look at that list and check if one rule matches. All rules need to be exclusive to avoid doubts if two or more rules match the same request and give different decision.

The number of interactions between our system and the user should be limited in order to annoy him as less as possible. Using meta-criteria to aggregate policies is a way to decrease interactions, but it isn't enough. This number of interactions also depends on the preferences learned and how fast the DSS learns them. One of the parts that affect the most this learning is how the criteria's aggregation is done to calculate requests' scores and the criteria's update. Using the simulator is a good tool to test several aggregation operators. During our simulations, we have played each set of requests on three different operators:

- The weighted mean, an operator used in the majority of DSS because of its simplicity.
- The Choquet integral [3], a more complex operator that uses importance of each class of criteria and interactions between them for more accuracy. We choose the Choquet integral because we thought that for this problem, criteria are not independant. Its implementation isn't easy. We have used Kappalab [4], a plug-in of R to made our Choquet operator.
- Our own operator is between the weighted mean and the Choquet integral, working not only on the request's criteria but also with all the combinations possible of those criteria. We have used this mixed operator to look if combinations of criteria will help the system find possible interactions between two or more criteria.

We ran ten simulations of 200 random requests to compare the three different operators and see if one of them shows better results. We evaluate four different metrics. First the number of interactions. It shows the level of abstraction of each operator. The more policies are created, the lower-level they are. Then, it indicates if an operator can adapt itself to a system with more criteria. The second metric is the number of requests processed by policies. It shows the learning speed. The more requests are handled by policies, the faster preferences are learned. The third metric is the level of completeness, ie the ratio between the number of requests covered by the policies created and the number of requests possible with the behaviour's rules that simulate the user. It shows, after 200 requests, the percentage of requests that will be handled by Kapuer. The last metric is the number of interactions made during the 200 requests. It's the sum of the number of policies and the number of requests non-processed by policies. For our first tests, we have implemented three classes of criteria:

- **What** data to protect with six criteria. "Contact list" and "Calendar" with the same meta-criterion "Data". "Name" and "E-mail address" with the meta-criterion "Info". "GPS" and "Camera" with the meta-criterion "Service".
- **Who** wants to have access to data with nine criteria. "Jimmy", "Lee" and "Billy" with the meta-criterion "Family". "Bob", "Jay" and "Fred" with the criterion "Friend". "Pierrick" and "Mick" with the meta-criterion "Colleague" and "John" with the meta-criterion "Unknown".

– **When** is the access requested with fourteen criteria for each half-day and two meta-criteria ("Morning" and "Afternoon")

Finally, users are simulated by two different behaviours. The first one, B_{cx}, is complex. It agrees to share data with members of the family all the time, with colleagues on morning, with friends on afternoon and doesn't share anything with unknowns. The second behaviour, B_{op}, is open to all requests. The results are shown in Figure 3 (note that coordinates don't use the same scales).

The results show that there isn't an operator way above the others. No one have the best results in all four metrics. Kapuer is interesting for users if it handles the more possible requests. After 200 requests, whatever the behaviour, the three operators are above 80% of completeness. So more than four out of five further requests will be handled by the system. The mixed operator even reaches 98.9% with B_{op}. This level has to be confronted with the number of policies created. As we already said, the more policies are created, the lower-level they are. Then if we strongly increase the number of criteria in the system, the system needs more time to learn user's preferences. If the created policies are low-level, it will lead to a lower level of completeness. Then, if Choquet and the weighted mean have good results in those cases, our operator will have better results in a system with more criteria and more important, Choquet and the weighted mean are not stable, results can vary a lot between two simulations whereas our operator is very stable.

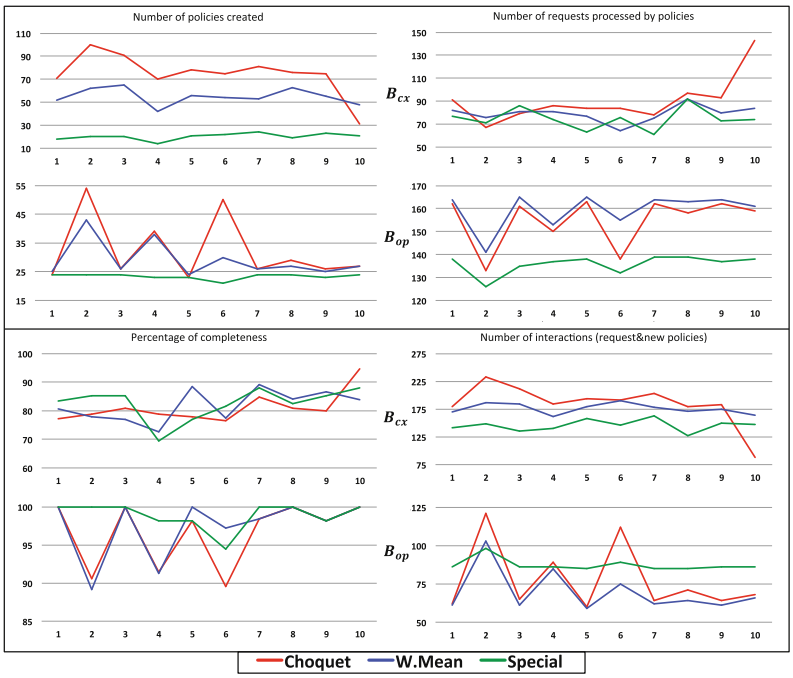

Fig. 3. Results of the simulations

The other important point for users is to limit interactions. We can see that a complex behaviour brings more interactions than an open one. The learning speed has an impact on interactions. Choquet and the weighted mean have more requests processed by policies. It shows that they create policies faster than our mixed operator, but with B_{cx} they create so many policies that they require more interactions than our operator. Three times during the simulations, Choquet and the weighted mean have had peaks in the results. Way more policies are created because they are low-level. With a low level of abstraction, the level of completeness is also lower and it increases the number of interactions. On the contrary, whatever the behaviour, our operator is more stable than the two other for the four types of metrics.

6 Conclusion and Future Work

Kapuer shows that a DSS can help the users protect their privacy, even if he has no skills in security. We have tested three operators. None is the best but our mixed operator has good results and is the most stable. A prototype of Kapuer [6] has been implemented on an Android device but it wasn't practical and fast enough to do lots of tests. This is why we have developped a simulator, allowing us to compare different operators on a same set of requests. The learning speed still needs to be increased. We are working on an initialization phase, few questions asked to the user before the first use of the system, to have a first idea of his preferences. It should allow Kapuer to be faster. Finally, we need to test Kapuer in real situations. To do that, we are currently improving the Android prototype and tests it with real users.

References

1. Bouyssou, D., Dubois, D., Pirlot, M., Prade, H.: Concepts et méthodes pour l'aide la d cision 3. Lavoisier (2006)
2. Dey, A.: Understanding and using context. Personal and ubiquitous computing (2001)
3. Grabisch, M.: Modelling data by the Choquet integral. In: Information Fusion in Data Mining, pp. 135–148 (2003)
4. Kappalab, `http://cran.r-project.org/web/packages/kappalab/` (last access December 2013)
5. OASIS XACML committee, eXtensible Access Control Markup Language (XACML) Version 2.0. (last access December 2013)
   `http://www.oasis-open.org/committees/xacml/`
6. Oglaza, A., Laborde, R., Zarate, P.: Authorization policies: Using Decision Support System for context-aware protection of user's private data. In: Proceedings of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom 2013) (July 2013)
7. Sandhu, R., Coyne, E., Feinstein, H., Youman, C.: Role-based access control models. Computer, 38–47 (1996)