Enhancing Security in the Cloud: when Traceability meets Access Control

Clara Bertolissi, Omar Boucelma, Worachet Uttha

Enhancing Security in the Cloud: when Traceability meets Access Control

Clara Bertolissi
Aix-Marseille Univ, CNRS,
Marseille, France,

Omar Boulcema
Aix-Marseille Univ, CNRS,
Marseille, France

Worachet Uttha
Nakhon Pathom Rajabhat University,
Thailand

***Abstract*—Cloud Computing technology is gaining momentum, however security concerns remain one of the top barrier to cloud projects. We propose a framework that ensures data control and privacy in the cloud by using traceability (aka Provenance) combined with expressive access control policies based on user categorization.**

I. Introduction

In a cloud environment, provenance may bring an added value to cloud providers. Simply stated, provenance consists in recording entities and activities involved in producing or transforming an object. Provenance may help answering questions such as: Who created this data ? What was the process used to create it ? When was it modified and by whom? Behind these questions, security issues such as data integrity, privacy, access control arise. In a distributed context, components/activities used during each step can locate on different sites each applying specific management policies.Because access control models and policies are the most known approaches to enforce protection on data and resources in a system, combining provenance with access controls may lead to an efficient system for enforcing trust in the cloud.

II. Framework description

We propose a solution combining access control features and systems' provenance data. For defining provenance, we adopt the PROV Data Model (PROV-DM [7]), a W3C Recommendation for provenance expression. It allows to represent objects and their dependencies as a directed acyclic graph composed of three vertice or object types (entity, activity, agent) and several types of edges, representing dependencies such as an activity used an entity, an entity was generated by an activity, etc (see Fig.1 for an example). For defining acces policies, in the aim of defining a framework as general as possible, we have chosen to adopt the CBAC metamodel[2] which has been shown to be expressive enough to accommodate a range of different access control models. We provide a set of rules that are checked each time a process (e.g.; data alteration) is invoked in a given system. These rules use provenance data and restrictions/authorization of users depending on the category they belong to. A category is a notion of grouping based on the attributes a user owns. We also consider dynamic categorization: the memberships of a user to a certain category (and thus her/his privileges), may be affected and change dynamically as a consequence of her/his actions. For instance, privileges can be automatically revoked depending on the number of refusals of (a specific) activity execution a user has received.

More precisely, in our framework, entities are denoted by constants in a many sorted domain including: a set agents, a set of named atomic activities, a set of entity identifiers, a set of categories. The core axiom of the model is as follows: $belongs\_to(agent, categ, entity) \wedge permission(categ, activity, entity) \Leftrightarrow allow(agent, activity, entity)$.

The idea is that the first relation, $belongs\_to$, specifies the "qualification" of the agent with respect to the attributes she/he has and the past (relevant) actions she/he has accomplished. The relation $permissions$ specifies whether the requested action can be performed against the object according to a certain level of qualification, i.e. a category. Notice that this relation is not dependent from the agent. It is used to model action validations by the system and it does not directly make use of user access privileges. This separation in an agent-dependent relation and a system-dependent relation easies the updates and maintenance operations of the policy. If the *agent* has a sufficient qualification level for belonging to a category to which the *activity* is permitted on the requested *entity*, then the access request *allow(agent,activity,entity)* is granted.

III. Related research

There are several research works that attempted to mix traceability with access control. Park *et al* [9] proposed PBAC, a model where the notion of object dependency lists, derived from process execution traces, are used for access requests evaluation. In [10], provenance-aware access control policies are discussed. An abstract provenance model TPM (Type Provenance Model) is proposed.TPM allows the expression of complex dependencies using regular expressions in a similar way as it is done in PBAC with object dependency lists. In [1], a provenance model (*cProv*) and a policy language are proposed. Their rules are generic and some of them are activity-oriented, very close to the an RBAC model, which can be expressed in CBAC. In [6] authors propose a formal model that assigns a trust degree (evaluated from provenance data) to each entity. The approach we propose provides strict authorization rules, instead of trust values granted to a particular type of activity per user. From our perspective, one the main ideas of our work is to come up with a tight integration of PROV and CBAC. The concept of category of users (or agents) has

no direct correspondence in the PROV-DM model, even with extended structures such as collections, which apply only to entities. The membership of an agent to a group can only be expressed in the past since a PROV document reflects only past and final facts. Even in PROV-O [5], the PROV Ontology, no relationships for the belonging of an agent to a group exist.

IV. Framework validation on an example

We choose Datalog programming for evaluation and testing of our model, since provenance data in PROV-N notation [8] can be easily expressed in Datalog and declarative policy rules are very similar to Datalog rules. [1]

In our example, we consider an academic course. The teacher asks students to submit 2 tasks via the University web application. Each task will be integrated incrementally by the web application on one student-specific file so that the teacher can view all one's work in one file. We assume that students can submit a task only after they have been graded on the previous one. Students cannot re-submit a task once the teacher has rated it. As all subject belonging to the same category will have the same privileges, in order to distinguish students having already uploaded some tasks, we may refine the category $student$ by adding two categories $uploadedT1$, $uploadedT2$. These categories are used for grouping students who have already uploaded the task 1 or 2, respectively, and may be defined as follows:

```
belong_to(Agent,uploadedT1,Entity):-
attribute(Agent, registered_student),
 wasGeneratedBy(Entity, uploadTask1),
 wasAssociatedWith(uploadTask1, Agent).
```

This rule means that agents qualified as registered students (as recorded in the University database) and having executed the action *UploadTask1* on the resource *entity* (as recorded in the dependency path of the corresponding provenance graph) are assigned to the category $uploadedT1$. In order to have information such as "how many times an entity was used"?, we may need to perform a computation on the provenance graph for calculating how many dependencies of each type an entity has. These will produce new predicates that may be used in the Datalog program.

```
permission(student, uploadTask1, Entity):-
wasUsedBy_count(Entity, rateTask1, 0).
```

The rules above means that a student is allowed to re-upload the same assignment if it has not been rated by the teacher.

Let us consider the scenario reported in Fig.1 with two students, *Bob* and *Mike*. By playing access queries we can simulate the system and verify the correctness of the access response. The termination of Datalog queries is guaranteed, see [3]. If we want to know whether Bob is allowed to upload his task 1 or not, we can ask to the Datalog engine the query

```
?- allow(mike, uploadTask1, taskMike).
false.
```

[1]The full Datalog specification is made available at `http://home.npru.ac.th/wuttha/research/PROV-CBAC`

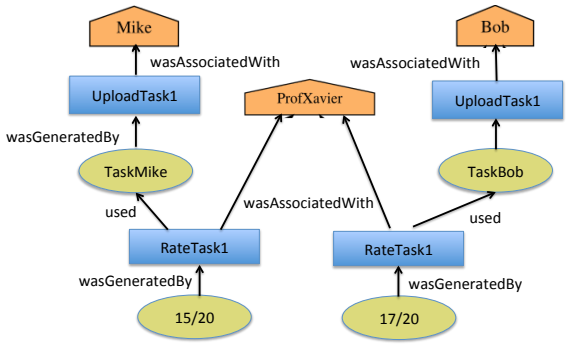

Fig. 1. Example: Excerpt of the Provenance Graph

We can also ask more general question by including variables in the query:

```
%% The only activity Mike can perform is upload task2
?- allow(mike, Activity, taskMike).
Activity = uploadTask2.

%% Mike is the only one who can upload his task2.
?- allow(Agent, uploadTask2, taskMike).
Agent = mike.

%% Mike is allowed to upload only his task.
?- allow(mike, uploadTask2, Entity).
Entity = taskMike.
```

This kind of simulation can help the policy designer or administrator to detect possible inconsistencies in the policy definition. Assuming the translation to Datalog is correct, the engine will compute a positive answer only if Bob is allowed to perform the access according to the specified policy.

V. Conclusion

We have described a framework that ensures trust in a distributed computing environment by means of data security enforcement. The framework combines Provenance and Category-based Access Control policies. We already have an implementation that accommodates RBAC policies [4] and we are planning to extend it with CBAC.

References

[1] M. Ali and L. Moreau. A Provenance-Aware Policy Language (cProvl) and a Data Traceability Model (cProv) for the Cloud. In In *Proc. of GreenCom'13*, p. 479–486, IEEE, 2013.

[2] Steve Barker. The Next 700 Access Control Models or a Unifying Meta-model? In *Proc. of SACMAT '09*, p. 187–196, 2009. ACM.

[3] W. Chen and D. S. Warren. Tabled Evaluation with Delaying for General Logic Programs. *J. ACM*, 43(1):20–74, January 1996.

[4] J. Lacroix and O. Boucelma. Design and Implementation of a Trust Service for the Cloud. In *Proc. of OTM '15*, p. 620–638. LNCS, 2015.

[5] T. Lebo, S. Sahoo, and D. McGuinness. PROV-O: The PROV Ontology, 2013.

[6] G. Lin, Y. Bie, and M. Lei. Trust Based Access Control Policy in Multi-domain of Cloud Computing. *J. of Computers*, 8(5), May 2013.

[7] L. Moreau and P. Missier. PROV-DM: The PROV Data Model, 2013.

[8] L.Moreau and P.Missier. PROV-N: The Provenance Notation, 2013.

[9] J. Park, D. Nguyen, and R. Sandhu. A provenance-based access control model. In *Proc. of PST'12*, p. 137–144, IEEE, 2012.

[10] L. Sun, J. Park, and R. Sandhu. Engineering Access Control Policies for Provenance-aware Systems. In *Proc. of CODASPY '13*, p.285–292, 2013. ACM.