Towards Privacy-Enhanced Authorization Policies and Languages

C.A. Ardagna, E. Damiani, S. De Capitani di Vimercati, and P. Samarati

Dipartimento di Tecnologie dell'Informazione
Università di Milano –26013 Crema - Italy
{ardagna,damiani,decapita,samarati}@dti.unimi.it

Abstract. The protection of privacy in today's global infrastructure requires the combined application solution from technology (technical measures), legislation (law and public policy), and organizational and individual policies and practices. Emerging scenarios of user-service interactions in the digital world are also pushing toward the development of powerful and flexible privacy-enhanced models and languages.
This paper aims at introducing concepts and features that should be investigated to fulfill this demand. In particular, the content of this paper is a result of our ongoing activity in the framework of the PRIME project (*Privacy and Identity Management for Europe*), funded by the European Commission, whose objective is the development of privacy-aware solutions for enforcing security.

1 Introduction

Traditional access control systems are based on regulations (*policies*) that establish who can, or cannot, execute which actions on which resources. However, in today's systems the definition of an access control model is complicated by the need to formally represent complex policies, where access decisions depend on the application of different rules coming, for example, from laws practices, and organizational regulations. Given the complexity of the scenario, these traditional policies are too limiting and do not satisfy all the above requirements. Although recent advancements allow the specifications of policies with reference to generic attributes/properties of the parties and the resources involved, they are not designed for enforcing privacy policies. For instance, privacy issues that are not addressed by traditional approaches include protecting user identities by providing *anonymity*, *pseudonymity*, *unlinkability*, and *unobservability* of users at communication level, system level, or application level. Therefore, the consideration of privacy issues introduces the need for rethinking authorization policies and models and the development of new paradigms for access control and in particular authorization specification and enforcement.

In this paper, we present our recent research work in the context of the PRIME project [12]. Our work deals with three main key aspects:

– *Resource representation.* Writing access control policies where resources to be protected are pointed at via data identifiers and access conditions are

evaluated against their attribute values is not sufficient anymore. Rather, it is important to be able to specify access control requirements about resources in terms of available *metadata* describing them.
- *Context representation.* Distributed environments have increased the amount of context information available at policy evaluation time (e.g., location-based one), and this information is achieving a more and more important role.
- *Subject identity.* Evaluating conditions on the subject requesting access to a resource often means accessing personal information either presented by the requestor as a part of the authentication process or available elsewhere. Identifying subjects raises a number of privacy issues, since electronic transactions (e.g., purchases) require disclosure of a far greater quantity of information than their physical counterparts.

A privacy-enhanced authorization model and language is then described allowing for definition and enforcement of powerful and flexible access restrictions based on generic properties associated with subjects and objects. We also bring forward the idea of exploiting the semantic web to allow the definition of access control rules based on generic assertions defined over concepts in the ontologies that control metadata content and provide abstract subject domain concepts, respectively [16]. These rules are then enforced on resources annotated with metadata regulated by the same ontologies.

The remainder of this paper is organized as follows. Section 2 presents the different types of privacy policies we have identified. Section 3 and Section 4 illustrate our privacy-enhanced model and language, respectively. Section 5 describes a possible representation of expressing our language by using an XML-based syntax. Finally, Section 6 presents our conclusions.

2 Privacy policies

To address the requirements mentioned in the previous section, different types of policies need to be introduced.

- *Access control policies.* They govern access/release of data/services managed by the party (as in traditional access control) [4].
- *Release policies.* They govern release of properties/credentials/PII of the party and specify under which conditions they can be disclosed [2].
- *Data handling policies.* They define the personal information release will be (or should be) deals with at the receiving party [15].
- *Sanitized policies.* They provide filtering functionalities on the response to be returned to the counterpart to avoid release of sensitive information related to the policy itself.

Access control policies. Access control policies define authorization rules concerning access to data/services. Authorizations correspond to traditional (positive) rules usually enforced in access control systems. For instance, an authorization

rule can require the proof of majority age and a credit card number (condition) to read (action) a specific set of data (object). Also, an obligation can specify that the credit card number must be deleted at the end of the transaction or that the server must log any request. When an access request is submitted to the party, it is first evaluated against the authorization rules applicable to it. If the conditions for the required access are evaluated to true, access is permitted. If none of the specified conditions that might grant the requested access can be fulfilled, then the access is denied. Finally, if the current information is insufficient to determine whether the access request can be granted or denied, additional information is needed and the client receives an undefined response with a list of requests that she must fulfill to gain the access. For instance, if some of the specified conditions can be fulfilled (e.g., by signing an agreement), then the party prompts the requester with the actions that would result in the required access.

Release policies. Release policies define the party's preferences regarding the release/disclosure of its Personal Identifiable Information (PII). More precisely, these policies specify to which party, for which purpose/action, and under which conditions/obligations a particular set of PII can be released/disclosed [2]. For instance, a release policy can state that credit card information can be disclosed only in the process of a `buy` action and upon presentation of a nondisclosure agreement (condition) by the party. The disclosure of PII may only be performed if the release policies are satisfied.

Data handling policies. Data handling policies specify how PII is used and processed [15]. More precisely, they should regulate how PII will be used (e.g., information collected through a service will be combined with information collected from other services and used in aggregation for market research purposes), how long PII will be retained (e.g., information will be retained as long as necessary to perform the service), and so on. Clients use these policies to define how her information will be used and processed by the counterpart. In this way, user can manage the information also after its release.

Sanitized policies. Sanitized policies provide filtering functionalities on the response to be returned to the counterpart to avoid release of sensitive information related to the policy itself (or to the status against which the policy has evaluated). This happens when an *undefined* decision together with a list of alternatives (policies) that must be fulfilled to gain the access to the data/service is returned to the counterpart. For instance, suppose that the policy returned by the access control is "citizenship=EU". The party can decide to return to the user either the policy as it is or a modified policy (obtained by applying the sanitized policies) simply requesting the user to declare its nationality (then protecting the information that access is restricted to EU citizens).

In the following, we deal with access control and release policies: data handling and sanitized policies will be added in future work.

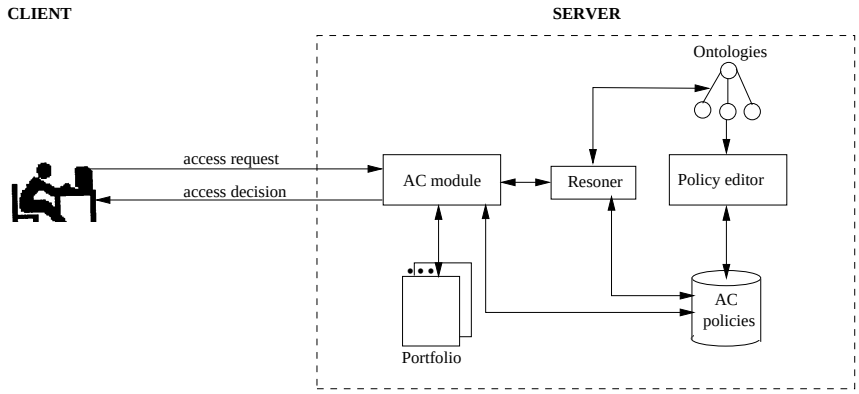

Fig. 1. Architecture

3 Scenario and basic elements of the privacy-enhanced model

We consider parties that interact with each other to offer services (see Figure 1). As in a usual client/server interaction, a client asks for a service and a server provides for the service. However, each party can be interchangeably as either a client or a server at different times, with respect to a specific instance of a service request. The access request is processed by the Access Control module (*AC module*). The AC module interacts with the *Reasoner* that takes the access control policies together with the subject, object, and credential ontologies as input and computes the expanded policies including semantically equivalent additional conditions. These conditions, specified in disjunction with the original ones, allow for increasing the original policy's expressive power. The AC module returns to the client a *yes*, *no*, or *undefined* decision. In the latter case, it returns the information about which conditions need to be satisfied for the access to be granted. In this last case, the problem of communicating such conditions to the counterpart arises.

The access control policies are based on generic properties (attributes) associated with the subjects requesting accesses and the resources (data/services) subjects interact with. In the following, we illustrate these basic elements of our model in details.

3.1 Portfolio

The set of properties associated with subjects and objects are represented by means of a *portfolio* [2]. More precisely, a portfolio includes two types of information: *declarations* and *credentials*. A declaration is a statement issued by the party while a credential is a statement issued and signed (i.e., certified) by authorities trusted for making the statement [8]. As an example, the driver license number maintained as a data value at a party and communicated in a

negotiation is a declaration. A digital copy of the driver license, released by the public administration to the party, and that the party can submit to a server to prove that it has a driver license or that the administration certifies some properties (e.g., address), is a credential. At a practical level, we view a credential as characterized by two elements: *i)* a signed content, and *ii)* the public digital signature verification key to verify the signature. We can also imagine the existence of (meta)information associated with a credential, outside the signed content. Such information cannot however be trusted as being certified by the authority that signed the credential. In the following, we consider credentials such a mean to allow query for specific data, such as name or address in a driver license, number or expiration date in a credit card. To refer to specific data in a credential we introduce the concept of *credential term*.

Definition 1. A *credential term* is an expression of the form `credential_name(`*predicate_list*`)`, where `credential_name` is the name of the credential, and *predicate_list* is a possibly empty list of elements of the form `predicate_name(`*arguments*`)`.

Intuitively, a credential term can be used to specify a condition on credentials (we will elaborate more on this in Sect. 4). Some examples of credential terms are: `driver-license(equal(name,`"John Doe"`))` and `identity-card(greater_than(age,`18`))`. The first term denotes the `driver-license` credential where attribute name should be equal to `John`. The second term denotes credential `identity-card` where attribute `age` should be greater than 18. Declarations and credentials in a portfolio may be organized into a partial order. For instance, an `identity-document` can be seen as an abstraction for credentials `driver-license`, `passport`, and `identity-card`. Finally, the functionalities offered by a server are defined by a set of services. Intuitively, each service can be seen as an application that clients can execute.

3.2 Ontologies and abstractions

Our model provides the support for *ontologies* that allow to make generic assertions on subjects and objects [13, 14]. More precisely, we use three ontologies: a *subject ontology*, an *object ontology*, and a *credential ontology*. The subject ontology contains terms that can be used to make generic assertions on subjects (e.g., in a medical scenario possible terms are `Physician`, `Patient`, `assists`). The object ontology contains domain-specific terms that are used to describe the resource content such as `Video` and `shows_how`. Finally, the credential ontology represent relationships among attributes and credentials (`part-of` and `is-a` relationships) to establish what kind of credentials can be provided to fulfill a declaration or credential request. For instance, an ontology can state that attributes `birth date` and `nationality` are `part_of driver-license`, `identity-card`, and `passport`. In this way, the reasoning process can point out all the credentials that a user, for example, can provide to prove the satisfaction of a given constraint. To fix ideas and make the discussion clear, suppose that a

user can use an on-line car rental service only if she is an European citizen. The access is then allowed if the user can prove her nationality and, according to the credential ontology, this can be done either by showing the `driver-license`, `identity-card`, or `passport`.

Abstractions can also be defined within the domains of users as well as objects. Intuitively, abstractions allow to group together users (objects, resp.) with common characteristics and to refer to the whole group with a name.

4 Privacy-aware language

We are now ready to describe the basic constructs of the language used to define the privacy policies and the syntax of the language.

4.1 Basic elements of the language

We have identified the following predicates:

- a predicate `declaration` where the argument is a list of predicates of the form `predicate_name(`*arguments*`)`;
- a binary predicate `credential` where the first argument is a credential term (see Definition 1) and the second argument is a public key term. Intuitively, a ground atom `credential(`c, K`)` is evaluated to true if and only if there exists a credential c verifiable with public key K.
- a set of standard binary built-in mathematic predicates, such as `equal()`, `greater_than()`, `lesser_than()`, and so on.
- a set of non predefined predicates that evaluate information stored at the site.

The above predicates constitute the basic literals that can be used in access control and release policies. Note that predicates `declaration` and `credential` have been introduced to distinguish between conditions on data declarations and conditions on credentials (we will elaborate more on this in the following sub-section).

4.2 Policy components

Syntactically, an access control rule (release rule, resp.) has the following form:

subject WITH *subject-expression* CAN *action* FOR *purpose* ON *object* WITH *object-expression* IF *conditions* FOLLOW *obligations*

where:

- *subject (object)* identifies the subject (object) to which the rule refers;
- *subject-expression (object-expression)* is an expression that allows the reference to a set of subjects (objects) depending on whether they satisfy given conditions that can be evaluated on the user's portfolio (object's profile);
- *action* is the action to which the rule refers (e.g., read, write, and so on)[1]

[1] Note that abstractions can also be defined on actions, specializing actions or grouping them in sets.

- *purpose* is the purpose (e.g., `scientific`) to which the rule refers and represents how the data is going to be used by the recipient;
- *conditions* is a boolean expression of generic conditions that an access request to which the rule applies has to satisfy;
- *obligations* is a boolean expression of obligations that the server must follow when manage the information/data/PII.

We now look at the different components in the rule.

Subject expression. These expressions allow the reference to a set of subjects depending on whether they satisfy given conditions that can be evaluated on the subject's portfolio. Note that the conditions specified through these expressions are very similar to generic conditions. The difference is that while the subject expression is evaluated on the user of the request, generic conditions specify generic constraints that are not evaluated on the requester. More precisely, *a subject expression is a boolean formula of terms of the form*:

- `declaration(`*predicate_list*`)`, where *predicate_list* is a possibly empty list of elements of the form `predicate_name(`*arguments*`)`. Intuitively, a declaration predicate is evaluated to true if each predicate specified in the *predicate_list* is evaluated to true.
- `credential(`*credential_term*,K`)`, where *credential_term* is defined as `credential_name(`*predicate_list*`)` (see Definition 1). Intuitively, a credential predicate is evaluated to true if there exists credential `credential_name` for which each predicate `predicate(`*arguments*`)` in *predicate_list* is evaluated to true and `credential_name` is verifiable with public key K.

Note that the predicates specified as arguments of the `declaration` and `credential` predicates can be: *i)* location-based predicates, *ii)* the standard built-in mathematic predicates, and *iii)* the non predefined predicates that evaluate information stored at the server.

To make it possible to refer to the user of the request being evaluated without the need of introducing variables in the language, we introduce the keyword **user**, whose appearance in a conditional expression is intended to be substituted with the actual parameters of the request in the evaluation at access control time.

Example 1. The following are examples of subject expressions:

- `declaration(equal(`**user**.name,Bob`),greater_than(`**user**.age,18`))` denoting requests made by a user whose name is `Bob` with `age` greater than `18`;
- `credential(passport(equal(`**user**.job,professor`)),`K_1`)` denoting requests made by users who are professors. This property should be certified by showing the `passport` credential verifiable with public key K_1

Object expression. These expressions allow the reference to a set of objects depending on whether they satisfy given conditions that can be evaluated on the object's profile. Note that the conditions specified through these expressions are very similar to generic conditions. The difference is that while the object expression evaluated on the object (or associated profile) to which the request being processed refers, generic conditions specify generic constraints that are not evaluated on the requested object. More precisely, *an object expression is a boolean formula of terms of the form*:

- `declaration(`*predicate_list*`)`, where *predicate_list* is a possibly empty list of elements of the form `predicate_name(`*arguments*`)`. Intuitively, a declaration predicate is evaluated to true if each predicate specified in the *predicate_list* is evaluated to true.

Note that the predicates specified as arguments of the `declaration` predicate can be: *i)* the standard built-in mathematic predicates, and *ii)* the non predefined predicates that evaluate information stored at the server.

Like for subjects, to make it possible to refer to the object to which the request being processed refers, without need of introducing variables in the language, we introduce the keyword **object**, whose appearance in a conditional expression is intended to be substituted with the actual parameters of the request in the evaluation at access control time.

Example 2. The following are examples of object expressions:

- `declaration(equal(`**object**`.creator,`**user**`))` denoting all objects created by the requester;
- `declaration(lesser_than(`**object**`.creation_date,1971))` denoting all objects created before 1971.

Conditions. We assume that the type of conditions that can be specified in the *conditions* element are only conditions that can be brought to satisfactions at run-time processing of the request. These conditions can be related to agreement acceptance, payment fulfillment, or registration. Conditions can be associated with data at different levels (i.e., attribute, credentials' attributes and credentials) and can be certified or uncertified. More precisely, *conditions are boolean formula of terms of the form*:

- `predicate_name(`*arguments*`)`.

Note that the predicates specified in the *conditions* element can be: *i)* trusted-based conditions stating that, for example, the requester should use a trusted platform, *ii)* the standard built-in mathematic predicates, and *iii)* the non predefined predicates that evaluate information stored at the server.

Example 3. The following is a simple example of condition.

- `fill_in_form(`**user**`,`*form1*`)` checks if the requester has filled in form *form1*.

Obligations. They establish how the released PII must be managed by the counterpart. For instance, obligations may state that some data should be deleted after three time accessed, the owner of some data should be notified after every access to the data, some data should be obfuscated or deleted after 3 months, and so on. Obligations can be attached to a particular instance of release data in order to give to the counterpart some rules that must be follow in the PII management.

5 An example

We now present an example of policy (other examples are omitted here for space constraints) and a possible way of expressing policies by using an XML-based syntax.

We define two namespaces: `xmlns:pol` is the namespace of the policy and `xmlns:ont` is the namespace for the ontology statements. Every policy can contain more than one rule combined through the combine-rule attribute. Each rule has three main components:

- `pol:target` is the target of the policy (subject, object, action, purpose);
- `pol:condition` includes generic conditions (neither related to subject nor object) such as assurance/trust conditions;
- `pol:obligation` includes further steps that the party must take in account when the access is granted.

We now analyze the target component more in details. The target includes the `pol:subject` tag corresponding to the *subject* field described in Section 4. Associated with the subject, there is the subject expression (`pol:subject-expression`) that contains boolean operators (and, or) and a set of constraints (`pol:constraint`). Every constraint has a type and is of the form "*left-value* operator *right-value*". The operator is a matching function, the left-value (`ont:datatype`) have to be a class referencing an ontology structure and the right-value (`ont:instanceref`) can be another class, an instance class, or a literal (e.g., in the rule below the constrain is **user**.`job` = "doctor"). The object and object expression have the same structure of the subject and subject expression, respectively. Finally, the target includes an action (`pol:action`) and a purpose (`pol:purpose`).

When a request is submitted to the system, the AC module selects all the applicable policies by using the subject, object, action, and purpose specified in the access request and then checks the (expanded) conditions inside the policies to determine the access result (yes/no/undefined).

Example 4. Suppose that an access control policy stated that "A registered user who works as a doctor, can read for research purposes data patientData with the agreement of the patient". This policy is expressed as follows.

`registeredUsers` WITH `declaration(equal(`**user**`.work, "doctor"))` CAN `read` FOR *research* ON `patientData`with`declaration(equal(`**object**`.patient_agreement,yes))` IF *no-condition* FOLLOW *no-obligation*

```
<pol:policy type="accessControl" combine-rule="first-grant"
 xmlns:pol="http://example.com/policy-namespace"
 xmlns:ont="http://example.com/ontology-namespace">
  <pol:rule>
    <pol:target>
      <pol:subject>registeredUsers</pol:subject>
      <pol:subject-expression>
        <pol:constraint type="declaration">
          <pol:function type="equal">
            <ont:datatype>
              <ont:user/> <ont:job/>
            </ont:datatype>
            <ont:instanceref>
              <ont:user/> <ont:job/>
              <ont:value>doctor</ont:value>
            </ont:instanceref>
          </pol:function>
        </pol:constraint>
      </pol:subject-expression>
      <pol:object>patientData</pol:object>
      <pol:object-expression>
        <pol:constraint type="declaration">
          <pol:function type="equal">
            <ont:datatype>
              <ont:object/> <ont:patient/> <ont:agreement/>
            </ont:datatype>
            <ont:value type="xsd:string">yes</ont:value>
          </pol:function>
        </pol:constraint>
      </pol:object-expression>
      <pol:action>read</pol:action>
      <pol:purpose>research</pol:purpose>
    </pol:target>
    <pol:condition/>
    <pol:obligation> ...  </pol:obligation>
  </pol:rule>
  <pol:rule> ... </pol:rule>
</pol:policy>
```

Fig. 2. A simple example of policy

Figure 2 illustrates the policy expressed by using the XML syntax described above. Note that our access control system operates also when the users want to remain anonymous or disclosure only some attributes about themselves, protecting users privacy.

6 Conclusions

This paper has presented the preliminary results of our ongoing activity in the framework of the PRIME project. Issues to be investigated include the filtering and renaming of policies and the addition of obligations. As discussed previously, since access control does not return only a "yes" or "no" access decision, but it returns the information about which conditions need to be satisfied for the access to be granted ("undefined" decision), the problem of communicating such conditions to the counterpart arises. The system should then provide meta-policies for protecting the policy when communication requisites.

7 Acknowledgments

This work was supported in part by the European Union within the PRIME Project in the FP6/IST Programme under contract IST-2002-507591 and by the Italian MIUR within the KIWI and MAPS projects.

References

1. Bonatti, P., Damiani, E., De Capitani di Vimercati, S., Samarati, P.: A Component-based Architecture for Secure Data Publication. Proc. of the 17th Annual Computer Security Applications Conference (2001), New Orleans, Louisiana.
2. Bonatti, P., Samarati, P.: A Unified Framework for Regulating Access and Information Release on the Web. Journal of Computer Security (2002), vol. 10, 241–272.
3. Ashley, P., Hada, S., Karjoth, G., Powers, C., Schunter, M.: Enterprise Privacy Authorization Language (EPAL 1.1). IBM Research Report (2003), `http://www.zurich.ibm.com/security/enterprise-privacy/epal`.
4. Samarati, P., De Capitani di Vimercati, S.: Access Control: Policies, Models, and Mechanisms. Foundations of Security Analysis and Design **LNCS 2171** (2001), Springer-Verlag.
5. eXtensible Access Control Markup Language (XACML) Version 1.1. OASIS, 2003, `http://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-1.1.pdf`.
6. Ardagna, C.A., Damiani, E., De Capitani di Vimercati, S., Samarati, P.: A Web Service Architecture for Enforcing Access Control Policies. Proc. of the First International Workshop on Views On Designing Complex Architectures (VODCA 2004), Bertinoro, Italy.
7. Ardagna, C.A., Damiani, E., De Capitani di Vimercati, S., Samarati, P.: XML-based Access Control Languages. Information Security Technical Report, (2004), vol. 9.
8. Gladman, B., Ellison, C., Bohm, N.: Digital signatures, certificates and electronic commerce, `http://www.clark.net/pub/cme/html/spki.html`.
9. Bettini, C., Jajodia, S., Sean Wang, X., Wijesekera, D.: Provisions and Obligations in Policy Management and Security Applications. In Proc. 28th Conf. Very Large Data Bases (VLDB'02), (2002), `citeseer.ist.psu.edu/bettini02provisions.html`.
10. Park, J., Sandhu, R.: The UCONabc Usage Control Model. ACM Transactions on Information and System Security (TISSEC), (2004), vol. 7, no. 1.

11. World Wide Web Consortium: Semantic Web. http://www.w3.org/2001/sw/.
12. Privacy and Identity Management for Europe (PRIME). http://www.prime-project.eu.org/.
13. Damiani, E., De Capitani di Vimercati, S., Fugazza, C., Samarati, P.: Semantics-aware Privacy and Access Control: Motivation and Preliminary Results. 1st Italian Semantic Web Workshop, (2004), Ancona, Italy.
14. Damiani, E., De Capitani di Vimercati, S., Fugazza, C., Samarati, P.: Extending Policy Languages to the Semantic Web. Proc. of the International Conference on Web Engineering, (2004), Munich, Germany.
15. Cranor, L., Langheinrich, M., Marchiori, M., Presler-Marshall, M., Reagle, J.: The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. http://www.w3.org/TR/P3P/.
16. Ardagna, C.A., Damiani, E., De Capitani di Vimercati, S., Fugazza, C., Samarati, P.: Offline Expansion of XACML Policies Based on P3P Metadata (to appear). ICWE 2005, 5th International Conference on Web Engineering, Sydney, Australia.