Negotiation of Prohibition: An Approach Based on Policy Rewriting

Nora Cuppens-Boulahia, Frédéric Cuppens, Diala Abi Haidar, Hervé Debar

1 Introduction

Traditionally, access control is enforced by centralized stand-alone architectures. In this case, the access controller “knows” all information necessary to evaluate the access control policy. As a consequence, when a subject sends a query to the access controller, this access controller does not need to interact with this subject to decide if this query must be accepted or rejected.

However, in more recent architectures, such a centralized evaluation of the access control policy is no longer appropriate. When a subject sends a query to the access controller, this controller needs to interact with the subject through a negotiation protocol. The objective of this protocol is to exchange additional information necessary to evaluate the policy. This information generally correspond to credentials the subject has to provide to prove that he or she satisfies the requirements to execute the query.

Notice that the negotiation protocol can actually behave in a symmetric way in the sense that the access controller may also exchange credentials to provide the subject with guarantees that this subject can interact securely with the controller.

The objective of the negotiation is to exchange credentials in order to decide if the query must be accepted or not. When the access control policy corresponds to a set of *permission* rules, this consists in determining if the query matches one of these permission rules. However, the access control policy may also include *prohibitions* that act as exception to the permissions. In that case, the negotiation protocol must decide if (1) there is a permission to accept the query and (2) there is no prohibition that would apply to deny the query.

Nora Cuppens-Boulahia, Frédéric Cuppens and Diala Abi Haidar
TELECOM Bretagne, 2 rue de la châtaigneraie, 35512 Cesson Sévigné Cedex, France

Diala Abi Haidar and Hervé Debar
France Telecom R&D Caen, 42 rue des coutures BP 6243, 14066 CAEN, France

However, we claim that it would not be fair if the negotiation protocol ask for credentials in order to activate prohibitions. To illustrate this claim, let us consider the two following access control rules: (R1) a member of the medical staff is permitted to consult the patient's medical summary, (R2) a medical secretary is prohibited to consult the patient's medical summary. Let us also assume that rule R2 has higher priority than rule R1. Assigning priority to access control rules will be further discussed in the remainder of this paper.

Let us now consider a subject who asks to consult a given medical summary. We argue that the negotiation protocol should not ask this subject to provide a credential proving that he or she is a medical secretary in order to activate prohibition R2. Instead, the negotiation protocol should ask this subject to prove that he or she is a medical staff member (so that permission R1 applies) and not a medical secretary (so that prohibition R2 does not apply). For this purpose, the subject may provide a credential proving that he or she is a physician if this is sufficient to derive that (1) a physician is a medical staff member (due to an inclusion hierarchy) and (2) a physician cannot be a medical secretary (due to a separation constraint).

Since it is not possible to directly negotiate prohibitions, we suggest an approach to solve this problem. The central idea consists in rewriting an access control policy that contains both permissions and prohibitions into an equivalent policy that contains only permissions. We show that this approach applies to both open and close policies. The resulting policy only contains permissions but requires to negotiate negative attributes. For instance, in our above example, the negotiation protocol must get evidence that the subject is *not* a medical secretary. Thus, another contribution of this paper consists in adapting a negotiation protocol so that negotiation of negative attributes is possible.

The remainder of this paper is organized as follows. In section 2, we further develop a scenario to motivate the problem addressed in this paper. Section 3 presents the model we use to specify access control policies and explains how to manage conflicts between permissions and prohibitions by assigning priority levels to security rules. In section 4, we define a rewriting procedure that transforms an access control policy into an equivalent set of permissions and show how this procedure applies to both open and close policies. Since our rewriting procedure can generally generate negative conditions, section 5 explains how to adapt a negotiation protocol in order to negotiate such negative conditions. Section 6 presents a discussion of our approach and compares it with related work. Finally, section 7 concludes the paper.

2 Motivating example

In this section, we present an example to illustrate our approach. We consider a database used in an organization to manage medical records. There is special type of medical record called medical summary.

The database can be accessed by medical staff members. There are several sub roles of medical staff member: medical secretary, nurse and physician. There are also two sub roles of physicians: senior physician and junior physician.

Subjects can ask to execute the activity of managing a medical records. There are two sub activities of managing called consult and update.

The access control policy associated with this database management system corresponds to the following rules:

- R1: A member of the medical staff is permitted to manage the patient's medical summary,
- R2: A medical secretary is prohibited to manage the medical records,
- R3: In a context of urgency, a medical secretary is permitted to consult the patient's medical summary,
- R4: A nurse is prohibited to update the patient's medical summary,
- R5: A physician is permitted to manage medical records,
- R6: A junior physician is prohibited to update medical records.
- R7: In a context of urgency, a junior physician is permitted to update the patient's medical summary.

When a subject queries the database to get an access to a medical record, this subject has to provide credentials to prove that the requested access is actually permitted. For this purpose, a subject can give credentials proving his or her role (medical secretary, nurse, physician, junior physician or senior physician), credential proving that someone is his or her patient (if this subject is a physician) and credentials proving that the context of urgency is active.

When a subject queries the database, several rules of the security policy may potentially apply. For instance, when a subject asks to update some medical summary, all the rules of the above policy may potentially apply (since update is a sub activity of manage and medical summary is a special type of medical record).

Since these rules are conflicting, it is first necessary to solve these conflicts by assigning priority levels to these rules. This is further explained in section 3. Based on these priority levels, we define a process to rewrite the initial policy into an equivalent set of access control rules but that only contains permissions. For example, if we assume that rule R6 has higher priority than rule R5, then our rewriting process will rewrite rule R5 into the two following rules:

- R5.1. A physician who is not a junior physician is permitted to manage his or her patient's medical records,
- R5.2. A physician is permitted to manage without updating his or her patient's medical records.

Then, the database access controller has to determine which rule actually applies to take the decision to accept or deny the access. For this purpose, the access control must determine which credentials are sufficient to grant an access. For example, let us assume that the subject that queries the database provides his credential proving that he or she is physician. In this case, if the query consists in updating some medical record, then rule R5.1 potentially applies. Thus, the negotiation process may ask this subject to prove that he or she is not a junior physician.

As mentioned in the introduction, the net advantage of our approach is that the negotiation process will not ask the subject to prove that he or she is a junior physician to check if the prohibition associated with rule R6 actually applies. We claim that it is clearly better to ask this user to prove that he or she is *not* a junior physician in order to derive that rule R5.1 actually applies.

3 Policy specification and conflict management

3.1 Access control specification

Access control models provide means to specify which permissions and prohibitions apply to subjects when they execute actions on objects [3, 9]. These permissions and prohibitions are generally modelled by rules having the form[1] $condition \rightarrow permission(S,A,O)$ or $condition \rightarrow prohibition(S,A,O)$ where $condition$ is a condition that must be satisfied on the state of the information system to derive the corresponding permission or prohibition. A conflict occurs if it is possible to derive that a given subject is both permitted and prohibited to execute a given action on a given object.

Managing conflicts in such models is a complex problem and [4] shows that detecting potential conflicts is actually undecidable. In [4], we also show the advantage of a more structured model as suggested in the OrBAC model [10, 11] and we shall use this model in the following to express the access control policy. One of the OrBAC contribution is the abstraction of the traditional triples $\langle subject, action, object\rangle$ into $\langle role, activity, view\rangle$. The entities $subject$, $action$ and $object$ are called *concrete entities* whereas the entities *role*, *activity* and *view* are called *organizational entities*.

A *view* is a set of objects that possess the same security-related properties within an organization thus these objects are accessed in the same way. Abstracting them into a view avoids the need to write one rule for each of them. Another useful abstraction is that of action into *activity*. An *activity* is viewed as an operation which is implemented by some actions defined in the organization. For example, the actions read (for a file) and select (for a database) may be considered as one *consult data* operation. This is why they can be grouped within the same activity for which we may define a single security rule. One of the main contributions of the OrBAC model is that it can model *context* that restricts the applicability of the rules to some specific circumstances [5]. Thus, *context* is another organizational entity of the OrBAC model.

[1] In the following, we shall assume that terms starting with a capital letter represent variables and that all free variables in formula are implicitly universally quantified.

The OrBAC model defines four predicates[2]:

- *empower*: *empower(s, r)* means that subject s is empowered in role r.
- *consider*: *consider(α, a)* means that action α implements the activity a.
- *use*: *use(o, v)* means that object o is used in view v.
- *hold*: *hold(s, α, o, c)* means that context c is true between subject s, action α and object o

Access control rules are specified in OrBAC by quintuples that have the following form:

- $SR(decision, role, activity, view, context)$

which specifies that the decision (*i.e.* permission or prohibition) is applied to a given role when requesting to perform a given activity on a given view in a given context. We call these *organizational security rules*. An example of such a security rule is:

- $SR(prohibition, nurse, update, medical\_summary, any_C)$

that corresponds to the rule R4 in our motivating example associated with the any_C context which is always true.

Concrete permissions or prohibitions that apply to triples $\langle subject, action, object\rangle$ are modelled using the predicate $sr(decision, subject, action, object)$ and logically derived from organizational security rules. The general derivation rule is defined as follows:

- $SR(Decision, R, A, V, C) \wedge empower(Subject, R) \wedge consider(Action, A) \wedge$
  $use(Object, V) \wedge hold(Subject, Action, Object, C)$
  $\rightarrow sr(Decision, Subject, Action, Object)$

3.2 Structuring organizational entities

The OrBAC model is based on four different types of organizational entities, namely *role*, *activity*, *view* and *context*. When defining our algorithm to rewrite a security policy in section 4, we shall need to aggregate elementary entities into composite entities. For instance, if r_1 and r_2 are two roles, then we shall consider that the intersection $r_1 \cap r_2$ and the disjunction $r_1 \cup r_2$ of these two roles is also a (composite) role. Similarly, the complement $\bar{r}$ or a role r is also a role.

For this purpose, we define an algebra for the four types of organizational entities. To simplify the presentation, we only formally define this algebra for the role entities. The algebras for the three other entities activity, view and context are similarly defined.

To define this algebra, we first consider a finite set $\mathscr{S}$ or subjects and a finite set $\mathscr{R}$ of elementary roles. The algebra associated with the role entity is then defined as follows:

[2] In OrBAC, the organization is made explicit in every predicate but here, to simplify, the organization is left implicit since we consider always only one organization.

Definition of the role algebra:

We define an algebra for the role entity as follows:

- no_R and any_R are two roles.
- If $r \in \mathscr{R}$ then r is an (elementary) role.
- If r is a role, then $\bar{r}$ is a role.
- If r_1 and r_2 are roles, then $r_1 \cap r_2$ and $r_1 \cup r_2$ are roles.
- Nothing else is a role.

In the following, we shall also use $r_1 \backslash r_2$ as a notation equivalent to $r_1 \cap \overline{r_2}$.

Interpretation of the role algebra:

To provide an interpretation of the algebra, we use the following notation for each elementary role r:

$\mid r \mid = \{s \in \mathscr{S}$ such that $empower(s,r)$ is true$\}$

Then the algebra is interpreted as follows:

- $\mid no_R \mid = \emptyset$
- $\mid any_R \mid = \mathscr{S}$
- $\mid \bar{r} \mid = C_{\mathscr{S}}^{|r|}$
- $\mid r_1 \cap r_2 \mid = \mid r_1 \mid \cap \mid r_2 \mid$
- $\mid r_1 \cup r_2 \mid = \mid r_1 \mid \cup \mid r_2 \mid$

Axiomatic:

The axiomatic of the role algebra is defined by axioms that specify that $\cap$ is commutative, associative, it distributes over $\cup$ plus the following axioms:

- $\overline{no_R} = any_R$
- $R \cap R = R$
- $R \cap no_R = no_R$
- $R \cap any_R = R$
- $\overline{\overline{R}} = R$
- $R_1 \cup R_2 = \overline{\overline{R_1} \cap \overline{R_2}}$

We also associates the four organizational entities with a hierarchy of inclusion and constraints of separation. We only present the model for the role entity. The models for the activity, view and context entities are similarly defined.

The inclusion hierarchy on the roles is defined using the *sub_role* predicate: If r_1 and r_2 are roles, then $sub\_role(r_1, r_2)$ means that r_1 is a sub role of r_2.

Separation constraints between roles are defined using the $separated\_role(r_1, r_2)$ predicate which states that role r_1 is separated from role r_2, *i.e.* a subject cannot be empowered in both r_1 and r_2.

We have the following axioms:

- $separated\_role(R_1, R_2) \leftrightarrow R_1 \cap R_2 = no_R$
- $sub\_role(R_1, R_2) \leftrightarrow R_1 \cap \overline{R_2} = no_R$
- sub_role is transitive
- $sub\_role(R_1, R_2) \wedge separated\_role(R_2, R_3) \rightarrow separated\_role(R_1, R_3)$

To illustrate this algebra, let *physician* be a role. According to our algebra $\overline{physician}$ is also a role which is defined through the complement of the role *physician*. That is, a subject is assigned to the role $\overline{physician}$ if he is not assigned to the role *physician*. If we have two roles *physician* and *employee* then $physician \cap employee$ and $physician \cup employee$ are also roles based respectively on intersection and disjunction of roles. A subject is empowered in the role $physician \cap employee$ if he or she is empowered in both roles *physician* and *employee*.

3.3 Prioritized access control rules

When the access control policy contains permissions and prohibitions, a conflict occurs when one can derive both $sr(permission, s, a, o)$ and $sr(prohibition, s, a, o)$ for some subject, action and object. The solution is based on assigning priorities to security rules so that when a conflict occurs between two rules, the rule with the higher priority takes precedence.

This is basically the approach suggested in the OrBAC model [4]. It actually provides means to detect and manage *potential* conflicts between organizational rules. A potential conflict exists between an organizational permission rule and an organizational prohibition rule if these two rules may possibly apply to the same subject, action and object. There is no such potential conflict between two organizational security rules if these rules are *separated*. Thus, in OrBAC, a potential conflict between two organizational security rules is defined as follows:

Definition 1. A potential conflict occurs between two security rules $SR(d_1, r_1, a_1, v_1, c_1)$ and $SR(d_2, r_2, a_2, v_2, c_2)$ if $d_1 \neq d_2$ and role r_1, activity a_1, view v_1 and context c_1 are respectively not separated from role r_2, activity a_2, view v_2 and context c_2.

Priorities should be associated with such potentially conflicting security rules in order to avoid situations of real conflict. Prioritization of security rules must proceed as follows [4]:

- Step 1: Detection of potentially conflicting rules.
- Step 2: Assignment of priority to potentially conflicting rules.

We then obtain a set of partially ordered security rules *SR(decision, role, activity, view, context, priority)*. Concrete security rules can be derived from the abstract security rules and are assigned with the same priority. It has been proved in previous works [4] the following theorem.

Theorem 1. *If every potential conflict is solved, then no conflict can occur at the concrete level.*

3.4 Application to our motivating example

The access control policy of our motivating example is formally specified by the following set of OrBAC security rules:

- R1: $SR(permission, medical\_staff, manage, medical\_summary, any_C)$
- R2: $SR(prohibition, secretary, manage, medical\_record, any_C)$
- R3: $SR(permission, secretary, consult, medical\_summary, urgency)$
- R4: $SR(prohibition, nurse, update, medical\_summary, any_C)$
- R5: $SR(permission, physician, manage, medical\_record, any_C)$
- R6: $SR(prohibition, junior\_physician, update, medical\_record, any_C)$
- R7: $SR(permission, junior\_physician, update, medical\_record, urgency)$

We also assume we have the following separation constraints:

- C1: $separated\_role(nurse, secretary)$
- C2: $separated\_role(nurse, physician)$
- C3: $separated\_role(secretary, physician)$

Notice that since we have $sub\_role(junior\_physician, physician)$ we can also derive:

- C4: $separated\_role(nurse, junior\_physician)$
- C5: $separated\_role(secretary, junior\_physician)$

Let us now detect and solve the potential conflicts of this access control policy:

- Step 1: Detection of potential conflicts.
  We have the following set of pairs of potentially conflicting rules:
  $Conflict = \{(R1,R2),(R1,R4),(R1,R6),(R2,R3),(R5,R6),(R6,R7)\}$
- Step 2: Resolution of potential conflicts.
  To solve the set of potential conflicts, we need to assign priority to every pair of potentially conflicting rules. For instance:
  $R1 < R2 < R3$
  $R1 < R4, R6 < R1$
  $R5 < R6 < R7$

4 Policy rewriting

We present an algorithm to rewrite a security policy that contains both permissions and prohibitions into an equivalent security policy that only contains permissions. In the initial policy we want to rewrite, we assume that every potential conflict is solved by priority assignment.

We first address the case of a close policy and then the case of an open policy. We recall that in the case of close policy, when no security rule applies to a given query, then the default decision is to reject the query. Whereas in an open policy, when no security rule applies to a given query, then the default decision is to accept the query.

4.1 Close policy case

Principle of the rewriting process: For every pair of potentially conflicting rule R_i and R_j such that R_i has higher priority than R_j and $decision(R_i) = prohibition$ and $decision(R_j) = permission$, rewrite R_i with R_j.
The rewriting process core: It keeps the rule with the higher priority R_i unchanged and it replaces the rule with the lower priority R_j by another rule after excluding from its application conditions the conditions of the higher priority rule R_i.
Let us write $SR(decision, r, ac, v, ctx, priority) = SR(decision, tuple_{SR}, priority)$, where $tuple_{SR} = \{(s,a,o,c)$ such that $s \in r, a \in ac, o \in v, c \in ctx\}$ and let us illustrate our algorithm using the following example of three conflicting rules:
$SR_1(permission, tuple_{SR_1}, priority_1)$
$SR_2(prohibition, tuple_{SR_2}, priority_2)$
$SR_3(permission, tuple_{SR_3}, priority_3)$
where each $tuple_{SR_i} = \{(s,a,o,c)$ such that $s \in r_i, a \in ac_i, o \in v_i, c \in ctx_i\}$ and $priority_1 < priority_2 < priority_3$.
The steps of our rewriting process are then the following:

1. The rule SR_3 is kept unchanged with its associated application condition $tuple_{SR_3}$.
2. Rewriting SR_2 with SR_3 is a process that replaces SR_2 by SR'_2 with:
   $tuple_{SR'_2} = \{(s,a,o,c)$ such that $(s,a,o,c) \in tuple_{SR_2} \backslash tuple_{SR_3}\}$
3. According to the principle of the rewriting process core, SR'_2 is kept unchanged and SR_1 is rewritten and replaced by SR'_1 with:
   $tuple_{SR'_1} = tuple_{SR_1} \backslash (tuple_{SR_2} \backslash tuple_{SR_3})$

$tuple_{SR'_1}$ can be simplified using some common properties of set theory. In a finite space E, we have the following properties over two sets S_1 and S_2:

$$S_1 \backslash S_2 = S_1 \cap C_E^{S_2} \tag{1}$$
$$C_E^{S_1 \cap S_2} = C_E^{S_1} \cup C_E^{S_2} \tag{2}$$
$$C_E^{C_E^{S_1}} = S_1 \tag{3}$$

Thus, using the property (1),(2) and (3), we get:

$$S_1 \backslash (S_2 \backslash S_3) = S_1 \cap (S_3 \cup C_E^{S_2}) \tag{4}$$

Coming back to our security rules and their associated conditions $tuple_{SR_i}$, $i \in \{1,2,3\}$, if we apply the above simplifications to $tuple_{SR'_1}$, we get:

$$tuple_{SR'_1} = tuple_{SR_1} \backslash (tuple_{SR_2} \backslash tuple_{SR_3}) = tuple_{SR_1} \cap (tuple_{SR_3} \cup C_E^{tuple_{SR_2}})$$

As the set $tuple_{SR_3}$ is already taken into account since we keep the rule of higher priority unchanged (*i.e* the rule SR_3), we can perform further simplification and we get:

$$tuple_{SR'_1} = tuple_{SR_1} \cap C_E^{tuple_{SR_2}} = tuple_{SR_1} \cap \overline{tuple_{SR_2}} \tag{5}$$

The simplification (5) is true in the case of 3 conflicting rules or even any number n of totally ordered conflicting rules. The correctness of this rewriting is proved in [6]. Thus, if we consider that we have n rules such as $priority_1 < priority_2 < priority_3 < ... < priority_n$ where $priority_n$ is the priority of SR_n, our rewriting algorithm for n ordered conflicting rules can be stated as the following:

- SR_n with its condition $tuple_{SR_n}$ are keep unchanged and
- for each j such that $1 \leq j, SR_{n-j}$ is replaced by SR'_{n-j} with the condition: $tuple_{SR'_{n-j}} = tuple_{SR_{n-j}} \backslash tuple_{SR_{n-(j-1)}}$

We get *in fine*:

$$\begin{aligned} tuple_{SR'_{n-j}} &= \mid r_{n-j} \backslash r_{n-(j-1)} \mid \times \mid ac_{n-j} \mid \times \mid v_{n-j} \mid \times \mid ctx_{n-j} \mid \\ &\cup \mid r_{n-j} \mid \times \mid ac_{n-j} \backslash ac_{n-(j-1)} \mid \times \mid v_{n-j} \mid \times \mid ctx_{n-j} \mid \\ &\cup \mid r_{n-j} \mid \times \mid ac_{n-j} \mid \times \mid v_{n-j} \backslash v_{n-(j-1)} \mid \times \mid ctx_{n-j} \mid \\ &\cup \mid r_{n-j} \mid \times \mid ac_{n-j} \mid \times \mid v_{n-j} \mid \times \mid ctx_{n-j} \backslash ctx_{n-(j-1)} \mid \end{aligned}$$

Actually, after applying the algorithm each rewritten rule is subdivided into four distinct sets of rules:

$$\begin{aligned} SR'_{n-j} \Leftrightarrow\ & SR'_{1.n-j}(decision, r_{n-j} \backslash r_{n-(j-1)}, ac_{n-j}, v_{n-j}, ctx_{n-j}, priority_{n-j}\} \\ & SR'_{2.n-j}(decision, r_{n-j}, ac_{n-j} \backslash ac_{n-(j-1)}, v_{n-j}, ctx_{n-j}, priority_{n-j}\} \\ & SR'_{3.n-j}(decision, r_{n-j}, ac_{n-j}, v_{n-j} \backslash v_{n-(j-1)}, ctx_{n-j}, priority_{n-j}\} \\ & SR'_{4.n-j}(decision, r_{n-j}, ac_{n-j}, v_{n-j}, ctx_{n-j} \backslash ctx_{n-(j-1)}, priority_{n-j}\} \end{aligned}$$

The rewriting process we have stated transforms a security policy into an equivalent policy that contains only permissions. All the conditions of prohibitions that are of higher priority are excluded from the permissions of less priority. Due to such an exclusion, if a prohibition rule of the policy before the application of our algorithm should have been applied to a given request, none of the resulting permissions of the rewritten policy should be matched. In this case, the default policy will be applied.

To illustrate our rewriting process, let us apply it to our motivating example. We shall obtain the following set of permissions:

- R1.1: $SR(permission, medical\_staff \backslash secretary \backslash nurse,$
  $manage, medical\_summary, any_C)$
- R1.2: $SR(permission, medical\_staff \backslash secretary,$
  $manage \backslash update, medical\_summary, any_C)$
- R3: $SR(permission, secretary, consult, medical\_summary, urgency)$

- R5.1: $SR(permission, physician \backslash junior\_physician,$
  $manage, medical\_record, any_C)$
- R5.2: $SR(permission, physician,$
  $manage \backslash update, medical\_record, any_C)$
- R7: $SR(permission, junior\_physician, update, medical\_record, urgency)$

Notice that the objective of the rewriting process is not to obtain a set of mutually independent permissions as suggested for instance in [1]. In our example, rules R5.1 and R5.2 are not mutually independent: if a subject assigned to role $physician \backslash junior\_physician$ asks for executing an action in $manage \backslash update$ on the view $medical\_record$, then both rules apply.

To obtain mutually independent rules, we could replace *physician* by *junior_physician* in rule R5.2. However, here, the objective of rewriting is actually not to obtain a "minimal" set of permissions. Instead, it is better for the negotiation process to obtain a set of "less" restrictive permissions. In our example, it would be inappropriate for the negotiation protocol to ask the subject to prove that he or she is a *junior_physician* if proving that he or she is a *physician* is sufficient to activate the rule.

4.2 Open policy case

The rewriting algorithm also applies when the security policy is open, i.e. when the default policy is to accept the request when no access control rule applies.

When the policy is open, we have simply to add a security rule specifying "everything is permitted":

- R0: $SR(permission, any_R, any_A, any_V, any_C)$

This security rule is associated with the lowest priority, i.e. for every other access control rule R_i of the policy, we have $R0 < R_i$.

We can then apply the rewriting algorithm without modification. Let us apply the approach to the following access control policy:

- R1: $SR(prohibition, secretary, manage, medical\_record, any_C)$
- R2: $SR(prohibition, nurse, update, any_V, any_C)$
- R3: $SR(permission, nurse, update, medical\_summary, urgency)$

Let us assume that R3 has higher priority than R2. After rewriting this policy, we shall get the following set of permissions:

- R0.1: $SR(permission, any_R \backslash secretary \backslash nurse, any_A, any_V, any_C)$
- R0.2: $SR(permission, any_R \backslash secretary, any_A \backslash update, any_V, any_C)$
- R0.3: $SR(permission, any_R, any_A \backslash manage, any_V, any_C)$
- R0.4: $SR(permission, any_R \backslash nurse, any_A, any_V \backslash medical\_record, any_C)$
- R0.5: $SR(permission, any_R, any_A \backslash update, any_V \backslash medical\_record, any_C)$

- R3: $SR(permission, nurse, update, medical\_summary, urgency)$

Rules R0.1 to R0.5 corresponds to rewriting rule R0 with prohibitions R1 and R2. Rule R3 is not rewritten since it has higher priority than rule R2 and is separated from rule R1.

5 Negotiation of negative attributes

The set theory we use in this paper is especially adapted to rewrite policies. We shall now explain how to define a negotiation protocol for the rewritten policies. For the purpose of negotiation, we need to specify conditions over attributes (*i.e.* credentials) to be requested from the requester. This is why we need to express our rewritten policy using conditions over the entities role, view, activity and context. Thus, we assume that every organizational entity involved in the negotiation is associated with a condition expressed in terms of attributes. This condition is a sufficient requirement to derive that a concrete entity (for instance a subject) is assigned to some organizational entity (for instance a role).

For example, the condition associated with the role *senior_physician* may be that the subject's occupation is *physician* and this subject starts this occupation for more than two years. Then, we have:
$occupation(S, physician) \wedge start\_occupation(S, physician, Start\_year) \wedge$
$year(current\_date, Current\_year) \wedge Current\_year - Start\_year \geq 2$
$\quad \rightarrow empower(s, senior\_physician)$
Now, if a subject involved in the negotiation has to prove that he or she is empowered in role *senior_physician*, then it will be requested to provide credentials to prove that his or her occupation is physician and that he or she is practicing this occupation for more than two years.

We have also to translate our set theory algebra into logical based conditions used in the negotiation process. This is straightforward because, if $Cond(E_1)$ and $Cond(E_2)$ respectively represent the sufficient conditions to be assigned into organizational entities E_1 and E_2, then we have the following equivalence:

$$Cond(E_1 \backslash E_2) \leftrightarrow Cond(E_1) \wedge not(Cond(E_2))$$
$$Cond(E_1 \cap E_2) \leftrightarrow Cond(E_1) \wedge Cond(E_2)$$
$$Cond(E_1 \cup E_2) \leftrightarrow Cond(E_1) \vee Cond(E_2)$$

As one can notice from the obtained rewritten security rules, we need to negotiate negative attributes such as $not(Cond(E_2))$. In the traditional centralized approach, the access controller will generally use "negation by failure" to evaluate negation. If the access controller cannot derive that some information is true, it will infer that this information is false. This corresponds to the close world assumption: The access controller knows every information necessary to evaluate the policy.

Of course, the close world assumption is not applicable to evaluate negative attributes in a negotiation process. Thus, the subject must provide credentials to prove that some condition is false.

If we assume that there is no credential that may be directly used to prove a negative attribute, then requester must provide credentials on positive conditions that are used to derive negative attributes proving that some condition is false. This derivation may be done using the inclusion hierarchy and separation constraint. For instance, having $separated\_entity(e_1, e_2)$, if the requester prove that he or she is assigned to the entity e_1, we can derive that he or she is not assigned to entity e_2. In addition to that if we have $sub\_entity(e_3, e_2)$, then we can derive that $separated\_entity(e_1, e_3)$. Thus, the given requester is not assigned to entity e_3.

For instance, in our motivation example, a subject can provide his or her credential proving that he or she is a senior physician to prove that he or she is not a medical secretary if (1) a senior physician is a sub role of physician (inclusion hierarchy) and (2) role physician is separated from role medical secretary (separation constraint).

6 Discussion and related work

Among other works done on negotiation of security policies we mainly discuss the Trustbuilder [15, 13, 14], Trust-χ [2] and XeNA [7] approaches.

TrustBuilder is a system for negotiation of trust in dynamic coalitions. It allows negotiating trust across organizational boundaries, between entities from different security domains. Using TrustBuilder, parties conduct bilateral and iterative exchanges of policies and credentials to negotiate access to system resources including services, credentials and sensitive system policies.

The TrustBuilder approach consists in gradually disclosing credentials in order to establish trust. The approach also incorporates policy disclosure; Only policies that are relevant to the current negotiation may be disclosed by the concerned parties. They specify what combinations of credentials one can present in order to gain access to a protected resource of the accessed service. In this way it is possible to focus the negotiation and base disclosures on need to know. Since these policies may contain sensitive information, their disclosure can also be managed by some strategies [12].

Trust-χ is another framework for trust negotiation specifically conceived for a peer-to-peer environment. Trust-χ proposes a language for the specification of policies and credentials needed in the negotiation process. Furthermore, it provides a variety of strategies for the negotiation. This latter consists of a set of phases to be sequentially executed. Trust-χ introduces *trust tickets* that are issued after a negotiation process succeeds. By certifying that previous negotiation process relative to a resource has succeeded, *i.e.* negotiating entities possess the required credentials, the trust tickets reduce as much as possible the number of credentials and policies needed in subsequent negotiation processes relative to the same resource thus speeding up these processes. Similarly to TrustBuilder, the Trust-χ disclosure policies state the conditions under which a resource can be revealed. Furthermore, *prerequisites*,(*i.e.* set of alternative policies to be disclosed before the policy they refer to) associated with sensitive policies manage their disclosure.

However, none of the previously described models deals with prohibitions.

XeNA is another negotiation approach based on the eXtensible Access Control Markup Language (XACML) [8, 7]. The proposed approach allows the expression of negative policies since XACML is a language that makes use of prohibitions. However, the authors do not explain how to deal with prohibitions in the negotiation policies. Their approach is based on a resource classification methodology [8]. It is the classification of a resource that determines if the access to this resource is negotiated (or not) and what are the negotiation requirements, *i.e.* needed credentials expressed in negotiation policies. They further propose a negotiation framework that uses this classification methodology and is based on the XACML architecture [7]. Two modules are introduced to manage the negotiation process: (1) the *negotiation module* is in charge of collecting the required information to establish a level of trust and to insure a successful evaluation of access and (2) the *exception treatment module* is called by the *negotiation module* in order to propose alternatives whenever an exception (*i.e.* non access or loop exception) is raised.

Thus, to our best knowledge, it is the first time that the problem of negotiating security policies that includes prohibition is addressed. We are currently implementing our approach as an extension of the above models.

7 Conclusion

We propose in this paper a new approach to negotiate security policies that include both permissions and prohibitions. Since it would be not fair to ask the subject to provide credentials in order to derive prohibitions, we suggest rewriting the policy so that it only contains permissions.

For this purpose and as suggested in the OrBAC model, the access control policy is defined in a structured way using the organizational entities of *role*, *activity*, *view* and *context* instead of the traditional concrete entities of *subject*, *action* and *object*. We also define a set theory algebra to aggregate elementary organizational entities into composite organizational entities. The rewriting algorithm uses, as preliminary steps, the approach suggested in [4] to detect and solve conflicts by assigning priorities to access control rules.

We then show that our rewriting algorithm provides means to transform an access control policy that contains both permissions and prohibitions into an equivalent one that only contains permissions. This rewritten access control policy is used in the negotiation process to determine which credentials are required to grant access to some requester. Since the rewritten policy generally specifies negative conditions, it is necessary to define strategies to negotiate these negative conditions. For this purpose, we actually assume that a credential cannot be directly used to prove a negative condition. Thus, we present an approach to derive negative attributes proving that some condition is false from credentials on positive attributes.

In future works we aim to implement this approach as an extension of existing prototypes, in particular TrustBuilder. We also plan to investigate how to negotiate policies that include obligations.

Acknowledgements This work is partially supported by the ANR Politess project.

References

1. J. G. Alfaro, F. Cuppens, and N. Cuppens-Boulahia. Towards Filtering and Alerting Rule Rewriting on Single-Component Policies. In *SAFECOMP*, Gdansk, Poland, September 2006.
2. E. Bertino, E. Ferrari, and A. C. Squicciarini. Trust-X: A Peer-to-Peer Framework for Trust Establishment. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):827–842, 2004.
3. E. Bertino, S. Jajodia, and P. Samarati. Supporting Multiple Access Control Policies in Database Systems. In *IEEE Symposium on Security and Privacy*, Oakland, USA, 1996.
4. F. Cuppens, N. Cuppens-Boulahia, and M. Ben Ghorbel. High level conflict management strategies in advanced access control models. *Electron. Notes Theor. Comput. Sci.*, 186:3–26, 2007.
5. F. Cuppens and A. Miège. Modelling Contexts in the Or-BAC Model. *ACSAC*, page 416, 2003.
6. N. Cuppens-Boulahia, F. Cuppens, D. Abi Haidar, and H. Debar. Negotiation of prohibition: An approach based on policy rewriting. Technical report, TELECOM Bretagne, 2008.
7. D. Abi Haidar, N. Cuppens, F. Cuppens, and H. Debar. Access Negotiation within XACML Architecture. *Second Joint Conference on Security in Networks Architectures and Security of Information Systems (SARSSI)*, June 2007.
8. D. Abi Haidar, N. Cuppens, F. Cuppens, and H. Debar. Resource Classification Based Negotiation in Web Services. *Third International Symposium on Information Assurance and Security (IAS)*, pages 313–318, August 2007.
9. S. Jajodia, S. Samarati, and V. S. Subrahmanian. A logical Language for Expressing Authorizations. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 1997.
10. A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *8th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, Lake Como, Italy, June 2003.
11. A. Miège. *Definition of a formal framework for specifying security policies. The Or-BAC model and extensions*. PhD thesis, ENST, June 2005.
12. K. Seamons, M. Winslett, and T. Yu. Limiting the Disclosure of Access Control Policies During Automated Trust Negotiation. In Network and Distributed System Security Symposium, San Diego, CA, April 2001.
13. K.E. Seamons, T. Chan, E. Child, M. Halcrow, A. Hess, J. Holt, J. Jacobson, R. Jarvis, A. Patty, B. Smith, T. Sundelin, and L. Yu. TrustBuilder: negotiating trust in dynamic coalitions. *Proceedings DARPA Information Survivability Conference and Exposition*, 2:49–51, April 2003.
14. B. Smith, K.E. Seamons, and M.D Jones. Responding to policies at runtime in TrustBuilder. *Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04)*, pages 149–158, June 2004.
15. T. Yu, M. Winslett, and K. E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security (TISSEC)*, 6(1):1–42, February 2003.