

# Inheritance hierarchies in the Or-BAC model and application in a network environment

Frédéric Cuppens<sup>1</sup>

Nora Cuppens-Boulahia<sup>1</sup>

Alexandre Miège<sup>1,2</sup>

<sup>1</sup>GET/ENST Bretagne, BP 78, 2 rue de la Châtaigneraie, 35512 Cesson Sévigné Cedex, France

<sup>2</sup>GET/ENST, 46 rue Barrault, 75634 Paris Cedex 13, France

## Abstract

*Role hierarchy was first introduced in the Role Based Access Control (RBAC) model. Inheritance of permissions is associated with this hierarchy. This is useful to design security policies in a modular way. In this paper, we extend this approach in the context of the Organization Based Access Control (Or-BAC) model. We first define hierarchies of roles, views and activities and formally model inheritance mechanism associated with each hierarchy. We then define hierarchy of organizations. We show that this provides efficient means to derive policies of security components from corporate security policies specification. We illustrate our approach in the context of network security policy, in particular to configure firewalls.*

## 1 Introduction

The inheritance mechanism was suggested in object oriented programming as an efficient way to design an application in a modular way. A similar mechanism is used in RBAC [14] when a hierarchy of roles is defined and associated with inheritance of permissions. The role hierarchy is a useful mean to structuring the security policy specification.

However, the concept of role hierarchy is not free of ambiguity [11, 12, 5]. Some of these ambiguities are directly related to the concept of role itself. If we consider the examples suggested in [14], there are actually several different interpretations of roles. Basically, a role allows a subject who is assigned to this role to perform some particular activities. This is the case of roles such as *physician*, *nurse* or *medical secretary*. However, in some examples, a role is related to a given *organization* [13]. For instance, we may consider roles such as *nurse in a cardiological department* or *nurse in a reanimation team*. This is interesting because permissions assigned to a given role may change from one organization to another. For example, a nurse may have different permissions if she performs her activities in a cardiological department or in a reanimation team. A role may also correspond to the activity of leading a given organization. Examples of such roles may be *director of an*

*hospital or head of a cardiological department.*

As we shall see in the following, these different interpretations of the role concept do not behave similarly with respect to permission inheritance. This is why it is important to have a model that provides means to make explicit such differences.

We shall analyze this problem in the context of the Or-BAC model [10]. This model is centered on the concept of *organization*. In Or-BAC, an organization corresponds to any entity that is in charge of managing a set of security rules (permissions or prohibitions). For instance, a given hospital is an organization. A concrete security component, such as a firewall, may be also viewed as an organization since it manages a set of security rules.

Role definition in Or-BAC is always related to a given organization. This is useful to avoid some ambiguities when defining role hierarchies and associating them with permission inheritance.

The Or-BAC model considers two other concepts, namely *activity* and *view*. A security policy assigned to a given organization is defined as permissions (or prohibitions) for roles to perform activities on views. In the following, we shall suggest defining hierarchies of activities and views and model inheritance mechanism associated with these hierarchies.

Finally, in Or-BAC, we can also define hierarchy of organizations. This possibility provides a very efficient mean to structuring the security policy specification, starting with high level organization such as an hospital and finishing with concrete security components such as a firewall.

In this paper, we aim to analyze and formally model inheritance of permissions and prohibitions through these different hierarchies. The remainder of this paper is organized as follows. Section 2 recalls main concepts of Or-BAC. Section 3 presents the various hierarchies respectively associated with roles, activities and views. Section 4 studies organization hierarchies. In section 5, we summarize how to specify a security policy in Or-BAC when hierarchies are used. Section 6 shows how our approach applies to network se-

curity policy specification. Finally, section 7 concludes and suggests several issues to this work.

## 2 Or-BAC

### 2.1 Basic concepts of Or-BAC

Or-BAC [10] is an access control model based on the organization concept. In Or-BAC, different organizations can specify their own access control policy using eight basic sets of entities: *Org*(anization), *Role*, *Activity*, *View*, *Subject*, *Action*, *Object* and *Context*. Basic predicates used in Or-BAC to model relationships between these eight entities are summarized in table 1.

As mentioned in the introduction, an organization is any entity that manages a set of security rules. A subject is an active entity that may be assigned to a role. We shall assume that  $Org \subseteq Subject$  so that a role may be assigned to an organization. For instance, roles “casualty department” or “rescue team” may be assigned to some organizations.

By means of the entity *Role*, we are able to structure the subjects and to update easily security policies when new subjects are added to the system. Since we have also to structure the objects and to add new objects to the system, a similar entity regarding objects is needed: the entity *View*. Roughly speaking, as in relational databases, a view corresponds to a set of objects that satisfy a common property.

Another entity is used to abstract actions: the entity *Activity*. Seeing that roles associate subjects that fulfil the same functions and views correspond to sets of objects that satisfy a common property, activities will join actions that share the same principles.

Subjects, objects and actions may have attributes. This is modelled by a set of binary predicates having the form  $att(ent, val)$  where  $ent$  is a subject, an object or an action and  $val$  is the value of attribute  $att$ . For instance, if  $med\_27$  is a medical record, then  $name(med\_27, John)$  means that  $med\_27$  is John’s medical record.

We assume that  $Subject \subseteq Object$  so that we can define views of subjects that we call *groups*. In Or-BAC, there is a clear difference between a role and a group. Permissions are assigned to roles whereas a group is simply a set of subjects that have some common properties. However, it is sometimes useful to assign the same role to every subject belonging to a given group. For this purpose, we can use the predicate  $G\_Empower(org, group, role)$  and specify the following rule:

$$\begin{aligned} - \text{GE: } & \forall org, \forall group, \forall role, \forall subject, \\ & Use(org, subject, group) \wedge \\ & G\_Empower(org, group, role) \\ & \rightarrow Empower(org, subject, role) \end{aligned}$$

Since the Or-BAC model allows the administrator to specify that some permission or prohibition only applies in specific

*contexts*, we also introduce the entity *Context*. Contexts are defined by logical rules whose conclusion is the predicate *Define* (see table 1 that gives the example of the *working\_hours* context). We say that a context  $c$  in organization  $org$  is defined by condition  $cond$  when there is a rule<sup>1</sup> having the form:  $Define(org, s, \alpha, o, c) \leftarrow cond$ . Specifying contexts in Or-BAC is further analyzed in [7].

### 2.2 Permission and prohibition

Permissions and prohibitions in Or-BAC are defined with predicates defined in figure 2. The access control policy is specified at two different levels: an abstract level that specifies permissions and prohibitions between role, activity and view, and a concrete level where permissions and prohibitions between subject, action and object are derived.

These two levels are related as follows. In a given organization  $org$ , a subject  $s$  is permitted to perform an action  $\alpha$  on an object  $o$  if (1)  $s$  is empowered to play a given role  $r$  in  $org$  and (2)  $\alpha$  implements a given activity  $a$  in  $org$  and (3)  $o$  is used in a given view  $v$  by  $org$ . If these three conditions are satisfied and if (4) the organization  $org$  grants to role  $r$  the permission to perform the activity  $a$  on the view  $v$ , then the request by the subject  $s$  to perform the action  $\alpha$  on the object  $o$  is accepted. Deriving concrete permissions from abstract permissions is modelled by the following rule:

$$\begin{aligned} - \text{RG}_1: & \forall org, \forall r, \forall a, \forall v, \forall s, \forall \alpha, \forall o, \\ & Permission(org, r, a, v, c) \wedge \\ & Empower(org, s, r) \wedge \\ & Consider(org, \alpha, a) \wedge \\ & Use(org, o, v) \wedge \\ & Define(org, s, o, \alpha, c) \\ & \rightarrow Is\_permitted(s, \alpha, o) \end{aligned}$$

Another similar rule (called  $\text{RG}_2$ ) is used to derive concrete *prohibitions* from abstract prohibitions.

### 2.3 Constraints

Constraints that apply to an access control policy was first suggested in the RBAC model (more precisely, in the  $\text{RBAC}_2$  sub-model [8]) and further analyzed in [1]. To specify constraints in the Or-BAC model, we introduce a predicate  $error()$ . A constraint is then modelled as a rule whose conclusion is  $error()$  (as suggested in [3, 9]).

For instance, we may specify that, in hospital  $H$ , a subject cannot be empowered in both roles *anesthetist* and *surgeon*:

$$\begin{aligned} - \text{C}_1: & \forall s, \\ & Empower(H, s, anesthetist) \wedge \\ & Empower(H, s, surgeon) \\ & \rightarrow error() \end{aligned}$$

1. We can assume that each context  $c$  is defined by a *unique* rule by using the fact that  $Define(org, s, \alpha, o, c) \leftarrow cond_1, \dots, Define(org, s, \alpha, o, c) \leftarrow cond_n$  is equivalent to  $Define(org, s, \alpha, o, c) \leftarrow (cond_1 \vee \dots \vee cond_n)$ .

Predicate name	Domain	Description
<i>Relevant_role</i>	$Org \times Role$	If <i>org</i> is an organization and <i>r</i> a role, then <i>Relevant_role(org,r)</i> means that playing role <i>r</i> is defined in organization <i>org</i> . Ex: <i>Relevant_role(H,physician)</i>
<i>Relevant_activity</i>	$Org \times Activity$	If <i>org</i> is an organization and <i>a</i> is an activity, then <i>Relevant_activity(org,a)</i> means that performing activity <i>a</i> is defined in organization <i>org</i> . Ex: <i>Relevant_activity(H,consult)</i>
<i>Relevant_view</i>	$Org \times View$	If <i>org</i> is an organization and <i>v</i> is a view, then <i>Relevant_view(org,o,v)</i> means that using view <i>v</i> is defined in organization <i>org</i> . Ex: <i>Relevant_view(H,medical_record)</i>
<i>Empower</i>	$Org \times Subject \times Role$	If <i>org</i> is an organization, <i>s</i> a subject and <i>r</i> a role, then <i>Empower(org,s,r)</i> means that <i>org</i> empowers subject <i>s</i> in role <i>r</i> . Ex: <i>Empower(H,John,physician)</i>
<i>Consider</i>	$Org \times Action \times Activity$	If <i>org</i> is an organization, $\alpha$ is an action and <i>a</i> is an activity, then <i>Consider(org,<math>\alpha</math>,a)</i> means that <i>org</i> considers that action $\alpha$ falls within the activity <i>a</i> . Ex: <i>Consider(H,"SELECT",consult)</i>
<i>Use</i>	$Org \times Object \times View$	If <i>org</i> is an organization, <i>o</i> is an object and <i>v</i> is a view, then <i>Use(org,o,v)</i> means that <i>org</i> uses object <i>o</i> in view <i>v</i> . Ex: <i>Use(H,med_27,medical_record)</i>
<i>Define</i>	$Org \times Subject \times Action \times Object \times Context$	If <i>org</i> is an organization, <i>s</i> a subject, $\alpha$ an action, <i>o</i> an object and <i>c</i> a context, then <i>Define(org,s,<math>\alpha</math>,o,c)</i> means that within organization <i>org</i> , context <i>c</i> holds between subject <i>s</i> , action $\alpha$ and object <i>o</i> . Ex: $\forall s, \forall \alpha, \forall o, Define(H, s, \alpha, o, working\_hours)$ $\leftarrow (08 : 00 \leq time(GLOBAL\_CLOCK) \wedge$ $time(GLOBAL\_CLOCK) \leq 19 : 00)$

FIG. 1 – Basic predicates of Or-BAC

In the following, we shall consider the following constraint that apply to any organization:

- $C_2: \forall org, \forall s, \forall r,$   
 $Empower(org, s, r) \wedge \neg Relevant\_role(org, r)$   
 $\rightarrow error()$

Rule  $C_2$  says that an organization *org* should not empower a subject *s* in role *r* if role *r* is not relevant in organization *org*.

There are other rules similar to  $C_2$  but for activities (called  $C_3$ ), views (called  $C_4$ ), permissions (called  $C_5$ ) and prohibitions (called  $C_6$ ).

### 3 Hierarchy within an organization

In Or-BAC, it is possible to consider hierarchies of roles (as suggested in [8]) but also of views and activities. Every hierarchy respectively defines a partial order relation over the set of roles, views and activities. We present general inheritance rules of permissions and prohibitions associated with these different hierarchies.

#### 3.1 Role hierarchy

Let us first address the case of inheritance between roles. In every organization, it is possible to associate a set of roles with a hierarchy. For this purpose, we introduce the predicate *sub\_role(org,r<sub>1</sub>,r<sub>2</sub>)*: in organization *org*, role *r<sub>1</sub>* is a sub-role of *r<sub>2</sub>*.

Notice that the role hierarchy depends on the organization. This means that the hierarchy may vary from one organization to another. Let us now model inheritance principles associated with this hierarchy.

Permission inheritance through the role hierarchy is modelled by the following rule:

- $RH_1: \forall org, \forall r_1, \forall r_2, \forall a, \forall v, \forall c,$   
 $sub\_role(org, r_1, r_2) \wedge$   
 $Permission(org, r_2, a, v, c)$   
 $\rightarrow Permission(org, r_1, a, v, c)$

This rule says that if role *r<sub>1</sub>* is a sub-role of role *r<sub>2</sub>* in organization *org*, then every permission assigned to role *r<sub>2</sub>* in organization *org* is also assigned to role *r<sub>1</sub>*.

Predicate name	Domain	Description
<i>Permission</i>	$Org \times Role \times Activity \times View \times Context$	If <i>org</i> is an organization, <i>r</i> a role, <i>a</i> an activity, <i>v</i> a view and <i>c</i> a context, then <i>Permission(org,r,a,v,c)</i> means that organization <i>org</i> grants to role <i>r</i> the permission to perform activity <i>a</i> on view <i>v</i> in context <i>c</i> . Ex: <i>Permission(H,physician,consult,medical_record,working_hours)</i>
<i>Prohibition</i>	$Org \times Role \times Activity \times View \times Context$	If <i>org</i> is an organization, <i>r</i> a role, <i>a</i> an activity, <i>v</i> a view and <i>c</i> a context, then <i>Prohibition(org,r,a,v,c)</i> means that organization <i>org</i> prohibits role <i>r</i> from performing activity <i>a</i> on view <i>v</i> in context <i>c</i> . Ex: <i>Prohibition(H,nurse,consult,medical_record,night)</i>
<i>Is_permitted</i>	$Subject \times Action \times Object$	If <i>s</i> is a subject, $\alpha$ an action, <i>o</i> an object, then <i>Is_permitted(s,<math>\alpha</math>,o)</i> means that <i>s</i> is concretely permitted to perform action $\alpha$ on object <i>o</i> . Ex: <i>Is_permitted(John,"SELECT",med_27)</i>
<i>Is_prohibited</i>	$Subject \times Action \times Object$	If <i>s</i> is a subject, $\alpha$ an action, <i>o</i> an object, then <i>Is_prohibited(s,<math>\alpha</math>,o)</i> means that <i>s</i> is concretely prohibited to perform action $\alpha$ on object <i>o</i> . Ex: <i>Is_prohibited(Mary,"DELETE",med_27)</i>

FIG. 2 –. Permissions and prohibitions specification in Or-BAC

Regarding prohibition inheritance, things are more complex. It is necessary to recognize that the relationships between roles in the hierarchy may be semantically different. We actually identify two different relationships:

- Relationship of specialization/generalization. For instance, role *surgeon* is a specialization of role *physician*. To model this first relationship we shall use the following predicate:  
*specialized\_role(org,r<sub>1</sub>,r<sub>2</sub>)*: in organization *org*, role *r<sub>1</sub>* is a specialized role of role *r<sub>2</sub>*.
- Relationship of organizational hierarchy. For instance, role *department director* may be defined as hierarchically higher than role *team head*. This second relationship is modelled by the following predicate:  
*senior\_role(org,r<sub>1</sub>,r<sub>2</sub>)*: in organization *org*, role *r<sub>1</sub>* is a senior role of role *r<sub>2</sub>*.

We consider that relationship *specialized\_role* is included in *sub\_role*:

- RH<sub>2</sub>:  $\forall org, \forall r_1, \forall r_2,$   
 $specialized\_role(org, r_1, r_2)$   
 $\rightarrow sub\_role(org, r_1, r_2)$

The consequence is that rule RH<sub>1</sub> applies to the specialization role hierarchy and thus permissions are inherited through this hierarchy. We also consider that prohibitions are inherited through the specialization role hierarchy:

- RH<sub>3</sub>:  $\forall org, \forall r_1, \forall r_2, \forall a, \forall v, \forall c,$   
 $specialized\_role(org, r_1, r_2) \wedge$

$$Prohibition(org, r_2, a, v, c) \\ \rightarrow Prohibition(org, r_1, a, v, c)$$

For instance, every prohibition of the role *physician* is inherited by the role *surgeon*. This is compatible with the intuition that a *surgeon* is a special case of *physician*.

By contrast, the relationship *senior\_role* is generally not included in *sub\_role*. This means that we may have:

$$\neg \exists org, \exists r_1, \exists r_2, \\ senior\_role(org, r_1, r_2) \wedge \neg sub\_role(org, r_1, r_2)$$

For instance, we can consider that role *hospital director* is hierarchically higher than *physician*. However, in some hospitals, role *hospital director* is a purely administrative role that is not assigned to a physician. In this case, there is no reason to conclude that *hospital director* is a sub role of physician.

Now let us assume that role *r<sub>1</sub>* is a senior role of *r<sub>2</sub>* and that *r<sub>1</sub>* is also a sub-role of *r<sub>2</sub>*. In this case, the idea is to consider that *r<sub>1</sub>* is “more powerful” than *r<sub>2</sub>*. This is compatible with rule RH<sub>1</sub> above that specifies that *r<sub>1</sub>* inherits the permissions assigned to *r<sub>2</sub>*. However, if we assume that *r<sub>1</sub>* inherits the prohibitions assigned to *r<sub>2</sub>*, this will not make *r<sub>1</sub>* more powerful than *r<sub>2</sub>*. This is why it would be better to consider that, in case of organizational hierarchy, prohibitions are inherited “upward”. This is modelled by the following rule:

$$\neg RH_4: \forall org, \forall r_1, \forall r_2, \forall a, \forall v, \forall c, \\ sub\_role(org, r_1, r_2) \wedge senior\_role(org, r_1, r_2) \wedge$$

$$\begin{aligned} & Prohibition(org, r_1, a, v, c) \wedge \\ & \rightarrow Prohibition(org, r_2, a, v, c) \end{aligned}$$

For instance, if we consider that *department director* is a senior role and also a sub role of *head team*, then *department director* inherits the permissions assigned to *team head* (from rule RH<sub>1</sub>) and *team head* inherits the prohibitions assigned to *department director* (from rule RH<sub>4</sub>).

To summarize, we have defined three different role hierarchies: *sub\_role*, *specialized\_role* (included in *sub\_role*) and *senior\_role* (generally not included in *sub\_role*). If we are only interesting in the problem of inheritance of permissions and prohibitions, we can actually consider only two hierarchies: *sub\_role* and *specialized\_role*. Regarding the *specialized\_role* hierarchy, rules RH<sub>1</sub>, RH<sub>2</sub> and RH<sub>3</sub> apply. Regarding the remainder of the *sub\_role* hierarchy, rules RH<sub>1</sub> and RH<sub>4</sub> apply.

Notice also that every inheritance rule presented in this section may have exceptions. For instance, one may specify that, in hospital *H*, physicians are prohibited to consult the medical records of people who are not their patients. Thus, applying rule RH<sub>3</sub>, a surgeon will inherit this prohibition. However, one can explicitly specify (as an exception) that, in hospital *H*, a surgeon is permitted to consult every medical records, even if they do not concern the surgeon's patient. How to manage exception is further discussed in section 5.2.

### 3.2 Activity hierarchy

We now suggest defining inheritance between activities. For this purpose, in every organization, the set of activities is associated with a hierarchy. This is modelled by the predicate *sub\_activity*(*org*, *a*<sub>1</sub>, *a*<sub>2</sub>): in organization *org*, activity *a*<sub>1</sub> is a sub-activity of *a*<sub>2</sub>.

The interpretation of this hierarchy is that, in organization *org*, activity *a*<sub>1</sub> is a specialization of activity *a*<sub>2</sub>. For instance, in hospital *H*, the activity of *managing* (for example medical records) may be specialized into the activities of *creating*, *consulting* and *updating*. Thus we have: *sub\_activity*(*H*, *creating*, *managing*) and similarly for *consulting* and *updating*.

This hierarchy is associated with permission inheritance. This is modelled by the following rule:

$$\begin{aligned} - \text{AH}_1: & \forall org, \forall r, \forall a_1, \forall a_2, \forall v, \forall c, \\ & Permission(org, r, a_2, v, c) \wedge \\ & sub\_activity(org, a_1, a_2) \\ & \rightarrow Permission(org, r, a_1, v, c) \end{aligned}$$

For instance, let us assume that, in hospital *H*, physicians are permitted to manage medical records of their patients. Applying rule AH<sub>1</sub> we can derive that physicians are also permitted to create, consult and update medical records of their patients.

We consider that a similar rule (called AH<sub>2</sub>) applies to inheritance of prohibitions. For instance, let us assume that, in hospital *H*, nurses are prohibited to manage medical records. Applying rule AH<sub>2</sub> we can derive that nurses are also prohibited to create, consult and update medical records.

### 3.3 View hierarchy

Using a similar approach, the set of views is associated with a hierarchy that depends on the organization. This is modelled by the predicate *sub\_view*(*org*, *v*<sub>1</sub>, *v*<sub>2</sub>): in organization *org*, view *v*<sub>1</sub> is a sub-view of *v*<sub>2</sub>.

Our interpretation is that, in *org*, view *v*<sub>1</sub> is a specialization of view *v*<sub>2</sub>. This is actually close to class inheritance hierarchy used in object-oriented hierarchy (*Isa* hierarchy).

In the context of Or-BAC, view hierarchies are associated with the permission inheritance that is modelled by the following rules:

$$\begin{aligned} - \text{VH}_1: & \forall org, \forall r, \forall a, \forall v_1, \forall v_2, \forall c, \\ & Permission(org, r, a, v_2, c) \wedge \\ & sub\_view(org, v_1, v_2) \\ & \rightarrow Permission(org, r, a, v_1, c) \end{aligned}$$

A similar rule (called VH<sub>2</sub>) applies to inheritance of prohibitions. For instance, in a hospital, we may consider that the view *surgeon\_record* is a sub-view of the view *medical\_record*. In this case, a role who is permitted or prohibited to perform a given activity on the view *medical\_record* will be permitted or prohibited to perform the same activity on the view *surgeon\_record*.

We can also define the concept of *derived* view as a special case of view specialization. Hence, we can define that a view *v*<sub>1</sub> is derived from view *v*<sub>2</sub> if there is a rule having the following form:

$$\begin{aligned} - & \forall org, \forall obj, \forall v_1, \forall v_2, \\ & (Use(org, obj, v_2) \wedge Condition) \rightarrow Use(org, obj, v_1) \end{aligned}$$

where *Condition* is a logical condition used to specialize view *v*<sub>2</sub> into view *v*<sub>1</sub>.

## 4 Organization hierarchy

Previous section showed how to define hierarchies between roles, activities and views within a given organization. In this section, we study how to define hierarchies of organizations. For this purpose, we introduce the predicate *sub\_organization*(*org*<sub>1</sub>, *org*<sub>2</sub>): organization *org*<sub>1</sub> is a sub-organization of organization *org*<sub>2</sub>. We assume that this predicate defines a partial order relation on the set of organizations.

For instance, if *H* is an hospital and *dept8* is the casualty department of this hospital, then we have: *sub\_organization*(*dept8*, *H*)

We may actually require that every sub-organization  $org_1$  of a given organization  $org_2$  is assigned to a role. This requirement is modelled by the following constraint:

- $C_7: \forall org_1, \forall org_2, \forall r,$   
 $sub\_organization(org_1, org_2) \wedge$   
 $\neg Empower(org_2, org_1, r)$   
 $\rightarrow error()$

For instance, in the above example, constraint  $C_7$  is satisfied if we have:  $Empower(H, dept8, casualty\_dept)$ .

Notice that some roles may be defined in a given organization but not in some of its sub-organizations. For instance if  $dept7$  is the management department of the hospital  $H$ , then the role nurse may be not defined in  $dept7$  (we have  $\neg Relevant\_role(dept7, nurse)$ ) whereas it is defined in  $H$  (we have  $Relevant\_role(H, nurse)$ ).

Conversely, if  $org_1$  is a sub-organization of  $org_2$ , then some roles may be defined in  $org_1$  whereas they are not defined in  $org_2$ .

Similar comments apply to views and activities: if  $org_1$  is a sub-organization of  $org_2$ , then the views (resp. activities) defined in  $org_1$  may be disjoint from the views (resp. activities) defined in  $org_2$ .

#### 4.1 Hierarchy inheritance

Let us assume that  $org_1$  is a sub-organization of  $org_2$ . For those roles of  $org_2$  that are relevant in  $org_1$ , we consider that the role hierarchy defined in  $org_2$  also applies in  $org_1$ . This is modelled by the following rule:

- $HH_1: \forall org_1, \forall org_2, \forall r_1, \forall r_2,$   
 $sub\_organization(org_2, org_1)$   
 $sub\_role(org_1, r_1, r_2) \wedge$   
 $relevant\_role(org_2, r_1) \wedge relevant\_role(org_2, r_2)$   
 $\rightarrow sub\_role(org_2, r_1, r_2)$

Similar principles apply to inheritance of specialized role hierarchy and also of activity and view hierarchies through the organization hierarchy. Thus, we obtain three other rules (respectively called  $HH_2$ ,  $HH_3$  and  $HH_4$ ) by replacing the  $sub\_role$  predicate in rule  $HH_1$  by the  $specialized\_role$  predicate (resp. the  $sub\_activity$  and  $sub\_view$  predicates).

#### 4.2 Permission and prohibition inheritance

We accept similar principles for inheritance of permissions and prohibitions through the organization hierarchy provided that the role, activity and view in the scope of the permission or prohibition are relevant in the sub-organization. This is modelled by the following rule:

- $OH_1: \forall org_1, \forall org_2, \forall r, \forall a, \forall v, \forall c,$   
 $sub\_organization(org_2, org_1)$   
 $Permission(org_1, r, a, v, c) \wedge$   
 $relevant\_role(org_2, r) \wedge$   
 $relevant\_activity(org_2, a) \wedge$   
 $relevant\_view(org_2, v)$   
 $\rightarrow Permission(org_2, r, a, v, c)$

A similar rule applies to inheritance of prohibition (rule called  $OH_2$ ).

## 5 Specifying a security policy in Or-BAC

### 5.1 Policy theory

To summarize, a security policy that includes inheritance hierarchies is modelled as a logical theory corresponding to the following definition.

**Definition 1:** In the Or-BAC model, a security policy  $pol$  is modelled as a logical theory  $T_{pol}$  defined as follows:

- Sets of facts using predicates  $Relevant\_role$ ,  $Relevant\_activity$  and  $Relevant\_view$
- Sets of facts using predicates  $Empower$ ,  $Use$ ,  $Consider$ ,  $Permission$  and  $Prohibition$
- Rule  $GE$  and facts using predicate  $G\_Empower$
- A set of rules for derived view definition (section 3.3)
- A set of facts using attribute binary predicates for describing attribute values of subjects, actions and objects
- A set of context definition rules, i.e. rules whose conclusion is the predicate  $Define(org, s, \alpha, o, c)$
- Sets of facts (inheritance hierarchies) using predicates  $sub\_role$ ,  $specialized\_role$ ,  $sub\_activity$  and  $sub\_view$
- Rules  $RG_1$  and  $RG_2$  for deriving concrete permissions and prohibitions (section 2.1)
- Rules  $RH_1$  to  $RH_4$  (role inheritance rules),  $AH_1$  and  $AH_2$  (activity inheritance rules) and  $VH_1$  and  $VH_2$  (view inheritance rules)
- Rules  $HH_1$  to  $HH_4$  (hierarchy inheritance rules)
- Rules  $OH_1$  and  $OH_2$  (organization inheritance rules)
- A set of constraints, i.e. rules whose conclusion is the predicate  $error()$ .

**Definition 2:** The security  $pol$  violates a constraint if it is possible to derive  $error()$  from  $T_{pol}: T_{pol} \vdash error()$

### 5.2 Conflicts

Since the Or-BAC model provides means to specify both permissions and prohibitions, it is possible that some conflicts arise. This occurs when a given user is both permitted and prohibited to perform a given action on a given object. To model such a situation, we introduce a predicate called  $conflict()$ . The following rule specifies a situation of conflict:

- $RC: \forall s, \forall \alpha, \forall o,$   
 $Is\_permitted(s, \alpha, o) \wedge Is\_prohibited(s, \alpha, o)$   
 $\rightarrow conflict()$

**Definition 3:** There is a conflict in the security  $pol$  if it is possible to derive  $conflict()$  from  $T_{pol}: T_{pol} \vdash conflict()$

Notice that exceptions in the inheritance rules may lead to conflicts. For instance, in the example presented in section 3.1, a surgeon may be both prohibited to consult a given medical record of someone who is not his or her patient (prohibition inherited from role *physician*) and permitted to do so (explicit permission assigned to role *surgeon*).

In [6], we suggest managing such a conflict by assigning priority to permissions and prohibitions. In our previous example, prohibition inherited from *physician* should have lower priority than explicit permission assigned to *surgeon* and thus this surgeon should be finally permitted to consult the medical record. However, this is not the purpose of this paper to further discuss how to manage conflicts in the Or-BAC model (see [6] for a detailed presentation).

## 6 Application

In this section, we model a local area network, its security architecture and its connectivity to the Internet. We choose to take back the example used in Firmato [2] so as to bring out how Or-BAC provides a natural statement of various entities and concepts used in the security architecture. Furthermore, we show that hierarchy notions of extended Or-Bac applied to the central entities, say organization, role, activity and view, avoid the use of artifices like "open" and "closed" groups suggested in [2].

### 6.1 Organization hierarchy

We want to model the access control policy of a corporate network used in an organization  $H$ .  $H$  has a two-firewall network configuration, as shown in Figure 3. As presented in [2], the external firewall guards the corporation's Internet connection. Behind it is the DMZ, which contains the corporation's externally visible servers. In our case these servers provide http/https (web), ftp, smtp (email), and dns services. The corporation actually only uses two hosts to provide these services, one for dns and the other (called *Multi\_server*) for all the other services. Behind the DMZ is the internal firewall which guards the corporation's intranet. This firewall actually has three interfaces: one for the DMZ, one for the private network zone, and a separate interface connecting to the firewall administration host. Within the private network zone, there is one distinguished host, *Admin\_serv*, which provides the administration for the servers in the DMZ.

In Or-BAC, we can introduce several organizations to model such a configuration. First, there is an organization  $H$  and to simplify, we shall actually identify  $H$  with its corporate network.  $H$  has two sub-organizations denoted  $H\_fw_1$  corresponding to the external firewall and  $H\_fw_2$  corresponding to the internal firewall. We may actually introduce other organizations, such as  $H\_private\_net$  if one would like to specify the policy to be enforced within  $H$  private

network. For instance, if  $H$  is an hospital, we might introduce roles such as *physician*, *nurse*, etc., to model this part of the policy. However, for the sake of simplicity, we shall not further refine this part. Notice that we could also use an organization called *internet* if we had to specify an explicit policy to be enforced by the Internet.

### 6.2 Subject

In this example, subjects correspond to hosts identified by their IP address. So if  $h$  is an host, then predicate  $address(h,a)$  means that the IP address of  $h$  is  $a$ . Roles are assigned to hosts as suggested in section 6.3 below. For this purpose, predicate *Empower* enables us to assign a role to a given host. However, it would be easier to cluster hosts into groups (also called *zone* in Firmato) and use  $G\_Empower$  to assign the same role to every host belonging to the same group. For instance, we can define the group *Private\_net* as follows:

$$\begin{aligned} & - \forall h, Use(H,h,Private\_net) \leftarrow \\ & \quad Use(H,h,Host) \wedge address(h,a) \wedge a \in 111.222.2.* \\ & \quad \wedge \\ & \quad \neg Use(H,h,Firewall\_interface) \end{aligned}$$

### 6.3 Role

Hosts may be assigned to roles presented in figure 4. All these roles are relevant to  $H$ . Figure 4 specifies those roles that are respectively relevant to organizations  $H\_fw_1$  and  $H\_fw_2$ . Notice that role *Firewall* is relevant to  $H\_fw_2$  (for administration purpose) but not to  $H\_fw_1$ . For each role, figure 4 also presents the sub-roles of this role. In this example, the sub-role hierarchy actually corresponds to a specialization role hierarchy.

### 6.4 Activity

Activities correspond to various services available in corporate network  $H$ . We define a first activity *all\_tcp* with different tcp activities (such as *smtp*, *ssh* and *https*) as sub-activities. Similarly, we define an activity *all\_icmp* with different icmp activities (such as *ping*) as sub-activities. We also define two other activities. *admin\_to\_gtwy* has two sub-activities: *ssh* and *ping*. *gtwy\_to\_admin* has also two sub-activities: *ssh* and *https*. All these activities are relevant in organizations  $H$ ,  $H\_fw_1$  and  $H\_fw_2$ .

The main difference here with the approach suggested in [2] is that we use hierarchies of activities whereas Firmato defines elementary services and groups of services. Our approach is more generic since permissions and prohibitions will all apply to a unique entity, the activity.

### 6.5 View

Views are used to structure objects on which network services apply. Thus, we define a view called *target* having

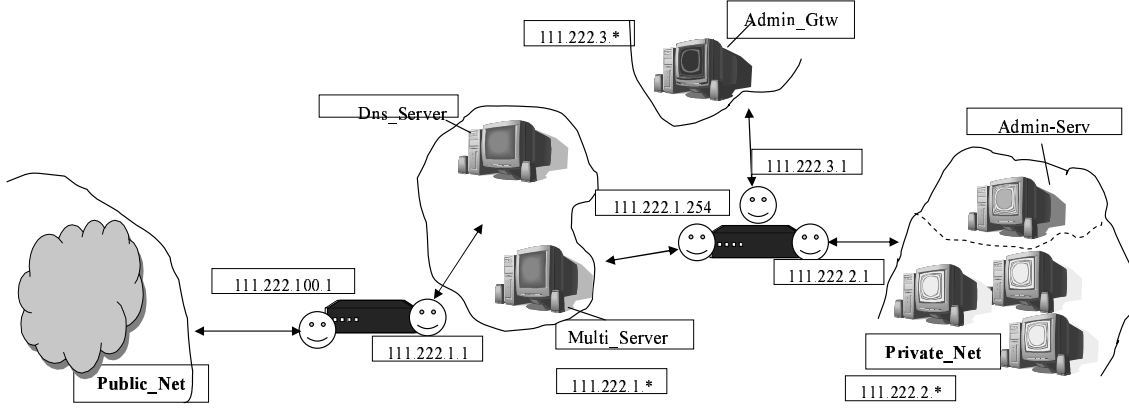


FIG. 3 –. Application example

Role name	Description	<i>sub_role</i>	Relevant to $H\_fw_1$	Relevant to $H\_fw_2$
<i>Public_host</i>	Role assigned to hosts in view <i>Public_Net</i>	-	X	
<i>Private_host</i>	Role assigned to hosts in view <i>Private_Net</i>	-		X
<i>Firewall</i>	Role assigned to firewall interfaces	<i>External_firewall</i> <i>Internal_firewall</i>		X
<i>External_firewall</i>	Role assigned to external firewall interfaces	-	X	X
<i>Internal_firewall</i>	Role assigned to internal firewall interfaces	-		X
<i>DNS_server</i>	Role assigned to the DNS server	-	X	X
<i>Ftp_server</i>	Role assigned to ftp server	<i>Multi_server</i>	X	X
<i>Mail_server</i>	Role assigned to mail server	<i>Multi_server</i>	X	X
<i>Web_server</i>	Role assigned to web server	<i>Multi_server</i>	X	X
<i>Multi_server</i>	Role assigned to multi-server	-	X	X
<i>Adm_fw.host</i>	Role assigned to hosts in view <i>Admin_gtw</i>	-	X	X
<i>Adm_serv.host</i>	Role assigned to hosts in view <i>Admin_serv</i>	-		X

FIG. 4 –. Role description

two attributes: *content* that corresponds to messages transmitted when using the service and *dest* that corresponds to the destination host of the service. The destination host is identified by its role.

Actually, the *content* attribute is not used in the example because we shall only consider filtering rules on the destination host. However, it would be useful to filter messages depending on their content.

We can then define sub-views derived from view *target* according to the role assigned to the destination host. For instance, we can define sub-view *to\_dns* as follows:

$$- \forall o, Use(H, o, to\_dns) \leftarrow Use(H, o, target) \wedge dest(o, dns)$$

This would lead to define as many views as there are roles. This would be quite fastidious. Instead, we suggest defining a function *to\_target* from roles into views. Views created by function *to\_target* are defined as follows:

$$- \forall o, \forall r, Use(H, o, to\_target(r)) \leftarrow Use(H, o, target) \wedge dest(o, r)$$

We consider that a view *to\_target(r)* is relevant in one of the organization of our example if *r* is a role relevant in this organization. We also consider that if role *r*<sub>1</sub> is a sub-role of role *r*<sub>2</sub>, then view *to\_target(r*<sub>1</sub>) is a sub-view of view *to\_target(r*<sub>2</sub>).

## 6.6 Security policy

We can now specify different permissions that apply to organization *H*. These permissions correspond to the security policy presented in [2]. Figure 5 lists how these permissions are modelled in Or-BAC. For the sake of simplicity, we do not specify prohibitions and all permissions are supposed to apply in every context (corresponding to *default* context that is always evaluated to *true*).

Compared to Firmato, one significant advantage of our approach is that it enables us to automatically derive permissions that respectively apply to *H\_fw*<sub>1</sub> and *H\_fw*<sub>2</sub> (using rule OH<sub>1</sub> for deriving permissions in sub-organizations of *H*). The results we obtain for *H\_fw*<sub>1</sub> is presented in figure 6. To illustrate this derivation process

let us consider the following permission:

$Permission(H,adm\_fw\_host,admin\_to\_gtwy,$   
 $to\_target(firewall),default)$

Since role  $adm\_fw\_host$ , activity  $admin\_to\_gtwy$  and view  $to\_target(firewall)$  are relevant in  $H\_fw_2$ , we can apply rule  $OH_1$  to derive:

$Permission(H\_fw_2,adm\_fw\_host,admin\_to\_gtwy,$   
 $to\_target(firewall),default)$

However, since view  $to\_target(firewall)$  is not relevant in  $H\_fw_1$ , we cannot derive a similar permission for  $H\_fw_1$ . But, view  $to\_target(external\_firewall)$  is a sub-view of  $to\_target(firewall)$ . Since  $to\_target(external\_firewall)$  is a relevant view in  $H\_fw_1$ , we can apply rules  $VH_1$  and  $OH_1$  to derive:

$Permission(H\_fw_1,adm\_fw\_host,admin\_to\_gtwy,$   
 $to\_target(external\_firewall),default)$

Notice that one permission, namely:

$Permission(H,private\_host,all\_tcp,$   
 $to\_target(public\_host),default)$

is not inherited by  $H\_fw_1$  nor  $H\_fw_2$ . This is because role  $private\_host$  is only relevant to  $H\_fw_2$  whereas view  $target(public\_host)$  is only relevant to  $H\_fw_1$ . So, no firewall alone can manage this permission. In this case, our proposal is to use this permission to configure both firewalls.

Our approach is actually based on fewer concepts than Firmato. In particular, we do not need to use the notion of “closed” groups. A closed group does not inherit from higher groups in the hierarchy. The example suggested in Firmato is the *firewall* group that should not inherit from *private\\_host*. We guess that the notion of closed group is complex to manage and actually not necessary. In our approach, we have simply to specify that *private-net* does not include *firewall-interface* (see the definition suggested for *private-net* in section 6.2). Our approach can also be used to handle more complex applications that include prohibitions, requirements on message contents or contextual rules.

## 7 Conclusion

In this paper we show how to model inheritance hierarchies in the Or-BAC model. Previous works related to inheritance hierarchies only considered role hierarchies (as suggested in the RBAC model). By contrast, we define role, activity, view and organization hierarchies and analyze inheritance of both permissions and prohibitions through these hierarchies.

Regarding the role hierarchy, we show that it is useful to distinguish between two different hierarchies: the specialization/generalization role hierarchy and the senior/junior role hierarchy. Permissions are inherited “downward” in both hierarchies (the more specialized role inherits from the less specialized role and the senior role inherits from the junior role). However, we suggest that prohibitions are inhe-

rited downward in the specialization/generalization hierarchy (as for permissions) whereas they are inherited upward in the senior/junior role hierarchy (the junior inherits from the senior role). Previous proposals did not make such a distinction. For instance, [3] always considers that prohibitions are inherited upward through the role hierarchy. We guess that our proposal eliminate some ambiguities of previous approaches.

Regarding the activity hierarchy, we only consider specialization hierarchy. We may actually define other activity “decomposition” such as decomposing an activity  $a$  into  $b; c$  denoting activity  $b$  followed by activity  $c$ . It is clear that if a given role  $r$  is permitted to perform activity  $a$ , then  $r$  should have also permission to perform activity  $b$ . However, defining permission associated with  $c$  is more complex; role  $r$  will be permitted to perform activity  $c$  only after having performed activity  $a$ . In Or-BAC, we guess that we can represent such a constraint using the notion of context. Modelling such activity decomposition in Or-BAC is an interesting problem that have several applications, in particular to specify security policies for workflow systems [4]. This represents further work that remains to be done.

We also only consider simple view hierarchy corresponding to specialization/generalization. We plan to analyze other relationships between views, in particular the aggregation relationship (also called *Part\_of* relationship). It would be interesting to have a general model that defines how permissions and prohibitions propagate from a given view to its different sub-parts. There are several applications to such a model, for instance UML modelling or XML security. However, some difficulties arise, in particular some activities that are relevant to a given view may not apply to some of its sub-parts. This is another problem that requires more investigation.

Finally, we define organization hierarchy and model inheritance of permissions and prohibitions in this hierarchy. We show how this hierarchy is useful to derive, from corporate security policies specification, policies of particular security components such as a firewall. We have implemented a translation module to concretely generate rules to automatically configure the NetFilter firewall. We also plan to apply a similar approach to generate policies of other components such as operating systems or database management systems and to use Or-BAC to model interoperability requirements between these various policies.

## Références

- [1] G.-J. Ahn and R. Sandhu. Role-Based Authorization Constraints Specification. *ACM Transactions on Information and System Security*, 3(4), November 2000.
- [2] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. In *20th IEEE Symposium on Security and Privacy*, pages 17–31, Oakland, California, May 1999.

```

Permission(H,adm_fw_host,admin_to_gtwy,to_target(firewall),default)
Permission(H,firewall,gtwy_to_admin,to_target(adm_fw_host),default)
Permission(H,private_host,all_tcp,to_target(public_host),default)
Permission(H,adm_server_host,all_tcp,to_target(dns_server),default)
Permission(H,adm_server_host,all_tcp,to_target(multi_server),default)
Permission(H,public_host,smtp,to_target(mail_server),default)
Permission(H,public_host,dns,to_target(dns_server),default)
Permission(H,public_host,ftp,to_target(ftp_server),default)
Permission(H,public_host,https,to_target(web_server),default)
Permission(H,private_host,smtp,to_target(mail_server),default)
Permission(H,private_host,dns,to_target(dns_server),default)
Permission(H,private_host,ftp,to_target(ftp_server),default)
Permission(H,private_host,https,to_target(web_server),default)
Permission(H,dns_server,dns,to_target(public_host),default)
Permission(H,ftp_server,ftp,to_target(public_host),default)
Permission(H,dns_server,dns,to_target(private_host),default)
Permission(H,ftp_server,ftp,to_target(private_host),default)

```

FIG. 5 –. Permissions in organization  $H$

```

Permission(H_fw1,adm_fw_host,admin_to_gtwy,to_target(external_firewall),default)
Permission(H_fw1,external_firewall,gtwy_to_admin,to_target(adm_fw_host),default)
Permission(H_fw1,public_host,smtp,to_target(mail_server),default)
Permission(H_fw1,public_host,dns,to_target(dns_server),default)
Permission(H_fw1,public_host,ftp,to_target(ftp_server),default)
Permission(H_fw1,public_host,https,to_target(web_server),default)
Permission(H_fw1,dns_server,dns,to_target(public_host),default)
Permission(H_fw1,ftp_server,ftp,to_target(public_host),default)

```

FIG. 6 –. Permissions in organization  $H\_fw_1$

- [3] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A Logical Framework for Reasoning about Access Control Models. *ACM Transactions on Information and System Security*, 6(1), February 2003.
- [4] Elisa Bertino, Elena Ferrari, and Vijay Atluri. Specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, 2(1), February 1999.
- [5] J. Crampton. On permissions, inheritance and role hierarchies. In *10th ACM Conference on Computer and Communication Security*, Washington, November 2003.
- [6] F. Cuppens and A. Miège. Conflict management in the or-bac model. Technical report, ENST Bretagne, December 2003.
- [7] F. Cuppens and A. Miège. Modelling contexts in the Or-BAC model. In *19th Annual Computer Security Applications Conference*, Las Vegas, December 2003.
- [8] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and System Security*, 4(3):222–274, August 2001.
- [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. Saurerel, and G. Trouessin. Organization Based Access Control. In *Proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, Lake Come, Italy, June 2003.
- [11] J. D. Moffett. Control Principles and Role Hierarchies. In *3rd ACM Workshop on Role-Based Access Control*, October 1998.
- [12] J. D. Moffett and E. C. Lupu. The use of role hierarchies in access control. In *4th ACM Workshop on Role-Based Access Control*, October 1999.
- [13] Sejong Oh and Ravi Sandhu. A Model for Role Administration Using Organization Structure. In *Seventh ACM Symposium on Access Control Models and Technologies, (SACMAT'02)*, pages 155–162, Monterey, California, USA, June 3–4 2002.
- [14] R. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.