Revocation Schemes for Delegation Licences

Meriam Ben-Ghorbel-Talbi[1,2], Frédéric Cuppens[1], Nora Cuppens-Boulahia[1], and Adel Bouhoula[2]

[1] Institut TELECOM/TELECOM Bretagne, 2 rue de la Chtaigneraie, 35576 Cesson Sévigné Cedex, France
{meriam.benghorbel,frederic.cuppens,nora.cuppens}@telecom-bretagne.eu
[2] SUP'COM Tunis, Route de Raoued Km 3.5, 2083 Ariana, Tunisie
bouhoula@planet.tn

Abstract. The paper presents revocation schemes in role-based access control models. We are particularly interested in two key issues: how to perform revocation and how to manage the revocation policy. We show how to deal with these two aspects in the delegation model based on the OrBAC formalism and its administration licence concept. This model provides means to manage several delegation types, such as the delegation or transfer of permissions and roles, multi-step delegation and temporary delegation. We state formally in this paper how to manage the revocation of these delegation schemes. Our model supports a wide spectrum of revocation dimensions such as propagation, dominance, dependency, automatic/user revocation, transfer revocation and role/permission revocation.

1 Introduction

In the field of access control, delegation is an important aspect and is managed as a special case of an administration task. An administration policy defines who is permitted to manage the security policy, i.e. who is permitted to create new security rules, or update or revoke existing security rules. A delegation policy defines who is permitted to manage existing security policies, i.e. who is permitted to delegate or revoke existing roles and privileges (e.g. permissions or obligations).

Many delegation schemes are defined in the literature, such as permanence, totality, monotonicity, multiple delegation and multi-step delegation. A complete access control model must provide a flexible administration model to manage these delegation aspects securely. But, it is also important to manage the revocation of such delegations. In [1] the authors propose a delegation model based on the OrBAC formalism and its administration licence concept [2]. This model provides means to deal with several delegation schemes thanks to the use of contextual and multi-granular licences. It supports the delegation and transfer of licences and roles, user-to-user and user-to-role delegation, permanent and temporary delegation, multiple delegation, simple and multi-step delegation, self-acted and agent-acted delegation. Moreover, several delegation constraints are

L. Chen, M.D. Ryan, and G. Wang (Eds.): ICICS 2008, LNCS 5308, pp. 190–205, 2008.

specified thanks to the taxonomy of contexts defined in OrBAC, namely prerequisite, provisional, temporal, spatial.

We base our work on this delegation model and we focus on the revocation mechanism. Our approach provides means to express various revocation dimensions suggested in the literature [3,4] such as propagation, dominance, grant dependency, automatic revocation. We focus our work on two aspects of revocation. The first aspect is how to perform the revocation, i.e. automatically or manually, and the effect of the revocation on other delegations (e.g. cascade revocation). The second issue is how to manage the right of revocation, i.e. who is permitted to revoke delegations.

We show that using the contextual licences our model is more flexible and expressive. We deal with the different revocation schemes in a simple manner, and unlike the other work on revocation management, we do not need to define many kinds of revocation actions. Moreover, we can specify several revocation constraints that were not supported previously. Existing models only support constraints on the role membership of the grantor or the grantee, whereas, in our model we may specify several conditions using prerequisite, temporal, spatial, user-declared and provisional contexts and use them to specify contextual licences.

This paper is organized as follows. In section 2 we start with the system description. We give the basic concept of the delegation model and we introduce some definitions. In section 3 we present the revocation mechanism. This section focuses on how revocation is performed and managed in our model. Then, section 4 presents related work and shows the advantages of our approach. Finally, concluding remarks and future work are made in section 5.

2 System Description

In the model presented in [1] the administration and delegation privileges are managed in a homogeneous unified framework using the OrBAC formalism [5]. This model provides means to specify the security policy at the organization level that is independent of the implementation of this policy. Thus, instead of modelling the policy by using the concrete concepts of subject, action and object, the OrBAC model suggests reasoning with the roles that subjects, actions or objects play in the organization. The role of a subject is simply called a role, whereas the role of an action is called an activity and the role of an object is called a view.

In OrBAC, there are eight basic sets of entities: Org (a set of organizations), S (a set of subjects), A (a set of actions), O (a set of objects), R (a set of roles), $\mathcal{A}$ (a set of activities), V (a set of views) and C (a set of contexts). In the following we give the basic OrBAC built-in predicates:

- ***Empower*** is a predicate over domains Org x S x R. If org is an organization, s a subject and r a role, then *Empower(org, s, r)* means that org empowers subject s in role r.

- ***Use*** is a predicate over domains *Org* x *O* x *V*. If *org* is an organization, *o* is an object and *v* is a view, then *Use(org, o, v)* means that *org* uses object *o* in view *v*.
- ***Consider*** is a predicate over domains *Org* x *A* x $\mathcal{A}$. If *org* is an organization, α is an action and *a* is an activity, then *Consider(org,* α*, a)* means that *org* considers that action α implements activity *a*.
- ***Hold*** is a predicate over domains *Org* x *S* x *A* x *O* x *C*. If *org* is an organization, *s* a subject, α an action, *o* an object and *c* a context, *Hold(org, s,* α*, o, c)* means that within organization *org*, context *c* holds between subject *s*, action α and object *o*.
- ***Permission***, ***Prohibition*** and ***Obligation*** are predicates over domains *Org* x R_s x A_a x V_o x *C*, where $R_s = R \cup S$, $A_a = \mathcal{A} \cup A$ and $V_o = V \cup O$. More precisely, if *org* is an organization, *g* is a role or a subject, *t* is a view or an object and *p* is an activity or an action, then *Permission(org, g, p, t, c)* (resp. *Prohibition(org, g, p, t, c)* or *Obligation(org, g, p, t, c)*) means that in organization *org*, grantee *g* is granted permission (resp. prohibition or obligation) to perform privilege *p* on target *t* in context *c*.

The OrBAC model is associated with a self administrated model called AdOrBAC [2,6]. This model is based on an object-oriented approach, thus we do not manipulate privileges directly (i.e. *Permission*, *Prohibition* and *Obligation*), but we use objects having a specific semantics and belonging to specific views, called administrative views. Each object has an identifier that uniquely identifies the object and a set of attributes to describe the object. A view is used to structure the policy specification.

The management of delegation is based on the AdOrBAC model. Delegation is modelled just like the administration mechanism, using specific views called delegation views [1]. For the sake of simplicity, we assume that the policy applies to a single organization and thus we omit the *Org* entity in the following.

Fig.1 is an outline of OrBAC views. The view ***Licence*** is used to specify and manage users privileges: permissions, prohibitions and obligations. In the following we only consider licences interpreted as permissions. Objects belonging to this view have the following attributes: *grantee*: subject to which the licence is granted, *privilege*: action permitted by the licence, *target*: object to which the licence grants access and *context*: specific conditions that must be satisfied to use the licence. The existence of a valid licence is interpreted as a permission by the following rule:

$$Permission(Sub, Act, Obj, Context) :- Use(L, licence), Grantee(L, Sub), Privilege(L, Act), Target(L, Obj), Context(L, Context).$$

The view ***Role_assignment*** is used to manage the assignment of subject to roles. Objects belonging to this view are associated with the following attributes: *assignee*: subject to which the role is assigned and *assignment*: role assigned by the role assignment. There is the following rule to interpret the objects of the *role_assignment* view:

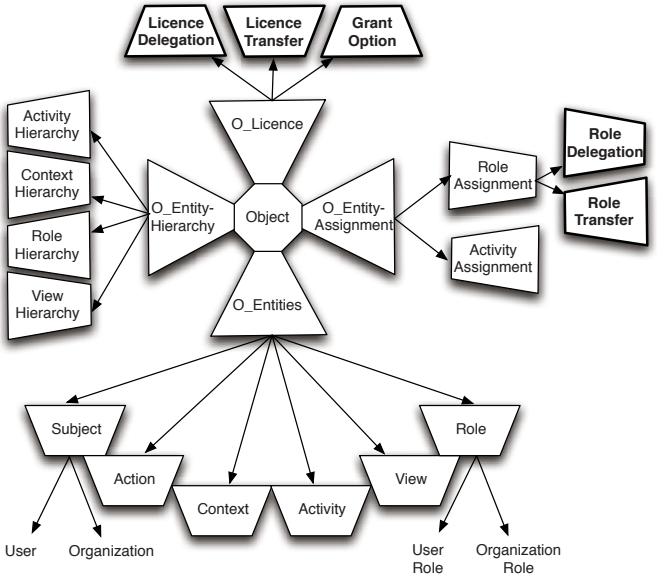

Fig. 1. OrBAC views

$Empower(Subject, Role)$:-
$Use(RA, role\_assignment), Assignee(RA, Subject), Assignment(RA, Role)$.

The views boldfaced in Fig. 1 are delegation views. ***Role_delegation*** and ***Role_transfer*** are two sub-views of *Role_assignment* defined to manage role delegation and transfer. ***Licence_delegation***, ***Licence_transfer*** and ***Grant_option*** are three sub-views of *Licence* defined to manage licence delegation and transfer. Licence delegation concerns only *Permission* and *Obligation*. Thus, the delegation of negative privileges in not supported, although the administration model supports the granting of prohibitions. In fact, defining prohibitions is considered an administration task and it is not meaningful to delegate a prohibition.

Due to space limitation, we only consider in this paper *Licence_delegation* and *Grant_ option* views. Objects belonging to these have the same attributes and semantics as objects belonging to the *Licence* view (i.e. an assignment of a permission to a role or to a user). These objects also have an additional attribute called *grantor*: subject who delegates the licence. This attribute is important to revoke the delegation and to recover the grantor privileges when the transfer is revoked.

Note that there is a distinction between the delegation of a right R and the delegation of the right to delegate R, and for that purpose two delegation views are used. Objects belonging to the view *Licence_delegation* are interpreted as a delegation of simple rights, and objects belonging to the *Grant_option* view are interpreted as a delegation of delegation rights. This means that the privilege

and the target of these objects concern delegation activities and delegation views, respectively.

Objects belonging to *Grant_option* have another additional attribute called *step*: the depth of the delegation chain. This attribute is used to control the propagation of the right to delegate. When the step is greater than 1 the grantee is allowed to propagate the delegation right (i.e. the right to delegate a licence or a role) to another user. Thus, a new permission is derived to the grantee as follows:

$$Permission(GR, delegate, grant\_option, C \ \& \ valid\_redelegation(LG)) \text{:-}$$
$$Use(LG, grant\_option), Grantee(LG, GR),$$
$$Context(LG, C), Step(LG, N), N > 1.$$

Note that, the context *valid_redelegation* is used to control the scope of the redelegation. Thus the grantee is allowed to delegate only the right (or a sub-right of the right) he was received and with a lower step. So, we can be sure that the redelegated right will never be higher than the original right delegated by the first grantor. This context is defined as follows:

$$Hold(U, delegate, LG', valid\_redelegation(LG)) \text{:-}$$
$$Use(LG', grant\_option), Sub\_Licence(LG', LG),$$
$$Step(LG', N'), Step(LG, N), N' < N.$$

where the predicate *Sub_Licence* is defined as follows:

$$Sub\_Licence(L, L') \text{:-}$$
$$Target(L, T), Target(L', T'), Sub\_Target(T, T'),$$
$$Privilege(L, P), Privilege(L', P'), Sub\_Privilege(P, P'),$$
$$Context(L, C), Context(L', C'), Sub\_Context(C, C').$$

O is considered a Sub_Target of O' if they are two views and O is a Sub_View of O', or if O is an object used in view O', or if they are equal (see [7] for more details about the OrBAC hierarchy):

$$Sub\_Target(O, O') \text{:-}$$
$$Sub\_View(O, O'); Use(O, O'); O = O'.$$

Predicates *Sub_Privilege* and *Sub_Context* are defined in a similar way.

2.1 Notation and Definitions

As we mentioned in the previous section, there is a distinction between the delegation of simple rights and the delegation of delegation rights. For this purpose, we denote L a simple delegated licence, i.e. objects belonging to the view *Licence_delegation*, and we denote LG a delegation licence, i.e. objects belonging to the view *Grant_option*.

Let O_L be the set of simple licences, O_{LG} the set of delegation licences and $O_{L_d} = O_L \cup O_{LG}$ the set of all delegated licences Ld.

We give now some definitions to deal with revocation in our model.

Definition 1. *Derivation relation.*

We distinguish between two derivation relations according to whether the type of derived licence is a simple or a delegation licence.

1. $\forall Ld, Ld' \in O_{LG}$, *Ld is* ***derived*** *from* Ld', *if:*
   - *the grantor of the licence Ld is the grantee of the licence* Ld',
   - *the licence Ld is a sub-licence of* Ld',
   - *the delegation step of Ld is lower than the delegation step of* Ld'.
2. $\forall Ld \in O_L, Ld' \in O_{LG}$, *Ld is* ***derived*** *from* Ld', *if:*
   - *the grantor of the licence Ld is the grantee of the licence* Ld',
   - *the licence Ld corresponds to the target definition of* Ld' *(i.e. Use(Ld, Target(Ld'))).*

Definition 2. *Delegation chain.*

1. *For each delegation licence* $LG \in O_{LG}$ *we can generate a delegation chain, which we call* $DC(LG)$. *A delegation chain is represented by a directed graph (see Fig 2.a). The nodes contain licences* $Ld \in O_{Ld}$, *and we denote* $N(Ld)$ *the node containing licence Ld. There is an arc from node* $N(Ld_1)$ *to node* $N(Ld_2)$, *if* Ld_2 *is derived from* Ld_1. *A node containing a simple licence* $L \in O_L$ *is always a leaf of the graph.*
2. *A node is rooted if it contains a licence that cannot be derived from any other licences. A delegation chain of a licence Ld is rooted if Ld is rooted.*
3. *When a node* N_i *is deleted (i.e. the licence contained in this node is revoked), a special arc labelled with a * is used to connect nodes* N_{i-1} *to* N_{i+1} *(see Fig2.b). In addition, we denote DC*(Ld) the delegation chain DC(Ld) that includes labelled arcs.*

Note that the delegation chain DC* is used to ensure that every delegated licence has a path that links it to the licences from which it is indirectly derived, even if some licences belonging to the delegation chain are removed. For instance, if we consider the delegation chain given in Fig. 2.b, then $DC(LG_1) = \{LG_2, L_3\}$ and $DC^*(LG_1) = \{LG_2, L_3, L_5\}$.

Definition 3. *Dependency.*

A licence Ld depends exclusively on a user U if there is no rooted delegation chain DC such that $Ld \in DC$ *and* $\forall Ld' \in DC$, $Grantor(Ld') \neq U$.

We assume that Dependent(Ld, U) is a predicate meaning that licence Ld depends exclusively on user U.

Theorem 1. *The delegation chains are computable in polynomial time.*

Proof. The delegation chain is based on the OrBAC model and its self-administration model. Policies associated with both of them can be expressed as recursive rules corresponding to a stratified Datalog program; the delegation chains are then obtained by computing a fixed point which is tractable in polynomial time. □

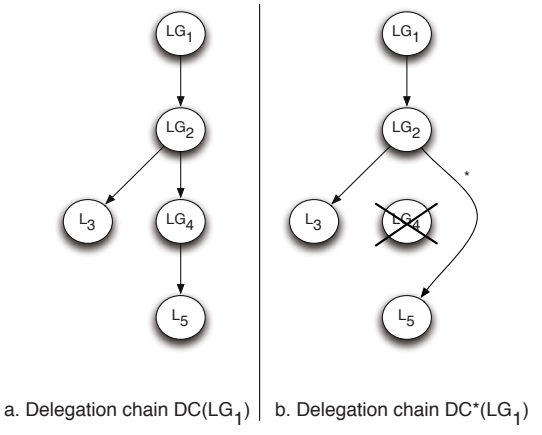

Fig. 2. Delegation Chain

3 Revocation

We focus in this section on two aspects of delegated right revocation. The first aspect is how to perform revocation. In our model revocation can be performed automatically when the delegation context does not hold, or manually by an authorized user. Following the classification defined in [4], user revocation can be categorized into three main dimensions: propagation, dominance and resilience. Propagation concerns the management of the revocation in the case of multi-step delegation, dominance concerns the revocation impact on other delegations associated with the same grantee, and resilience concerns the persistence of the revocation in time, i.e. the revocation by removal of the permission (non-resilient) or by adding a negative permission that has higher priority (resilient). In our delegation model we do not consider resilient revocation. Indeed, we consider that delegation concerns only positive permissions and granting prohibitions is an administrative task only.

Obviously, to perform a revocation the user must have the permission to revoke the delegation. This is the second issue of revocation: how to manage the right of revocation. For this purpose, we consider the grant dependency characteristic. Grant Dependent (GD) revocation means that the user can only revoke his/her delegations. Grant Independent (GI) means that the delegation can be revoked by any authorized user. For instance, a user empowered in a role R can revoke any grantee empowered to this role by delegation.

3.1 User Revocation

As mentioned in the previous section, delegation is performed by adding objects into delegation views. The existence of these objects is then interpreted as a permission. The revocation of the delegation is performed by removing these

objects from the delegation views. Each object is associated with a grantor attribute that indicates the subject that is performing the delegation. By default, each object can be removed by his/her grantor. This can be specified implicitly or explicitly in our model (see section 3.3). Thus, we can ensure that each delegation can be revoked.

The effect of the licence revocation on the other delegations depends on the revoker needs. He/she can choose, for instance, to revoke the whole delegation chain (cascade revocation) or to revoke all other delegations associated with the same grantee (strong revocation).

Propagation. In the case of multi-step delegation, the delegation of each delegation licence (i.e. using the view *Grant_option*) can generate a delegation chain. Hence, the revoker can choose to revoke only the licence he/she has delegated or also all licences derived from it.

Definition 4. *Simple revocation.*

This is the simplest revocation scheme. The revocation involves deleting the licence from the delegation view and does not affect the other delegated licences. We formally define the request to revoke a delegation as follows:

Request(U, revoke, L) ***then***
  forall $L_i \in$ *Derived_licences(L)* **and** $L_j \in$ *Parent_licences(L)* **do**
    Add a labeled arc from $N(L_j)$ *to* $N(L_i)$,
    Remove(L).
end

We assume that, Derived_licences(L) is the set of all licences L' *such that there is an arc (labelled or not labelled) from node* $N(L)$ *to node* $N(L')$ *and Parent_licences(L) is the set of licences* L'' *such that there is an arc (labelled or not labelled) from node* $N(L'')$ *to node* $N(L)$.

Definition 5. *Cascade revocation.*

This involves the revocation of all the licences belonging to the delegation chain. But, this revocation should not affect the licences belonging to the delegation chain (DC) of other licences. The reason is that if the grantor of a licence L has received the permission to delegate this licence from two or more different delegation licences, then if one of these delegations is revoked, the grantor still has the right to delegate L (an illustrative example is given below). In our model, cascade revocation is defined as follows:

Request(U, Cascade_revoke, L) ***then***
  Request(U, revoke, L),
  forall $L_i \in$ *Derived_licences(L)* **do**
    if $L_i \notin DC(L'), L' \in O_{LG}$ **then**
      Request(U, Cascade_revoke, L_i*).*
    end
end

Example 1. We consider the example shown in Fig.3, and we assume that LG_1 is revoked with the cascade option. On the first pass, licence LG_1 is removed. On

the second pass, LG_2 and LG_3 are revoked and a labelled arc is added to connect node LG_0 to LG_4. Note that LG_3 is revoked because it belongs to DC*(LG_0) and not to $DC(LG_0)$. Finally, the cascade revocation process is stopped because LG_4 belongs to $DC(LG_5)$.

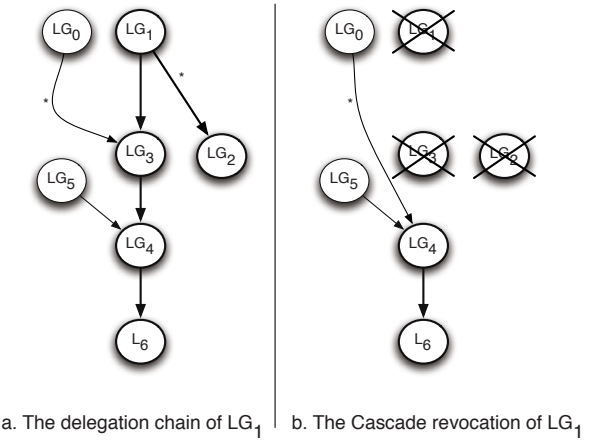

Fig. 3. Cascade revocation

Dominance. The revocation described so far is a weak revocation. This means that the revocation of a licence L will only affect this licence (in the case of simple revocation) or the licences belonging to its delegation chain (in the case of cascade revocation). However, a strong revocation will affect the other licences associated with the same grantee.

Definition 6. *Strong revocation.*

The strong revocation of licence L means that all the licences equal to L (or are a sub-licences of L) that depend on the revoker and associated with the same grantee must be revoked. This is defined as follows:

Request(U, Strong_revoke, L) ***then***
Request(U, revoke, L),
forall $Ld_i \in O_{Ld}$ **do**
  if *Grantee*(Ld_i) = *Grantee*(L) **and** *Sub_licence*(Ld_i,L) **and**
  Dependent(Ld_i,U) **then**
    Request(U, revoke, Ld_i*).*
  end
end

As mentioned in Definition 3, Dependent(Ld, U) *is a predicate meaning that licence Ld depends exclusively on user U.*

Definition 7. *Strong-Cascade revocation.*

We consider the strong-cascade revocation of a licence L as a strong revocation of all licences belonging to the delegation chain of L. This is defined as follows:

Request(U, Strong_Cascade_revoke, L) **then**
Request(U, Strong_revoke, L)
forall $L_i \in$ *derived_licences(L)* **do**
  if $L_i \notin$ *DC(L')*, $L' \in O_{LG}$ **then**
    Request(U, Strong_Cascade_revoke, L_i*)*
  end
end

Example 2. We consider the example shown in Fig.4, where we represent in each node the delegated licence, the grantor and the grantee of this licence. We assume that licence LG_5 is a sub-licence of LG_3 and LG_7 is a sub-licence of LG_4.

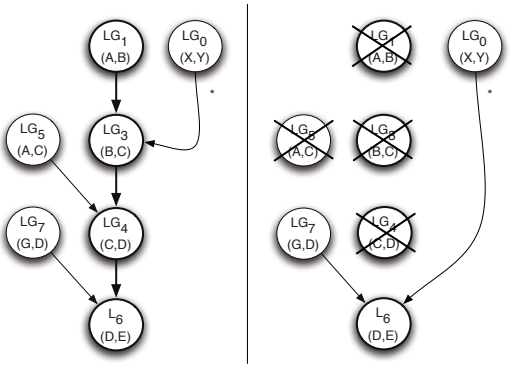

Fig. 4. Strong-Cascade Revocation

If A the grantor of licence LG_1 revokes this licence with the strong-cascade option, then the whole delegation chain of LG_1 will be revoked with the strong cascade option as well. On the first pass, licence LG_1 is revoked by A. On the second pass licence LG_3 is revoked and also licence LG_5 since it is a sub_licence of LG_3 and it depends on A. On the third pass, LG_4 is revoked but not LG_7 because it is independent of A. L_6 is not revoked because it belongs to $DC(LG_7)$.

Theorem 2. *The revocation requests are computable in polynomial time.*

Proof. A request to revoke a licence requires a recursive search in the delegation chain of this licence, therefore using theorem 1, it is computable in polynomial time. □

Other revocation schemes. The revocation discussed in this section only concerns delegated licences, however the delegation model supports both role

and licence delegation. Role revocation is omitted for the sake of simplicity, but we can deal with this aspect similarly as described above. We have simply to consider the set of delegated roles, O_R (the set of objects belonging to the view *Role_delegation*) and we replace the set O_L (the set of delegated licences) by $O_D = O_L \cup O_R$ (the set of delegations). The remainder is unchanged, except for strong revocation where *sub_licence* is replaced by *sub_role* in the case of role revocation.

Additionally, the delegation model supports role and licence transfer. But there is no need to use specific functions to deal with the transfer revocation. In fact, the grantor automatically recovers his/her rights when the transfer is revoked (see [1] for more details). As we have mentioned earlier, when the licence is removed the permission delegated to the grantee is no longer derived, but in the case of transfer, the prohibition associated with the grantor is no longer derived either.

3.2 Automatic Licence Revocation

In our model, the licence revocation can also be performed automatically using the notion of context. In fact and contrary to other existing models (see section 4), *Licences* are associated with an attribute called *Context* used to specify conditions (see [8] for more details). Concrete permissions are derived from *Licences* only if the context is active. Hence, the delegation is revoked automatically if the context does not hold. Note that in this case the licence is not removed as in user revocation, but the permission is not derived.

For instance, we consider the case when the user U_1 delegates licence L to user U_2 with the following attributes: Grantor: u_1, Grantee: u_2, Privilege: *read*, Target: $file_1$, Context: *weekend*. This means that the following permission is derived for user U_2: $Permission(u_2, read, file_1, weekend)$.

This permission is derived only if the context *weekend* holds, thus the user U_2 is allowed to read the file $file_1$ only during the weekend. We can say that the delegation is revoked automatically during the other days of the week.

Note that in the case of multi-step delegation, when a delegation licence LG is revoked automatically (i.e. the delegation context of LG does not hold), then all the delegation licences derived from it are revoked automatically as well (i.e. the delegation context of these licences does not hold as well). This is due to the fact that when we derive a licence LG' from LG, then LG' must be a sub-licence of LG. Therefore, the delegation context of LG' must be equal to or a sub-context of the delegation context of LG.

Moreover, in the case of licence transfer, the grantor revocation is done automatically using contextual prohibition. In fact, when a transfer is performed, a prohibition associated with the highest priority level is automatically assigned to the grantor while the delegation is active (see [1]). Therefore, the grantor will lose the permission he/she has delegated. The context of the prohibition and of the delegated permission are the same. So, when the delegation context does not hold, the prohibition is no longer derived and the grantor will automatically retrieve his/her permission.

3.3 Managing Revocation

A revocation policy defines who is permitted to revoke delegations (roles or licences). In the literature some models [9,10,11,12] consider that the right to revoke his/her delegations is implicitly defined, i.e. the grantor of the right R is automatically allowed to revoke it. Other models [13,14,15] use specific functions to manage the revocation policy, such as *can_revoke*, *can_revokeDG*, *can_revokeGI*, *can_u2u_revokeGD*. This approach is more general since it dissociates the right to revoke from the right to delegate a privilege. Therefore, it supports more revocation schemes such as Grant Independent.

In our model, we follow the second approach, but we do not use specific functions to define the right to revoke. In fact, we manage delegation and revocation by the assignment of subjects (users or roles) to permissions just like in the AdOrBAC model. These permissions apply to the activities *delegate* and *revoke*, and are associated with a delegation target (the delegation views). So, they allow users to add and remove objects from the delegation views.

Note that using the derivation rules, we can implicitly specify that grantors who are permitted to delegate a right are also permitted to revoke this right. But in addition, we can specify explicitly who is permitted to revoke delegations and, thanks to the use of contexts, we can specify different conditions to restrict the revocation rights. In the following we give some context examples dealing with Grant Dependent and Independent revocation.

Example 3. Grant Dependent revocation.

In the case of Grant Dependent revocation only the grantor is allowed to revoke the delegated licence. To deal with this aspect we define the prerequisite contexts gd_L as follows[1]:

$Hold(U, revoke, L, gd_L)$:-
$Use(L, licence\_delegation); Use(L, grant\_option), Grantor(L, U)$.

Using this context, the administrator can specify that all users are authorized to revoke their delegated licences. This is defined by the following:

$Permission(default\_Role, revoke, licence\_delegation, gd_L)$.

$Permission(default\_Role, revoke, grant\_option, gd_L)$.

where *default_Role* is a role to which all authorized users are empowered.

We can also consider the case of role revocation as follows:

$Hold(U, revoke, RD, gd_R)$:-
$Use(RD, role\_delegation), Grantor(RD, U)$.

Similarly, the administrator can specify that all users are authorized to revoke their delegated roles as follows:

$Permission(default\_role, revoke, role\_delegation, gd_R)$.

Example 4. Grant Independent revocation.

[1] The operator ';' corresponds to a disjunction.

We can specify many contexts to deal with this aspect according to the administrator's needs. Contexts provide our model with high flexibility and expressiveness.

For instance, we may consider that a user can revoke a licence (respectively a role) if this licence (or this role) is derived from this user but depends exclusively on him/her. Hence, he/she cannot revoke a licence that also depends on other users. We define for this purpose the contexts *Ancestor Dependent* ad_L and ad_R to revoke a licence and a role, respectively:

$Hold(U, revoke, L, ad_L)$:-
$Use(L, licence\_delegation); Use(L, grant\_option), Dependent(L, U)$.

$Hold(U, revoke, RD, ad_R)$:-
$Use(RD, role\_delegation), Dependent(RD, U)$.

We can also define the context *Role Dependent (rd)* to say that any user empowered in role R can revoke this role:

$Hold(U, revoke, RD, rd)$:-
$Use(RD, role\_delegation), Assignment(RD, R), Empower(U, R)$.

Note that to manage the revocation policy, the administrator only specifies who is permitted to perform a simple revocation. We assume that the right to perform the other revocation schemes, namely cascade, strong and strong-cascade revocation, is automatically derived from the simple revocation. For instance, the *request (U, cascade-revoke, L)* is considered as a set of simple requests to revoke the licences that belong to the delegation chain of L. Hence, if the user is allowed to revoke these licences then he is allowed to revoke L with the cascade option. A straightforward refinement of our model involves specifying that the revocation request is atomic or not, i.e. if the user is not allowed to revoke all the related licences, then we have to specify if the request to revoke with the cascade, strong or strong-cascade option, can be partially accepted or must be totally rejected.

Moreover, using the contextual licences, we can specify complex security rules to manage revocation. Indeed, we can define several conditions using the taxonomy of contexts (e.g. temporal, prerequisite, provisional [8]). These conditions may concern the grantor or the grantee attributes, previous actions, the delegated right (i.e. the role, the target, the privilege), the time, circumstances (e.g. urgency). For instance, we can specify that a given user U_1 is allowed to revoke the role R transferred by the grantor U_3 to the grantee U_2, only if U_3 is not on vacation. We can also define that U_1 can revoke U_2 if this user has performed a given action A in a given object O. We may also specify that U_1 is authorized to revoke U_2, if U_2 is the assistant of U_1, or is associated with the same department as U_1. These kind of conditions are not supported by the existing models since they only specify constraints on the role membership of the grantor or the grantee.

4 Related Work

In [4] authors classify the revocation into three dimensions: resilience, propagation and dominance. These dimensions are combined to provide eight different revocation schemes. This paper proposes to study permission revocation in a generic access control framework with a grant for both positive and negative permissions, where negative permissions dominate positive ones. The concept of inactive permissions is used to deal with resilient revocation, namely, positive permissions are inactivate when a negative permission is granted.

[9] addresses the revocation of certificates. The proposed framework supports revocation schemes such as propagation, dominance and grant dependency. Strong revocation means that the user is permitted to revoke any certificate that belongs to the delegation chain of a certificate issued by him/her. This corresponds to ancestor-dependent revocation in our paper.

Work [3,10,11,12,13,14,15] deals with revocation in role-based access control models. RBDM0 [3] is the first delegation model of Barka and Sandhu. It addresses role revocation and supports automatic revocation using a time out, and grant independent revocation (which corresponds to role-dependent revocation in our paper). RBDM1 [13] is an extension of this model, which supports cascade revocation, strong revocation, grant-dependent and role-dependent revocation. The strong revocation of a role is considered as a revocation of both explicit and implicit memberships (i.e. all roles junior to that role are revoked), this is a specific case of strong revocation presented in our paper. Role-role revocation is defined using the relation: *Can-Revoke* $\subseteq R$ x R. *Can-Revoke(x, y)* $\in$ *Can-Revoke* means that the user empowered in role x can revoke the membership of the delegate member y in role x. RDM2000 [15] is based on RBDM0. It supports cascade and strong revocation (strong option has the same definition as in RBDM0). Revocation is managed using the following relations: *Can-RevokeGD* $\subseteq R$ to manage the grant-dependent revocation and *Can-RevokeGI* $\subseteq R$ to manage the role-dependent one.

The two models [11,12] address the revocation of permissions in workflow systems. The first model supports automatic revocation using the notion of case (an instance of a workflow process) and grant-dependent revocation (the right to revoke his/her own delegations is implicitly defined). It also supports cascade revocation and uses a delegation graph similar to our model, namely nodes represent accepted delegations. The second model deals with automatic revocation using lifetime, cascade revocation and dependency. Users are automatically allowed to revoke their own delegations, and the grant-independent option is considered as revocation by the security administrator.

Compared to these works, our model is more expressive since it supports various revocation schemes such as automatic/manual, simple/cascade, weak /strong, grant dependent/independent (e.g. role dependent, ancestor dependent). Only resilient revocation is not considered because we assume that granting negative permissions is an administration task only. These different schemes apply to the revocation of delegation/transfer of roles/permissions. Moreover, thanks to the use of contexts, our model is more flexible and simpler to manage.

Namely, there are no specific permissions or actions for each revocation task like in related work. Hence, we have simply to define contexts to specify the different revocation schemes. On the other hand, using contextual permissions, we can specify several constraints to manage revocation that are not supported by other work. Indeed, they only support constraints on the role membership of the grantor or the grantee, whereas in our model we can specify several kinds of conditions concerning grantor/grantee attributes, previous actions, delegated rights, the time.

5 Conclusion

In this paper we have proposed to deal with revocation in role-based access control models. Our work is based on the delegation model presented in [1]. This model is flexible and various delegation schemes are defined.

Our revocation study has focused on two issues: how to perform revocation and how to manage the revocation policy. Revocation can be performed automatically when the delegation context does not hold, or manually by an authorized user. The effect of revocation on other delegations varies according to the revoker's needs. The revocation can be with a cascade so all the delegation chains are revoked, or with a strong option so all the delegations assigned to the grantee are revoked. We can also combine these two features to revoke with a strong-cascade option.

To manage the revocation policy, we have to specify who is permitted to delete objects from the delegation views. Thanks to the use of contexts we can specify various conditions to restrict the revocation rights. We have given some context examples to specify that all grantors are permitted to revoke their delegations, or all users empowered in a role R are permitted to revoke R, or finally, all users are permitted to revoke licences (or roles) that belong to their delegation chain.

Although many revocation schemes are supported, such as propagation, dominance, automatic revocation, grant dependency, transfer revocation, we have dealt with these different revocations in a simple manner and there is no need to deal with each level separately. Thanks to the facilities provided by the OrBAC model (i.e. contextual licences, the use of views), we do not use specific functions to manage revocation like in other related work. Therefore, it is easier to extend our study to deal with other revocation schemes and constraints, since we have simply to define new contexts.

In this paper we have focused on the delegation and revocation of licences that are interpreted as permissions. Future work will be to enrich our model to study the delegation and revocation of obligations.

References

1. Ben-Ghorbel-Talbi, M., Cuppens, F., Cuppens-Boulahia, N., Bouhoula, A.: Managing Delegation in Access Control Models. In: Proceedings of the 15th International Conference on Advanced Computing and Communications (ADCOM 2007), Guwahati, Inde, pp. 744–751. IEEE Computer Society Press, Los Alamitos (2007)

2. Cuppens, F., Cuppens-Boulahia, N., Coma, C.: Multi-Granular Licences to Decentralize Security Administration. In: Proceedings of the First international workshop on reliability, availability and security (SSS/WRAS 2007), Paris, France (November 2007)
3. Barka, E., Sandhu, R.: A Role-based Delegation Model and Some Extensions. In: Proceedings of the 23rd National Information Systems Security Conference (NISSC 2000), Baltimore, MD (October 2000)
4. Hagström, Å, Jajodia, S., Parisi-Persicce, F., Wijesekera, D.: Revocation - a Classification. In: Proceedings of the 14th Computer Security Foundation Workshop (CSFW 2001), Cape Breton, Nova Scotia, Canada, IEEE Computer Society, Los Alamitos (2001)
5. Abou-El-Kalam, A., Benferhat, S., Miège, A., Baida, R.E., Cuppens, F., Saurel, C., Balbiani, P., Deswarte, Y., Trouessin, G.: Organization Based Access Control. In: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2003). IEEE Computer Society, Los Alamitos (2003)
6. Cuppens, F., Miège, A.: Administration Model for Or-BAC. International Journal of Computer Systems Science and Engineering (CSSE) 19(3) (May 2004)
7. Cuppens, F., Cuppens-Boulahia, N., Miège, A.: Inheritance Hierarchies in the Or-BAC Model and Application in a Network Environment. In: Proceedings of the 3rd Workshop on Foundations of Computer Security (FCS 2004), Turku, Finland (July 2004)
8. Cuppens, F., Cuppens-Boulahia, N.: Modeling Contextual Security Policies. International Journal of Information Security (November 2007)
9. Firozabadi, B.S., Sergot, M.: Revocation Schemes for Delegated Authorities. In: Proceedings of the Third International Workshop on Policies for Distributed Systems and Networks (2002)
10. Nguyen, T.A., Su, L., Inman, G., Chadwick, D.: Flexible and Manageable Delegation of Authority in RBAC. In: Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW 2007). IEEE Computer Society, Los Alamitos (2007)
11. Wainer, J., Kumar, A., Barthelmess, P.: DW-RBAC: A Formal Security Model of Delegation and Revocation in Workflow Systems. Information Systems 32(3), 365–384 (2007)
12. Wei, Y., Shu, Q.: A Delegation-Based Workflow Access Control Model. In: Proceedings of the First International Symposium on Data, Privacy, and E-Commerce (ISDPE 2007). IEEE Computer Society, Los Alamitos (2007)
13. Barka, E., Sandhu, R.: Role-Based Delegation Model/ Hierarchical Roles (RBDM1). In: Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC 2004), Tucson, Arizona (December 2004)
14. Lee, Y., Park, J., Lee, H., Noh, B.: A Rule-Based Delegation Model for Restricted Permission Inheritance RBAC. In: Proceedings of the 2nd International Conference (ACNS 2004), Yellow Mountain (June 2004)
15. Zhang, L., Ahn, G.-J., Chu, B.-T.: A Rule-Based Framework for Role-Based Delegation and Revocation. ACM Transactions on Information and System Security (TISSEC) 6, 404–441 (2003)