Refinement for Administrative Policies

M.A.C. Dekker[1,2] and S. Etalle[2]

[1] Security group, TNO ICT, The Netherlands
[2] Distributed and Embedded Systems group, University of Twente, The Netherlands

Abstract. Flexibility of management is an important requisite for access control systems as it allows users to adapt the access control system in accordance with practical requirements. This paper builds on earlier work where we defined administrative policies for a general class of RBAC models. We present a formal definition of administrative refinement and we show that there is an ordering for administrative privileges which yields administrative refinements of policies. We argue (by giving an example) that this privilege ordering can be very useful in practice, and we prove that the privilege ordering is tractable.

1 Introduction

Role-based access control (RBAC) [1] is a well-known standard for access control, aimed to make the assignment of users to privileges more easy. In practice however, for example in hospitals or enterprises, RBAC policies can be very large and dynamic, consisting of thousands of roles [6], and changing frequently. In such cases *policy management* can be a daunting task. The usual approach to this problem is to divide the work and to delegate (bits of) administrative authority to other users. The advantage is that users can adapt the access control policy to changing circumstances more easily, without an administrative bottleneck. Not only does this reduce the cost of maintaining the access control policy, it also avoids bad security practices, such as sharing passwords or keys that should really remain secret. For example, it may be convenient to allow the head nurse to delegate database access to other nurses when they need it for particular tasks, without having to recur to the hospital's security officer. On the other hand, this kind of flexibility also introduces security risks as changes made to the RBAC policy could entail privacy breaches.

The issue of designing *flexible* yet *safe* policy administration mechanisms for RBAC has received much attention recently [3, 4, 6, 9, 14]. To mention some of the research: In ARBAC [9] *administrative privileges* are assigned to a separate hierarchy of *administrative roles* and defined by specifying a range of roles that can be changed. Crampton and Loizou [4]

take a more general approach, by using the same hierarchy for both administrative privileges and ordinary user privileges. Using the concept of *administrative scope*, they define which roles should have administrative privileges over other roles. In a similar approach, Wang and Osborn [12] divide the role-graph (a type of RBAC policy) into administrative domains. Each administrative domain has one administrator with privileges about the (roles in the) domain. In the Role-Control Center [6], administrative privileges over roles are defined in terms of *views*, which are subsets of the role-hierarchy, and they can only be assigned to users that are assigned to these roles. There seems to be no consensus (yet) about which administrative privileges belong to which roles; each of the above mentioned frameworks differs on this issue. Some models motivate their choice by considerations that include the meaning of a *role* in a company, or the concepts of ownership, and responsibility, as one would find it in a company. On the other hand, Li et al. argue that interpreting the RBAC role hierarchy as a business organization chart can be misleading [8].

This paper aims to be a contribution to the above-mentioned lines of research on management of RBAC policies. In this paper we introduce the concept of *administrative refinement*, and we show that this concept yields a more flexible, and at the same time more safe administrative model. This paper builds on earlier work [5], where we argued informally that there is a natural ordering for administrative privileges. In this paper we present a formal definition of administrative refinement, and we show that it yields an ordering on the administrative privileges, which allows for a more flexible policy management. Furthermore, we present the formal proof that this privilege ordering is tractable. Note that in this paper we do not assume any features that go beyond the *General Hierarchical RBAC model* (such as constraints), and that we do not restrict which administrative privileges can be assigned to which roles. We are hence led to believe that our results are also applicable to a range of more advanced RBAC models.

2 Preliminaries

We first introduce shortly the *General Hierarchical RBAC* model, as defined in the ANSI RBAC standard, because it is the most commonly used RBAC model [1, 6]. In Section 3 we extend this model with administrative privileges, yielding a general class of administrative policies.

The goal of an RBAC policy is to specify which users are permitted to perform which actions on which objects. We denote the sets of users,

roles, actions, and objects, by U, R, A, and O. Permissions for performing actions on objects are called *user privileges*, forming a set $P \subseteq A \times O$, e.g. $(read, ehrtable)$, $(print, colorA4)$.

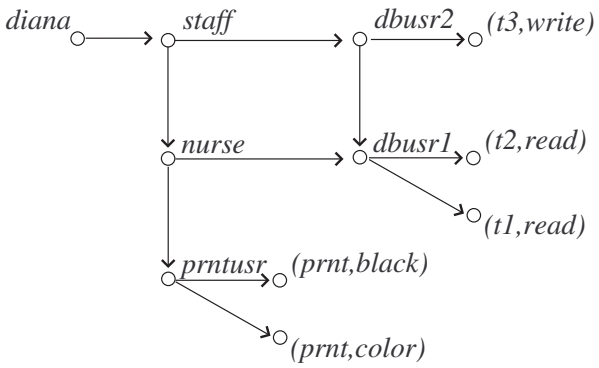

Fig. 1. Sample Non-Administrative RBAC policy.

A non-administrative RBAC policy assigns users to roles, roles to user privileges, and it defines an order on the roles; the *role-hierarchy*[3].

Definition 1 (Non-administrative Policies). *Let U, R, and P be sets of users, roles, and user privileges,* a non-administrative RBAC policy ϕ *is a tuple*

$$\phi = (UA, RH, PA),$$

where $UA \subseteq U \times R$ determines which users are member of which roles, $RH \subseteq R \times R$ is the role-hierarchy, and $PA \subseteq R \times P$ determines which roles have which privileges.

The set of non-administrative RBAC policies is denoted $\Phi_{U,R,P}$. To simplify our exposition we treat a policy ϕ as a *directed graph*, defined by the *set* of directed edges $UA \cup RH \cup PA$. If there is a path from one vertex v to another v' we write $v \rightarrow_\phi v'$. Below we sometimes omit the subscript ϕ when the policy is clear from the context.

The RBAC *reference monitor* uses the policy ϕ as follows. Any user u can start a *session*. The reference monitor allows the user to *activate*

[3] In the RBAC standard the relation RH is defined to be acyclic, reflexive and transitive, i.e. it is defined to be a *partial order*. Li et al., however, showed that this definition causes problems when changes are made to the role-hierarchy [8]. Here, for the sake of generality we do not assume that RH is a partial order.

a role r in a session iff $u \rightarrow_\phi r$. The privileges of the user's session are all the privileges p such that $r \rightarrow_\phi p$ for some role r activated in the session. Sessions are an important safety mechanism, allowing users to apply the *principle of least privilege*. Here, for the sake of simplicity we ignore the details about sessions. For details about sessions we refer the reader to the ANSI RBAC standard [1]. Let us give a simple example of a non-administrative RBAC policy.

Example 1 (Basic RBAC). Consider the setting of a hospital, where a database system *dbms* stores electronic health records in a number of tables *t1, t2, t3* etc. The health records can only be seen or changed by authorized personnel. To enforce this the system *dbms* uses the RBAC policy depicted in Figure 1: The employee *Diana* can activate the role *nurse* or the role *staff*. In the former case she can *read* the tables *t1* and *t2*, while in the latter case she can also *write* the table *t3*.

3 Administrative RBAC Policies

The RBAC standard specifies a number of *administrative functions and controls*, which can be used by an administrative authority to make policy changes [1]. In this paper we express administrative authority in terms of administrative privileges to model which users (or roles) can make which policy changes. There are two types of privileges: privileges for making new edges (denoted here by $\Box$), and privileges for removing edges (denoted here by $\Diamond$). We assign the administrative privileges to roles just like the user privileges are assigned to roles in standard RBAC. This approach is also advocated in the literature[4] and the intuition behind it is that the RBAC policy can also be used to specify who can change the RBAC policy [4, 12].

Clearly, administrative privileges must be an *infinite* set, even if we assume that the sets of users, roles and user privileges are *finite*. The reason is that administrative privileges over administrative privileges are also administrative privileges. For example, consider the privilege to give someone else the privilege to change the members of a role. The number of *administrative levels* (the number of nestings of the $\Box$ connective to be introduced below) is often restricted in existing literature (sometimes to one [10] or to two levels [14]). We agree that in some settings multiple

[4] Existing literature focusses however on defining constraints on which roles can have which administrative privileges. For example, to prevent low roles from obtaining privileges about higher roles [4]. For the sake of generality we do not make choices with respect to such constraints.

levels of administration are not useful, however, here we prefer to take a general approach, leaving it up to security officers to choose which administrative privileges to use in their systems.

We formalize the full set of privileges by defining a *grammar* that encompasses both user privileges and administrative privileges.

Definition 2 (Privilege Grammar). *Let U, R, P be sets of users, roles and user privileges, the set of all privileges $P^{\dagger}_{U,R,P}$ is defined by the following grammar:*

$$p \ ::= \ q \mid \Box(u,r) \mid \Diamond(u,r) \mid \Box(r,r') \mid \Diamond(r,r') \mid \Box(r,p) \mid \Diamond(r,p).$$

where $u \in U$, $r, r' \in R$, and $q \in P$.

Each administrative privilege corresponds to an administrative action in a straightforward way. For example, the privilege $\Box(u,r)$ allows to add a member u to the role r. The privilege $\Diamond(u,r)$ allows to remove a member u from the role r. For simplicity we do not model privileges to change the sets U, R, or P, and we assume that they are chosen sufficiently large and fixed. The rationale is that changes to U, R, or P do not actually change the policy, rather they change which policies are well-formed. For example, in practice the set of users could be chosen to be *all strings starting with 'uid'*, which is independent of which users are assigned to roles in the RBAC policy.

As mentioned, $\Box$ and $\Diamond$ are connectives and the set $P^{\dagger}$ is infinite. (Even if U, R, P are finite.) For example, one could have an expression $\Box(r, \Box(u, r'))$, which expresses the privilege to give to role r, the privilege to a user u to the role r'. We can now define administrative policies.

Definition 3 (Administrative Policies). *Let U, R, P be sets of users, roles and user privileges,* an administrative RBAC policy *ϕ is a tuple*

$$\phi = (UA,\ RH,\ PA^{\dagger}),$$

where $UA \subseteq U \times R$ is a set of user assignments, $RH \subseteq R \times R$ a role-hierarchy, and $PA^{\dagger} \subseteq R \times P^{\dagger}$ are the assignments to user or administrtaive privileges.

The set of administrative policies is denoted $\Phi^{\dagger}_{U,R,P}$, which is a superset of the policy set $\Phi_{U,R,P}$ from standard RBAC (see Definition 1). Administrative policies allow users to make policy changes. We model this formally by defining administrative commands.

Definition 4 (Administrative Commands). *Let U, R, P, be sets of users, roles and user privileges, an* administrative command *is a term*

$$\texttt{cmd}(u, a, v, v'),$$

where $u \in U$, $a \in \{\Box, \Diamond\}$ and $v, v' \in U \cup R \cup P^{\dagger}$.
A command queue *is a list of administrative commands, denoted $cq = \texttt{cmd}(u, a, v_1, v_2) : \texttt{cmd}(u', a', v_1', v_2')...$, where $:$ denotes the list constructor.*

The set of command queues is denoted CQ. The empty command queue is denoted ε. The administrative functionality of the RBAC reference monitor is modeled by a command queue, and an administrative RBAC policy. The RBAC reference monitor changes the policy by executing administrative commands in the command queue. We formalize this by a transition function.

Definition 5 (Administrative Transition). *Let $cq \in CQ$ be a command queue, and $\phi \in \Phi^{\dagger}$ an administrative policy, the* administrative transition function*, denoted $\Rightarrow\!:\!\Rightarrow\!: CQ \times \Phi^{\dagger} \to CQ \times \Phi^{\dagger}$, is*

$$\begin{aligned}
\langle \texttt{cmd}(u, \Box, v, v') : cq,\ \phi \rangle &\ \Rightarrow\ \langle cq,\ \phi \cup (v, v') \rangle, \text{ if } u \to_{\phi} r \text{ and } r \to_{\phi} \Box(v, v'). \\
\langle \texttt{cmd}(u, \Diamond, v, v') : cq,\ \phi \rangle &\ \Rightarrow\ \langle cq,\ \phi \setminus (v, v') \rangle, \text{ if } u \to_{\phi} r \text{ and } r \to_{\phi} \Diamond(v, v'). \\
\langle \texttt{cmd}(\ldots) : cq,\ \phi \rangle &\ \Rightarrow\ \langle cq,\ \phi \rangle, \text{ otherwise.}
\end{aligned}$$

Note that if an administrative command is not allowed by the policy ϕ, then the command is removed from the queue, without changing the policy ϕ. Below, a sequence of executions of commands in the queue is called a *run*, denoted by $\Rightarrow^*$. We give a brief example by applying this model in a practical situation.

Example 2. Consider the policy in Example 1. Alice, the security officer, wants to delegate some of her administrative authority to the employees of the Human Resource department (HR). In this way, members of *HR* can appoint new staff members or nurses, without having to recur to Alice each time. To delegate these administrative privileges, Alice uses an administrative policy.

Figure 2 shows Alice's policy: Members of *HR* can assign and revoke certain users to *staff* and *nurse* roles. Additionally, to protect the confidentiality of health records in the tables *t2* and *t3* Alice delegated a revocation privilege about the role *dbusr2* to the role *dbusr3*. The administrative policy hence not only describes who can access which resources, but also which roles have privileges to change to the policy.

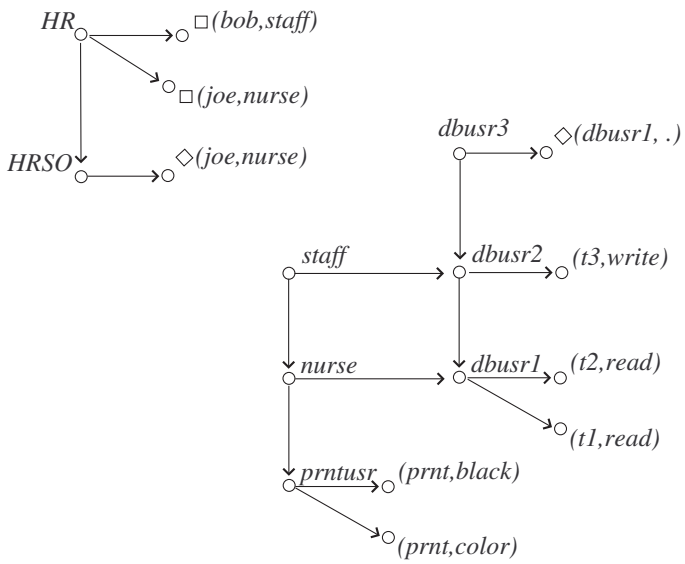

Fig. 2. The administrative RBAC policy deployed by Alice, the security officer.

4 Administrative Refinement

In the previous section we have defined a general class of administrative policies for the General Hierachical RBAC model. In existing literature [3, 4, 6, 9, 14], the administrative privileges in RBAC policies are treated just like ordinary user privileges. In this section we show that this is more restrictive than necessary for safety, and that a more flexible approach can be very useful in practice. This section is organized as follows. First we formalize the notion of *administrative refinement*. In section 4.1 we show that the privilege ordering for *assignment privileges* [5] yields administrative refinements of policies, and in section 4.2 we present the formal proof that the privilege ordering is decidable.

Ignoring policy changes for the moment, an access control policy ψ is safer than a policy ϕ, if ψ grants users to less privileges than ϕ does. We call this *non-administrative refinement*.

Definition 6 (Non-Administrative Refinement). *Let $\phi, \psi \in \Phi^{\dagger}$ be two RBAC policies. We say that ψ is a non-administrative refinement of ϕ, denoted $\phi \succeq \psi$, iff for any $v \in U \cup R$ and any* user privilege *$p \in P$, $v \rightarrow_{\psi} p$ implies $v \rightarrow_{\phi} p$.*

We give a basic example to illustrate this definition.

Example 3 (Non-Administrative Refinement). Consider the policy depicted in Figure 1. Clearly, by removing any of the edges in the policy one obtains a refinement of the policy. For example, by removing Diana from the *staff* role. There is a more fine-grained type of refinement that rearranges edges. For example, if we replace the edge between *Diana* and *staff* with an edge between *Diana* and *nurse*, then we have another refinement of the policy. On the other hand, if we replace the edge between *nurse* and *dbusr1* with an edge between *nurse* and *dbusr2*, we do not obtain a refinement, as nurses get more privileges.

We can now define administrative refinement. The goal of an administrative policy is to allow certain policy changes. Basically, an administrative refinement of a policy is a policy that allows *safer* policy changes. Note that a policy change made by one user may allow other users to make new policy changes, and so on. Therefore, to determine the possible policy changes that are allowed, we must take into account which users are performing administrative actions, and in which order[5]. We formalize administrative refinement as follows.

Definition 7 (Administrative Refinement). *Let $\phi, \psi \in \Phi^\dagger$ be administrative RBAC policies. We say that ψ is an* administrative refinement *of ϕ, denoted $\phi \succeq^\dagger \psi$, if, for any queue of administrative commands $cq \in CQ$, there is a queue of administrative commands $cq' \in CQ$, such that $\phi' \succeq \psi'$, where $\langle cq, \phi\rangle \Rightarrow^* \langle \varepsilon, \phi'\rangle$, and $\langle cq', \psi\rangle \Rightarrow^* \langle \varepsilon, \psi'\rangle$, and cq' is such that, it contains the same number of commands, and the n-th command in cq and the n-th command in cq' are both of the form $\mathtt{cmd}(u, ., .)$, where n ranges over the number of commands in the queue cq.*

Basically the definition states that, if ψ allows a certain policy change then either the same policy change is also allowed by the policy ϕ, or it is a policy change that results in a *safer* policy. It is easy to see that administrative refinement implies non-administrative refinement; take $cq = cq' = \varepsilon$. In other words, if $\phi \succeq^\dagger \psi$ holds then also $\phi \succeq \psi$ holds.

4.1 Ordering Administrative Privileges

In this section, we introduce first a privilege ordering on administrative privileges [5] and we show that the ordering of the administrative priv-

[5] Taking into account the order is more precise than in the HRU model [7] where it is assumed that there is a group of untrusted users who can collude in any order, which does not allow to distinguish the policy $lowrole \rightarrow \Box(r, p)$ from $highrole \rightarrow \Box(r, p)$ (but the latter is more safe).

ileges yields administrative refinements of a policy. At the end of this section we show how the privilege ordering can be used in practice to allow more flexible policy management.

Consider a simple setting where a sub-administrator has the explicit right to assign a user u to a *high* role in the role-hierarchy. There is no reason to forbid the sub-administrator to assign the user to a lower role. This can be seen as follows. If u becomes a member of the high role, then u can activate also the lower roles and obtain their privileges, as if u was assigned to it explicitly. In existing RBAC literature administrative privileges are not interpreted in this way. The ordering of privileges, just described here, can be defined formally as follows.

Definition 8 (Privilege Ordering). *Let $\phi \in \Phi^\dagger$ be an administrative policy, let p, p_1, p_2 be privileges in $P^\dagger$, and let v_1, v_2, v_3, v_4 be users (U) or roles (R). We define the relation $\rightsquigarrow_\phi$ as the smallest relation satisfying:*

$$p \rightsquigarrow_\phi p \tag{1}$$

$$\Box(v_2, v_3) \rightsquigarrow_\phi \Box(v_1, v_4), \text{ if } v_1 \rightarrow_\phi v_2 \text{ and } v_3 \rightarrow_\phi v_4. \tag{2}$$

$$\Box(v_2, p_1) \rightsquigarrow_\phi \Box(v_1, p_2), \text{ if } v_1 \rightarrow_\phi v_2 \text{ and } p_1 \rightsquigarrow_\phi p_2. \tag{3}$$

The ordering $\rightsquigarrow_\phi$ is both reflexive and transitive. In practice the privilege ordering can be used to allow users, with administrative privileges, to be implicitly authorized for weaker administrative privileges.

It can be shown (see the Theorem 1 below) that by replacing an administrative privilege by a weaker one (with respect to the ordering), one obtains an administrative refinement of the policy. In other words, giving administrative users also the weaker administrative privileges allows them to perform also *safer* administrative operations than the ones originally allowed.

Theorem 1. *Let $\phi \in \Phi^\dagger$ be an administrative policy, let $(r, p) \in \phi$ be a privilege assignment, and let q be a privilege such that $p \rightsquigarrow_\phi q$, then the policy $\psi = (\phi \setminus (r, p)) \cup (r, q)$ is an administrative refinement of ϕ, that is $\phi \succeq^\dagger \psi$.*

Proof. (Sketch) The proof is by case analysis over the different cases in definition 8.

The first case is trivial, since the relation $\succeq^\dagger$ is reflexive. For the second case take a policy ϕ with privilege assignment $(r, \Box(v_2, v_3))$, and $v_1 \rightarrow_\phi v_2$, and $v_3 \rightarrow_\phi v_4$. Let ψ be the same policy where this privilege assignment is replaced by $(r, \Box(v_1, v_4))$. So ϕ allows the command

$$\texttt{cmd}(u, \Box, v_2, v_3),$$

which changes ϕ to $\phi' = \phi \cup (v_2, v_3)$, while ψ allows the command

$$\texttt{cmd}(u, \Box, v_1, v_4),$$

which changes ψ to $\psi' = \psi \cup (v_2, v_3)$. To show that $\phi \succeq^\dagger \psi$ it is sufficient to show that $\phi' \succeq \psi$: In ψ' v_1 has the privileges of $v4$, but in ϕ' v_1 has the same privileges, due to the edges $v_1 \rightarrow_\phi v_2$, $v_3 \rightarrow_\phi v_4$ and $v_2 \rightarrow v_3$.

For the third case take a policy ϕ with privilege assignment $(r, \Box(v_2, p_1))$, and $v_1 \rightarrow_\phi v_2$, and $p_1 \rightsquigarrow_\phi p_2$. Let ψ be the same policy where this privilege assignment is replaced by the weaker privilege $(r, \Box(v_1, p_2))$. So ϕ allows the command

$$\texttt{cmd}(u, \Box, v_2, p_1),$$

which changes ϕ to $\phi' = \phi \cup (v_2, p_1)$, while ψ allows the command

$$\texttt{cmd}(u, \Box, v_1, p_2),$$

which changes ψ to $\psi' = \psi \cup (v_1, p_2)$. In case p_1 is a user privilege, p_1 equals p_2 and the proof is the same as for the second case. We simply show that ψ' is a non-administrative refinement of ϕ'. On the other hand, if p_1 is an administrative privilege we must show that the subsequent commands allowed by ψ' yield refinements of the policies created by commands allowed by ϕ'. This can be shown by induction over the structure (the number of nestings of $\Box$) of p_1.

Let us now give an example of how the privilege ordering can be used in a practical situation.

Example 4 (A Flexworker). Consider the administrative RBAC policy depicted in Figure 2. The role *HR* has the administrative privilege to add new members to the staff role. There is also a role below staff called nurse, with additional privileges. Bob is a flexworker, Jane is from the *HR* department.

Bob arrives at the hospital and his job is to put some order in the health record database. To do the job he needs to have *dbusr2* privileges. Jane a member of the role *HR* can give the necessary clearance to Bob. Jane can give Bob staff privileges (the dashed edge in Figure 3). If she does so, then she must urge Bob to apply the principle of least privilege, by activating only the role *dbusr2*, and not e.g. the staff or the nurse role, which would yield excessive privileges, for instance medical privileges. But Jane can only hope that Bob does so.

The privilege ordering implies that Jane can assign Bob directly to the *dbusr2* role (the dotted edge in Figure 3) *because* of her privilege to

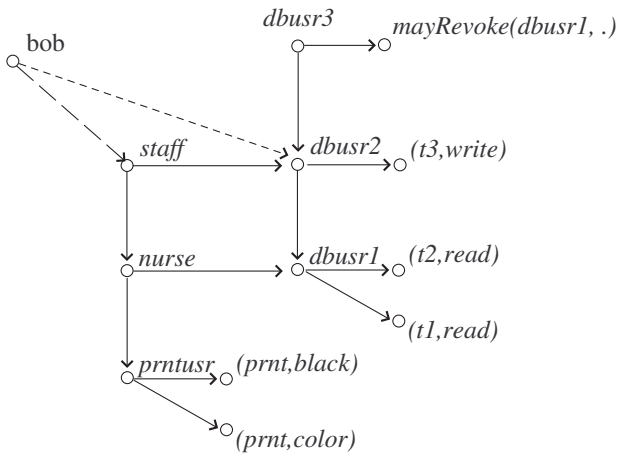

Fig. 3. A practical example of the use of administrative refinement.

add Bob to the staff role. In a way, instead of preaching the principle of least privilege to Bob, Jane applies it for him.

Remark 1 (Less privileges, safer policies). In this paper we have defined a policy to be safer when the policy gives users less privileges. The principle of least privilege, and the way it is supported by the RBAC session mechanism, is a well-known example of the usefulness of this definition. One could perhaps argue that there could be practical situations where having less privileges is not more safe. For example one could imagine a privilege to append to a log file. Removing this privilege could cause programs to run unsafely, that is without writing logs. We believe however that such peculiarities should be resolved at the application layer. For example by changing the program so that it halts when no logs can be written.

4.2 Tractability

In this section we address an important practical issue. We prove that the ordering relation (Definition 8) is tractable. Since the full set P of privileges is infinite, this result is not immediate. For instance, a naive forward search does not necessarily terminate (see the example at the end of this section). The proof indicates how a decision algorithm, deciding which privileges are to be given to which roles, can be implemented at an RBAC reference monitor.

Lemma 1 (Decidability of the Ordering Relation). *Let $\phi \in \Phi^\dagger$ be an administrative policy, and $p, q \in P^\dagger$ be two privileges, it is decidable whether $p \rightsquigarrow_\phi q$.*

Proof. The proof is by structural induction over q.

The base cases are when q is not of the form $\Box(r, r')$. We show that for the three base cases $p \rightsquigarrow_\phi q$ is decidable:

- Either q is a user privilege from P. In this case $p \rightsquigarrow_\phi q$ holds only when $p = q$ (see rule (1) in Definition 8).
- Or q is of the form $\Box(v, v')$ for some $v, v' \in U \cup R$, in which case only rule (2) needs to be checked, which has finite premises.

For the induction step, suppose that q is $\Box(r', p')$, for some role r' and privilege p'. Now, $p \rightsquigarrow q$ can only hold if the premises of rule (3) hold. The premises of rule (3) are decidable, either because they are finite, or because the induction hypothesis is applicable (in $p' \rightsquigarrow q'$, q' is structurally smaller than q, regardless of p').

Let us show how the proof above can be used in practice, as a procedure for checking whether one privilege is weaker than another.

Example 5. Consider Example 4 again. Can Jane assign Bob to the *dbusr2* role? We have to check that the role staff inherits the privilege $\Box(bob, dbusr2)$. Using the first part of Definition 8, one finds that the staff role has the privilege $\Box(bob, staff)$. Now we should decide whether

$$\Box(bob, staff) \rightsquigarrow \Box(bob, dbusr2).$$

This follows trivially from the first rule of Definition 8.

To give a more involved example, suppose that the system administrator Alice has the privilege $\Box(staff, \Box(bob, staff))$. Can Charlie give to staff, the privilege $\Box(bob, dbusr2)$ directly? We have to check whether

$$\Box(staff, \Box(bob, staff)) \rightsquigarrow \Box(staff, \Box(bob, dbusr2)).$$

This is indeed the case by using rule (3) first, and then rule (2).

Now, for the sake of exposition, let us remove the edge between the *staff* and the *dbusr2* role. Let us show how to determine that the previous relation does not hold: Now only rule (3) applies, in which case we must decide whether $\Box(bob, staff) \rightsquigarrow \Box(bob, dbusr2)$. This is a base case of the induction described in the proof of Lemma 1: Only rule (2) remains to be checked and than we can conclude that it does not hold.

It could be useful to find *all* the privileges p' weaker than a given p. To our surprise, in some cases the set of all privileges p' weaker than a given privilege p, is infinite. Let us give an example.

Example 6 (Infinitely many weaker privileges). Consider a policy where $(r_2, \Box(r_1, r_2)) \in PA$. We should stress here that this is by no means an artificial, or peculiar policy: Members of r_2 can make members of r_1 member too.

Suppose we are interested in finding all the privileges weaker than $\Box(r_1, r_2)$. The first weaker privilege we discover by applying rule (2) in definition 8:

$$\Box(r_1, \Box(r_1, r_2)).$$

Using this result in rule (3), we find another weaker privilege,

$$\Box(r_1, \Box(r_1, \Box(r_1, r_2))),$$

and we can use this again in rule (3), and so on.

Remark 2. The outer nesting in the last term in the previous example is in a sense redundant. Instead of assigning the privilege $\Box(r_1, r_2)$ to r_1, one assigns the privilege to do so, to r_1. This only requires the users in role r_1 to perform an extra administrative step, which seems unnecessary. It is cumbersome for the user in r_1, and it does not introduce any safeguards. We conjecture that for all practical purpose one could stop after n applications of rule (3), where n is the length of the longest chain in RH. We do not make this observation more formal here.

5 Related Work

The problem of administration of an RBAC system was first addressed by Sandhu et al. [10]. Later, numerous articles have been published extending or improving the administration model proposed there [3, 4, 6, 9, 11, 13, 14]. We discuss some of them.

Barka et al. [3] distinguish between original and delegated user role assignments. Delegations are modeled using special sets, and different sets are used for single step and double step delegations (which must remain disjoint). A function is used to verify if membership to a role can be delegated. Privileges can also be delegated, provided they are in the special set of delegatable privileges belonging to the role. In their work, each level of delegation requires the definition of tens of sets and functions, whereas in our model administrative privileges, of an arbitrary complexity, are

simply assigned to roles, just like the ordinary privileges. The PDBM model [14] defines a cascaded delegation. This form of delegation is also expressible in our grammar (by nesting the $\square$ connective). In the PDBM model, however, each delegation requires the addition of a separate role, which is cumbersome given the fact that there are already many roles to manage. In our model the administrative privileges are assigned to roles just like the ordinary privileges. It is not required to add any additional roles.

A number of proposals define general constraints on the administrative privileges. For example, the constraint that a user must first have a privilege, before being allowed to delegate it to other users. Note that, as mentioned earlier, in this paper no particular choice is made with respect to such constraints. Zhang et al. [13] implement rule based constraints on delegations. They demonstrate their model using a Prolog program. Basically, they analyze the properties of a centralized RBAC system, focussing on so-called *separation of duty* policies. Crampton [4] defines the concept of administrative scope. Basically a role r is in the scope of a role r' if there is no role above r' that is not below r. They show how administrative scope can be used to constrain delegations to evolve in a natural progression in the role hierarchy. Bandman et al. [2] use a general constraint language to specify constraints on who can receive certain delegations.

6 Conclusion

With this work we make a contribution to the design of flexible administration models for RBAC. Flexible administration is important to cut the cost of maintenance and to enable the RBAC system to adapt to changing circumstances. In general, the flexibility of management is a very important requisite for access control systems. Discretionary access control systems are prevalently used (see for instance Linux, Windows) because users can change the policies about their files so easily. Mandatory access control systems, on the other hand (such as RBAC) are deployed to a lesser extent because they are too inflexible. There are settings where flexibility is required, but discretionary access control is inappropriate. A good example is the setting of the protection of electronic health records. The high availability requirements for health records require flexibility, and at the same time, policies protecting health records are not at the discretion of medical personnel creating and using them. RBAC, with a

flexible decentralized policy management mechanism could be an interesting solution in such settings.

The issue of designing *flexible* yet *safe* policy administration mechanisms for RBAC has received much attention recently [3, 4, 6, 9, 14]. With this paper we contribute to these lines of research. We introduce the notion of administrative refinement of policies, and we show how it can be used to allow more flexible management of the RBAC policy. Concretely, our contribution is a the definition of a general class of administrative policies, and a formal definition of administrative refinement. We have shown that there is a natural ordering for administrative privileges which yields administrative refinements of policies, and we have presented the proof that this ordering is tractable. We also showed how useful our extension is in practice. Our approach allows administrative users to be implicitly authorized for weaker administrative operations, which is thus more flexible and more safe as well.

Revocation privileges are included in our model, but we have not identified (yet) a separate ordering for revocation privileges. We believe that this is an interesting possibility for further research.

References

1. RBAC Standard, ANSI INCITS 359-2004, 2004.
2. O. L. Bandmann, B. Sadighi Firozabadi, and M. Dam. Constrained delegation. In M. Abadi and S. M. Bellovin, editors, *Proc. of the Symp. on Security and Privacy (S&P)*, pages 131–140. IEEE Computer Society Press, 2002.
3. E. Barka and R. S. Sandhu. Framework for role-based delegation models. In J. Epstein, L. Notargiacomo, and R. Anderson, editors, *Annual Computer Security Applications Conference (ACSAC)*, pages 168–176, 2000.
4. J. Crampton and G. Loizou. Administrative scope: A foundation for role-based administrative models. *Transactions on Information System Security (TISSEC)*, 6(2):201–231, 2003.
5. M.A.C. Dekker, J. Cederquist, J. Crampton, and S. Etalle. Extended privilege inheritance in RBAC. In *Proc. of the Symp. on Information, Computer and Communications Security (ASIACCS)*, page to be published. ACM Press, 2007.
6. D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-based Access Control*. Computer Security Series. Artech House, 2003.
7. M. A Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Comunications of the ACM*, 19(5), 1976.
8. N. Li, J. Byun, and E. Bertino. A critique of the ANSI standard on role based access control. *IEEE Security and Privacy*, page in press.
9. R. S. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for role-based administration of roles. *Transactions on Information and System Security (TISSEC)*, 2(1):105–135, 1999.
10. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

11. J. Wainer and A. Kumar. A fine-grained, controllable, user-to-user delegation method in RBAC. In E. Ferrari and G. Ahn, editors, *Proc. of the Symp. on Access Control Models and Technologies (SACMAT)*, pages 59–66. ACM Press, 2005.
12. H. Wang and S. L. Osborn. An administrative model for role graphs. In *Proc. of the IFIP TC-11 WG 11.3 Annual Working Conference on Data and Application Security (DBSec)*, pages 302–315. Kluwer, 2003.
13. L. Zhang, G. Ahn, and B. Chu. A rule-based framework for role-based delegation and revocation. *Transactions on Information and System Security (TISSEC)*, 6(3):404–441, 2003.
14. X. Zhang, S. Oh, and R. S. Sandhu. PBDM: a flexible delegation model in RBAC. In D. Ferraiolo, editor, *Proc. of the Symp. on Access Control Models and Technologies (SACMAT)*, pages 149–157. ACM Press, 2003.