Discretionary and Mandatory Controls for Role-Based Administration

Jason Crampton

Information Security Group, Royal Holloway, University of London
jason.crampton@rhul.ac.uk

Abstract. Role-based access control is an important way of limiting the access users have to computing resources. While the basic concepts of role-based access control are now well understood, there is no consensus on the best approach to managing role-based systems. In this paper, we introduce a new model for role-based administration, using the notions of discretionary and mandatory controls. Our model provides a number of important features that control the assignment of users and permissions to roles. This means that we can limit the damage that can be done by malicious administrative users. We compare our approach to a number of other models for role-based administration, and demonstrate that our model has several advantages.

1 Introduction

Role-based access control (RBAC) is an increasingly popular model for limiting the access users have to resources provided by a computer system. A role provide a means of associating a group of users (typically corresponding to some particular job within an organization) to some set of permissions (corresponding to the functions and duties performed by users with that particular job).

"Administration" in the context of access control is a generic term that is taken to mean the management of the sets and relations underpinning the access control model. Adding users or changing the access rights associated with those users are part of the administrative process, for example. The administration of role-based systems using role-based principles has been less widely studied than RBAC itself, although some important progress has been made [1,2,3]. A number of important ideas have emerged from this work: RBAC96 introduces the idea of administrative permissions [3]; ARBAC97 requires that the parameters of administrative operations should satisfy certain conditions [2]; the RHA model introduces the idea of administrative scope which divides the role hierarchy into a number of different administrative domains [1].

The ANSI-RBAC standard was released in 2004 [4]. It is very strongly influenced by the RBAC96 model [3] and the NIST proposal for an RBAC standard [5]. It states the administrative functions that must be supported by an RBAC system that is compliant with the standard. However, it provides no model for administration, nor does it specify how those functions should be implemented.

E. Damiani and P. Liu (Eds.): Data and Applications Security 2006, LNCS 4127, pp. 194–208, 2006.

We believe, therefore, that there is a pressing need for a comprehensive model for administration in role-based systems, and it is this need that we address in this paper. Moreover, we believe that existing models for administration each have certain limitations: primarily, the assignment of administrative permissions to roles in RBAC96 is insufficiently structured and leads to a lack of control over the propagation of permissions, while the highly structured approaches of ARBAC97 and RHA are insufficiently flexible.

In this paper, we construct a general model for role-based administration that takes advantage of some of the features of existing models and introduces some new features. That is, we employ administrative permissions, divide the role hierarchy into different administrative domains, and require that the parameters of an administrative operation meet certain conditions. In particular, we will show that this approach provides far more control over which roles can perform different administrative operations and provides stronger control over which operations are permitted. We can insist that only a human resources role can update the assignment of users to roles, for example, and that the assignment of a user to a role only succeeds if the user, the role and the administrative role performing the assignment satisfy certain conditions. Our approach also means that we are able to require that different roles perform different types of operations within each administrative domain; what we call *administrative separation of duty*.

In the next section we introduce our model for administration. We introduce some prerequisite concepts from mathematics and RBAC, and then define administrative permissions, domains and commands. We then define the extensions to RBAC96 that are required to support role-based administration. In Sect. 4 we introduce the use of separation of duty constraints for administration. In Sect. 5 we illustrate our approach using an example from the literature. In this section we also compare our approach to related work. We conclude the paper in Sect. 6 by summarizing our contribution and suggesting some ideas for future work.

2 Foundations for a New Model for Administration

2.1 Mathematical Preliminaries

Let $\leqslant$ be a reflexive, anti-symmetric, transitive binary relation on X. Then we say $(X, \leqslant)$ is a *partially ordered set* or *poset*. When the order relation is obvious from context we will simply write "X is a poset". We may write $y \geqslant x$ whenever $x \leqslant y$.

Let $Y \subseteq X$: we say $y \in Y$ is a *maximal element* in Y if $y \leqslant x$ implies that $x = y$ for all $x \in Y$. Informally, there is no element bigger than y in Y. A *minimal element* of Y is defined analogously. We write $\max(Y)$ (respectively $\min(Y)$) to denote the set of maximal (minimal) elements in Y. Let $x, y \in X$: we say y is the *parent* of x if for any $z \in X$ such that $x < z \leqslant y$, $y = z$. In other words, y is the parent of x if there is no element of X that can "fit between" x and y in the ordering. Given $Y \subseteq X$, we use the following notation:

$\downarrow Y = \{x \in X : x \leqslant y, y \in Y\}$; $\uparrow Y = \{x \in X : x \geqslant y, y \in Y\}$. We write $\downarrow x$ rather than $\downarrow \{x\}$ and $\uparrow x$ rather than $\uparrow \{x\}$.

2.2 RBAC Preliminaries

A hierarchical role-based access control model[1] has the following features [3,4]: a partially ordered set of roles $(R, \leqslant)$; a set of permissions P and a permission-role assignment relation $PA \subseteq P \times R$; a set of users U and a user-role assignment relation $UA \subseteq U \times R$. The set of roles available to a user $u \in U$ is defined to be $\{r \in R : r \leqslant r', (u, r') \in UA\}$, and the set of permissions available to u is defined to be $\{p \in P : (p, r) \in PA, r \leqslant r', (u, r') \in UA\}$.

2.3 Administrative Permissions

In the early RBAC literature, permissions were simply "uninterpreted symbols", because the precise nature of permissions is "implementation and system dependent" [3]. In the ANSI-RBAC standard, which is based on the RBAC96 model, permissions are defined by an object and an action (or operation). In the context of administrative permissions, it is instructive to actually specify the object and action for each permission.

The RBAC96 model defines five sets, P, U, $(R, \leqslant)$, $UA \subseteq U \times R$, and $PA \subseteq P \times R$, and we require administrative permissions that can update each of these sets. We assume that the partially ordered set of roles is implemented by a role hierarchy relation $RH \subseteq R \times R$. We assume that there are two generic (primitive) actions `add` and `delete`, and that either of these actions may be applied to one of the objects that define the role-based model. In other words, we may add or delete entries from each of P, U, RH, UA, and PA. We will write `addX` to denote the permission $(\texttt{add}, X)$ and `delX` to denote the administrative permission $(\texttt{delete}, X)$, where $X \in \{P, U, RH, UA, PA\}$.

2.4 Administrative Commands

A request to invoke an administrative permission is made by an administrative role. The request will specify a number of parameters, determined by the permission requested. For example, the parameters required when invoking the `addUA` permission, which is a request to assign a user u to a role r, will obviously be u and r, as well as the administrative role a in which the requesting user is acting. In this paper, we will write parameterized administrative requests as function calls and refer to them as *administrative commands* (in the style of Harrison-Ruzzo-Ullman commands [6]).

Table 1 summarizes the ten administrative commands. It should be noted that we have only stated the immediate effect of an administrative command. In general, additional changes may need to be made: in particular, when a role

[1] Both RBAC96 [3] and the ANSI-RBAC standard [4] distinguish between flat RBAC (RBAC_0 [3]) and hierarchical RBAC (RBAC_1 [3]). This distinction is unnecessary as an unordered set is a partially ordered set with an empty order relation.

Table 1. Administrative commands

Command	Effect	Description
$\texttt{addU}(a, u)$	$U \leftarrow U \cup \{u\}$	Create a new user account
$\texttt{addP}(a, p)$	$P \leftarrow P \cup \{p\}$	Create a new permission
$\texttt{addR}(a, r, C, P)$	$R \leftarrow R \cup \{r\}$	Add role with children C and parents P
$\texttt{addE}(a, c, p)$	$RH \leftarrow RH \cup \{(c, p)\}$	Add edge (c, p)
$\texttt{addUA}(a, u, r)$	$UA \leftarrow UA \cup \{(u, r)\}$	Create a new user-role assignment
$\texttt{addPA}(a, p, r)$	$PA \leftarrow PA \cup \{(p, r)\}$	Create a new permission-role assignment

is added or deleted, the set of edges (RH) will also need to be updated. We have omitted these changes in order to simplify the presentation. We assume each command is initiated by an administrative role a. For each add command, there is a corresponding delete command; these commands are not shown in the table.

2.5 Administrative Domains

A fundamental part of our model is the concept of an *administrative domain*. Intuitively, an administrative domain is a set of roles within the role hierarchy that can be administered by the same role. Given a role hierarchy, we would expect that an appropriate choice of domains would be quite apparent. However, formalizing the notion of an administrative domain is rather more difficult.

In fact, it is easier to consider the set of administrative domains. This is because it seems reasonable that there should be no overlap between different administrative domains. The justification for this lies in the notions of "ownership" and "responsibility". Suppose that role r belongs to two different administrative domains D_1 and D_2 and each domain is associated with an administrative role a_1 and a_2. Then if a_1 assigns a user u to r, u may acquire certain roles and permissions within D_2 without a_2's knowledge or approval. This suggests that the set of roles should be partitioned (in a mathematical sense) into different domains.

There is, however, one situation where overlapping domains would be intuitively reasonable, particularly within the context of RBAC. That is, when one domain D_2 is completely contained within a second D_1. In this case, when a_1 assigns u to a role within D_1, any roles that u obtains in D_2 through inheritance are also contained within D_1. Hence, we define the notion of a nested partition of a set and use this to define an administrative partition of a set of roles.

Definition 1. *Let X be a set. A family of sets $\mathcal{Y} = \{Y_1, \ldots, Y_n\}$, $\emptyset \subset Y_i \subseteq X$, is a* nested partition *of X if $X = \bigcup_{i=1}^{n} Y_i$, and for all $Y_i, Y_j \in \mathcal{Y}$, one of the following conditions holds:* (i) $Y_i \cap Y_j = \emptyset$ (ii) $Y_i \subseteq Y_j$ (iii) $Y_j \subseteq Y_i$.

Let $(R, \leqslant)$ be a set of roles and let $\mathcal{D} = \{D_1, \ldots, D_n\}$, $D_i \subseteq R$. We say $\mathcal{D}$ is an *administrative partition* of R if $\mathcal{D}$ is a nested partition of R. Each element of an administrative partition is called an *administrative domain*. In other words, $\mathcal{D}$ is an administrative partition if every role is contained in at least one domain and every pair of domains either has empty intersection or one is completely

contained in the other. Clearly, $\mathcal{D} = \{R\}$ is a (trivial) set of administrative domains. At the other extreme, $\mathcal{D} = \{\{r_1\}, \ldots, \{r_n\}\}$, where $R = \{r_1, \ldots, r_n\}$, is a set of administrative domains. Note that a set of administrative domains is partially ordered by subset inclusion. In fact, we have the following two results.

Proposition 2. *Let $\mathcal{D}$ be a set of administrative domains. Then for any $D \in \mathcal{D}$, D has at most one parent.*

Proof. We need to show that if a domain D has a parent D', then D' is unique. Suppose, in order to obtain a contradiction, that there exist two distinct domains D' and D'' that are both parents of D. Now $D' \cap D'' \neq \emptyset$, since $D \subseteq D' \cap D''$. Hence, since D' and D'' are distinct administrative domains, either $D' \subset D''$ or $D'' \subset D'$. We can assume without loss of generality that $D' \subset D''$ and hence D'' is not a parent of D (since we have $D \subset D' \subset D''$). This is the required contradiction and the result now follows.

Corollary 3. *Let $\mathcal{D}$ be an administrative partition of R. Then the graph of the reflexive transitive reduction of the poset $(\mathcal{D}, \subseteq)$ is a forest.*

3 A New Administrative Model for RBAC96

We augment the standard RBAC96 model with a set of administrative domains $\mathcal{D}$ and a domain-role assignment relation $DA \subseteq \mathcal{D} \times R$. We say r has *administrative control* over D if $(D, r) \in DA$. We say r is an *administrative role* if $(D, r) \in DA$, for some $D \in \mathcal{D}$. (If $\mathcal{D} = \{R\}$, we do not require the DA relation.) Let $D, D' \in \mathcal{D}$ and $D \subseteq D'$. If r has administrative control over D' then we assume that r also has administrative control over D. Hence, the ordering on the set of domains induces an ordering on the set of administrative roles.

We define a new relation assigning administrative permissions to roles $APA \subseteq AP \times R$, where AP is the set of administrative permissions. An administrative command is issued by (a user acting in) an administrative role a and is interpreted as an attempt to invoke an administrative permission with particular parameters (chosen by the requesting user). An administrative command is permitted if the *discretionary administrative property* and the *mandatory administrative property* are satisfied:

Property 1. *The administrative command* $\texttt{cmdX}(a, \ldots)$ *satisfies the* discretionary administrative property *if* $(a, (\texttt{cmd}, X)) \in APA$, *where* $\texttt{cmd} \in \{\texttt{add}, \texttt{delete}\}$.

Property 2. *The administrative command* $\texttt{cmdX}(a, \ldots)$ *satisfies the* mandatory administrative property *if the role parameters of the command belong to a domain over which a has administrative control. In particular:*

- *for* $\texttt{addR}(a, r, C, P)$ *to succeed, there must exist a domain D over which a has administrative control and $C, P \subseteq D$;*
- *for* $\texttt{addE}(a, c, p)$ *and* $\texttt{delE}(a, c, p)$ *to succeed, there must exist a domain D over which a has administrative control and $c, p \in D$;*

– *for* $\texttt{delR}(a, r)$, $\texttt{addUA}(a, u, r)$, $\texttt{addPA}(a, p, r)$, $\texttt{delUA}(a, u, r)$, *and* $\texttt{delPA}(a, p, r)$ *to succeed, there must exist a domain* D *over which* a *has administrative control and* $r \in D$.

Note that if r has administrative control over D' then r also has administrative control over every domain $D \in \mathcal{D}$ such that $D \subseteq D'$. In other words, the mandatory administrative property is similar to the simple security property of the Bell-LaPadula model [7], which states that a subject s can read any object o with a security label that is less than or equal to that of s.

3.1 User-Role Assignment

It is fairly easy to see that the existence of a role hierarchy means that administrative commands could affect more than one administrative domain. In this section we introduce a further mandatory control on the execution of user-role assignment commands.

This control is motivated by the following observation: the assignment of a user u to a role r results in u being implicitly assigned (via inheritance in the role hierarchy) to all roles $r' < r$. In other words, if an administrative role assigns u to a role r within its domain D, it may have an impact on the roles available to u in an administrative domain $D' \supset D$. Hence, a very natural (mandatory) requirement on user-role assignment should be that u is already assigned to more junior roles that are not in the administrative domain of the role to which u is to be assigned. More formally, we have the following security property for user-role assignment.

Property 3. *The command* $\texttt{addUA}(a, u, r)$ *satisfies the* mandatory *UA* property *if there exists a domain* D *over which* a *has administrative control such that* $r \in D$ *and* u *is already assigned to all roles in* $\max(\downarrow r \setminus D)$.

Let us assume that an administrative role a has the permission to create new user accounts. An important consequence of the mandatory *UA* property is that an administrative role assigned to domain D cannot assign a new user to a role $r \in D$ unless $\downarrow r \subseteq D$. This means that it is impossible for an administrative user (assigned to some administrative role), by virtue of the mandatory restrictions on user-role assignment, to assign users to roles over which he has no control. In other words, the damage that a malicious administrative user can do is limited to the domain(s) he controls.

3.2 Permission-Role Assignment

Just as the act of assigning a user to a role r may have an impact outside the domain controlled by the administrative role performing the assignment, the act of assigning a permission to a role may also have undesirable consequences. Consider the command $\texttt{addPA}(a, p, r)$. If r belongs to a domain controlled by a then we might expect that this command should be permitted. However, consider the case when $r < r'$, p is not currently assigned to r', and r' does not belong

to any domain controlled by a. In this case, a is "giving away" the permission p and the permission leaks from the domain(s) to which it has previously been confined. Hence we introduce a mandatory security property for permission-role assignment.

Property 4. *The command* $\texttt{addPA}(a, p, r)$ *satifies the* mandatory *PA* property *if there exists a domain* D *over which* a *has administrative control such that* $r \in D$ *and* p *is already assigned to all roles in* $\min(\uparrow r \setminus D)$.

An important consequence of the mandatory *PA* property is that an administrative user cannot downgrade permissions, by assigning them to less senior roles in the domain(s) he controls, beyond a certain level. As with the mandatory *UA* property, the mandatory *PA* property limits the damage that a malicious administrative user can do.

3.3 Automatic Assignment of Domains

Note that the union of the set of administrative domains is R. Therefore, we need to specify which domain a newly created role should belong to. Intuitively, we wish to assign the role to the most appropriate domain automatically. More formally, we can specify that a newly created role r should belong to the smallest domain D that contains P, where P is the set of parents of r.[2] (Note that "smallest domain" is well defined since the set of administrative domains can be represented as a forest.)

3.4 Choosing Administrative Domains

Of course, the choice of appropriate administrative domains is an important practical aspect of the model proposed in this paper. In fact, the notion of administrative scope, introduced in the RHA model, can be particularly useful in choosing administrative domains.

Definition 4 (Crampton [1]). *The* administrative scope *of a role* r*, denoted* $\sigma(r)$*, is defined in the following way:* $\sigma(r) = \{r' \leqslant r : \uparrow r' \subseteq \downarrow r \cup \uparrow r\}$.

In other words, $r' \in \sigma(r)$ if any role bigger than r' is comparable to r in the role hierarchy. The intuition is that if $r' \in \sigma(r)$, then r "knows about" all the upward inheritance from r', and hence it is appropriate for r to be able to administer r'. However, the reason that administrative scope is particularly useful in the context of this paper is that it can be used to define a nested partition of R.

Lemma 5 (Crampton [8, Lemma 2]). *Let* $r, r' \in R$. *Then*

$$\sigma(r) \cap \sigma(r') = \begin{cases} \sigma(r) & \text{if } r \in \sigma(r'), \\ \sigma(r') & \text{if } r' \in \sigma(r), \\ \emptyset & \text{otherwise.} \end{cases}$$

[2] Note also that if role r is deleted, every domain to which r belongs has r removed from it.

Corollary 6. *$\Sigma(R) = \bigcup_{r \in R} \sigma(r)$ defines a nested partition of R.*

Proof. For all $r \in R$, $r \in \sigma(r)$. Hence $\Sigma(R) = R$. The remaining condition for $\Sigma(R)$ to be a nested partition follows from Lemma 5.

Let $R' \subseteq R$ and write $\Sigma(R')$ for $\bigcup_{r \in R'} \sigma(r)$. Then any $R' \subseteq R$ such that $\Sigma(R') = R$ can be used to define a set of administrative domains. Figure 1 illustrates the non-trivial domains (cardinality greater than 1) identified using administrative scope. Each domain is enclosed by a broken line. $\{\sigma(a), \sigma(b), \sigma(c), \sigma(d)\}$ would be a suitable choice of administrative domains; each of these domains is enclosed by a heavier broken line. The domain forest in this example is simply a root node for $\sigma(a)$, and three child nodes, one for each of the domains determined by the administrative scope of b, c and d.

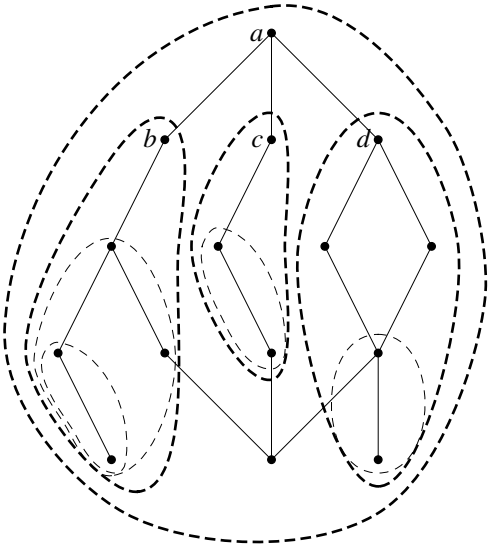


Fig. 1. Administrative domains from administrative scope

3.5 Checking Access Requests Using the Domain Forest

Given an administrative command with administrative role parameter a and role parameters X, we can check whether the mandatory administrative properties are satisfied by considering the sub-tree of the domain forest rooted at a. We simply check whether each role in X is contained in one of the domains in the sub-tree. The simplicity with which the satisfaction of mandatory properties can be checked contrasts sharply with the difficulty of checking whether an administrative command is permitted in the ARBAC97 model [1].

Of course, additional resources are required to store and maintain the domain forest. In particular, the domain forest will need to be updated following an `addR` or `delR` command. Nevertheless, we believe that the benefits that are obtained by using the domain forest more than offset this disadvantage.

4 Administrative Separation of Duty

We now introduce a second facet of our model for administration. We have already seen that the mandatory *UA* and *PA* properties can prevent certain types of administrative abuse. We now consider the use of separation of duty constraints to further reduce the possibility that a single malicious user can compromise the security of a system through a sequence of administrative commands.

Separation of duty is an important control principle in management whereby sensitive combinations of duties are partitioned between different individuals in order to prevent the violation of business rules. The research community has taken an active interest in incorporating separation of duty controls into computer systems since the late 1980s. One of the rules of the Clark-Wilson model [6] requires that separation of duty requirements must be met. In recent years, a number of papers have studied separation of duty in the context of RBAC [9,10,11,12,13,14].

Many of these papers have considered rather complex separation of duty requirements. In the context of role-based administration, we claim that the requirements are rather simple, so we will confine our attention to a very simple model for separation of duty. The model is based on the RBAC96 notion of mutually exclusive roles. However, in our case we will consider mutually exclusive permissions.

Definition 7. *An* administrative separation of duty (ASD) constraint *is a set of administrative permissions. An ASD constraint C is* satisfied *provided no user is assigned to all the permissions in C. An* ASD policy *is a family of ASD constraints.*

As a motivating example, let us assume that we wish to separate the following functions: the creation of new user accounts (or simply users) and the assignment of users to roles. We might wish to do this so that one user cannot create a new user account and assign it to roles, preferring instead that a user acting in some IT support role is responsible for creating users and a user acting in some human resources role is responsible for deciding which roles are relevant to a user's job. In order to realize this requirement, we specify the separation of duty constraint $\{\texttt{addU}, \texttt{addUA}\}$. This constraint is satisfied provided no user is assigned to both permissions (via the *UA* and *APA* relations).

In the case when $|C| = 2$, we can ensure the satisfaction of C by assigning each permission to different administrative roles a_1 and a_2, and ensuring that no user is assigned to both roles. In other words, we can convert the ASD constraint on administrative permissions into an ASD constraint on administrative roles. In fact, we can ensure the satisfaction of any ASD constraint $\{p_1, \ldots, p_n\}$ by re-writing it as a set of ASD constraints on administrative roles. Specifically, we ensure that each permission p_i is assigned to a different administrative role a_i and define the constraints $\{\{a_i, a_j\} : 1 \leqslant i < j \leqslant n\}$. Now we can be assured that the ASD constraint on administrative permissions is satisfied provided the ASD policy on administrative roles is satisfied. It is quite straightforward to ensure that this policy is always satisfied:

– at system initialization we simply check each user's administrative role assignments;
– before allowing any command of the form $\texttt{addUA}(a, u, a')$, where a' is an administrative role, we must check that, if there exists an ASD constraint of the form $\{a', a''\}$, then u is not already assigned to a''.

Suppose, for example, that $\{a_1, a_2\}$ is an ASD constraint on administrative roles and u is already assigned to a_1. Then the command $\texttt{addUA}(a, u, a_2)$ must fail, because allowing it to succeed would violate the ASD constraint.

Another constraint that is likely to be useful in practice is $\{\texttt{addUA}, \texttt{addPA}\}$, which would prevent any administrative role from establishing a link between a user and a particular permission by assigning them to the same role. In practice, we might assign `addUA` to human resources and `addPA` to senior operational roles within each domain. Clearly, the set of constraints that will be used in practice will vary depending on the environment and on the personnel available.

5 Related Work

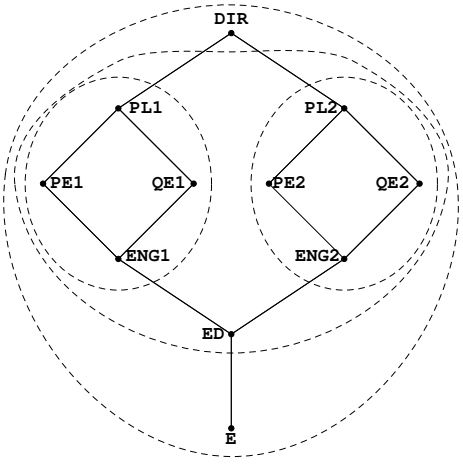
As we noted in the introduction, there have been two generic approaches to administration of role-based systems. We now consider these approaches in more detail and compare them to our approach. In order to provide a concrete basis for this discussion, we introduce an example from the literature. This also provides an opportunity to illustrate the use of our model.

Figure 2(a) shows a typical role hierarchy that has been used as an illustrative example by Sandhu [2]. The hierarchy should be considered in the context of a software engineering company, with two projects under the leadership of roles `PL1` and `PL2`. All employees are assigned to the `E` role and all software engineers are assigned to the `ED` (engineering department) role. Each project has its own security officer role (`PSO1` and `PSO2`); the engineering department has a security officer role (`DSO`); and the company has a senior security officer (`SSO`).

It is clear that there are four main administrative domains, one corresponding to each of the projects, one corresponding to the department (which incorporates both project teams) and one corresponding to the whole role hierarchy. These domains (sets of roles) are enclosed by dashed lines in Fig. 2. The `PSO` roles have control over their respective projects, while the `DSO` and `SSO` roles have control over the department and organization domains. Hence we would suggest the following administrative configuration: we define the set of administrative domains to be $\{D_{P_1}, D_{P_2}, D_{Eng}, R\}$, where $D_{P_i} = \{\texttt{ENGi}, \texttt{PEi}, \texttt{QEi}, \texttt{PLi}\}$[3] and $D_{Eng} = \{\texttt{ED}\} \cup D_{P_1} \cup D_{P_2}$; and we define the domain assignment relation to be $\{(D_{P_1}, \texttt{PSO1}), (D_{P_2}, \texttt{PSO2}), (D_{Eng}, \texttt{DSO}), (R, \texttt{SSO})\}$. Note that $D_1, D_2 \subseteq D_{Eng} \subseteq R$ induces the following partial order on the set of administrative roles: $\texttt{PSO1}, \texttt{PSO2} < \texttt{DSO} < \texttt{SSO}$. In addition we would need to assign administrative permissions to each administrative role.

The mandatory *UA* property allows the `PSO` roles to assign u to a role in their respective domains if u is already assigned to `ED`; `DSO` can assign u to a role in

[3] Note that $\sigma(\texttt{PLi}) = \{\texttt{ENGi}, \texttt{PEi}, \texttt{QEi}, \texttt{PLi}\}$.



(a) A role hierarchy

PSO1	ED	{E1, PE1, QE1}
PSO2	ED	{E2, PE2, QE2}
DSO	ED	{E1, ..., PL1, E2, ..., PL2}
SSO	E	{ED, ..., DIR}

(b) The can_assign relation

PSO1	{E1, PE1, QE1}
PSO1	{E2, PE2, QE2}
DSO	{E1, ..., PL1, E2, ..., PL2}
SSO	{ED, ..., DIR}

(c) The can_revoke relation

Fig. 2. An ARBAC97 example [2]

D_{Eng} if u is already assigned to E; and SSO can assign any user to any role in R. In other words, the larger the domain controlled by an administrative role, the more trust is placed in that role regarding administrative operations.

5.1 "Mandatory" Approaches

The ARBAC97 model was designed specifically as a management model for RBAC96. It includes three sub-models URA97, PRA97, and RRA97, for managing user-role assignment, permission-role assignment and role-role assignment (the role hierarchy), respectively.

Each of the three sub-models uses relations that control what administrative roles are permitted to do. Each of these relations (can_assign and can_revoke in URA97, can_assignp and can_revokep in PRA97, and can_modify in RRA97) define sets of roles that each administrative role is permitted to change. An element $(a, q, R') \in$ can_assign, for example, means that an administrative role a can assign any user who is assigned to the "prerequisite" role $q \in R$ to any role in the set $R' \subseteq R$.

Figure 2 shows examples of the two URA97 relations. The project security officers (PSO1 and PSO2), for example, can assign a user to roles in their respective projects, provided the user is already assigned to ED. The departmental and senior security officer roles (DSO and SSO) have greater powers to assign users to roles. Similarly, the different security officer roles can revoke roles from different sets of roles in the hierarchy, as specified by the can_revoke relation.

Note that our model for administration is very much simpler than ARBAC97. We only require a domain-role assignment relation and the *APA* relation.

Indeed, Fig. 2 omits the other ARBAC97 relations, `can_assignp`, `can_revokep` and `can_modify`.

The ARBAC97 approach mixes the definition of administrative domains with administrative rights. Our proposed approach to role-based administration separates the specification of administrative domains and the assignment of administrative rights. One useful consequence of this is that we only need to specify a set of administrative domains and a domain-role assignment relation.

The ARBAC97 model assumes that `can-assign` and the other relations in the model are static. This means that if a new role is added to the hierarchy, for example, no constraints can be imposed on the assignment of users (and permissions) to that role. We believe that the static nature of ARBAC97 relations is a considerable disadvantage.

The role sets specified in ARBAC97 relations are always expressed as ranges within the partially ordered set of roles. Moreover, it is impossible to delete a role that is the end-point of a range in an ARBAC97 relation. A particularly significant limitation of ARBAC97 is the requirement in RRA97 that all ranges in the `can_modify` relation are encapsulated [2] and that all hierarchy operations should preserve the encapsulation of ranges defined in the `can_modify` relation. One side effect of this requirement is that the creation of a role with either no parent or no child will violate the encapsulation of some encapsulated range in the hierarchy and hence such operations are prohibited. In addition, encapsulated ranges are actually quite rare in partially ordered sets.[4] In short, the use of ranges leads to significant usability problems for the ARBAC97 model.

The ARBAC02 model extends ARBAC97 by introducing organizational units [15]. The main motivation for this seems to be to reduce the number of steps that are required when assigning users (and permissions) to roles. This problem arises as a consequence of the use of prerequisite roles in the `can_assign` and `can_assignp` relations. Organizational units are nothing more than groups of roles and hence can be thought of as administrative domains. The designers of ARBAC02 do not impose any structure on organizational units. Can one role belong to two different organizational units, for example? Organizational units are only used to simplify the prerequisite conditions in the assignment relations `can_assign` and `can_assignp`, which seems a wasted opportunity.

The RHA family of models defines the notion of administrative scope. Whether a request to perform an administrative action is permitted is defined in terms of administrative scope. In this respect, administrative scope is similar to the different role ranges that are defined in each of the ARBAC97 relations. The big advantage of RHA over ARBAC97 is that a single set of roles (namely administrative scope) is used to determine the success or otherwise of administrative commands. In many ways, the RHA family of models is simpler and more versatile than ARBAC97 [1].

However, RHA shares one weakness with ARBAC97: it is vulnerable to changes in the role hierarchy, because the administrative domains are defined in terms of the role hierarchy structure itself. In ARBAC97, domains are defined

[4] A comprehensive analysis of the shortcomings of ARBAC97 can be found in [1].

by encapsulated ranges; in RHA, domains are defined by administrative scope. Hence, it is necessary to perform some additional checks before allowing hierarchy operations, in order to check that administrative domains are preserved by the operation. Again, it turns out that this is easier to check whether domains are preserved in RHA than in ARBAC97, but it is still an overhead [8]. In other words, we believe the approach advocated in this paper, in which domains are simply specified by the system administrator and then left to evolve according to the successful execution of commands by administrative roles, is likely to lead to more useable systems.

5.2 "Discretionary" Approaches

The RBAC96 family of models does not explicitly include administrative functionality. The original paper on RBAC96 suggests that the model can be augmented by administrative permissions and roles, as well as an administrative role hierarchy and an administrative permission-role assignment relation. This approach assumes that there is a single administrative domain D.

There are two significant disadvantages to this approach. Firstly, it is rather difficult to provide fine-grained administrative control: an administrative role either has an administrative permission or it doesn't. Secondly, it is difficult to reason about the propagation of permissions to users, for reasons similar to those that make it difficult to reason about the propagation of access rights in a protection matrix [6].

X-GTRBAC is an XML-based RBAC model that includes temporal constraints on the activation of roles. Bhatti *et al* have proposed an administrative model for X-GTRBAC based on the use of administrative permissions [16]. The interesting aspect of X-GTRBAC Admin is its introduction of administrative domains and the association of administrative permissions with administrative domains. However, X-GTRBAC Admin simply defines domains and associates roles and permissions with each domain. Like ARBAC02, it makes no attempt to impose any structure on administrative domains or to exploit the existence of domains in any way. As such, X-GTRBAC Admin simply extends the administrative model of RBAC96 by the introduction of administrative domains, but does not take advantage of the additional possibilities that this provides.

6 Conclusion

In this paper we have introduced a new model for role-based administration. To our knowledge, this is the first model to combine the use of administrative permissions and conditions on the parameters of an administrative command. These requirements are characterized as discretionary and mandatory administrative controls, respectively. The mandatory controls limit the extent to which administrative users can propagate user- and permission-role assignments, making it more difficult for an administrative user to compromise or damage the access control system (either deliberately or accidentally).

We believe that our model offers a number of advantages over existing approaches to role-based administration. In particular, it is more flexible than approaches such as ARBAC97 and RHA because it does not require administrative domains to have a particular structure (beyond requiring that the set of domains forms a nested partition of R). Nevertheless, it does incorporate mandatory controls, which means it provides greater control over the evolution of the access control system than unstructured approaches such as RBAC96 and X-GTRBAC Admin.

One obvious aspect of future work will be the development of a prototype implementation. On a more theoretical level, it would be interesting to see whether further mandatory properties are required for hierarchy operations. For example, should we limit the ability of a senior administrative role to add an edge between roles in two different domains? Our recent paper considers what conditions need to be placed on hierarchy operations in order to preserve administrative scope in the RHA model and encapsulated ranges in ARBAC97 [8]. It will be interesting to investigate whether analogous conditions are required or can be used for the less structured model described in this paper.

Acknowledgements. I would like to thank the anonymous referees for their comments, which have helped to improve the final version of the paper.

References

1. Crampton, J., Loizou, G.: Administrative scope: A foundation for role-based administrative models. ACM Transactions on Information and System Security **6**(2) (2003) 201–231
2. Sandhu, R., Bhamidipati, V., Munawer, Q.: The ARBAC97 model for role-based administration of roles. ACM Transactions on Information and System Security **1**(2) (1999) 105–135
3. Sandhu, R., Coyne, E., Feinstein, H., Youman, C.: Role-based access control models. IEEE Computer **29**(2) (1996) 38–47
4. American National Standards Institute: ANSI INCITS 359-2004 for Role Based Access Control. (2004)
5. Ferraiolo, D., Sandhu, R., Gavrila, S., Kuhn, D., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Transactions on Information and System Security **4**(3) (2001) 224–274
6. Harrison, M., Ruzzo, W., Ullman, J.: Protection in operating systems. Communications of the ACM **19**(8) (1976) 461–471
7. Bell, D., LaPadula, L.: Secure computer systems: Mathematical foundations. Technical Report MTR-2547, Volume I, Mitre Corporation, Bedford, Massachusetts (1973)
8. Crampton, J.: Understanding and developing role-based administrative models. In: Proceedings of the 12th ACM Conference on Computer and Communications Security. (2005) 158–167
9. Ahn, G.J., Sandhu, R.: Role-based authorization constraints specification. ACM Transactions on Information and System Security **3**(4) (2000) 207–226

10. Crampton, J.: Specifying and enforcing constraints in role-based access control. In: Proceedings of the 8th ACM Symposium on Access Control Models and Technologies. (2003) 43–50
11. Gligor, V., Gavrila, S., Ferraiolo, D.: On the formal definition of separation-of-duty policies and their composition. In: Proceedings of the 1998 IEEE Symposium on Security and Privacy. (1998) 172–183
12. Jaeger, T., Tidswell, J.: Practical safety in flexible access control models. ACM Transactions on Information and System Security **4**(2) (2001) 158–190
13. Nyanchama, M., Osborn, S.: The role graph model and conflict of interest. ACM Transactions on Information and System Security **2**(1) (1999) 3–33
14. Simon, R., Zurko, M.: Separation of duty in role-based environments. In: Proceedings of 10th IEEE Computer Security Foundations Workshop. (1997) 183–194
15. Oh, S., Sandhu, R.: A model for role administration using organization structure. In: Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies. (2002) 155–162
16. Bhatti, R., Joshi, J., Bertino, E., Ghafoor, A.: X-GTRBAC Admin: A decentralized administration model for enterprise-wide access control. In: Proceedings of the 9th ACM Symposium on Access Control Models and Technologies. (2004) 78–86