Extended Privilege Inheritance in RBAC[*]

M.A.C. Dekker[†] J. Cederquist[‡] J. Crampton[§] S. Etalle[¶]

December 20, 2006

Abstract

In existing RBAC literature, administrative privileges are inherited just like ordinary user privileges. We argue that from a security viewpoint this is too restrictive, and we believe that a more flexible approach can be very useful in practice. We define an ordering on the set of administrative privileges, enabling us to extend the standard privilege inheritance relation in a natural way. This means that if a user has a particular administrative privilege, then she is also implicitly authorized for weaker administrative privileges. We prove the non-trivial result that it is possible to decide whether one administrative privilege is weaker than another and show how this result can be used to decide administrative requests in an RBAC security monitor.

1 Introduction

Role-based access control (RBAC) [11] is a non-discretionary access control mechanism that simplifies the assignment of access rights to users. The basic idea is that while there are many access rights and users, rights and users can be grouped using a relatively small number of roles, ordered in a *role hierarchy.* In practice however, an RBAC system in a large enterprise may involve thousands of roles [5]. Keeping the access rights and roles up to date with changes in the enterprise may be a too big task for a single administrator. The usual approach to this problem is to divide the work and to allow delegation of part of the administrator's authority to other users. It is convenient to make this delegation mechanism flexible, in order to reduce the likelihood of bottlenecks and (administrative) users sharing keys or passwords that should remain secret. On the other hand, the delegation mechanism should be *safe*: users should not obtain rights other than those explicitly delegated by the administrator.

[*]This is an extended version of a paper to be published in the proceedings of ASIA CCS 2007, published by ACM press.

[†]Security group, TNO ICT, The Netherlands

[‡]SQIG-IT, IST, TU Lisbon, Portugal

[§]Information Security Group, Royal Holloway, University of London, United Kingdom

[¶]Distributed and Embedded Systems group, University of Twente, The Netherlands

Several lines of research address the problem of delegation of administrative privileges in RBAC systems; Ferraiolo et al. [5] compare some of the different approaches. The main issue in these lines of research is how to model *administrative privileges*, as opposed to the ordinary *user privileges*, and to decide who should have them. In ARBAC [10] 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 the administrative privileges and the ordinary user privileges. Using the concept of *administrative scope*, they define which roles should have administrative privileges over other roles. In the Role-Control Center [5], 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 assigned to these roles.

In existing RBAC literature [2, 4, 5, 10, 15], administrative privileges are inherited just like ordinary user privileges. We argue that this approach is more restrictive than necessary for safety: consider a simple setting where an administrator has delegated the authority to some user u to assign a user u' to a *high* role in the role-hierarchy. When user u uses this authority, user u' becomes assigned to the high role, and consequently u' can also play *lower* roles. There is no security motivation for not letting user u assign user u' directly to (one of) the lower roles. User u' could play the lower roles anyway. However, in standard RBAC, administrative privileges are not interpreted in this way[1]

In this paper, we define an ordering on the administrative privileges, enabling us to extend the standard privilege inheritance relation in a natural way. This means that if a user has a particular administrative privilege, then she is also implicitly authorized for weaker administrative privileges. Basically, this allows for a more flexible use of administrative privileges. Additionally, we show that the new relation is tractable, and we sketch a possible implementation of the extended inheritance relation. We argue that decentralized management of RBAC becomes more flexible with this extension.

2 Preliminaries

In this section we give some definitions about the standard RBAC model [11].

Definition 2.1 (RBAC State) *Given the sets U of users, R of roles and P of privileges, we define an RBAC state as a tuple,*

$$(UA, RH, PA),$$

[1]It could be argued that an entry in, say, an ARBAC97 relation or the existence of a particular administrative scope, by implication gives additional administrative privileges. This is not, however, mentioned explicitly.

where,

$$\begin{aligned}UA &\subseteq U \times R\\ RH &\subseteq R \times R \text{ (a directed graph on } R\text{)}\\ PA &\subseteq R \times P.\end{aligned}$$

Here UA determines which users are assigned to which roles in R. The graph RH is the so-called *role-hierarchy* and PA determines which privileges are assigned to which roles. A user can play the roles to which it is assigned, and the roles that are below them in the hierarchy. Sometimes $(r, r') \in RH$ is written as $r > r'$. Below we denote the reflexive transitive closure over the graph RH with $\geqslant$ (also known as the partial order on RH).

In the RBAC model [11], a user does not get all the privileges associated to the roles it can play: The user has to start a *session*, in which one or more of the user's roles can be *activated*. The session gets only all the privileges of the activated roles. Basically, this allows users to operate from sessions with less privileges than they are entitled to, implementing the so-called *principle of least privilege*. In this paper, for the sake of brevity, we will not be explicit about sessions nor the activations of roles.

Remark 2.2 (About cycles in the role-graph) *In some of the existing literature on RBAC, it is required that RH be acyclic, to avoid redundancy. For example, if both $(a, b) \in RH$ and $(b, a) \in RH$, then using the two different names a and b is redundant. Similarly, sometimes RH is required to be* transitively reduced*; for example, the transitive reduction of $\{(a, b), (b, c), (a, c)\}$ removes the last element, because there would be a path anyway from a to c using the other edges.*

For the sake of brevity, we ignore such constraints. Actually, we do not assume any set of constraints on RH or PA throughout this paper. The results in this paper apply equally to acyclic and cyclic directed graphs. Moreover, the extension of the privilege inheritance relation, to be introduced in the next section, does not introduce extra cycles (in some cases it removes cycles).

Given an RBAC state, we say that a role r has a privilege p, if $(r, p) \in PA$. Additionally, it is common (see below) that privileges of lower roles are available as well, i.e. without the need to activate the lower roles first. This is known as *privilege inheritance*. See for example the RBAC state shown in Figure 1a. The role r_1 inherits the privilege p assigned to role r_2. With privilege inheritance the edge from r_1 to p in Figure 1b is redundant.

Definition 2.3 (Privilege Inheritance) *Given an RBAC state (UA, RH, PA), a role r has the privilege p, denoted $r \rightsquigarrow p$, only if*

$$r \geqslant r' \text{ and } (r', p) \in PA \text{ for some } r' \in R.$$

When a user activates a role in a session, this session acquires all the privileges of the role.

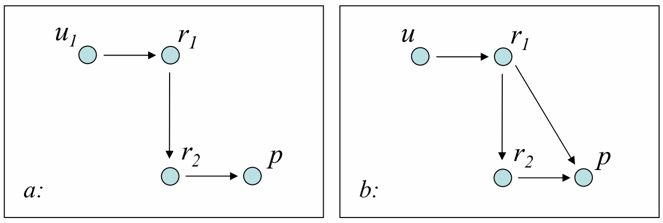

Figure 1: Two sample RBAC states. With privilege inheritance the extra edge on the right is redundant.

The advantage of privilege inheritance in RBAC is that it allows users in a high role, to also use privileges of a lower role, without activating that lower role. This is a well-known feature of RBAC, which can be used to avoid repetitive definitions in the RBAC state (like the edge between r_1 and p).

3 A Different View on Administrative Privileges

Privileges can be divided into *user privileges* and *administrative privileges* [11]. While user privileges allow actions on objects (such as printing files or viewing records), administrative privileges, allow actions on the RBAC state itself, e.g. adding an edge from one role to another. Here we assume that user privileges form a *finite* set of atomic privileges, denoted by Q, that corresponds to a finite set of actions on objects. On the other hand, the set of administrative privileges is necessarily infinite, because privileges about administrative privileges are included as well. We formalize the full set of privileges by defining a *grammar* that encompasses both user privileges and administrative privileges.

Definition 3.1 (Privilege Grammar) *Given the sets U of users, R of roles and Q of user privileges, the set of all privileges P is defined by the following grammar:*

$$p \ ::= \ q \ | \mathit{addUser}(u, r) \ | \ \mathit{addEdge}(r, r') \ | \ \mathit{addPrivilege}(r, p),$$

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

Each administrative privilege corresponds to an administrative action. The privilege $\mathit{addUser}(u, r)$ allows the action of adding a member u to the role r. The privilege $\mathit{addEdge}(r_1, r_2)$ allows the action of adding an edge from role r_1 to r_2. The construct *addPrivilege* is a grammatical connective and consequently - as mentioned above - the set P is infinite despite the fact that the sets of users, roles and user privileges are all finite. In the existing literature the number of administrative levels (in other words, the number of nestings of the *addPrivilege* connective) is sometimes restricted to one [11] or to two levels [15]. We agree that administrative privileges with multiple levels of administration might not be useful in some implementations. However, here we take a general approach, and we let the administrators choose which administrative privileges to use. We should mention also that most existing models constrain the roles that can have administrative privileges, for example to prevent low roles obtaining privileges to change membership of higher roles [4]. We do not make choices with respect to such constraints.

In most existing literature on the administration of RBAC systems some constraints are assumed that restrict which roles should have administrative privileges over others. For example, in the original RBAC model, administrative privileges can only be assigned to a separate set of administrative roles. In the RCC model [5] a role can only assign a privilege if the role itself has that privilege. In the RHA model [4] a role only has the privilege to administer other roles that are in its administrative scope. We do not exclude any of these choices and we assume that, in principle, any role can be assigned any administrative privilege. In the sequel, we focus on the inheritance of administrative privileges.

3.1 Extended Privilege Inheritance

An RBAC state is denoted by a triple (UA, RH, PA), containing the user-role assignments, the edges between roles and the privilege assignments to roles. A well-known feature of RBAC is *privilege inheritance* [11], by which a role has the privileges to which it is explicitly assigned, and additionally, the privileges of lower roles.

Definition 3.2 (Standard Privilege Inheritance) *Let (UA, RH, PA) denote an RBAC state and let $\geqslant$ denote the reflexive transitive closure of RH, we say that a role r has the privilege p, denoted by $r \rightsquigarrow p$, iff*

$$r \geqslant r' \text{ and } (r', p) \in PA \text{ for some } r' \in R.$$

We argue that the standard privilege inheritance is inadequate for administrative privileges. Take for example a role r with the privilege to add an edge e from r_2 to r_3. The role r does not have the privilege to add an edge from r_2 to any role below r_3, nor the privilege to add an edge from any role above r_2 to r_3. However, from a security point of view this makes no sense, because with edge e in place there would be anyway a path to roles below r_3, or a path from roles above r_2. The standard RBAC privilege inheritance however does not capture this, and treats administrative privileges like ordinary user privileges. In Figure 2 we show the six different cases where administrative privileges yield weaker administrative privileges. For example, in Figure 2a, the (administrative) privilege to assign user u to role r_1 (the dashed edge) is stronger than the privilege to assign user u to role r_2. In Figure 2b, the privilege to add an edge between r_1 and r_2 (the dashed arrow) is stronger than the privilege to add only user u to role r_2. And so on. We formalize this ordering by the following definition.

Definition 3.3 (Privilege Ordering) *Let (UA, RH, PA) be an RBAC state, let p, p_1, p_2 be privileges in P, let Q be the subset of user privileges in P, and let r_1, r_2, r_3, r_4 be roles in R. We define the relation $\rightarrow$ as the smallest relation satisfying:*

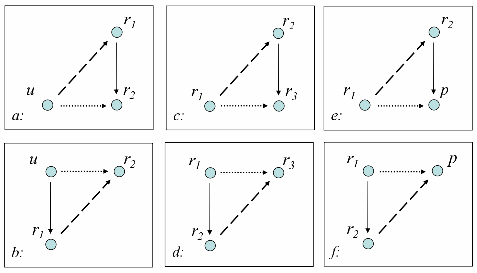

Figure 2: The right to add the dashed edge is stronger than the right to add the dotted edge.

1. *$p \to p$, if $p \in Q$*
2. *$addUser(u, r_1) \to addUser(u, r_2)$, if $r_1 \geqslant r_2$*
3. *$addEdge(r_1, r_2) \to addUser(u, r_3)$, if $r_2 \geqslant r_3$ and $(u, r_1) \in UA$*
4. *$addEdge(r_2, r_3) \to addEdge(r_1, r_4)$, if $r_1 \geqslant r_2$ and $r_3 \geqslant r_4$*
5. *$addEdge(r_2, r_3) \to addPrivilege(r_1, p_2)$, if $r_1 \geqslant r_2$, $r_3 \geqslant r_4$, $(r_4, p_1) \in PA$ and $p_1 \to p_2$*
6. *$addPrivilege(r_2, p_1) \to addPrivilege(r_1, p_2)$, if $r_1 \geqslant r_2$ and $p_1 \to p_2$*

The ordering $\to$ is both reflexive and transitive.

The ordering of privileges yields an extension of the standard privilege inheritance relation.

Definition 3.4 (Extended Privilege Inheritance) *Let (UA, RH, PA) be an RBAC state, let r be a role in R and p be a role in P, and let $\rightsquigarrow$ denote the* standard *privilege inheritance, reported in Definition 3.2. We say that the extended privilege inheritance $r \rightsquigarrow^* p$ holds iff*

$$r \rightsquigarrow p' \text{ and } p' \to p, \text{ for some } p' \in P.$$

The extended privilege inheritance relation is useful because it allows users, with administrative privileges, to be implicitly authorized for weaker administrative privileges. Thereby, it gives administrative users the possibility to perform safer administrative operations than the ones originally allowed. We now give a practical example of its usage.

Example 3.5 (Visiting Researcher) *Charlie, the security administrator, gives the staff the privilege to add visiting researchers to the staff role. There is also a role below staff called wifi, with the privilege to use the wireless network. Alice is a visiting researcher and Bob is a member of the staff. Alice only needs access to the wifi network, so Bob would like Alice to use the wifi role. Charlie (who*

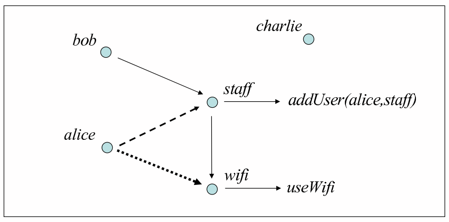

Figure 3: A practical example of the use of the extended inheritance relation.

just left) did not provide this privilege explicitly to the staff. This scenario is illustrated in Figure 3.

In the standard RBAC model, Bob can only assign Alice to the staff role. Given the fact that Alice only needs wifi access, Bob urges Alice to apply the principle of least privilege, and to activate only the wifi role. However, Bob can only hope that Alice does so. With the extended privilege inheritance relation Bob can assign Alice to the wifi role because *of his privilege to add users to the staff role. In a way, instead of preaching the principle of least privilege to Alice, Bob applies it for her.*

3.2 Tractability

Now we address a practical issue. We prove that the extended privilege inheritance relation (Definition 3.4) 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 below). The proof also indicates how a decision algorithm, deciding which privileges are to be given to which roles, can be implemented at an RBAC security monitor.

First notice that since RH and PA are finite sets, the standard privilege inheritance $\leadsto$ is decidable. To show how to decide whether $r \leadsto^* p$, we first prove that there is an algorithm that can decide, whether $p \to q$, for any p, q in P.

Lemma 3.6 (Decidability of the Ordering Relation) *Given an RBAC state S, and two privileges p, q, it is decidable whether $p \to q$.*

Proof 1 *The proof is by structural induction over q.*

The base cases are when q is not of the form addPrivilege$(.,.)$. We show that for the three base cases $p \to q$ is decidable:

- *Either q is a user privilege from Q. In this case $p \to q$ holds only when $p = q$ (see rule (1) in Definition 3.4).*
- *Or q is of the form addUser$(.,.)$ in which case only rule (2) needs to be checked, which has finite premises.*
- *Or q is of the form addEdge$(.,.)$, in which case the rules (2) and (3) of Definition 3.3 need to be checked. Both have finite premises.*

For the induction step, suppose that q is $addPrivilege(r', q')$, for some role r' and privilege q'. Now, $p \to q$ can only hold if either p is of the form $addEdge(.,.)$ and the premises of rule (5) holds, or p is of the form $addPrivilege(.,.)$ and the premises of rule (6) holds. In both cases, the premises are decidable, either because they are finite, or because the induction hypothesis is applicable (in $p' \to q'$, q' is structurally smaller than q, regardless of p').

Theorem 3.7 (Decidability of Extended Privilege Inheritance)
Given an RBAC state, a role r and a privilege p in P, there is an algorithm to determine whether $r \rightsquigarrow^* p$.

Proof 2 *The standard privilege inheritance $\rightsquigarrow$ is decidable, yielding a finite set of privileges p' inherited by r. Now for each privilege p' we need to check whether $p' \to p$. This was shown to be decidable in the previous lemma.*

We now give an example of how the above described procedure can be used in practice.

Example 3.8 *Consider Example 3.5 again.*
Can Bob assign Alice to the wifi role? We have to check that the role staff inherits the privilege $addUser(alice, wifi)$. Using the first part of Definition 3.4, one finds that the staff role has the privilege $addUser(alice, staff)$. Now we should decide whether

$$addUser(alice, staff) \to addUser(alice, wifi).$$

This follows trivially from the first rule of Definition 3.3.

To give a more involved example, suppose that the system administrator Charlie has the privilege $addPrivilege(staff, addUser(alice, staff))$. Can Charlie also give the staff role the privilege $addUser(alice, wifi)$? We have to check whether

$$addPrivilege(staff, addUser(alice, staff)) \to \\ addPrivilege(staff, addUser(alice, wifi)).$$

This is indeed the case by using rule (6) first, and then rule (2).

Now, for the sake of exposition, let us remove the edge between the staff and the wifi role. Let us show how to determine that the previous relation does not hold: Only rule (6) applies, in which case we must decide whether $addUser(alice, staff) \to addUser(alice, wifi)$. This is a base case of the induction described in the proof of Lemma 3.6: 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. However, in some cases the set of all privileges p' weaker than a given privilege p, is infinite. For the interested reader, we give an example of this in the appendix.

4 Related Work

The problem of administration of an RBAC system was first addressed by Sandhu et al. [11]. Later, numerous articles have been published extending or improving the administration model proposed there [2, 3, 4, 5, 10, 13, 14, 15]. We discuss some of them.

Barka et al. [2] 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 [15] defines a cascaded delegation. This form of delegation is also expressible in our grammar. In the PDBM model, however, each delegation requires the addition of a separate role, whereas, in our model the privileges for delegations are assigned to roles just as 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. [14] 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. [1] use a general constraint language to specify constraints on who can receive certain delegations. A more complex issue (another type of constraint) is the transfer of delegations [3]. Here the delegator looses the right it is delegating. Such delegations may be useful in practice, and we are interested to see how they can be implemented in our model.

Role-based trust management systems [6, 7, 8, 12] and distributed certificate systems, such as SDSI [9], are related lines of research. In these systems, a number of agents exchange security statements. Specifically, agents may make hierarchies similar to those in RBAC, simply by uttering certain security statements. In such models it is often assumed that users are free to utter security statements, while the focus is on wether to trust such statements (typically by some trust calculation by the receiver). In the RBAC setting however this assumption is very inappropriate. Statements changing the RBAC hierarchy should not be uttered by users, unless they have the explicit privilege to do so. Despite this difference, our result does apply also to role-based trust management models. The extended privilege inheritance relation would then correspond to the notion of *refinement* of policies or trust statements.

5 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. Concretely, our contribution is an extension of the standard RBAC privilege inheritance relation. We defined an ordering on administrative privileges, that enabled us to extend the standard privilege inheritance relation in the natural way. This means that if a user has a particular administrative privilege, then she is also implicitly authorized for weaker administrative privileges. We showed that this relation is tractable. Our extension can be seen as an application of the *principle of least privilege* at the level of administration.

Flexibility of management is an important requisite when deploying access control systems in practice. For example, discretionary access control systems are widely used because they are so flexible. RBAC on the other hand is less flexible, but it can be used to implement also mandatory security policy (for instance like in the Bell LaPadula model). To allow for a more flexible management, we extend the standard RBAC privilege inheritance. Basically, users with administrative privileges can also use *lesser* administrative privileges. Our extension can be seen as an application of the *principle of least privilege* to administration.

Of course, the definitions of the RBAC state can be chosen such that the extended and the standard inheritance relation yield the same result. In most cases however this is cumbersome. For example, the dotted edge in Figure 3 could also be explicitly added as an administrative privilege for the staff role. However, when the edge between the staff role and the wifi role is removed, then this makes no sense anymore. This kind of dependencies may complicate changing the role-hierarchy. Such repetitive definitions are not needed, when using the extended privilege inheritance.

Finally, a number of improvements and additions can be made. We do not express privileges such as $\forall r.addEdge(r', r))$, which may be useful in practice, but we believe that special care is needed to deal with quantifiers. Finally, as we do not make a particular choice regarding constraints on the administrative privileges, it would be interesting to investigate how our results can be combined with, for example, the work by Crampton and Loizou [4] or that of Bandmann et al. [1].

Acknowledgements

Marnix Dekker was funded by TNO and SenterNovem through the Dutch IOP Gencom project *PAW*. Jan Cederquist was funded by the Dutch NWO Account project and partially, while at SQIG-IT IST TU Lisbon, supported by the Portugese FEDER/FCT project QuantLog POCI/MAT/55796/2004.

References

[1] 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.

[2] 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. IEEE Computer Society Press, 2000.

[3] J. Crampton and H. Khambhammettu. Delegation in role-based access control. In D. Gollmann and A. Sabelfeld, editors, *Proc. of the European Symp. on Research in Computer Security (ESORICS)*, LNCS, pages 174–191. Springer, Berlin, 2006.

[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] D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-based Access Control*. Computer Security Series. Artech House, 2003.

[6] T. Jim. SD3: A trust management system with certified evaluation. In R. Needham and M. Abadi, editors, *Proc. of the Symp. on Security and Privacy (S&P)*, pages 106–115. IEEE Computer Society Press, 2001.

[7] N. Li, J. Mitchell, and W. Winsborough. Design of a role-based trust-management framework. In M. Abadi and S. M. Bellovin, editors, *Proc. of the Symp. on Security and Privacy (S&P)*, pages 114–130. IEEE Computer Society Press, 2002.

[8] N. Li, W. H. Winsborough, and J. C. Mitchell. Distributed credential chain discovery in trust management: extended abstract. In P. Samarati, editor, *Proc. of the Conf. on Computer and Communications Security (CCS)*, pages 156–165. ACM Press, 2001.

[9] R. L. Rivest and B. Lampson. SDSI – A simple distributed security infrastructure. Presented at CRYPTO'96 Rump session, 1996.

[10] 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.

[11] 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.

[12] R. Tamassia, D. Yao, and W. H. Winsborough. Role-based cascaded delegation. In T. Jaeger and E. Ferrari, editors, *Proc. of the Symp. on Access Control Models and Technologies (SACMAT)*, pages 146–155. ACM Press, 2004.

[13] 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.

[14] 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.

[15] 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.

A Infinitely many weaker privileges

Consider a state where $(r_2, \mathit{addEdge}(r_1, r_2)) \in \mathit{PA}$. Suppose now we are interested in finding all the privileges weaker than $\mathit{addEdge}(r_1, r_2)$. The first weaker privilege we discover by applying rule (5) in definition 3.3:

$$\mathit{addPrivilege}(r_1, \mathit{addEdge}(r_1, r_2)).$$

Using this result in rule (6), we find another weaker privilege,

$$\mathit{addPrivilege}(r_1, \mathit{addPrivilege}(r_1, \mathit{addEdge}(r_1, r_2))),$$

and we can use this again in rule (6), and so on.

Note that the outer nesting in the last term is in a sense redundant. Instead of assigning the privilege $\mathit{addEdge}(r_1, r_2)$ to r_1, one assigns the privilege to do so, to r_1. This only requires the users in role r_1 to perform another administrative step: The extra nesting is useless. It seems that we can stop after n applications of rule (6), where n is the length of the longest chain in RH, but we do not make this observation more formal here.