Security Analysis in Role-Based Access Control

NINGHUI LI
Purdue University
and
MAHESH V. TRIPUNITARA
Motorola Labs

The administration of large role-based access control (RBAC) systems is a challenging problem. In order to administer such systems, decentralization of administration tasks by the use of delegation is an effective approach. While the use of delegation greatly enhances flexibility and scalability, it may reduce the control that an organization has over its resources, thereby diminishing a major advantage RBAC has over discretionary access control (DAC). We propose to use security analysis techniques to maintain desirable security properties while delegating administrative privileges. We give a precise definition of a family of security analysis problems in RBAC, which is more general than safety analysis that is studied in the literature. We show that two classes of problems in the family can be reduced to similar analysis in the RT[$\leftarrow, \cap$] role-based trust-management language, thereby establishing an interesting relationship between RBAC and the RT framework. The reduction gives efficient algorithms for answering most kinds of queries in these two classes and establishes the complexity bounds for the intractable cases.

Categories and Subject Descriptors: K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; D.4.6 [**Operating Systems**]: Security and Protection—*Access Controls*

General Terms: Security, Theory, Languages

Additional Key Words and Phrases: Role-based access control, role-based administration, delegation, trust management

1. INTRODUCTION

The administration of large role-based access control (RBAC) systems is a challenging problem. A case study carried out with Dresdner Bank, a major

Most of this work was performed while the second author was a Ph.D. candidate at Purdue University. A preliminary verion of this paper appears in the *Proceedings of the ACM Symposium on Access Control, Models and Technologies* (SACMAT) [Li and Tripunitara 2004].
Authors' addresses: Ninghui Li, CERIAS, Purdue University, 656 Oval Drive, West Lafayette, IN 47907; email: ninghui@cs.purdue.edu; Mahesh V. Tripunitara, Motorola Labs, IL 02, 2712, 1301 E. Algonquin Road, Schaumburg, IL 60196; email: tripunit@motorola.com.

European bank, resulted in an RBAC system that has around 40,000 users and 1300 roles [Schaad et al. 2001]. In systems of such size, it is impossible for a single system security officer (SSO) to administer the entire system. In recent years, several administrative models for RBAC have been proposed, e.g., ARBAC97 [Sandhu et al. 1999], ARABCRA02 [Oh and Sandhu 2002], and CL03 (Crampton and Loizou) [Crampton and Loizou 2003]. In all these models, delegation is used to decentralize the administration tasks.

A major advantage that RBAC has over discretionary access control (DAC) is that if an organization uses RBAC as its access-control model, then the organization (represented by the SSO in the system) has central control over its resources. This is different from DAC, in which the creator of a resource determines who can access the resource. In most organizations, even when a resource is created by an employee, the resource is still owned by the organization and the organization wants some level of control over how the resource is to be shared. In most administrative models for RBAC, the SSO delegates to other users the authority to assign users to certain roles (thereby granting those users certain access permissions), to remove users from certain roles (thereby revoking certain permissions those users have), etc. While the use of delegation in the administration of an RBAC system greatly enhances flexibility and scalability, it may reduce the control that the organization has over its resources, thereby diminishing a major advantage RBAC has over DAC. As delegation gives a certain degree of control to a user that may be only partially trusted, a natural security concern is whether the organization nonetheless has some guarantees about who can access its resources. To the best of our knowledge, the effect of delegation on the persistence of security properties in RBAC has not been considered in the literature as such.

In this paper, we propose to use security analysis techniques [Li et al. 2005] to maintain desirable security properties while delegating administrative privileges. In security analysis, one views an access-control system as a state-transition system. In an RBAC system, state changes occur via administrative operations. Security analysis techniques answer questions such as whether an undesirable state is reachable and whether every reachable state satisfies some safety or availability properties. Examples of undesirable states are a state in which an untrusted user gets access and a state in which a user who is entitled to an access permission does not get it.

Our contributions in this paper are as follows.

- We give a precise definition of a family of security analysis problems in RBAC. In this family, we consider queries that are more general than queries that are considered in safety analysis [Harrison et al. 1976; Koch et al. 2002a; Lipton and Snyder 1977; Sandhu 1988].
- We show that two classes of the security analysis problems in RBAC can be reduced to similar ones in RT[$\leftarrow, \cap$], a role-based trust-management language for which security analysis has been studied [Li et al. 2005]. The reduction gives efficient algorithms for answering most kinds of queries in

these two classes and establishes the complexity bounds for the intractable cases.

Our contributions are significant in that our work presents a way to capture and represent a large class of security properties of interest in complex RBAC systems, such as the one discussed by Schaad et al. [2001]. Our work also shows how several kinds of these security properties can be efficiently verified. Our establishment of complexity bounds for the intractable cases gives us a clear understanding of the difficulty of the problems so that future work can develop efficient heuristics. In Section 2.2, we discuss how security analysis is used in RBAC systems, which further demonstrates the significance of our contributions.

The rest of this paper is organized as follows. In Section 2, we define a family of security analysis problems in RBAC and summarize our main results. We give an overview of the results for security analysis in RT[$\twoheadleftarrow, \cap$] in Section 3. We present the reduction from security analysis in RBAC to that in RT[$\twoheadleftarrow, \cap$] in Section 4. Related work is discussed Section 5. We conclude with Section 6. An appendix contains proofs not included in the main body.

2. PROBLEM DEFINITION AND MAIN RESULTS

In Li et al. [2005], an abstract version of security analysis is defined in the context of trust management. In this section, we restate the definition in the context of general access-control schemes.

Definition 1. (*Access-Control Schemes*) An access-control scheme is modeled as a state-transition system $\langle \Gamma, Q, \vdash, \Psi \rangle$, in which Γ is a set of states, Q is a set of queries, Ψ is a set of state-change rules, and $\vdash : \Gamma \times Q \rightarrow \{true, false\}$ is called the entailment relation, determining whether a *query* is true or not in a given state. A *state*, $\gamma \in \Gamma$, contains all the information necessary for making access-control decisions at a given time. When a query, $q \in Q$, arises from an access request, $\gamma \vdash q$ means that the access corresponding to the request q is granted in the state γ, and $\gamma \nvdash q$ means that the access corresponding to q is not granted. One may also ask queries other than those corresponding to a specific request, e.g., whether every principal that has access to a resource is an employee of the organization. Such queries are useful for understanding the properties of a complex access-control system.

A state-change rule, $\psi \in \Psi$, determines how the access-control system changes state. Given two states γ and γ_1 and a state-change rule ψ, we write $\gamma \mapsto_\psi \gamma_1$ if the change from γ to γ_1 is allowed by ψ, and $\gamma \overset{*}{\mapsto}_\psi \gamma_1$ if a sequence of zero or more allowed state changes leads from γ to γ_1. If $\gamma \overset{*}{\mapsto}_\psi \gamma_1$, we say that γ_1 is ψ-*reachable* from γ, or simply γ_1 is *reachable*, when γ and ψ are clear from the context.

An example of an access-control scheme is the HRU scheme, that is derived from the work by Harrison et al. [1976]. The HRU scheme is based on the access-matrix model [Graham and Denning 1972; Lampson 1971]. We assume the existence of three countably infinite sets: $\mathcal{S}$, $\mathcal{O}$, and $\mathcal{A}$, which are the sets of

all possible subjects objects and access rights. We assume further that $\mathcal{S} \subseteq \mathcal{O}$. In the HRU scheme:

- Γ is the set of all possible access matrices. Formally, each $\gamma \in \Gamma$ is identified by three finite sets, $S_\gamma \subset \mathcal{S}$, $O_\gamma \subset \mathcal{O}$, and $A_\gamma \subset \mathcal{A}$, and a function $M_\gamma[\,]$: $S_\gamma \times O_\gamma \rightarrow 2^{A_\gamma}$, where $M_\gamma[s, o]$ gives the set of rights s has over o. An example of a state, γ, is one in which $S_\gamma = \{\mathsf{Admin}\}$, $O_\gamma = \{\mathsf{employeeData}\} \cup S_\gamma$, $A_\gamma = \{own, read\}$, and $M_\gamma[\mathsf{Admin}, \mathsf{Admin}] = \emptyset$, and $M_\gamma[\mathsf{Admin}, \mathsf{employeeData}] = \{own, read\}$. In this state, two objects exist, of which one is a subject, and the system is associated with the two rights, *own* and *read*.
- $\mathcal{Q}$ is the set of all queries of the form: $a \in [s, o]$, where $a \in \mathcal{A}$ is a right, $s \in \mathcal{S}$ is a subject, and $o \in \mathcal{O}$ is an object. This query asks whether the right a exists in the cell corresponding to subject s and object o.
- The entailment relation is defined as follows: $\gamma \vdash a \in [s, o]$ if, and only if, $s \in S_\gamma$, $o \in O_\gamma$, and $a \in M_\gamma[s, o]$. For example, let the query q_1 be $read \in M[\mathsf{Admin}, \mathsf{employeeData}]$. and the query q_2 be $own \in M[\mathsf{Admin}, \mathsf{Admin}]$ Then, for the state, γ, discussed above, $\gamma \vdash q_1$ and $\gamma \vdash q_2$.
- Each state-transition rule ψ is given by a set of commands. Given ψ, the change from γ to γ_1 is allowed if there exists command in ψ such that the execution of the command in the state γ results in the state γ_1. An example of ψ is the following set of commands.

command createObject(s, o)	*command grant_a*(s, s', o)
create object o	*if own* $\in [s, o]$
enter own into $[s, o]$	*enter a into* $[s', o]$

The set of queries is not explicitly specified in Harrison et al. [1976]. It is conceivable to consider other classes of queries, e.g., comparing the set of all subjects that have a given right over a given object with another set of subjects. In our framework, HRU with different classes of queries can be viewed as different schemes.

Definition 2. (*Security Analysis in an Abstract Setting*) Given an access-control scheme $\langle \Gamma, \mathcal{Q}, \vdash, \Psi \rangle$, a security analysis instance takes the form $\langle \gamma, q, \psi, \Pi \rangle$, where $\gamma \in \Gamma$ is a state, $q \in \mathcal{Q}$ is a query, $\psi \in \Psi$ is a state-change rule, and $\Pi \in \{\exists, \forall\}$ is a quantifier. An instance $\langle \gamma, q, \psi, \exists \rangle$ asks whether there exists γ_1 such that $\gamma \overset{*}{\mapsto}_\psi \gamma_1$ and $\gamma_1 \vdash q$. When the answer is affirmative, we say q is *possible* (given γ and ψ). An instance $\langle \gamma, q, \psi, \forall \rangle$ asks whether for every γ_1 such that $\gamma \overset{*}{\mapsto}_\psi \gamma_1$, $\gamma_1 \vdash q$. If so, we say q is *necessary* (given γ and ψ).

For our example HRU scheme from above, adopt γ as the start state. In γ, there is only one subject (namely, Admin) and the access matrix is empty. The system is associated with the two rights, *own* and r. Let the query q be $r \in M[\mathsf{Alice}, \mathsf{employeeData}]$ for $\mathsf{Alice} \in \mathcal{S}$ and $\mathsf{employeeData} \in \mathcal{O}$. Let the state-change rule ψ be the set of two commands *createObject* and *grant_r*. Then, the security analysis instance $\langle \gamma, q, \psi, \exists \rangle$ is true. The reason is that although in the start state γ, Alice does not have the r right over the object employeeData, there exists a reachable state from γ in which she has such access. The security

analysis instance $\langle \gamma, q, \psi, \forall \rangle$ is false, as there exists at least one state reachable from γ (γ itself) that does not entail the query.

Security analysis generalizes safety analysis. As we discuss in the following section, with security analysis we can study not only safety, but also several other interesting properties, such as availability and mutual exclusion.

2.1 A Family of Security Analysis Problems in Role-Based Access Control

We now define a family of security analysis problems in the context of RBAC by specifying Γ, Q, and $\vdash$, while leaving Ψ abstract. By considering different possibilities for Ψ, one obtains different classes of RBAC security analysis problems in this family. We consider two specific instances of Ψ in Sections 2.3 and 2.4.

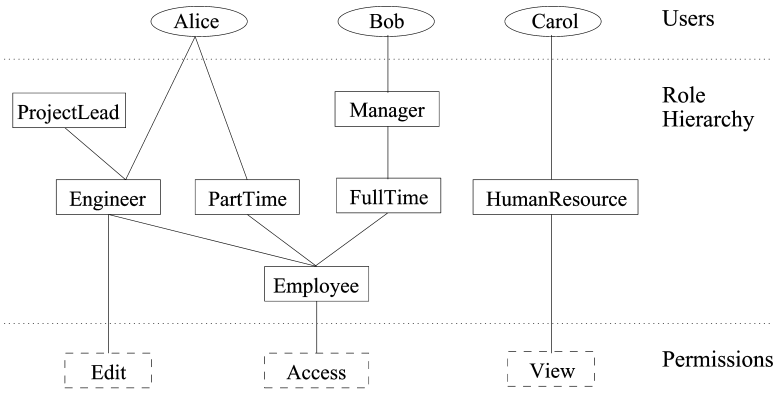
We assume a basic level of familiarity with RBAC; readers are referred to Ferraiolo et al. [2001] and Sandhu et al. [1996] for an introduction to RBAC. We assume that there are three countable sets: $\mathcal{U}$ (the set of all possible users), $\mathcal{R}$ (the set of all possible roles), and $\mathcal{P}$ (the set of all possible permissions). The family of analysis problems is given by specializing the analysis problem defined in Definition 2 to consider access-control schemes that have Γ, Q, and $\vdash$ specified as follows.

2.1.1 *States* (Γ). Γ is the set of all RBAC states. An RBAC state, γ, is a 3-tuple $\langle UA, PA, RH \rangle$, in which the user assignment relation $UA \subseteq \mathcal{U} \times \mathcal{R}$ associates users with roles, the permission assignment relation $PA \subseteq \mathcal{P} \times \mathcal{R}$ associates permissions with roles, and the role hierarchy relation $RH \subseteq \mathcal{R} \times \mathcal{R}$ is a partial order among roles in $\mathcal{R}$. We denote the partial order by $\succeq$. $r_1 \succeq r_2$ which means that every user who is a member of r_1 is also a member of r_2 and every permission that is associated with r_2 is also associated with r_1.

Example 1. Figure 1 is an example of an RBAC state. It reflects an organization that has engineers and whose human-resource needs are outsourced (i.e., human-resource personnel are not employees). Everyone in the organization is an employee and, therefore, a member of the role Employee. Some of the employees are full-time (members of the role FullTime), and the others are part-time (members of the role PartTime). All managers are full-time employees. All employees have access to the office and, therefore, have the permission Access. Engineers may edit code (have the permission Edit) and human-resource personnel may view employee details (have the permission View).

We now discuss some example members of UA, PA, and RH. The user Alice is an engineer who is a part-time employee. Therefore, (Alice, Engineer) and (Alice, PartTime) are members of UA. All employees have access to the office and, therefore, (Access, Employee) is a member of PA. Project leads are engineers and, therefore, (ProjectLead, Engineer) is a member of RH (i.e., ProjectLead $\succeq$ Engineer).

Given a state $\gamma = \langle UA, PA, RH \rangle$, every role has a set of users who are members of that role and every permission is associated with a set of users who have that permission. We formalize this by having every state γ define a function $\mathsf{users}_\gamma : \mathcal{R} \cup \mathcal{P} \to 2^U$, as follows. For any $r \in \mathcal{R}$ and $u \in \mathcal{U}$, $u \in \mathsf{users}_\gamma[r]$ if, and only if, either $(u,r) \in UA$ or there exists r_1 such that $r_1 \succeq r$ and $(u,r_1) \in UA$.



RH = { (Engineer, Employee) , (FullTime, Employee),
(PartTime, Employee) , (ProjectLead, Engineer),
(Manager, FullTime) }.
PA = { (Access, Employee) , (View, HumanResource),
(Edit, Engineer) }.
UA = { (Alice, PartTime) , (Alice, Engineer),
(Bob, Manager) , (Carol, HumanResource) }.

Fig. 1. An example RBAC state with a role hierarchy, users, and permissions. Roles are shown in solid boxes, permissions in dashed boxes, and users in ovals. A line segment represents a role–role relationship, the assignment of a permission to a role, or the assignment of a user to a role.

For any $p \in \mathcal{P}$ and $u \in U$, $u \in \mathsf{users}_\gamma[p]$ if, and only if, there exists r_1 such that $(p, r_1) \in PA$ and $u \in \mathsf{users}_\gamma[r_1]$. Note that the effect of permission propagation through the role hierarchy is already taken into consideration by the definition of $\mathsf{users}_\gamma[r_1]$.

Example 2. Let the RBAC state shown in Figure 1 be γ. Then, for the role Engineer, $\mathsf{users}_\gamma[\mathsf{Engineer}] = \{\mathsf{Alice}\}$. Similarly, for the permission Access, $\mathsf{users}_\gamma[\mathsf{Access}] = \{\mathsf{Alice}, \mathsf{Bob}\}$.

2.1.2 *Queries* (Q). The purpose of a query is to encode some property of a state that is of interest. For this, we introduce the notion of *user sets* by extending our definition of the function users_γ. The intuition is as follows. Given a state, a user set evaluates to a set of users. A query encodes a comparison of user sets, which evaluates (in the entailment of a query) to a comparison of two sets of users. As we demonstrate, such a representation for a query is quite powerful; indeed, we are able to capture several properties of interest. The reason is that properties regarding users, roles, and permissions can all be captured using user sets.

A query q has the form $s_1 \sqsupseteq s_2$, where $s_1, s_2 \in S$, and S is the set of all *user sets*, defined to be the least set satisfying the following conditions: (1) $\mathcal{R} \cup \mathcal{P} \subseteq S$, i.e., every role r and every permission p is a user set; (2) $\{u_1, u_2, \ldots, u_k\} \in S$, where $k \geq 0$ and $u_i \in U$ for $1 \leq i \leq k$, i.e., a finite set of users is a user set; and (3) $s_1 \cup s_2, s_1 \cap s_2, (s_1) \in S$, where $s_1, s_2 \in S$, i.e., the set of all user sets is closed with

respect to union, intersection, and paranthesization. We extend the function users_γ in a straightforward way to give a valuation for all user sets. The extended function users_γ: $S \to 2^U$ is defined as follows: $\mathsf{users}_\gamma[\{u_1, u_2, \ldots, u_k\}] = \{u_1, u_2, \cdots, u_k\}$, $\mathsf{users}_\gamma[(s)] = \mathsf{users}_\gamma[s]$, $\mathsf{users}_\gamma[s_1 \cup s_2] = \mathsf{users}_\gamma[s_1] \cup \mathsf{users}_\gamma[s_2]$, and $\mathsf{users}_\gamma[s_1 \cap s_2] = \mathsf{users}_\gamma[s_1] \cap \mathsf{users}_\gamma[s_2]$. We say a query $s_1 \sqsupseteq s_2$ is *semistatic* if one of s_1, s_2 can be evaluated independent of the state, i.e., no role or permission appears in it. The reason we distinguish semistatic queries is that (as we assert in Sections 4.1 and 4.2) a security analysis instance involving only such queries can be solved efficiently.

2.1.3 *Entailment* ($\vdash$). Given a state γ and a query $s_1 \sqsupseteq s_2$, $\gamma \vdash s_1 \sqsupseteq s_2$ if, and only if, $\mathsf{users}_\gamma[s_1] \supseteq \mathsf{users}_\gamma[s_2]$.

Example 3. Continuing from the previous examples, an example of a query, q, is $\mathsf{FullTime} \cap \mathsf{Access} \sqsupseteq \{\mathsf{Alice}\}$, for the role FullTime, the permission Access and the user Alice. This query is semistatic; the user set {Alice} can be evaluated (to itself) independent of the state.

The query q asks whether Alice is a full-time employee that has access to the office. To find out whether γ entails q or not, we evaluate q as follows. We evaluate the user set FullTime to the set of users {Bob}. We evaluate the user set Access to the set of users {Alice, Bob}. We intersect the two sets of users to obtain the set of users {Bob}. The user set {Alice} does not need further evaluation; it is already a set of users. We now check whether the set of users {Alice} is a subset of the set of users {Bob} and determine that $\gamma \vdash q$. If another query q' is $\mathsf{Edit} \sqsupseteq \mathsf{ProjectLead}$ (i.e., whether project leads can edit code), then $\gamma \vdash q'$.

The state of an RBAC system changes when a modification is made to a component of $\langle UA, PA, RH \rangle$. For example, a user may be assigned to a role or a role hierarchy relationship may be added. In existing RBAC models, both constraints and administrative models affect state changes in an RBAC system. For example, a constraint may declare that roles r_1 and r_2 are mutually exclusive, meaning that no user can be a member of both roles. If a user u is a member of r_1 in a state, then the state is not allowed to change to a state in which u is a member of r_2 as well. An *administrative model* includes administrative relations that dictates who has the authority to change the various components of an RBAC state and what are the requirements these changes have to satisfy. Thus, in RBAC security analysis, a state-change rule may include constraints, administrative relations, and possibly other information.

In this section, we leave the state-change rule abstract for the following reasons. First, there are several competing proposals for constraint languages [Ahn and Sandhu 2000; Jaeger and Tidswell 2001; Crampton 2003] and for administrative models in RBAC [Sandhu et al. 1999; Oh and Sandhu 2002; Crampton and Loizou 2003; Ferraiolo et al. 2003]; a consensus has not been reached within the community. Furthermore, RBAC is used in diverse applications. It is conceivable that different applications would use different classes of constraints and/or administrative models; therefore, different classes of problems in this family are of interest.

Given a state γ and a state-change rule ψ, one can ask the following questions using security analysis.

- *Simple Safety* : is $s \sqsupseteq \{u\}$ possible? This asks whether there exists a reachable state in which the user set s includes the (presumably untrusted) user u. A "no" answer means that the system is safe.
- *Simple Availability* : is $s \sqsupseteq \{u\}$ necessary? This asks whether in every reachable state, the (presumably trusted) user u is always included in the user set s. A "yes" answer means that the resources associated with the user set s are always available to the user u.
- *Bounded Safety* : is $\{u_1, u_2, \ldots, u_n\} \sqsupseteq s$ necessary? This asks whether in every reachable state, the user set s is bounded by the set of users $\{u_1, u_2, \ldots, u_n\}$. A "yes" answer means that the system is safe. A special case of bounded safety is *mutual exclusion*, which asks: is $\emptyset \sqsupseteq (s_1 \cap s_2)$ necessary? This asks whether in every reachable state, no user is a member of both user sets s_1 and s_2. A "yes" answer means that the two user sets are mutually exclusive.
- *Liveness* : is $\emptyset \sqsupseteq s$ possible? This asks whether the user set s always has at least one user. A "no" answer means that the liveness of the resources associated with s holds in the system.
- *Containment* : is $s_1 \sqsupseteq s_2$ necessary? This asks whether in every reachable state, every user in the user set s_2 is in the user set s_1. Containment can be used to express a safety property, in which case a "yes" answer means that the safety property holds. An example of containment for the RBAC state in Figure 1 and some state-change rule is: "is Employee $\sqsupseteq$ Access necessary?," for the role Employee and the permission Access. This asks whether in every reachable state, every user who has the permission Access (i.e., has access to the office) is a member of the role Employee (i.e., is an employee). A "yes" answer means that our desired safety property holds.
  Containment can also express availability properties. For example, "is Access $\sqsupseteq$ Employee necessary?" asks whether the permission Access (i.e., access to the office) is always available to members of the role Employee (i.e., employees). A "yes" answer means that the availability property holds.

We point out that that all the above properties (except for containment) use semistatic queries and, therefore, as we mention in the context of queries in this section, we can efficiently determine whether those properties are satisfied.

2.2 Usage of RBAC Security Analysis

In an RBAC security analysis instance $\langle \gamma, q, \psi, \Pi \rangle$, the state γ fully determines who can access which resources. In addition to administrative policy information, the state-change rule ψ also contains information about which users are trusted. In any access control system, there are *trusted users*; these are users who have the authority to take the system to a state that violates security requirements, but are trusted not to do so. An SSO is an example of a trusted user.

Security analysis provides a means to ensure that security requirements (such as safety and availability) are always met, as long as users identified as

trusted behave according to the usage patterns discussed in this section. In other words, security analysis helps ensure that the security of the system does not depend on users other than those that are trusted.

Each security requirement is formalized as a security analysis instance, together with an answer that is acceptable for secure operation. For example, in the context of the RBAC system whose state in shown in Figure 1, a security requirement may be that only employees may access the office. This can be formalized as an instance $\langle \gamma, q, \psi, \forall \rangle$, where γ is the current state, q is Employee $\sqsupseteq$ Access, and ψ specifies administrative policy information. The rule ψ should precisely capture the capabilities of users that are not trusted. In other words, any change that could be made by such users should be allowed by ψ. The rule ψ could restrict the changes that trusted users can make, because these are trusted not to make a change without verifying that desirable security properties are maintained subsequent to the change. For the example discussed above, the acceptable answer is "yes," as we want to ensure that everyone who has the permission Access is an employee. The goal is to ensure that such a security requirement is always satisfied.

Suppose that the system starts in a state γ such that the answer to $\langle \gamma, q, \psi, \forall \rangle$ is "yes." Further, suppose a trusted user (such as the SSO) attempts to make a change that is not allowed by ψ, e.g., the SSO decides to grant certain administrative privileges to a user u. Before making the change, SSO performs security analysis $\langle \gamma', q, \psi', \forall \rangle$, where γ' and ψ' are resulted from the prospective change. Only if the answer is "yes," does the SSO actually make the change. The fact that ψ limits the SSO from making changes does not mean that we require that the SSO never make such changes. It reflects the requirement that the SSO perform security analysis and make only those changes that do not violate security properties.

This way, as long as trusted users are cooperating, the security of an access-control system is preserved. One can delegate administrative privileges to partially trusted users with the assurance that desirable security properties always hold. By using different ψ's, one can evaluate which sets of users are trusted for a given security property. In general, it is impossible to completely eliminate the need to trust people. However, security analysis enables one to ensure that the extent of this trust is well understood.

2.3 Assignment and Trusted Users (AATU)

In this paper, we present solutions to two classes of security analysis problems in RBAC. Both classes use variants of the URA97 component of the ARBAC97 administrative model for RBAC [Sandhu et al. 1999]. URA97 specifies how the *UA* relation may change.

The first class is called Assignment And Trusted Users (AATU), in which a state-change rule ψ has the form $\langle can\_assign, T \rangle$. The relation $can\_assign \subseteq R \times C \times 2^R$ determines who can assign users to roles and the preconditions these users have to satisfy. C is the set of conditions, which are expressions formed using roles, the two operators $\cap$ and $\cup$, and parentheses. $\langle r_a, c, rset \rangle \in can\_assign$ means that members of the role r_a can assign any user whose role

memberships satisfy the condition c, to any role $r \in rset$. For example, $\langle r_0, (r_1 \cup r_2) \cap r_3, \{r_4, r_5\}\rangle \in can\_assign$ means that a user that is a member of the role r_0 is allowed to assign a user that is a member of at least one of r_1 and r_2 and is also a member of r_3, to be a member of r_4 or r_5. $T \subseteq U$ is a set of trusted users; these users are assumed not to initiate any role assignment operation for the purpose of security analysis. The set T is allowed to be empty.

Definition 3. (*Assignment And Trusted Users – AATU*) The class AATU is given by parameterizing the family of RBAC analysis problems in Section 2.1 with the following set of state-change rules. Each state-change rule ψ has the form $\langle can\_assign, T\rangle$ such that a state change from $\gamma = \langle UA, PA, RH\rangle$ to $\gamma_1 = \langle UA_1, PA_1, RH_1\rangle$ is allowed by $\psi = \langle can\_assign, T\rangle$ if $PA = PA_1$, $RH = RH_1$, $UA_1 = UA \cup \{(u,r)\}$, where $(u,r) \notin UA$ and there exists $(r_a, c, rset) \in can\_assign$ such that $r \in rset$, u satisfies c, and $\mathsf{users}_\gamma[r_a] \not\subseteq T$ (i.e., there exists at least one user who is a member of the role r_a and is not in T, so that such a user can perform the assignment operation).

Example 4. In this example, we consider the question of whether a particular user, Alice, can become a ProjectLead given a system in AATU. In our example, we do not want Alice to become a ProjectLead unless the trusted administrator Carol is involved. We encode this question as a security analysis instance.

For the state, γ, shown in Figure 1 and discussed in the previous examples, a state-change rule, ψ, in the class AATU is $\langle can\_assign, T\rangle$, where

$$can\_assign = \{\langle \mathsf{Manager}, \mathsf{Engineer} \wedge \mathsf{FullTime}, \{\mathsf{ProjectLead}\}\rangle, \langle \mathsf{HumanResource}, true, \{\mathsf{FullTime}, \mathsf{PartTime}\}\rangle\}$$

$$T = \{\mathsf{Carol}\}$$

That is, ψ authorizes managers to assign a user to the role ProjectLead provided that the user is a member of the roles Engineer and FullTime. In addition, ψ authorizes anyone that is a member of the role HumanResource to assign users to the roles FullTime and PartTime. Setting T to {Carol} implies that we wish to analyze what kinds of states can be reached via changes made by users other than Carol.

Let q be the query $\mathsf{ProjectLead} \sqsupseteq \{\mathsf{Alice}\}$. Then, $\gamma \vdash q$. The analysis instance $\langle \gamma, q, \psi, \exists\rangle$ asks whether there exists a reachable state in which Alice is a project lead. The instance is false. This is because for Alice to become a member of ProjectLead, she would first need to be a full-time employee and only Carol can grant anyone membership to FullTime. As Carol is in T, she cannot initiate any operation. If we consider, instead, the state-change rule ψ', with the same *can_assign* as ψ from above, but with $T = \emptyset$, then the analysis instance $\langle \gamma, q, \psi', \exists\rangle$ is true.

2.3.1 *Main Results for AATU*

- If q is semistatic (see Section 2.1), then an AATU instance $\langle \gamma, q, \psi, \Pi\rangle$ can be answered efficiently, i.e., in time polynomial in the size of the instance.
- Answering general AATU instances $\langle \gamma, q, \psi, \forall\rangle$ is decidable, but intractable (**coNP**-complete).

2.4 Assignment and Revocation (AAR)

In this class, a state-change rule ψ has the form $\langle can\_assign, can\_revoke\rangle$, where can_assign is the same as in AATU, and $can\_revoke \subseteq R \times 2^R$ determines who can remove users from roles. That $\langle r_a, rset\rangle \in can\_revoke$ means that the members of role r_a can remove a user from a role $r \in rset$. No explicit set of trusted users is specified in AAR, unlike AATU. In AATU and AAR, the relations can_assign and can_revoke are fixed in ψ. This means that we are assuming that changes to these two relations are made only by trusted users.

Definition 4. (*Assignment And Revocation—AAR*) The class AAR is given by parameterizing the family of RBAC analysis problems in Section 2.1 with the following set of state-change rules. Each state-change rule ψ has the form $\langle can\_assign, can\_revoke\rangle$ such that a state-change from $\gamma = \langle UA, PA, RH\rangle$ to $\gamma_1 = \langle UA_1, PA_1, RH_1\rangle$ is allowed by $\psi = \langle can\_assign, can\_revoke\rangle$ if $PA = PA_1$, $RH = RH_1$, and either (1) $UA_1 = UA \cup \{(u,r)\}$ where $(u,r) \notin UA$ and there exists $(r_a, c, rset) \in can\_assign$ such that $r \in rset$, u satisfies c, and $\mathsf{users}_\gamma[r_a] \neq \emptyset$, i.e., the user u being assigned to r is not already a member of r and satisfies the precondition c, and there is at least one user that is a member of the role r_a that can perform the assignment operation; or (2) $UA_1 \cup (u,r) = UA$ where $(u,r) \notin UA_1$, and there exists $(r_a, rset) \in can\_revoke$ such that $r \in rset$ and $\mathsf{users}_\gamma[r_a] \neq \emptyset$, i.e., there exists at least one user in the role r_a that can revoke the user u's membership in the role r.

We assume that an AAR instance satisfies the following three properties. (1) The administrative roles are not affected by can_assign and can_revoke. The administrative roles are given by those that appear in the first component of any can_assign or can_revoke tuple. These roles should not appear in the last component of any can_assign or can_revoke tuple. This condition is easily satisfied in URA97, as it assumes the existence of a set of administrative roles that is disjoint from the set of normal roles. (2) If a role is an administrative role (i.e., appears as the first component of a can_assign or can_revoke tuple), then it has at least one user assigned to it. This is reasonable, as an administrative role with no members has no effect on the system's protection state. (3) If a can_assign tuple exists for a role, then a can_revoke tuple also exists for that role.

Example 5. In this example, we ask whether it is possible that only project leads have access to the office and whether Alice can ever edit code, both in the same AAR system. The former is an example of an availability question, while the latter is an example of a safety question. We encode both questions as security analysis instances.

For the state, γ, from Figure 1, an example of a state-change rule in AAR is $\psi = \langle can\_assign, can\_revoke\rangle$, where

$$\begin{aligned} can\_assign \;=\; & \{\langle \mathsf{Manager}, \mathsf{Engineer} \wedge \mathsf{FullTime}, \{\mathsf{ProjectLead}\}\rangle, \\ & \langle \mathsf{HumanResource}, true, \{\mathsf{FullTime}, \mathsf{PartTime}\}\rangle\} \end{aligned}$$

$$\begin{aligned} can\_revoke \;=\; & \{\langle \mathsf{Manager}, \{\mathsf{ProjectLead}, \mathsf{Engineer}\}\rangle, \\ & \langle \mathsf{HumanResource}, \{\mathsf{FullTime}, \mathsf{PartTime}\}\rangle\} \end{aligned}$$

We point out that the *can_assign* we use in this example is the same as the *can_assign* we use in Example 2.3. Then, if q is the query ProjectLead $\sqsupseteq$ Access (i.e., only project leads have access to the office), the AAR analysis instance $\langle \gamma, q, \psi, \exists \rangle$ is true. If q' is the query Edit $\sqsupseteq$ {Alice} (i.e., Alice can edit code), then the analysis instance $\langle \gamma, q', \psi, \forall \rangle$ is false.

2.4.1 *Main Results for AAR*

- If q is semistatic (see Section 2.1), then an AAR instance $\langle \gamma, q, \psi, \Pi \rangle$ can be answered efficiently, i.e., in time polynomial in the size of the instance.
- Answering general AAR instances $\langle \gamma, q, \psi, \forall \rangle$ is **coNP**-complete.

2.5 Discussion of the Definitions

Our specifications of *can_assign* and *can_revoke* are from URA97, which is one of the three components of ARBAC97 [Sandhu et al. 1999]. The state-change rules considered in AAR are similar to those in URA97, but they differ in the following two ways. (1) URA97 allows negation of roles to be used in a precondition; AAR does not allow this. (2), URA97 has separate administrative roles; AAR does not require the complete separation of administrative roles from ordinary roles. AATU differs from URA97 in two additional ways. (1) AATU does not have revocation rules. (2) AATU has a set of trusted users, which does not exist in URA97.

The other components of ARBAC97 are PRA97 and RRA97, for administering permission-role assignment/revocation and the role hierarchy, respectively. In this paper, we study the effect of decentralizing user-role assignment and revocation and assume that changes to the permission-role assignment relation and the role hierarchy are centralized, i.e, made only by trusted users. In other words, whoever is allowed to make changes to permission-role assignment and the role hierarchy will run the security analysis and only make changes that do not violate the security properties. The administration of the user-role relation is most likely to be delegated, as that is the component of an RBAC state that changes most frequently.

AATU and AAR represent two basic cases of security analysis in RBAC. Although we believe that they are useful, they are only the starting point. Many other more sophisticated cases of security analysis in RBAC remain open. For example, it is not clear how to deal with negative preconditions in role assignment and how to deal with constraints, such as mutually exclusive roles.

3. OVERVIEW OF SECURITY ANALYSIS IN RT[$\leftarrow$, $\cap$]

In Li et al. [2005], study security analysis in the context of the RT family of role-based trust-management languages [Li et al. 2002, 2003]. In particular, security analysis in RT[$\leftarrow$, $\cap$] and its sublanguages is studied. RT[$\leftarrow$, $\cap$] is a slightly simplified (yet expressively equivalent) version of the RT_0 language introduced in Li et al. [2003] (RT[$\leftarrow$, $\cap$] is called SRT in Li et al. [2005]). In this section, we summarize the results for security analysis in RT[$\leftarrow$, $\cap$]. We summarize the concepts from and results for RT[$\leftarrow$, $\cap$] so that we can leverage those results in the security analysis of the RBAC schemes that we consider in

Simple Member
syntax: $K.r \longleftarrow K_1$
meaning: $\mathsf{members}(K.r) \supseteq \{K_1\}$
LP clause: $m(K, r, K_1)$

Simple Inclusion
syntax: $K.r \longleftarrow K_1.r_1$
meaning: $\mathsf{members}(K.r) \supseteq \mathsf{members}(K_1.r_1)$
LP clause: $m(K, r, ?Z) :- m(K_1, r_1, ?Z)$

Linking Inclusion
syntax: $K.r \longleftarrow K.r_1.r_2$
meaning: $\mathsf{members}(K.r) \supseteq \bigcup_{K_1 \in K.r_1} \mathsf{members}(K_1.r_2)$
LP clause: $m(K, r, ?Z) :- m(K, r_1, ?Y),\ m(?Y, r_2, ?Z)$

Intersection Inclusion
syntax: $K.r \longleftarrow K_1.r_1 \cap K_2.r_2$
meaning: $\mathsf{members}(K.r) \supseteq \mathsf{members}(K_1.r_1) \cap \mathsf{members}(K_2.r_2)$
LP clause: $m(K, r, ?Z) :- m(K_1, r_1, ?Z),\ m(K_2, r_2, ?Z)$

Fig. 2. Statements in RT[$\twoheadleftarrow$, $\cap$]. There are four types of statements. For each type, we give the syntax, the intuitive meaning of the statement, and the LP (logic-programming) clause corresponding to the statement. The clause uses one ternary predicate m, where $m(K, r, K_1)$ means that K_1 is a member of the role $K.r$. Symbols that start with "?" represent logical variables.

this paper (AATU and AAR). In Section 4, we reduce security analysis in AATU and AAR to that in RT[$\twoheadleftarrow$, $\cap$].

3.1 Syntax of RT[$\twoheadleftarrow$, $\cap$]

The most important concept in the RT languages is also that of *roles*. A role in RT[$\twoheadleftarrow$, $\cap$] is denoted by a principal (corresponding to a user in RBAC) followed by a role name, separated by a dot. For example, when K is a principal and r is a role name, $K.r$ is a role. Each principal has its own name space for roles. For example, the "employee" role of one company is different from the "employee" role of another company. A *role* has a value that is a set of principals that are members of the role.

Each principal K has the authority to designate the members of a role of the form $K.r$. Roles are defined by *statements*. Figure 2 shows the four types of statements in RT[$\twoheadleftarrow$, $\cap$]; each corresponds to a way of defining role membership. A simple-member statement $K.r \longleftarrow K_1$ means that K_1 is a member of K's r role. This is similar to a user assignment in RBAC. A simple inclusion statement $K.r \longleftarrow K_1.r_1$ means that K's r role includes (all members of) K_1's r_1 role. This is similar to a role–role dominance relationship $K_1.r_1 \succeq K.r$. A linking inclusion statement $K.r \longleftarrow K.r_1.r_2$ means that $K.r$ includes $K_1.r_2$ for every K_1 that is a member of $K.r_1$. An intersection inclusion statement $K.r \longleftarrow K_1.r_1 \cap K_2.r_2$ means that $K.r$ includes every principal who is a member of both $K_1.r_1$ and $K_2.r_2$. Linking and intersection inclusion statements do not directly correspond to constructs in RBAC, but they are useful in expressing memberships in roles that result from administrative operations. Our reduction algorithms in Sections 4.1 and 4.2 use linking and intersection inclusion statements to capture user-role memberships affected by administrative operations.

3.2 States

An RT[$\leftarrow, \cap$] state γ^T consists of a set of RT[$\leftarrow, \cap$] statements. The semantics of RT[$\leftarrow, \cap$] is given by translating each statement into a datalog clause. (Datalog is a restricted form of logic programming (LP) with variables, predicates, and constants, but without function symbols.) (See Figure 2 for the datalog clauses corresponding to RT[$\leftarrow, \cap$] statements.) We call the datalog program resulting from translating each statement in γ^T into a clause that is the *semantic program* of γ^T, denoted by $SP(\gamma^T)$.

Given a datalog program, $\mathcal{DP}$, its semantics can be defined through several equivalent approaches. The model–theoretic approach views $\mathcal{DP}$ as a set of first-order sentences and uses the minimal Herbrand model as the semantics. We write $SP(\gamma^T) \models m(K, r, K')$ when $m(K, r, K')$ is in the minimal Herbrand model of $SP(\gamma^T)$.

3.3 State-Change Rules

A state-change rule is of the form $\psi^T = (\mathcal{G}, \mathcal{S})$, where $\mathcal{G}$ and $\mathcal{S}$ are finite sets of roles.

- Roles in $\mathcal{G}$ are called *growth-restricted* (or *g-restricted*); no statements defining these roles can be added. (A statement defines a role if it has the role to the left of "$\longleftarrow$".) Roles not in $\mathcal{G}$ are called *growth-unrestricted* (or *g-unrestricted*).
- Roles in $\mathcal{S}$ are called *shrink-restricted* (or *s-restricted*); statements defining these roles cannot be removed. Roles not in $\mathcal{S}$ are called *shrink-unrestricted* (or *s-unrestricted*).

3.4 Queries

Li et al. [2005] consider the following three forms of queries:

- *Membership*: $A.r \sqsupseteq \{D_1, \ldots, D_n\}$
  Intuitively, this means that all the principals $D_1, \ldots, D_n$ are members of $A.r$. Formally, $\gamma^T \vdash A.r \sqsupseteq \{D_1, \ldots, D_n\}$ if, and only if, $\{Z \mid SP(\gamma^T) \models m(A, r, Z)\} \supseteq \{D_1, \ldots, D_n\}$.
- *Boundedness*: $\{D_1, \ldots, D_n\} \sqsupseteq A.r$
  Intuitively, this means that the member set of $A.r$ is bounded by the given set of principals. Formally, $\gamma^T \vdash \{D_1, \ldots, D_n\} \sqsupseteq A.r$ if, and only if, $\{D_1, \ldots, D_n\} \supseteq \{Z \mid SP(\gamma^T) \models m(A, r, Z)\}$.
- *Inclusion*: $X.u \sqsupseteq A.r$
  Intuitively, this means that all the members of $A.r$ are also members of $X.u$. Formally, $\gamma^T \vdash X.u \sqsupseteq A.r$ if, and only if, $\{Z \mid SP(\gamma^T) \models m(X, u, Z)\} \supseteq \{Z \mid SP(\gamma^T) \models m(A, r, Z)\}$.

Each form of query can be generalized to allow compound role expressions that use linking and intersection. These generalized queries can be reduced to the forms above by adding new roles and statements to the state. For instance, $\{\} \sqsupseteq A.r \cap A_1.r_1.r_2$ can be answered by adding $B.u_1 \longleftarrow A.r \cap B.u_2$, $B.u_2 \longleftarrow B.u_3.r_2$, and $B.u_3 \longleftarrow A_1.r_1$ to γ^T, in which $B.u_1$, $B.u_2$, and $B.u_3$ are new g/s-restricted roles, and by posing the query $\{\} \sqsupseteq B.u_1$.

3.4.1 *Main Results for Security Analysis in* RT[←, ∩]. Membership and boundedness queries (both whether a query is possible and whether a query is necessary) can be answered in time polynomial in the size of the input. The approach taken in Li et al. [2005] uses logic programs to derive answers to those security analysis problems. This approach exploits the fact that RT[←, ∩] is monotonic in the sense that more statements will derive more role membership facts. This follows from the fact that the semantic program is a positive logic program.

Inclusion queries are more complicated than the other two kinds. In Li et al. [2005], only the ∀ case (i.e., whether an inclusion query is necessary) is studied. It is not clear what the security intuition is of an ∃ inclusion query (whether an inclusion query is possible); therefore, it is not studied in Li et al. [2005]. The problem of deciding whether an inclusion query is necessary, i.e., whether the set of members of one role is always a superset of the set of members of another role is called *containment analysis*. It turns out that the computational complexity of containment analysis depends on the language features. In RT[ ], the language that allows only simple member and simple inclusion statements, containment analysis is in **P**. It becomes more complex when additional policy language features are used. Containment analysis is **coNP**-complete for RT[∩] (RT[ ] plus intersection inclusion statements), **PSPACE**-complete for RT[←] (RT[ ] plus linking inclusion statements), and decidable in **coNEXP** for RT[←, ∩].

4. SOLVING AATU AND AAR BY REDUCTIONS TO SECURITY ANALYSIS IN RT[←, ∩]

In this section, we solve AATU (Definition 3) and AAR (Definition 4). Our approach is to reduce each of them to security analysis in RT[←, ∩]. Each reduction is an efficiently computable mapping from an instance of AATU/AAR to a security analysis instance in RT[←, ∩]. We precisely articulate the properties of the reductions in Propositions 1 and 4, respectively. Intuitively, the reductions preserve the results of security analysis across the mapping.

4.1 Reduction for AATU

The reduction algorithm AATU_Reduce is given in Figure 4; it uses the subroutines defined in Figure 3. Given an AATU instance $\langle \gamma = \langle UA, PA, RH \rangle, q = s_1 \sqsupseteq s_2, \psi = \langle can\_assign, T \rangle, \Pi \in \{\exists, \forall\} \rangle$, AATU_Reduce takes $\langle \gamma, q, \psi \rangle$ and outputs $\langle \gamma^T, q^T, \psi^T \rangle$, such that the RT[←, ∩] analysis instance $\langle \gamma^T, q^T, \psi^T, \Pi \rangle$ has the same answer as the original AATU instance.

In the reduction, we use one principal for every user that appears in γ, and the special principal Sys to represent the RBAC system. The RT[←, ∩] role names used in the reduction include the RBAC roles and permissions in γ and some additional temporary role names. The RT[←, ∩] role Sys.r represents the RBAC role r and the RT[←, ∩] role Sys.p represents the RBAC permission p. Each $(u,r) \in UA$ is translated into the RT[←, ∩] statement Sys.$r \longleftarrow u$. Each $r_1 \succeq r_2$ is translated into the RT[←, ∩] statement Sys.$r_2 \longleftarrow$ Sys.r_1 (as r_1 is senior

```
1 Subroutine Trans(s, γ^T) {
2   /* Trans(s,γ^T) returns an RT[←,∩] role corresponding
3       to the user set s*/
4   if s is an RBAC role then return Sys.s;
5   else if s is an RBAC permission then return Sys.s;
6   else if s is a set of users then {
7     name=newName(); foreach u ∈ s {
8       γ^T += Sys.name ⟵ u;}
9       return Sys.name; }
10  else if (s = s1 ∪ s2) then {
11    name=newName(); γ^T +=Sys.name ⟵ Trans(s1,γ^T);
12    γ^T += Sys.name ⟵ Trans(s2,γ^T);
13    return Sys.name; }
14  else if (s = s1 ∩ s2) then {
15    name=newName();
16    γ^T +=Sys.name ⟵ Trans(s1,γ^T) ∩ Trans(s2,γ^T);
17    return Sys.name; }
18 } /* End Trans */
19
20 Subroutine QTrans(s, γ^T) {
21   /* Translation for users sets that are used at top
22      level in a query */
23   if s is a set of users then return s;
24   else return Trans(s,γ^T);
25 } /* End QTrans */
26
27 Subroutine HTrans(s, γ^T) {
28   if s is an RBAC role then return HSys.s;
29   else if (s = s1 ∪ s2) then {
30     name=newName(); γ^T += Sys.name ⟵ HTrans(s1,γ^T);
31     γ^T += Sys.name ⟵ HTrans(s2,γ^T); return Sys.name; }
32   else if (s = s1 ∩ s2) then {
33     name=newName();
34     γ^T +=Sys.name ⟵ HTrans(s1,γ^T) ∩ HTrans(s2,γ^T);
35     return Sys.name; }
36 } /* End HTrans */
```

Fig. 3. Subroutines Trans, QTrans, and HTrans are used by the two reduction algorithms. We assume call-by-reference for the parameter γ^T.

to r_2, any member of r_1 is also a member of r_2.) Each $(p, r) \in PA$ is translated into $\mathsf{Sys}.p \longleftarrow \mathsf{Sys}.r$ (each member of the role r has the permission p.)

The translation of the *can_assign* relation is less straightforward. Each $\langle r_a, r_c, r\rangle \in$ *can_assign* is translated into the RT[←, ∩] statement $\mathsf{Sys}.r \longleftarrow \mathsf{Sys}.r_a.r \cap \mathsf{Sys}.r_c$. The intuition is that a user u_a, who is a member of the role r_a assigning the user u to be a member of the r role, is represented as adding the RT[←, ∩] statement $u_a.r \longleftarrow u$. As u_a is a member of the $\mathsf{Sys}.r_a$ role, the user u is added as a member to the $\mathsf{Sys}.r$ role if, and only if, the user u is also a member of the r_c role.

In the reduction, all the Sys roles (i.e., $\mathsf{Sys}.x$) are fixed (i.e., both g- and s-restricted). In addition, for each trusted user u in T, all the roles starting with u are also g-restricted; this is because we assume that trusted users will not perform operations to change the state (i.e., user-role assignment operations).

```
37 AATU_Reduce (⟨ γ = ⟨UA, PA, RH⟩, q = s1 ⊒ s2, ψ = ⟨can_assign, T⟩ ⟩)
38 {
39   /* Reduction algorithm for AATU */
40
41   γ^T = ∅; q^T = QTrans(s1, γ^T) ⊒ QTrans(s2, γ^T);
42   foreach (ui, rj) ∈ UA { γ^T += Sys.rj ⟵ ui; }
43   foreach (ri, rj) ∈ RH { γ^T += Sys.rj ⟵ Sys.ri; }
44   foreach (pi, rj) ∈ PA { γ^T += Sys.pi ⟵ Sys.rj; }
45   foreach (ai, s, rset) ∈ can_assign {
46     if (s==true) { foreach r ∈ rset {
47         γ^T += Sys.r ⟵ Sys.ai.r; } }
48     else { tmpRole=Trans(s, γ^T);
49       foreach r ∈ rset { name=newName();
50         γ^T += Sys.name ⟵ Sys.ai.r;
51         γ^T += Sys.r ⟵ Sys.name ∩ tmpRole
52   } } }
53   foreach RT role name x appearing in γ^T {
54     G+=Sys.x; S+=Sys.x; foreach user u ∈ T { G+=u.x; } }
54   return ⟨γ^T, q^T, (G, S)⟩;
55 } /* End AATU_Reduce */
```

Fig. 4. Reduction algorithm for AATU.

We may also make roles starting with trusted users s-restricted; however, this has no effect, as no statement defining these roles exists in the initial state.

Example 6. Consider the state-change rule ψ we discussed in Example 4, in which *can_assign* consists of the two tuples ⟨Manager, Engineer∧ FullTime, ProjectLead⟩ and ⟨HumanResource, *true*, {*FullTime*, *PartTime*}⟩, and $T = \{\text{Carol}\}$. Let γ be the RBAC state (shown in Figure 1) and let q be the query ProjectLead ⊒ Alice. We then represent the output of AATU_Reduce$(\langle \gamma, q, \psi \rangle)$ as $\langle \gamma^T, q^T, \psi^T \rangle$. q^T is Sys.ProjectLead ⊒ {Alice}. The following RT statements in γ^T result from UA:

Sys.Engineer ⟵ Alice Sys.PartTime ⟵ Alice
Sys.Manager ⟵ Bob Sys.HumanResource ⟵ Carol

The following statements in γ^T result from RH:

Sys.Employee ⟵ Sys.Engineer Sys.Employee ⟵ Sys.FullTime
Sys.Employee ⟵ Sys.PartTime Sys.Engineer ⟵ Sys.ProjectLead
Sys.FullTime ⟵ Sys.Manager

The following statements in γ^T result from PA:

Sys.View ⟵ Sys.HumanResource Sys.Access ⟵ Sys.Employee
Sys.Edit ⟵ Sys.Engineer

The following statements in γ^T result from *can_assign*. The first two statements reflect the ability of a member of HumanResource to assign users to FullTime and PartTime with no precondition and the remaining statements reflect the ability of a member of Manager to assign users to ProjectLead provided that they are already members of FullTime and Engineer.

Sys.FullTime ⟵ Sys.HumanResource.FullTime
Sys.PartTime ⟵ Sys.HumanResource.PartTime

$\mathsf{Sys.NewRole}_1 \longleftarrow \mathsf{Sys.Engineer} \cap \mathsf{Sys.FullTime}$
$\mathsf{Sys.NewRole}_2 \longleftarrow \mathsf{Sys.Manager.ProjectLead}$
$\mathsf{Sys.ProjectLead} \longleftarrow \mathsf{Sys.NewRole}_1 \cap \mathsf{Sys.NewRole}_2$

$\gamma^T = \langle G, S \rangle$, where G is the growth-restricted set of roles and S is the shrink-restricted set of roles. G consists of every role of the form $\mathsf{Sys}.x$ and every role of the form $\mathsf{Carol}.x$. The latter is included in G because Carol is in the set of trusted users T. S consists of every role of the form $\mathsf{Sys}.x$. It is clear that the security analysis instance $\langle \gamma^T, q^T, \psi^T, \exists \rangle$ is false, as Alice can never become a member of Sys.ProjectLead. If we adopt as the state-change rule ψ_1^T, that is the same as ψ^T except that $T = \emptyset$, then roles of the form $\mathsf{Carol}.x$ would be growth-unrestricted. There also exists a state γ_1^T that is reachable from γ^T, which has the following statements, in addition to all the statements in γ^T.

$\mathsf{Carol.FullTime} \longleftarrow \mathsf{Alice}$ $\quad$ $\mathsf{Bob.ProjectLead} \longleftarrow \mathsf{Alice}$

These statements are necessary and sufficient for $\mathsf{Sys.ProjectLead} \longleftarrow \mathsf{Alice}$ to be inferred in γ_1^T. Thus, the security analysis instance $\langle \gamma^T, q^T, \psi_1^T, \exists \rangle$ is true.

The following proposition asserts that the reduction is sound, meaning that one can use RT security analysis techniques to answer RBAC security analysis problems.

PROPOSITION 1. *Given an AATU instance* $\langle \gamma, q, \psi, \Pi \rangle$*, let* $\langle \gamma^T, q^T, \psi^T \rangle = \mathsf{AATU\_Reduce}(\langle \gamma, q, \psi \rangle)$*, then*:

- Assertion 1: *For every RBAC state* γ' *such that* $\gamma \overset{*}{\mapsto}_{\psi} \gamma'$*, there exists an* $\mathsf{RT}[\twoheadleftarrow, \cap]$ *state* $\gamma^{T\prime}$ *such that* $\gamma^T \overset{*}{\mapsto}_{\psi^T} \gamma^{T\prime}$ *and* $\gamma' \vdash q$ *if and only if* $\gamma^{T\prime} \vdash q^T$.
- Assertion 2: *For every* $\mathsf{RT}[\twoheadleftarrow, \cap]$ *state* $\gamma^{T\prime}$ *such that* $\gamma^T \overset{*}{\mapsto}_{\psi^T} \gamma^{T\prime}$*, there exists an RBAC state* γ' *such that* $\gamma \overset{*}{\mapsto}_{\psi} \gamma'$ *and* $\gamma' \vdash q$ *if and only if* $\gamma^{T\prime} \vdash q^T$.

See Appendix 1 for the proof. As we discuss in detail in Tripunitara and Li [2004], the above proposition asserts that AATU_Reduce is security-preserving in the sense that an AATU analysis instance is true if, and only if, the $\mathsf{RT}[\twoheadleftarrow, \cap]$ analysis instance that is the output of AATU_Reduce is true. That is, AATU_Reduce preserves the answer to every security analysis instance. We argue the need for assertion 1 in the proposition by considering the case that there exists a reachable state γ' in the RBAC system, but no corresponding reachable state $\gamma^{T\prime}$ in the $\mathsf{RT}[\twoheadleftarrow, \cap]$ system produced by AATU_Reduce. Let the corresponding query be q. If $\gamma' \vdash q$, then let Π be $\exists$, and if $\gamma' \vdash q$, then let Π be $\forall$. In the former case, the security analysis instance in RBAC is true, but the instance in the $\mathsf{RT}[\twoheadleftarrow, \cap]$ system that is the output of AATU_Reduce is false. In the latter case, the analysis instance in RBAC is false, but the instance in $\mathsf{RT}[\twoheadleftarrow, \cap]$ is true. Therefore, for AATU_Reduce to preserve the answer to every analysis instance, we need assertion 1.

Similarly, we argue the need for assertion 2 by considering the contrary situation. Let $\gamma^{T\prime}$ be a reachable state in $\mathsf{RT}[\twoheadleftarrow, \cap]$ for which there exists no corresponding state in RBAC. Let the corresponding query in $\mathsf{RT}[\twoheadleftarrow, \cap]$ be q^T. If $\gamma^{T\prime} \vdash q^T$, then let Π be $\exists$, and let Π be $\forall$ otherwise. Again, AATU_Reduce would

not preserve the answer to a security analysis instance and we would not be able to use the answer to an analysis instance in RT[$\leftarrow$, $\cap$] as the answer to the corresponding instance in RBAC.

THEOREM 2. *An AATU instance* $\langle \gamma, q, \psi, \Pi \rangle$ *can be solved efficiently, i.e., in time polynomial in the size of the instance, if* q *is semistatic.*

PROOF. Let the output of AATU_Reduce corresponding to the input $\langle \gamma, q, \psi \rangle$ be $\langle \gamma^T, q^T, \psi^T \rangle$. If q is semistatic, we observe that q^T is semistatic as well. Furthermore, AATU_Reduce runs in time polynomial in its input. We know from Li et al. [2005] that in RT[$\leftarrow$, $\cap$], a security analysis instance with a semistatic query can be answered in time polynomial in the size of γ^T. Therefore, in conjunction with Proposition 1, we can conclude that a security analysis instance with a semistatic query in the RBAC system can be answered in time polynomial in the size of the system (i.e., the size of $\langle \gamma, q, \psi \rangle$). □

THEOREM 3. *An AATU instance* $\langle \gamma, q, \psi, \Pi \rangle$ *is* **coNP***-complete.*

PROOF. We show that the general AATU problem is **coNP**-hard by reducing the monotone 3SAT problem to the complement of the AATU problem. Monotone 3SAT is the problem of determining whether a boolean expression in conjunctive normal form with, at most, three literals in each clause such that the literals in a clause are either all positive or all negative, is satisfiable. Monotone 3SAT is known to be **NP**-complete [Garey and Johnson 1979].

Let ϕ be an instance of monotone 3SAT. Then $\phi = c_1 \wedge \ldots c_l \wedge \overline{c_{l+1}} \wedge \ldots \wedge \overline{c_n}$ where $c_1, \ldots, c_l$ are the clauses with positive literals and $\overline{c_{l+1}}, \ldots, \overline{c_n}$ are the clauses with negative literals. Let $p_1, \ldots, p_s$ be all the propositional variables in ϕ. For each clause with negative literals $\overline{c_k} = (\neg\, p_{k_1} \vee \neg\, p_{k_2} \vee \neg\, p_{k_3})$, define $d_k = \neg\, \overline{c_k} = (p_{k_1} \wedge p_{k_2} \wedge p_{k_3})$. Then, ϕ is satisfiable if, and only if, $c_1 \wedge \ldots c_l \wedge \neg(d_{l+1} \vee \ldots \vee d_n)$ is satisfiable. Let $\eta = (c_1 \wedge \ldots \wedge c_l) \rightarrow (d_{l+1} \vee \ldots \vee d_n)$, where $\rightarrow$ is logical implication. Then, $c_1 \wedge \ldots c_l \wedge \neg(d_{l+1} \vee \ldots \vee d_n) = \neg\, \eta$. Therefore, ϕ is satisfiable if, and only if, η is not valid. We now construct γ, ψ, and q in an AATU instance such that $q = z_1 \sqsupseteq z_2$ is true for user sets z_1 and z_2 in all states reachable from γ if, and only if, η is valid.

In γ, we have a role a (which is for administrators) and UA contains (A, a), where A is a user (i.e., the role a is not empty in terms of user-membership). With each propositional variable p_i in η, we associate a role r_i. For each r_i, we add $\langle a, true, r_i \rangle$ to *can_assign*. That is, anyone can be assigned to the role r_i. We let T (the set of trusted users) be empty. For each j such that $1 \leq j \leq l$, we associate the clause $c_j = (p_{j_1} \vee p_{j_2} \vee p_{j_3})$, with a user set $s_j = (r_{j_1} \cup r_{j_2} \cup r_{j_3})$. For each k such that $(l+1) \leq k \leq n$, we associate the clause $d_k = (p_{k_1} \wedge p_{k_2} \wedge p_{k_3})$, with a user set $s_k = (r_{k_1} \cap r_{k_2} \cap r_{k_3})$. In our query $q = z_1 \sqsupseteq z_2$, we let $z_1 = s_{l+1} \cup \ldots \cup s_n$ and $z_2 = s_1 \cap \ldots \cap s_l$. We now need to show that $z_1 \sqsupseteq z_2$ in every state reachable from γ if, and only if, η is valid. We show that $z_1 \sqsupseteq z_2$ is *not* true in every state reachable from γ if, and only if, η is *not* valid.

For the "only if" part, we assume that there exists a state γ' that is reachable from γ such that in γ' there exists a user u that is a member of the user set z_2, but not z_1. Consider a truth-assignment I for the propositional variables in η as follows: if u is a member of the role r_i in γ', then $I(p_i) = true$. Otherwise,

```
AAR_Reduce (⟨γ = ⟨UA, PA, RH⟩, q = s1 ⊒ s2,
            ψ = ⟨can_assign, can_revoke⟩⟩)
{ /* Reduction algorithm for AAR */
  γ^T = ∅;  q^T = QTrans(s1, γ^T) ⊒ QTrans(s2, γ^T);
  foreach (ui, rj) ∈ UA {
    γ^T += HSys.rj ⟵ ui; γ^T += RSys.rj ⟵ ui;
    γ^T += Sys.rj ⟵ RSys.rj; }
  foreach (ri, rj) ∈ RH {
    γ^T += Sys.rj ⟵ Sys.ri; γ^T += HSys.rj ⟵ HSys.ri; }
  foreach (pi, rj) ∈ PA { γ^T += Sys.pi ⟵ Sys.rj; }
  foreach (ai, s, rset) ∈ can_assign {
    if (s==true) {
      foreach r ∈ rset {
        γ^T += HSys.r ⟵ BSys.r; γ^T += Sys.r ⟵ ASys.r; }
    } else { tmpRole = HTrans(s, γ^T);  /* precondition */
      foreach r ∈ rset {
        γ^T += HSys.r ⟵ BSys.r ∩ tmpRole;
        γ^T += Sys.r ⟵ ASys.r ∩ tmpRole; }
  } }
  foreach RT role name x appearing in γ^T {
    G+=Sys.x; S+=Sys.x; G+=HSys.x; S+=HSys.x; G+=RSys.x;
    S+=BSys.x; S+=RSys.x; S+=ASys.x;
  } /* when a can_revoke rule exists for r, ASys.r and
       RSys.r can shrink */
  foreach (ai, rset) ∈ can_revoke {
    foreach r in rset { S-=RSys.r; S-=ASys.r; } }
  return ⟨γ^T, q^T, (G, S)⟩;
} /* End AAR_Reduce */
```

Fig. 5. AAR_Reduce: the reduction algorithm for AAR.

$I(p_i) = \mathit{false}$. Under I, η is not true, as $(c_1 \wedge \ldots \wedge c_l)$ is true, but $(d_{l+1} \vee \ldots \vee d_n)$ is false. Therefore, η is not valid.

For the "if" part, we assume that η is not valid. Therefore, there exists a truth-assignment I such that $(c_1 \wedge \ldots \wedge c_l)$ is true, but $(d_{l+1} \vee \ldots \vee d_n)$ is false. Consider a state γ' that has the following members in UA in addition to the ones in γ: for each p_i that is true under I, $(u, r_i) \in UA$. Otherwise, $(u, r_i) \notin UA$. γ' is reachable from γ and in γ', $z_1 \sqsupseteq z_2$ is not true.

To prove that the problem is in **coNP**, we need to show that when an instance is false, there exists evidence of size polynomial in the input that can be verified efficiently. The evidence is a user u and a sequence of n state-changes from the start-state to some state γ' such that in γ', u is a member of the user set z_2, but not of z_1. We know that n is bounded by the number of roles in the system as there can be only as many user-to-role assignment operations for a particular user as the number of roles. The verification of this evidence is certainly efficient. Therefore, the problem is in **coNP**. □

We observe from the above proof that the AATU problem remains **coNP**-complete even when every precondition that occurs in *can_assign* is specified as *true*; the expressive power of the queries is sufficient for reducing the monotone 3SAT problem to the general AATU problem.

4.2 Reduction for AAR

The reduction algorithm for AAR is given in Figure 5. The reduction algorithm includes in the set of principals a principal for every user in U and five special

principals: Sys, RSys, HSys, ASys, and BSys. Again, the Sys roles simulate RBAC roles and permissions. In this reduction, we do not distinguish whether a role assignment operation is effected by one user or another and use only one principal, ASys, to represent every user that exercises the user-role assignment operation. The roles of the principal RSys contain all the initial role memberships in UA; these may be revoked in state changes. HSys.r maintains the history of the RBAC role r; its necessity is argued using the following scenario. A user is a member of r_1, which is the precondition for being added to another role r_2. After one assigns the user to r_2 and revokes the user from r_1. The user's membership in r_2 should maintain, even though the precondition is no longer satisfied (a similar justification for this approach is provided in the context of ARBAC97 [Sandhu et al. 1999], as well). BSys is similar to ASys, but it is used to construct the HSys roles. An administrative operation to try to add a user u_i to the role r_j is represented by adding the statement ASys.$r_j \longleftarrow u_i$ and BSys.$r_j \longleftarrow u_i$ to γ^T. An administrative operation to revoke a user u_i from the role r_j is represented by removing the statements RSys.$r_j \longleftarrow u_i$ and ASys.$r_j \longleftarrow u_i$ if either exists in γ^T.

Example 7. Consider the state-change rule ψ we discuss in Example 5, in which *can_assign* consists of the two tuples ⟨Manager, Engineer∧ FullTime, ProjectLead⟩ and ⟨HumanResource, *true*, {FullTime, PartTime}⟩, and *can_revoke* consists of the two tuples ⟨Manager, {Engineer, ProjectLead}⟩ and ⟨HumanResource, {FullTime, PartTime}⟩. Let γ be the RBAC state, shown in Figure 1, and let q be the query ProjectLead $\sqsupseteq$ Alice. We then represent the output of AATU_Reduce$(\langle \gamma, q, \psi \rangle)$ as $\langle \gamma^T, q^T, \psi^T \rangle$. q^T is Sys.ProjectLead $\sqsupseteq$ {Alice}. The following RT statements in γ^T result from UA:

HSys.Engineer ⟵ Alice
HSys.PartTime ⟵ Alice
HSys.Manager ⟵ Bob
HSys.HumanResource ⟵ Carol
RSys.Engineer ⟵ Alice
RSys.PartTime ⟵ Alice
RSys.Manager ⟵ Bob
RSys.HumanResource ⟵ Carol
Sys.Engineer ⟵ RSys.Engineer
Sys.FullTime ⟵ RSys.FullTime
Sys.HumanResource ⟵ RSys.HumanResource
Sys.PartTime ⟵ RSys.PartTime

The following statements in γ^T result from RH:

Sys.Employee ⟵ Sys.Engineer
Sys.Employee ⟵ Sys.FullTime
Sys.Employee ⟵ Sys.PartTime
Sys.Engineer ⟵ Sys.ProjectLead
Sys.FullTime ⟵ Sys.Manager
HSys.Employee ⟵ HSys.Engineer
HSys.Employee ⟵ HSys.FullTime
HSys.Employee ⟵ HSys.PartTime
HSys.Engineer ⟵ HSys.ProjectLead
HSys.FullTime ⟵ HSys.Manager

The following statements in γ^T result from PA:

Sys.View ⟵ Sys.HumanResource
Sys.Edit ⟵ Sys.Engineer
Sys.Access ⟵ Sys.Employee

The following statements in γ^T result from *can_assign*:

HSys.FullTime ⟵ BSys.FullTime
Sys.FullTime ⟵ ASys.FullTime

HSys.PartTime ⟵ BSys.PartTime Sys.PartTime ⟵ ASys.PartTime
Sys.NewRole$_1$ ⟵ HSys.Engineer ∩ HSys.FullTime
HSys.ProjectLead ⟵ BSys.ProjectLead ∩ Sys.NewRole$_1$
Sys.ProjectLead ⟵ ASys.ProjectLead ∩ Sys.NewRole$_1$

$\psi^T = \langle G, S \rangle$, where G is the growth-restricted set of roles and S is the shrink-restricted set of roles. Unlike *can_assign*, *can_revoke* results only in some roles not being added to S. G is comprised of all roles of the form Sys.x, HSys.x and RSys.x (but not BSys.x or ASys.x). S is comprised of all roles of the form Sys.x, HSys.x, RSys.x and ASys.x, except the roles RSys.Manager, ASys.Manager, RSys.Engineer, ASys.Engineer, RSys.FullTime, ASys.FullTime, RSys.PartTime, and ASys.PartTime. This is because those roles appear in *can_revoke* rules and, therefore, may shrink.

There exists a state γ_1^T that is reachable from γ^T that has the following statements in addition to the ones in γ^T.

BSys.FullTime ⟵ Alice ASys.ProjectLead ⟵ Alice

We can now infer that in γ_1^T, HSys.FullTime⟵Alice and, therefore, HSys.NewRole$_1$ ⟵ Alice, and so, Sys.ProjectLead ⟵ Alice. Thus, the security analysis instance $\langle \gamma^T, q^T, \psi^T, \exists \rangle$ is true. If we consider, instead, the query q_1^T, which is Sys.PartTime $\sqsupseteq$ Alice, then as RSys.PartTime is a shrink-unrestricted role, there exists a state γ_2^T that is reachable from γ^T in which the statement RSys.PartTime ⟵ Alice is absent. Therefore, we would conclude that Sys.ProjectLead does not include Alice. Consequently, the analysis instance $\langle \psi^T, q_1^T, \gamma^T, \forall \rangle$ is false.

We are also able to demonstrate the need for the roles associated with the principals HSys and BSys. Consider the state γ_2^T that can be reached from γ_1^T by removing the statement RSys.FullTime ⟵ Alice. Now, Sys.FullTime does not include Alice. This is equivalent to Carol revoking the membership of the user Alice to the role FullTime. This affects the precondition that one can be assigned to the role ProjectLead only if one is already a member of the roles Engineer and FullTime. Nonetheless, we observe that $\gamma_2^T \vdash q^T$, as indeed it should. That is, Alice should continue to be a member of ProjectLead even if subsequent to her becoming a member of ProjectLead, her membership is removed from FullTime. We observe that this is the case because the role BSys.FullTime is shrink-restricted and, therefore, one cannot remove the statement BSys.FullTime ⟵ Alice once it has been added and, consequently, HSys.FullTime ⟵ Alice is true, and, therefore, Alice continues to be a member of the role ProjectLead (i.e., is included in Sys.ProjectLead). Of course, Alice can later have her membership revoked from the role ProjectLead (by Bob) and this is equivalent to the statement ASys.ProjectLead ⟵ Alice being removed.

The following proposition asserts that the reduction is sound.

PROPOSITION 4. *Given an AAR instance* $\langle \gamma, q, \psi, \Pi \rangle$*, let* $\langle \gamma^T, q^T, \psi^T \rangle$ = AAR_Reduce($\langle \gamma, q, \psi \rangle$)*, then*:

- Assertion 1: *For every RBAC state* γ' *such that* $\gamma \overset{*}{\mapsto}_{\psi} \gamma'$, *there exists an* RT[$\leftarrow$, $\cap$] *state* $\gamma^{T\prime}$ *such that* $\gamma^T \overset{*}{\mapsto}_{\psi^T} \gamma^{T\prime}$ *and* $\gamma' \vdash q$ *if, and only if,* $\gamma^{T\prime} \vdash q^T$.
- Assertion 2: *For every* RT[$\leftarrow$, $\cap$] *state* $\gamma^{T\prime}$ *such that* $\gamma^T \overset{*}{\mapsto}_{\psi^T} \gamma^{T\prime}$, *there exists an RBAC state* γ' *such that* $\gamma \overset{*}{\mapsto}_{\psi} \gamma'$ *and* $\gamma' \vdash q$ *if, and only if,* $\gamma^{T\prime} \vdash q^T$.

The proof is in Appendix 2. Our comments regarding the need for assertions 1 and 2 to preserve answers to security analysis instances, which we make in the previous section in the context of AATU_Reduce, apply to the above proposition in the context of AAR_Reduce as well. If either of the assertions does not hold, then we cannot use the answer to the RT[$\leftarrow$, $\cap$] analysis instance as the answer to the corresponding RBAC instance.

THEOREM 5. *An AAR instance* $\langle \gamma, q, \psi, \Pi \rangle$ *can be solved efficiently, i.e., in time polynomial in the size of the instance, if* q *is semistatic.*

PROOF. Let the output of AAR_Reduce for the input $\langle \gamma, q, \psi \rangle$ be $\langle \gamma^T, q^T, \psi^T \rangle$. If q is semistatic, so is q^T. As AAR_Reduce runs in time polynomial in its input and q^T can be answered in time polynomial in the size of γ^T (which is shown by Li et al. [2005]), q can be answered in time polynomial in the size of the system (i.e., the size of $\langle \gamma, q, \psi \rangle$). Thus, an AAR instance with a semistatic query can be solved efficiently. □

THEOREM 6. *An AAR instance* $\langle \gamma, q, \psi, \Pi \rangle$ *is* **coNP***-complete.*

PROOF. We deduce that an AAR instance is in **coNP** from the fact that AAR_Reduce runs in time polynomial in the size of the system and the corresponding security analysis problem in the RT[$\cap$] system that is the output of AAR_Reduce is **coNP**-complete. (RT[$\cap$] is a sublanguage of RT[$\leftarrow$, $\cap$] that allows only the first, second, and fourth kinds of statements from Figure 2.) That is, if q is not true in every state reachable from γ, then we offer as counterproof the algorithm AAR_Reduce and the counterproof in the RT[$\leftarrow$, $\cap$] system that q^T is not true in every state reachable from γ^T.

We can show that the general AAR problem is **coNP**-hard in almost exactly the same way that we show the result for the AATU problem in the proof for Theorem 3. The only difference is that for every role r_i, which is associated with a propositional variable p_i, apart from a rule in *can_assign*, we add the rule $\langle a, r_i \rangle$ to *can_revoke*. We construct the query q the same way as in that proof and show in the same way that q is true in every state reachable from γ if, and only if, η is valid. □

5. RELATED WORK

Simple safety analysis, i.e., determining whether an access-control system can reach a state in which an unsafe access is allowed, was first formalized by Harrison et al. [1976] in the context of the well-known access matrix model [Lampson 1971; Graham and Denning 1972], and was shown to be undecidable in the HRU model [Harrison et al. 1976]. There are special cases for which safety is decidable for the HRU model; safety is decidable if (1) no subjects or objects are allowed to be created, (2) at most one condition is used in a

command, but subjects or objects cannot be destroyed, or (3) only one operation is allowed in a command.

Following that, there have been various efforts in designing access-control systems in which simple safety analysis is decidable or efficiently decidable, e.g., the take-grant model [Lipton and Snyder 1977], the schematic protection model [Sandhu 1988], and the typed access matrix model [Sandhu 1992].

One may be tempted to reduce the security analysis problem defined in this paper to a problem in one of the other models, such as HRU, and use existing results. However, this approach has several difficulties. First, we consider different kinds of queries, while only safety is considered in other models. It is not clear, for instance, how one would reduce containment in RBAC to safety in HRU. Second, even when we restrict our attention to simple safety, the reduction of either AATU or AAR into HRU results in a set of command schemas that does not fall into any known decidable special case of HRU. (1) New users are implicitly created when being assigned to roles. (2) Because of preconditions in AATU and AAR, an assignment operation requires checking both the command initiator's privileges and the user's role memberships. The resulting HRU command schema would not be mono-conditional. (3) Adding a user to a role results in the user attaining several permissions simultaneously. The resulting command in HRU is unlikely to be mono-operational. Last, but not least, even if some further restricted subcases of RBAC security analysis can be reduced to decidable subcases of HRU, no efficient algorithm exists for those cases. For example, even in the subcase where no subjects or objects are allowed to be created, safety analysis in HRU remains **PSPACE**-complete (which implies that it is **NP**-hard).

Recently, Li et al. [2005] proposed the notion of security analysis and studied security analysis in the context of RT[$\leftarrow$, $\cap$], a role-based trust-management language. They showed that a security analysis instance in RT[$\leftarrow$, $\cap$] involving only semistatic queries can be solved efficiently (in time polynomial in the size of the start-state in the analysis) and, for more general queries, they showed that the analysis is decidable, but intractable.

Crampton and Loizou [2003] claim that "if administrative (or control) permissions are assigned to subjects, then the safety problem is undecidable. Indeed, Munawer and Sandhu [1999] and Crampton [2002] have shown independently that the safety problem for RBAC96 is undecidable." We disagree with this claim, and we show in this paper that simple safety (and even more sophisticated analysis) can be decidable when administrative permissions are given to subjects. The simulation by Munawer and Sandhu [1999] suggests only that when an overly complicated administrative model is added to RBAC96, safety analysis may be undecidable.

The work by Koch et al. [2002a] considers safety in RBAC with the RBAC state and state-change rules posed as a graph formalism [Koch et al. 2002b]. They show that safety (defined as whether a given graph can become a subgraph of another graph) is decidable provided that a state-change rule does not both remove and add components to the graph that represents the protection state. It is not clear what import the property of safety, as defined in the context of the graph-based formalism, has in the context of an RBAC system. In particular,

it is not clear whether the notion of safety as defined in that work captures the notions of simple or bounded safety, or containment that we discuss in Section 2.1. Also, specific complexity bounds for deciding safety are also not provided in that work and, therefore, it is not clear how useful the decidability result for safety is. In particular, we do not know whether safety can be decided efficiently. Furthermore, the administrative model (set of state-change rules) considered in that work is limited in that all roles are considered to be of the same type and, therefore, roles correspond to nodes in the graph, each of which has the same label as another. Consequently, we cannot express preconditions to user-role assignment as we can with ARBAC97 and the administrative models considered in this paper. Such preconditions, as we discuss in Section 2.3, are expressions formed using roles. Recently, the graph-based formalism [Koch et al. 2002b] has been extended to consider a more realistic and flexible administrative model [Koch et al. 2004]. This new administrative model considers state-change rules that consist of commands such as *addEdge* and *deleteEdge*. The commands do not satisfy a criterion for the decidability of the safety property that was shown in Koch et al. [2002a]; some of the commands remove and add components to the graph. Whether safety (as defined for the graph-based formalism) is indeed decidable or not, given the new state-change rules, is not known. Our work differs from that work in that we consider a general class of queries, and provide specific algorithms and complexity bounds. In addition, our state-change rules are based on ARBAC97, whose usefulness has been argued in the literature [Sandhu et al. 1999].

Previous work on ensuring security properties in RBAC takes the approach of using constraints [Ahn and Sandhu 2000; Crampton 2003; Jaeger and Tidswell 2001]. In this approach, a set of desirable properties are explicitly specified as constraints on the relations in an RBAC state. Each time the state of an access-control system is about to change, these constraints are checked. A change is allowed only when these constraints are satisfied. We believe that security analysis and constraints are complementary. Constraints directly specify desirable properties on the state of an RBAC system. Security analysis uses conditions specified on what kinds of state changes are allowed and infer security properties on all reachable states. An advantage of using constraints is that sophisticated conditions can be specified and enforced efficiently. In the security analysis approach, fewer security properties can be analyzed efficiently, because of the need to analyze potentially infinitely many reachable states. On the other hand, the constraint approach requires that the system controls all changes to the RBAC state, because of the need to perform constraint checking. Security analysis can handle decentralized control by allowing the parts of a state that are not controlled by the system to change freely. It can be used to help enforce security properties even when the RBAC system itself is maintained in a decentralized manner and one cannot ensure that constraints are checked when some part of the RBAC state changes. Another advantage of security analysis is that it can be performed less often than checking constraints. Analysis only needs to be performed when changes not allowed by the state-transition rule are made, while constraints need to be evaluated each time a state changes.

6. CONCLUSION AND FUTURE WORK

We have proposed the use of security analysis techniques to maintain desirable security properties while delegating administrative privileges. More specifically, we have defined a family of security analysis problems in RBAC and two classes of problems in this family, namely AATU and AAR, based on the URA97 component of the ARBAC97 administrative model for RBAC. We have also shown that AATU and AAR can be reduced to similar analysis problems in the RT[$\leftarrow$, $\cap$] trust-management language, establishing an interesting relationship between RBAC and the RT (role-based trust management) framework. The reduction gives efficient algorithms for answering most kinds of queries in these two classes and helps establish the complexity bounds for the intractable cases.

While security analysis is especially effective in cases that the associated problems are tractable, as we have demonstrated in this paper, several security analysis problems can be intractable or even undecidable. Consequently, administrators may be constrained in the kinds of queries they can pose or the states in which they can allow the RBAC system to be. In any case, unless efficient heuristics can be developed for the intractable cases, security analysis may not be effective or usable.

Much work remains to be done for understanding security analysis in RBAC. The family of RBAC security analysis defined in this paper can be parameterized with more sophisticated administrative models, e.g., those that allow negative preconditions, those that allow changes to the role hierarchy or role-permission assignments, and those that allow the specification of constraints, such as mutually exclusive roles.

Commercial products, such as database management systems, include support for RBAC and for decentralized administration. We believe that security analysis will be effective in such contexts; a detailed discussion those RBAC schemes and security analysis in their context is part of future work. Security analysis is also applicable in several other access-control schemes, including UCON [Park and Sandhu 2004; Zhang et al. 2004, 2005], which extends RBAC. The use of security analysis in such schemes is also part of future work.

Appendix 1. Proof for Proposition 1

PROOF. *For Assertion 1:* A state change in AATU occurs when a user assignment operation is successfully performed. For every RBAC state γ' such that $\gamma \overset{*}{\mapsto}_{\psi} \gamma'$, let $\gamma_0, \gamma_1, \ldots, \gamma_m$ be RBAC states such that $\gamma = \gamma_0 \mapsto_{\psi} \gamma_1 \mapsto_{\psi} \cdots \mapsto_{\psi} \gamma_m = \gamma'$. We construct a sequence of RT[$\leftarrow$, $\cap$] states ${\gamma^T}_0, {\gamma^T}_1, \ldots, {\gamma^T}_m$ as follows: $\gamma_0^T = \gamma^T$; for each $i = [0..m-1]$, consider the assignment operation that changes γ_i to γ_{i+1}, let it be the operation in which a user u_1 adds (u,r) to the user-role assignment relation. The state γ_{i+1}^T is obtained by adding $u_1.r \longleftarrow u$ to γ_i^T. Let ${\gamma^T}'$ be γ_m^T.

Step one: Prove that if $\gamma' \vdash q$ then ${\gamma^T}' \vdash q^T$. It is sufficient to prove the following: for each $i \in [0..m]$, if γ_i implies that a certain user u is a member of a role r (or has the permission p), then ${\gamma^T}_i$ implies that u is a member of the RT[$\leftarrow$, $\cap$] role Sys.r (or Sys.p). We use induction on i to prove this. The

base case (i = 0) follows directly from the AATU_Reduce algorithm; lines 42–44 reproduces *UA*, *RH*, *PA* in the RT[$\twoheadleftarrow$, $\cap$] state γ_0^T. For the step, assumes that the induction hypothesis holds for $\gamma_0, \ldots, \gamma_i$, consider γ_{i+1}. Let the operation leading to γ_{i+1} be one in which u_1 assigns u to a role r. Since both sequences of states are increasing, we only need to consider role memberships implied by γ_{i+1} but not γ_i; these are caused (directly or indirectly) by this assignment. There must exists a $\langle r_a, c, r\rangle \in$ *can_assign* to enable this assignment; thus in γ_i, u_1 is a member of the role r_a and u satisfies the condition c. By induction hypothesis, in γ_i^T, u_1 is a member of Sys.r_a and u satisfies the condition c. From the translation and the construction of γ_{i+1}^T, γ_{i+1}^T has the following statements: $u_1.r \longleftarrow u$, Sys.$r \longleftarrow$ Sys.$r_a.r$, and Sys.$r \longleftarrow$ Sys.*name* $\cap$ *tmpRole* (where *tmpRole* corresponds to the precondition c). Furthermore, in γ_{i+1}^T, u_1 is a member of the role r_a and u satisfies the condition c. Therefore, u is a member of the Sys.r role in γ_{i+1}^T.

Step two: Prove that if ${\gamma^T}' \vdash q^T$ then $\gamma' \vdash q$. It is sufficient to show that if an RT[$\twoheadleftarrow$, $\cap$] role membership is implied by ${\gamma^T}'$, then the corresponding RBAC role membership (or permission possession) is also implied. A detailed proof uses induction on the number of rounds in which a bottom-up datalog evaluation algorithm outputs a ground fact. Here, we only point out the key observations. (For details of similar proofs, see the Appendix in Li et al. [2005].) A RT[$\twoheadleftarrow$, $\cap$] role membership is proved by statements generated on lines 42–52. The first three cases correspond to the *UA*, *RH*, *PA*. For the last case, there must exist a statement $u_1.r \longleftarrow u$ in ${\gamma^T}'$, and it implies that u is a member of the role Sys.r. By the construction of ${\gamma^T}'$, the user u has been assigned to the role r during the changes leading to γ'.

For Assertion 2: Given an RT[$\twoheadleftarrow$, $\cap$] state ${\gamma^T}'$ such that $\gamma^T \overset{*}{\mapsto}_{\psi^T} {\gamma^T}'$, we can assume without loss of generality that ${\gamma^T}'$ adds to γ^T only simple member statements. We also only need to consider statements defining $u_i.r_j$, where u_i is a user in γ and r_j is a role in γ. Consider the set of all statements in ${\gamma^T}'$ having the form $u_i.r_j \longleftarrow u_k$. For each such statement, we perform the following operation on the RBAC state, starting from γ, have u_i assign u_k to the role r_j. Such an operation may not succeed either because u_i is not in the right administrative role or because u_k does not satisfy the required precondition. We repeat to perform all operations that could be performed. That is, we loop through all such statements and repeat the loop whenever the last loop results in a new successful assignment. Let γ' be the resulting RBAC state. It is not difficult to see that γ' implies the same role memberships as ${\gamma^T}'$, using arguments similar to those used above. □

Appendix 2. Proof for Proposition 4

PROOF. *For Assertion 1:* A state change in AAR occurs when a user assignment or a revocation operation is successfully performed. Given any RBAC state γ' such that $\gamma \overset{*}{\mapsto}_{\psi} \gamma'$, let $\gamma_0, \gamma_1, \ldots, \gamma_m$ be RBAC states such that $\gamma = \gamma_0 \mapsto_{\psi} \gamma_1 \mapsto_{\psi} \cdots \mapsto_{\psi} \gamma_m = \gamma'$. We construct a sequence of RT[$\twoheadleftarrow$, $\cap$] states ${\gamma^T}_0, {\gamma^T}_1, \ldots, {\gamma^T}_m$ as follows: ${\gamma^T}_0 = \gamma^T$; for each $i = [0..m-1]$, consider the operation that changes γ_i to γ_{i+1}. If it is an assignment operation in which a

user u_1 adds (u,r) to the user-role assignment relation, the state γ_{i+1}^T is obtained by adding $\mathsf{Sys}.r \longleftarrow u$ and $\mathsf{BSys}.r \longleftarrow u$ to $\gamma_i{}^T$. For each revocation that revokes a user u from a role r, we remove (if they exist) from the RT$[\twoheadleftarrow, \cap]$ state, the statements $\mathsf{ASys}.r \longleftarrow u$ and $RSys.r \longleftarrow u$. Let $\gamma^{T\prime}$ be γ_m^T.

Step 1: Prove that if $\gamma' \vdash q$ then $\gamma^{T\prime} \vdash q^T$. *Step 1a:* We prove that in $\gamma^{T\prime}$, $\mathsf{HSys}.r$ captures all users that are ever a member of the role r at some time, i.e., for each $i \in [0..m]$, if $u \in \mathsf{users}_{\gamma_i}[r]$, then u is a member of the RT$[\twoheadleftarrow, \cap]$ role $\mathsf{HSys}.r$ in γ_m^T $\left(SP(\gamma^T{}_m) \models m(\mathsf{HSys}, r, u)\right)$. We prove this by induction on i. The basis ($i = 0$) is true, since in γ^T we reproduce UA and RH in the definition of the HSys roles (see lines 60–64 in Figure 5); furthermore, the HSys roles never shrink. For the step, we show that if $(u,r) \in UA_{i+1}$, then u is a member of the RT$[\twoheadleftarrow, \cap]$ role $\mathsf{HSys}.r$ in γ_m^T. This is sufficient for proving the induction hypothesis because the effect of propagation through role hierarchy is captured by the definition of HSys roles. If $(u,r) \in UA_{i+1}$, then either $(u,r) \in UA$ (in which case $\mathsf{HSys}.r \longleftarrow u \in \gamma^{T\prime}$), or there is an assignment operation that assigns u to r (in which case $\mathsf{BSys}.r \longleftarrow u \in \gamma^{T\prime}$). Let $(r_a, c, r) \in can\_assign$ be an administrative rule used for this assignment, then in γ_i, the user u satisfies c. By induction hypothesis u's role memberships in γ_i is captured in u's role memberships in $\mathsf{HSys}.r$; therefore, u would satisfy the translated precondition $tmpRole$. Therefore, u is a member of the role $\mathsf{HSys}.r$ in γ_m^T (because of the statement $\mathsf{HSys}.u \longleftarrow \mathsf{BSys}.r \cap tmpRole$).

Step 1b: We prove that in $\gamma^{T\prime}$ the Sys roles capture all the role memberships in γ'. It is sufficient to prove the following: let UA' be the user assignment relation in γ'. If $(u,r) \in UA'$, then u is a member of the role $\mathsf{Sys}.r$ in $\gamma^{T\prime}$. If $(u,r) \in UA$, then either $(u,r) \in UA$ and this is never revoked (in which case $\mathsf{RSys}.r \longleftarrow u \in \gamma^T$ and this statement is never removed, therefore $\mathsf{RSys}.r \longleftarrow u \in \gamma^{T\prime}$), or there is an assignment operation in C, and this assignment is not revoked after it (in which case $\mathsf{ASys}.r \longleftarrow u \in \gamma^{T\prime}$).

Step two: Prove that if $\gamma^{T\prime} \vdash q^T$, then $\gamma' \vdash q$. It is sufficient to show that if an RT$[\twoheadleftarrow, \cap]$ role membership is implied by $\gamma^{T\prime}$, then the corresponding RBAC role membership (or permission possession) is also implied. A detailed proof uses induction on the number of rounds in which a bottom-up datalog evaluation algorithm outputs a ground fact. Here, we only point out the key observation. A RT$[\twoheadleftarrow, \cap]$ role membership is proved by statements generated on lines 60–65 or 71–74. The first three cases correspond to the UA, RH, PA. For the last case, there must exist a statement $\mathsf{ASys}.r \longleftarrow u$ in $\gamma^{T\prime}$ and it implies that u is a member of the role $\mathsf{Sys}.r$. By the construction of $\gamma^{T\prime}$, the user u has been assigned to the role r during the changes leading to γ' and the assignment is not revoked after that.

We only need to also consider statements defining $u_i.r_j$, where u_i is a user in γ and r_j is a role in γ. Consider the set of all statements in $\gamma^{T\prime}$ having the form $u_i.r_j \longleftarrow u_k$. For each such statement, we perform the following operation on the RBAC state, starting from γ, have u_i assign u_k to the role r_j. Such an operation may not succeed either because u_i is not in the right administrative role or because u_k does not satisfy the required precondition. We repeatedly perform all operations that could be performed. That is, we loop through all

such statements and repeat the loop whenever the last loop results in a new successful assignment. Let γ' be the resulting RBAC state. It is not difficult to see that γ' implies the same role memberships as $\gamma^{T\prime}$, using arguments similar to those used above.

For Assertion 2: Among the RT[$\leftarrow, \cap$] roles, Sys roles and HSys roles are fixed; ASys roles can grow or shrink; RSys roles can shrink but not grow; BSys roles can grow but not shrink. Given an RT[$\leftarrow, \cap$] state $\gamma^{T\prime}$ such that $\gamma^T \overset{*}{\mapsto}_{\psi^T} \gamma^{T\prime}$, we can assume without loss of generality that $\gamma^{T\prime}$ adds to γ^T only simple member statements. Consider the set of all statements in $\gamma^{T\prime}$ defining ASys, BSys, and RSys roles. We construct the RBAC state γ' as follows. (1) For every statement BSys.$r \longleftarrow u$ in $\gamma^{T\prime}$, assign the user u to the role r. Repeat through all such statements until no new assignment succeeds. Using arguments similar to those used for proving assertion 1, it can be shown that now the RBAC roles have the same memberships as the HSys roles. (2) Do the same thing for all the ASys.$r \longleftarrow u$ statements. At this point, all the role memberships for the Sys roles in $\gamma^{T\prime}$ are replicated in the RBAC roles, because all the HSys memberships have been added. (3) Remove the extra role membership in the RBAC state, i.e., those not in the Sys roles. The ability to carry out this step depends upon the requirement (in Definition 4) that if there is a *can_assign* rule for a role, then there is also revoke rule for the role. □

ACKNOWLEDGMENTS

This work was supported by an NSF CAREER grant (CNS-0448204) and an NSF ITR grant (CCR-0325951), and by sponsors of CERIAS. We also thank the anonymous reviewers for their helpful comments.

REFERENCES

AHN, G.-J. AND SANDHU, R. S. 2000. Role-based authorization constraints specification. *ACM Transactions on Information and System Security 3*, 4 (Nov.), 207–226.

CRAMPTON, J. 2002. Authorizations and antichains. Ph.D. thesis, Birbeck College, University of London, UK.

CRAMPTON, J. 2003. Specifying and enforcing constraints in role-based access control. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies (SACMAT 2003)*. Como, Italy. 43–50.

CRAMPTON, J. AND LOIZOU, G. 2003. Administrative scope: A foundation for role-based administrative models. *ACM Transactions on Information and System Security 6*, 2 (May), 201–231.

FERRAIOLO, D. F., SANDHU, R. S., GAVRILA, S., KUHN, D. R., AND CHANDRAMOULI, R. 2001. Proposed NIST standard for role-based access control. *ACM Transactions on Information and Systems Security 4*, 3 (Aug.), 224–274.

FERRAIOLO, D. F., CHANDRAMOULI, R., AHN, G.-J., AND GAVRILA, S. 2003. The role control center: Features and case studies. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*.

GAREY, M. R. AND JOHNSON, D. J. 1979. *Computers And Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA.

GRAHAM, G. S. AND DENNING, P. J. 1972. Protection—principles and practice. In *Proceedings of the AFIPS Spring Joint Computer Conference*. Vol. 40. AFIPS Press, Montvale, N.J. 417–429.

HARRISON, M. A., RUZZO, W. L., AND ULLMAN, J. D. 1976. Protection in operating systems. *Communications of the ACM 19*, 8 (Aug.), 461–471.

JAEGER, T. AND TIDSWELL, J. E. 2001. Practical safety in flexible access control models. *ACM Transactions on Information and System Security 4*, 2 (May), 158–190.

KOCH, M., MANCINI, L. V., AND PARISI-PRESICCE, F. 2002a. Decidability of safety in graph-based models for access control. In *Proceedings of the Seventh European Symposium on Research in Computer Security (ESORICS 2002)*. Springer, New York. 229–243.

KOCH, M., MANCINI, L. V., AND PARISI-PRESICCE, F. 2002b. A graph-based formalism for RBAC. *ACM Transactions on Information and System Security 5*, 3 (Aug.), 332–365.

KOCH, M., MANCINI, L. V., AND PARISI-PRESICCE, F. 2004. Administrative scope in the graph-based framework. In *Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies (SACMAT 2004)*. 97–104.

LAMPSON, B. W. 1971. Protection. In *Proceedings of the 5th Princeton Conference on Information Sciences and Systems*. Reprinted in *ACM Operating Systems Review 8*, 1, 18–24 (Jan 1974).

LI, N. AND TRIPUNITARA, M. V. 2004. Security analysis in role-based access control. In *Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies (SACMAT 2004)*. 126–135.

LI, N., WINSBOROUGH, W. H., AND MITCHELL, J. C. 2003. Distributed credential chain discovery in trust management. *Journal of Computer Security 11*, 1 (Feb.), 35–86.

LI, N., MITCHELL, J. C., AND WINSBOROUGH, W. H. 2002. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, Washington, DC. 114–130.

LI, N., MITCHELL, J. C., AND WINSBOROUGH, W. H. 2005. Beyond proof-of-compliance: Security analysis in trust management. *Journal of the ACM 52*, 3 (May), 474–514. (Preliminary version appeared in *Proceedings of 2003 IEEE Symposium on Security and Privacy*.)

LIPTON, R. J. AND SNYDER, L. 1977. A linear time algorithm for deciding subject security. *Journal of the ACM 24*, 3, 455–464.

MUNAWER, Q. AND SANDHU, R. S. 1999. Simulation of the augmented typed access matrix model (ATAM) using roles. In *Proceedings of INFOSECU99 International Conference on Information and Security*.

OH, S. AND SANDHU, R. S. 2002. A model for role admininstration using organization structure. In *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies (SACMAT 2002)*.

PARK, J. AND SANDHU, R. S. 2004. The $UCON_{ABC}$ usage control model. *ACM Transactions on Information and System Security 7*, 128–174.

SANDHU, R. S. 1988. The schematic protection model: Its definition and analysis for acyclic attenuating systems. *Journal of the ACM 35*, 2, 404–432.

SANDHU, R. S. 1992. The typed access matrix model. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, Washington, DC. 122–136.

SANDHU, R. S., COYNE, E. J., FEINSTEIN, H. L., AND YOUMAN, C. E. 1996. Role-based access control models. *IEEE Computer 29*, 2 (Feb.), 38–47.

SANDHU, R. S., BHAMIDIPATI, V., AND MUNAWER, Q. 1999. The ARBAC97 model for role-based administration of roles. *ACM Transactions on Information and Systems Security 2*, 1 (Feb.), 105–135.

SCHAAD, A., MOFFETT, J., AND JACOB, J. 2001. The role-based access control system of a European bank: A case study and discussion. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*. ACM Press, New York. 3–9.

TRIPUNITARA, M. V. AND LI, N. 2004. Comparing the expressive power of access control models. In *Proceedings of 11th ACM Conference on Computer and Communications Security (CCS-11)*. ACM Press, New York. 62–71.

ZHANG, X., PARK, J., PARISI-PRESICCE, F., AND SANDHU, R. S. 2004. A logical specification for usage control. In *Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies (SACMAT 2004)*.

ZHANG, X., PARISI-PRESICCE, F., SANDHU, R. S., AND PARK, J. 2005. Formal model and policy specification of usage control. *ACM Transactions on Information and System Security 8*, 351–387.

Received November 2004; revised March 2006; accepted June 2006