Security Analysis in Role-Based Access Control

Ninghui Li
ninghui@cs.purdue.edu

Mahesh V. Tripunitara
tripunit@cerias.purdue.edu

Center for Education and Research in Information Assurance and Security
and Department of Computer Sciences
Purdue University
656 Oval Drive, West Lafayette, IN 47907

ABSTRACT

Delegation is often used in administrative models for Role-Based Access Control (RBAC) systems to decentralize administration tasks. 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 also show that two classes of problems in the family can be reduced to similar analysis in the RT_0 trust-management language, thereby 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 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

Keywords

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 European bank, resulted in an RBAC system that has around 40,000 users and 1300 roles [22]. In systems of such size, it is impossible for a single system security officer (SSO) to administer the entire system. Several administrative models for RBAC have been proposed in recent years, e.g., ARBAC97 [18], ARABC02 [17], and CL03 (Crampton and Loizou) [4]. 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 [13] 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 [8, 10, 15, 19].
- We show that two classes of the security analysis problems in RBAC can be reduced to similar ones in RT_0, a role-based trust-management language for which security analysis has been studied [13]. The reduction gives efficient algorithms for answering most kinds of queries in these two classes and establishes the complexity bounds for the intractable cases.

SACMAT'04, June 2–4, 2004, Yorktown Heights, New York, USA.

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. Related work is discussed in Section 3. We gave an overview of the results for security analysis in RT_0 in Section 4 and present the reduction from security analysis in RBAC to that in RT_0 in Section 5. We conclude with Section 6. An appendix contains proofs not included in the main body.

2. PROBLEM DEFINITION AND MAIN RESULTS

In [13], 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 modelled 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 $\vdash$: $\Gamma \times Q \to \{true, false\}$ 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.

Definition 2. (*Security Analysis in an Abstract Setting*) Given an access control scheme $\langle \Gamma, Q, \vdash, \Psi \rangle$, a security analysis instance takes the form $\langle \gamma, q, \psi, \Pi \rangle$, where $\gamma \in \Gamma$ is a state, $q \in Q$ 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).

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 [6, 21] for an introduction to RBAC. We assume that there are three countable sets: U (the set of all possible users), R (the set of all possible roles), and P (the set of all possible permissions).

Definition 3. (*A Family of RBAC Security Analysis Problems*) This family 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.

States (Γ): An RBAC state γ is a 3-tuple $\langle UA, PA, RH \rangle$, in which the user assignment relation $UA \subseteq U \times R$ associates users with roles, the permission assignment relation $PA \subseteq P \times R$ associates permissions with roles, and the role hierarchy relation $RH \subseteq R \times R$ is a partial order among roles in R. We denote the partial order by $\succeq$. $r_1 \succeq r_2$ 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.

Given a state γ, 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 $\mathsf{users}_\gamma : R \cup P \to 2^U$, as follows. For any $r \in R$ and $u \in U$, $u \in \mathsf{users}_\gamma[r]$ if and only if either $(u, r) \in UA$ or there exists r_1 such that $r_1 \succeq r$ and $(u, r_1) \in UA$. 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 $\mathsf{users}_\gamma[r_1]$.

Queries (Q): 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) $R \cup P \subseteq S$, i.e., every role r and every permission p is a user set; (2) $\{u_1, u_2, \cdots, u_k\} \in S$, 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 $\mathsf{users}_\gamma : S \to 2^U$ is defined as follows: $\mathsf{users}_\gamma[\{u_1, u_2, \cdots, u_k\}] = \{u_1, u_2, \cdots, u_k\}$, $\mathsf{users}_\gamma[(s)] = \mathsf{users}_\gamma[s]$, $\mathsf{users}_\gamma[s_1 \cup s_2] = \mathsf{users}_\gamma[s_1] \cup \mathsf{users}_\gamma[s_2]$, 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 *semi-static* if one of s_1, s_2 can be evaluated independent of the state, i.e., no role or permission appears in it. We distinguish semi-static queries because they are easier to answer.

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 $\mathsf{users}_\gamma[s_1] \supseteq \mathsf{users}_\gamma[s_2]$.

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 Definition 3, we leave the state-change rule abstract for the following reasons. First, there are several competing proposals for constraint languages [1, 9, 3] and for administrative models in RBAC [18, 17, 4, 5]; 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* E.g., is $r_1 \sqsupseteq \{u_1\}$ possible? This asks whether there exists a reachable state in which the (presumably untrusted) user u_1 becomes a member of r_1. A 'no' answer means that the system is safe.

- *Simple Availability* E.g., is $p_1 \sqsupseteq \{u_1\}$ necessary? This asks whether in every reachable state, the (presumably trusted) user u_1 always has the permission p_1. A 'yes' answer means that the permission p_1 is always available to the user u_1.
- *Bounded Safety* E.g., is $\{u_1, u_2, u_3\} \sqsupseteq (p_1 \cap p_2)$ necessary? This asks whether in every reachable state, only users in the set $\{u_1, u_2, u_3\}$ have both the permissions p_1 and p_2. A 'yes' answer means that the system is safe.
- *Liveness* E.g., is $\emptyset \sqsupseteq p_1$ possible? This asks whether the permission p_1 is always accessible to at least one user. A 'no' answer means that the liveness of the permission p_1 holds in the system.
- *Mutual Exclusion* E.g., is $\emptyset \sqsupseteq (r_1 \cap r_2)$ necessary? This asks whether in every reachable state, no user is a member of both r_1 and r_2. A 'yes' answer means that the two roles are mutually exclusive.
- *Containment* E.g., is $r_1 \sqsupseteq p_1$ necessary? This asks whether in every reachable state, every user who has the permission p_1 (e.g., has access to an internal document) is a member of the role r_1 (e.g., is an employee). This example expresses a safety property. A 'yes' answer means that the safety property holds.

  Containment can also express availability properties, e.g., "is $p_1 \sqsupseteq r_1$ necessary?" asks whether the permission p_1 is always available to members of the role r_1. A 'yes' answer means that the availability property holds.

Observe that all the above examples (except for containment) use semi-static queries. We distinguish semi-static queries from other queries as they are easier to answer.

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, a security requirement may be that only employees may access a sensitive document. This can be formalized as an instance $\langle \gamma, q, \psi, \forall \rangle$, where γ is the current state, q is $Employees \sqsupseteq p$ where p is the permission to access the sensitive document, $Employees$ is the role that contains all employees of an organization, 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 p 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 solve two classes of security analysis problems in RBAC. Both classes use variants of the URA97 component of the ARBAC97 administrative model for RBAC [18]. 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 4. (*Assignment And Trusted Users – AATU*) The class AATU is given by parameterizing the family of RBAC analysis problems in Definition 3 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).

Main results for AATU

- If q is semi-static (see Definition 3), 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**-hard).

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 5. (*Assignment And Revocation – AAR*) The class AAR is given by parameterizing the family of RBAC analysis problems in Definition 3 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 pre-condition 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 $(r_a, c, rset) \in can\_revoke$ 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) None of the administrative roles is empty. (3) If a can_assign tuple exists for a role, then a can_revoke tuple also exists for that role.

Main results for AAR

- If q is semi-static (see Definition 3), 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 usage of can_assign and can_revoke is inspired by URA97, which is one of the three components of ARBAC97 [18]. The state-change rules considered in AAR are similar to those in URA97, but they differ in the following two ways. One, URA97 allows negation of roles to be used in a precondition; AAR does not allow this. Two, 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. One, AATU does not have revocation rules. Two, 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. It has been observed that permission-role assignment and the role hierarchy are changed less often than user-role assignment. As the user-role relation is the component that changes most frequently, we expect that user-role assignment is the most likely to be decentralized.

AATU and AAR represent two basic cases of security analysis in RBAC. Although we believe that they are useful cases, 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. 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. [8] in the context of the well-known access matrix model [11, 7], and was shown to be undecidable in the HRU model [8]. 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 [15], the schematic protection model [19], and the typed access matrix model [20].

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 pre-conditions 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. [13] proposed the notion of security analysis, in which they study queries other than simple safety. They study security analysis in the context of Trust Management (TM). Although RT_0, the TM language studied in [13], supports delegation and is more expressive than the access matrix model in certain ways, and the kinds of analysis include problems other than simple safety analysis, somewhat surprisingly, all the security analysis problems considered there are decidable; furthermore, most problems are efficiently decidable.

Munawer and Sandhu [16] presented a simulation of the Augmented Typed Access Matrix Model (ATAM) in an RBAC model. In the simulation, they use an administrative model that is far more powerful than ARBAC97 or any other administrative model considered in the literature. In particular, they assume the existence of administrative permissions each of which can simulate the effect of an ATAM command. An ATAM command is more general than an HRU command. It checks the existence and nonexistence of rights

in the cells corresponding to subjects and objects specified by the parameters, and if all conditions are satisfied, executes a sequence of operations, such as entering a new right in a cell.

Crampton and Loizou [4] claim that "if administrative (or control) permissions are assigned to subjects, then the safety problem is undecidable. Indeed, Munawer and Sandhu [16] and Crampton [2] have shown independently that the safety problem for RBAC96 is undecidable." We disagree with this claim, as 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 [16] suggests only that when an overly complicated administrative model is added to RBAC96, safety analysis may be undecidable.

The work by Koch et al [10] considers safety in RBAC with the RBAC state and state change rules posed as a graph formalism. They show that safety in RBAC is decidable provided that a state change rule does not both remove and add components to the graph that represents the protection state. The administrative model (set of state change rules) considered in that work is limited in that it is expressed in terms of the types of nodes and edges in the graph. Consequently, it is not powerful enough to allow constructs such as pre-conditions involving user-role memberships. Such pre-conditions are part of ARBAC97 [18] and the administrative models we consider in this paper. Also, our work differs from that work in that we consider a more general class of queries than safety, and we provide specific algorithms and complexity bounds.

Previous work on ensuring security properties in RBAC takes the approach of using constraints [1, 3, 9]. 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 evaluated each time a state changes.

4. OVERVIEW OF SECURITY ANALYSIS IN $\mathsf{RT}[\leftarrow, \cap]$

In [13], Li et al. study security analysis in the context of the RT family of Role-based Trust-management languages [12, 14]. In particular, security analysis in $\mathsf{RT}[\leftarrow, \cap]$ and its sub-languages is studied. $\mathsf{RT}[\leftarrow, \cap]$ is a slightly simplified (yet expressively equivalent) version of the RT_0 language introduced in [14]. In this section we summarize the results for security analysis in $\mathsf{RT}[\leftarrow, \cap]$. In Section 5 we reduce security analysis in AATU and AAR to that in $\mathsf{RT}[\leftarrow, \cap]$.

Syntax of $\mathsf{RT}[\leftarrow, \cap]$ The most important concept in the RT languages is also that of *roles*. A role in $\mathsf{RT}[\leftarrow, \cap]$ 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 which is a set of principals that are members of the role.

Each principal K has the authority to designate the members of each role of the form $K.r$. Roles are defined by *statements*. Figure 1 shows the four types of statements in $\mathsf{RT}[\leftarrow, \cap]$; 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 $K_2.r_2$.

States An $\mathsf{RT}[\leftarrow, \cap]$ state γ^T consists of a set of $\mathsf{RT}[\leftarrow, \cap]$ statements. The semantics of $\mathsf{RT}[\leftarrow, \cap]$ 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 1 for the datalog clauses corresponding to $\mathsf{RT}[\leftarrow, \cap]$ 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 $SP(\gamma^T)$.

State-change Rules A state-change rule has 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*).

Queries Li et al. [13] 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, $\gamma^T \vdash A.r \sqsupseteq \{D_1, \ldots, D_n\}$ 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 $\{Z \mid SP(\gamma^T) \models m(X, u, Z)\} \supseteq \{Z \mid SP(\gamma^T) \models m(A, r, Z)\}$.

Simple Member	syntax:	$K.r \longleftarrow K_1$	
	meaning:	$\mathsf{members}(K.r) \supseteq \{K_1\}$	
	LP clause:	$m(K, r, K_1)$	$(m1)$
Simple Inclusion	syntax:	$K.r \longleftarrow K_1.r_1$	
	meaning:	$\mathsf{members}(K.r) \supseteq \mathsf{members}(K_1.r_1)$	
	LP clause:	$m(K, r, ?Z) :- m(K_1, r_1, ?Z)$	$(m2)$
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:	$m(K, r, ?Z) :- m(K, r_1, ?Y),\ m(?Y, r_2, ?Z)$	$(m3)$
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:	$m(K, r, ?Z) :- m(K_1, r_1, ?Z),\ m(K_2, r_2, ?Z)$	$(m4)$

Figure 1: Statements in $\mathsf{RT}[\leftarrow, \cap]$. 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:

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 $\{\} \sqsupseteq B.u_1$.

Main results for security analysis in $\mathsf{RT}[\leftarrow, \cap]$

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 [13] uses logic programs to derive answers to those security analysis problems. This approach exploits the fact that $\mathsf{RT}[\leftarrow, \cap]$ 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 [13], only the $\forall$ case (i.e., whether an inclusion query is necessary) is studied. It is not clear what the security intuition is of an $\exists$ inclusion query (whether an inclusion query is possible); therefore, it is not studied in [13]. 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 $\mathsf{RT}[\,]$, the language that allows only simple member and simple inclusion statements, containment analysis is in $\mathbf{P}$. It becomes more complex when additional policy language features are used. Containment analysis is $\mathbf{coNP}$-complete for $\mathsf{RT}[\cap]$ ($\mathsf{RT}[\,]$ plus intersection inclusion statements), $\mathbf{PSPACE}$-complete for $\mathsf{RT}[\leftarrow]$ ($\mathsf{RT}[\,]$ plus linking inclusion statements), and decidable in $\mathbf{coNEXP}$ for $\mathsf{RT}[\leftarrow, \cap]$.

5. REDUCING AATU AND AAR TO SECURITY ANALYSIS IN $\mathsf{RT}[\leftarrow, \cap]$

In this section, we solve AATU (Definition 4) and AAR (Definition 5). Our approach is to reduce each of them to security analysis in $\mathsf{RT}[\leftarrow, \cap]$.

5.1 Reduction for AATU

The reduction algorithm AATU_Reduce is given in Figure 3; it uses the subroutines defined in Figure 2. 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 $\mathsf{RT}[\leftarrow, \cap]$ 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 $\mathsf{Sys}.r$ represents the RBAC role r and the RT role $\mathsf{Sys}.p$ represents the RBAC permission p. Each $(u, r) \in UA$ is translated into the RT statement $\mathsf{Sys}.r \longleftarrow u$. Each $r_1 \succeq r_2$ is translated into the RT statement $\mathsf{Sys}.r_2 \longleftarrow \mathsf{Sys}.r_1$ (as r_1 is senior 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-restricted and s-restricted). In addition, for each trusted user u in T, all the roles starting with u is 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). 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.

The following proposition shows 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 RT 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 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 A.1 for the proof.

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

Figure 2: Subroutines Trans, QTrans, and HTrans: They are used by the two reduction algorithms. We assume call-by-reference for the parameter γ^T.

```
31 AATU_Reduce (⟨ γ = ⟨UA, PA, RH⟩,  q = s1 ⊒ s2,  ψ = ⟨can_assign, T⟩ ⟩)
32 {
33   /* Reduction algorithm for the first class of analysis problems */
34   γ^T = ∅;  q^T = QTrans(s1,γ^T) ⊒ QTrans(s2,γ^T);
35   foreach (ui, rj) ∈ UA { γ^T += Sys.rj ⟵ ui; }
36   foreach (ri, rj) ∈ RH { γ^T += Sys.rj ⟵ Sys.ri; }
37   foreach (pi, rj) ∈ PA { γ^T += Sys.pi ⟵ Sys.rj; }
38   foreach (ai, s, rset) ∈ can_assign {
39     tmpRole=Trans(s,γ^T);
40     foreach r ∈ rset {
41       name=newName();  γ^T += Sys.name ⟵ Sys.ai.r; γ^T += Sys.r ⟵ Sys.name ∩ tmpRole;  } }
42   foreach RT role name x appearing in γ^T {
43     G+=Sys.x;  S+=Sys.x;  foreach user u ∈ T { G+=u.x; } }
44   return ⟨γ^T, q^T, (G, S)⟩;
45 } /* End AATU_Reduce */
```

Figure 3: Reduction Algorithm for AATU

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 semi-static. The general AATU problem is* **coNP***-hard.*

PROOF. Sketch: Follows directly from Proposition 1 and the results on security analysis in RT$[\leftarrow, \cap]$. Observe that AATU_Reduce runs in time polynomial in the size of the input. The **coNP**-hardness of the problem can be shown by reducing the monotone 3SAT problem to the AAR problem. The proof is similar to the proof for **coNP**-hardness of security analysis in RT$[\cap]$ (Section A.3 of [13]). In summary, the monotone 3SAT problem can be reduced to determining whether a propositional formula having the form $\phi_1 \Rightarrow \phi_2$ is *not* valid, where ϕ_1 and ϕ_2 are propositional formulas constructed using conjunction and disjunction. Such a formula can be encoded using a query in a containment analysis. In fact, the AAR problem remains **coNP**-hard even when no precondition occurs in can_assign; the expressive power of the queries is sufficient for reducing the monotone 3SAT problem. □

5.2 Reduction for AAR

The reduction algorithm for AAR is given in Figure 4. 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 [18] 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.

The following proposition shows that the reduction is sound.

PROPOSITION 3. *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 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 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 $\gamma^{T\prime} \vdash q^T$.*

THEOREM 4. *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 semi-static. The general AAR problem is* **coNP***-complete.*

PROOF. Sketch: Follows directly from Proposition 3 and the results on security analysis in RT$[\cap]$. Observe that AAR_Reduce runs in time polynomial in the size of the input, and the result is an instance of security analysis in RT$[\cap]$, which is **coNP**-complete. This shows that the general AAR problem is in **coNP**. That AAR is **coNP**-hard can be proved using arguments similar to those for AATU. □

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_0 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.

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.

Acknowledgements

Portions of this work were supported by NSF ITR and by sponsors of CERIAS. We thank the anonymous reviewers for their helpful comments. We also thank Ji-Won Byun, Ziad El Bizri, Jiantao Li and Klorida Miraj for their reviews and suggestions on presentation.

7. REFERENCES

[1] G.-J. Ahn and R. Sandhu. Role-based authorization constraints specification. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):207–226, Nov. 2000.

[2] J. Crampton. *Authorizations and Antichains*. PhD thesis, Birbeck College, University of London, UK, 2002.

[3] J. Crampton. Specifying and enforcing constraints in role-based access control. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies (SACMAT 2003)*, pages 43–50. June 2003.

[4] J. Crampton and G. Loizou. Administrative scope: A foundation for role-based administrative models. *ACM Transactions on Information and System Security (TISSEC)*, 6(2):201–231, May 2003.

[5] D. Ferraiolo, G.-J. Ahn, and S. Gavrila. The role control center: Features and case studies. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies*, June 2003.

[6] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and Systems Security (TISSEC)*, 4(3):224–274, Aug. 2001.

[7] G. S. Graham and P. J. Denning. Protection — principles and practice. In *Proceedings of the AFIPS Spring Joint Computer Conference*, volume 40, pages 417–429. May 16–18 1972.

[8] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, Aug. 1976.

[9] T. Jaeger and J. E. Tidswell. Practical safety in flexible access control models. *ACM Transactions on Information and System Security (TISSEC)*, 4(2):158–190, May 2001.

[10] M. Koch, L. V. Mancini, and F. Parisi-Presicce. Decidability of safety in graph-based models for access control. In *Proceedings of the Seventh European Symposium on*

```
51 AAR_Reduce (⟨ γ = ⟨UA, PA, RH⟩,  q = s1 ⊒ s2,  ψ = ⟨can_assign, can_revoke⟩ ⟩)
52 { /* Reduction for AAR */
53   γ^T = ∅;  q^T = QTrans(s1,γ^T) ⊒ QTrans(s2,γ^T);
54   foreach (ui, rj) ∈ UA { γ^T += HSys.rj ⟵ ui; γ^T += RSys.rj ⟵ ui;
55         γ^T += Sys.rj ⟵ RSys.rj; }
56   foreach (ri, rj) ∈ RH { γ^T += Sys.rj ⟵ Sys.ri; γ^T += HSys.rj ⟵ HSys.ri; }
57   foreach (pi, rj) ∈ PA { γ^T += Sys.pi ⟵ Sys.rj; }
58   foreach (ai, s, rset) ∈ can_assign {
59     if (s==true) {
60       foreach r ∈ rset { γ^T += HSys.r ⟵ BSys.r; γ^T += Sys.r ⟵ ASys.r; } }
61     } else { tmpRole = HTrans(s,γ^T);  /* precondition */
62       foreach r ∈ rset
63         { γ^T += HSys.r ⟵ BSys.r ∩ tmpRole; γ^T += Sys.r ⟵ ASys.r ∩ tmpRole; }
64   } }
65   foreach RT role name x appearing in γ^T {
66     G+=Sys.x; S+=Sys.x; G+=HSys.x; S+=HSys.x; G+=RSys.x; S+=BSys.x;
67     S+=RSys.x; S+=ASys.x;
68   } /* when a can_revoke rule exists for r, ASys.r and RSys.r can shrink */
69   foreach (ai, rset) ∈ can_revoke { foreach r in rset { S-=RSys.r; S-=ASys.r; } }
70   return ⟨γ^T, q^T, (G, S)⟩;
71 } /* End AAR_Reduce */
```

Figure 4: AAR_Reduce**: the reduction algorithm for AAR**

Research in Computer Security (ESORICS 2002), pages 229–243. Springer, Oct. 2002.

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

[12] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130. May 2002.

[13] N. Li, W. H. Winsborough, and J. C. Mitchell. Beyond proof-of-compliance: Safety and availability analysis in trust management. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 123–139. May 2003.

[14] N. Li, W. H. Winsborough, and J. C. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, Feb. 2003.

[15] R. J. Lipton and L. Snyder. A linear time algorithm for deciding subject security. *Journal of the ACM*, 24(3):455–464, 1977.

[16] Q. Munawer and R. Sandhu. Simulation of the augmented typed access matrix model (ATAM) using roles. In *Proceedings of INFOSECU99 International Conference on Information and Security*, 1999.

[17] S. Oh and R. Sandhu. A model for role admininstration using organization structure. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies (SACMAT'02)*, June 2002.

[18] R. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for role-based aministration of roles. *ACM Transactions on Information and Systems Security (TISSEC)*, 2(1):105–135, Feb. 1999.

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

[20] R. S. Sandhu. The typed access matrix model. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 122–136. May 1992.

[21] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[22] A. Schaad, J. Moffett, and J. Jacob. 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*, pages 3–9. 2001.

APPENDIX

A.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 $\gamma_0, \gamma_1, \cdots, \gamma_m$ 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 states $\gamma_0^T, \gamma_1^T, \cdots, \gamma_m^T$ 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 $\gamma^{T\prime}$ be γ_m^T.

Step one: Prove that if $\gamma' \vdash q$ then $\gamma^{T\prime} \vdash q^T$. 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 γ_i^T implies that u is a member of the RT 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 35–37 reproduces UA, RH, PA in the RT state γ_0^T. For the step, assumes that the induction hypothesis holds for $\gamma_0, \cdots, \gamma_i$, 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 $\mathsf{Sys}.r \longleftarrow \mathsf{Sys}.name \cap tmpRole$ (where $tmpRole$ corresponds to the precondition c). Furthermore, in γ^T_{i+1}, u_1 is a member of the role r_a and u satisfies the condition c. Therefore, u is a member of the $\mathsf{Sys}.r$ role in γ^T_{i+1}.

Step two: Prove that if $\gamma^{T\prime} \vdash q^T$ then $\gamma' \vdash q$. It is sufficient to show that if an RT 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 observations. (For details of similar proofs, see the Appendix in [13].) A RT role membership is proved by statements generated on lines 35–37 or 40–41. 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 $\gamma^{T\prime}$, 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'.

For Assertion 2: Given an RT 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. Also, we 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 $\gamma^{T\prime}$ 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 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 $\gamma^{T\prime}$; using arguments similar to those used above. □

A.2 Proof for Proposition 3

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 $\gamma_0, \gamma_1, \cdots, \gamma_m$ 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 states $\gamma^T_0, \gamma^T_1, \cdots, \gamma^T_m$ 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^T_{i+1} is obtained by adding $\mathsf{Sys}.r \longleftarrow u$ and $\mathsf{BSys}.r \longleftarrow u$ to γ^T_i. For each revocation that revokes a user u from a role r, we remove (if they exist) from the RT state the statements $\mathsf{ASys}.r \longleftarrow u$ and $RSys.r \longleftarrow u$. Let $\gamma^{T\prime}$ be γ^T_m.

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 role $\mathsf{HSys}.r$ in γ^T_m $\big(SP(\gamma^T_m) \models m(\mathsf{HSys}, r, u)\big)$. 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 54 and 56 in Figure 4); 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 role $\mathsf{HSys}.r$ in γ^T_m. 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^T_m (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 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 role membership is proved by statements generated on lines 55, 56, 57, or 63. 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.

Also, we 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 $\gamma^{T\prime}$ 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 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 $\gamma^{T\prime}$; using arguments similar to those used above.

For Assertion 2: Among the RT roles, Sys roles and HSys roles are fixed; ASys roles can grow or shrink; RSys roles can shrink but not grow; and BSys roles can grow but not shrink. Given an RT 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 $\mathsf{BSys}.r \longleftarrow u$ 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 $\mathsf{ASys}.r \longleftarrow u$ 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 5) that if there is a can_assign rule for a role, then there is also revoke rule for the role. □