A Modal Logic for Role-Based Access Control*

Thumrongsak Kosiyatrakul, Susan Older, and Shiu-Kai Chin

EECS Department, Syracuse University, Syracuse, New York 13244, USA
skchin@syr.edu

Abstract. Making correct access-control decisions is central to security, which in turn requires accounting correctly for the identity, credentials, roles, authority, and privileges of users and their agents. In networked systems, these decisions are made more complex because of delegation and differing access-control policies. Methods for reasoning rigorously about access control and computer-assisted reasoning tools for verification are effective for providing assurances of security. In this paper we extend the access-control logic of [11,1] to also support reasoning about role-based access control (RBAC), which is a popular technique for reducing the complexity of assigning privileges to users. The result is an access-control logic which is simple enough for design and verification engineers to use to assure the correctness of systems with access-control requirements but yet powerful enough to reason about delegations, credentials, and trusted authorities. We explain how to describe RBAC components such as user assignments, permission assignments, role inheritance, role activations, and users' requests. The logic and its extensions are proved to be sound and implemented in the HOL (Higher Order Logic version 4) theorem prover. We also provide formal support for RBAC's static separation of duty and dynamic separation of duty constraints in the HOL theorem prover. As a result, HOL can be used to verify properties of RBAC access-control policies, credentials, authority, and delegations.

1 Introduction

The ubiquitous use of inter-networked computers makes controlled access to information and services simultaneously essential and complex. Access is ultimately granted based on establishing a relationship between a principal and her privileges with respect to a particular object. In networked systems, requests and authority may be delegated. This complicates the task of establishing the identity and authority of principals behind access requests.

One interesting specialty logic for reasoning about access-control policies and decisions is the access-control logic of Abadi and colleagues [11,1]. This modal logic brings clarity and consistency to reasoning about access requests because it provides a formal model of principals, statements, credentials, authority, trust,

* Partially supported by the CASE Center at Syracuse University, a New York State Center for Advanced Technology supported by the New York State Office of Science Technology and Academic Research.

V. Gorodetsky, I. Kotenko, and V. Skormin (Eds.): MMM-ACNS 2005, LNCS 3685, pp. 179–193, 2005.

and delegations. However, this logic lacks the capability for specifying and reasoning about role-based access control (RBAC) [4,6]. RBAC policies are particularly well-suited for large-scale computing systems, because they reduce the administrative complexity of associating users with permissions by decoupling the two: users are authorized for *roles*, and permissions are assigned to roles. RBAC also supports a decentralized view of access control.

Our objective is to unify within a single logic the ability to describe and reason about access-control requests and decisions based on the relationships between principals, statements, and trusted authorities while accounting for credentials, delegation, and RBAC roles. We therefore extend the access-control logic of Abadi to encompass three major RBAC components: (1) user-role associations, (2) role-permission associations, and (3) role-inheritance relations. We express user-role associations as delegations (roles delegate their authority to users to act on their behalf); role-permission associations and role-inheritance relations are expressed as relations among principals and sets of statements by which certain statements of one principal may be attributed to another principal. With these extensions, we can (1) model RBAC policies within the access-control logic, and (2) reason about RBAC-based access-control decisions.

Other researchers have used modal logic for describing security policies and properties [8,2]. Those frameworks are more general than ours, but require a high level of sophistication on the part of users. Our objective was to identify a simple logic accessible to engineers that nonetheless could describe a wide variety of access-control concerns. Our experiences teaching the Abadi logic to computer science and engineering Master's students indicate that the logic meets those criteria [12,13].

To verify the soundness of the access-control logic and its extensions, we use the HOL (Higher Order Logic version 4) theorem prover [7,10]. Defining the access-control logic within HOL serves several purposes. First, HOL is used to verify the soundness of the access-control logic. Second, because HOL is an open system, all of our proofs can be easily checked by third parties. Finally, the existence of an executable and verifiable access-control logic implemented in HOL makes both the access-control logic and a means for formal verification available to design and verification engineers.

In addition to user-role associations, role-permission associations, and role hierarchies, RBAC allows the specification of constraints that prevent users from (1) being assigned to roles that are in conflict (static separation of duty), and (2) activating certain roles simultaneously (dynamic separation of duty). These constraints are outside the direct scope of the access-control logic, which focuses on access-control decisions for a specific policy. In contrast, the separation of duty constraints impose limits on what should be considered a well-defined or consistent policy in the first place. However, like the access-control logic, these constraints can be described and verified within the higher-order logic of HOL. Hence, we are able to use higher-order logic to verify the consistency of a specific RBAC policy prior to using the access-control logic to reason about access-control decisions based on that policy.

The rest of this paper is organized as follows. Section 2 provides a brief RBAC tutorial. Section 3 describes the syntax and and semantics of our logic, which builds on the work of Abadi and colleagues [11,1]. Section 4 explains how RBAC relations are described in our extended logic. Section 5 presents the HOL definitions of static and dynamic separation of duty constraints. Finally, our conclusions are in Section 6.

2 Role-Based Access Control (RBAC)

Role-Based Access Control (RBAC) [5,4,6] replaces direct user-permission associations in traditional access control through a combination of user-role and role-permission associations. Rather than assigning individuals specific permissions that may change as their duties and status change, an RBAC policy assigns users to roles and grants permissions to roles. In RBAC, an access request q made by a user U will be granted if and only if U is authorized to act in a role R that has been granted the permission q.

RBAC policies involve three essential entities: a set of users, a set of roles, and a set of permissions. RBAC also defines a set UA of user assignments and a set PA of permission assignments: $(U, R) \in UA$ means that user U has the right to act in role R, and $(p, R) \in PA$ means that permission p is assigned to role R.

2.1 Role Inheritance

RBAC also includes a partial order over roles called *role inheritance*. When role R_1 inherits role R_2, denoted $R_1 \succeq R_2$, every user U explicitly assigned to role R_1 is also implicitly assigned to role R_2; likewise, every permission p explicitly associated with role R_2 is implicitly associated with role R_1. The sets $authorized\_users(R)$ and $authorized\_permissions(R)$ define the authorized users and authorized permissions of a role R are given respectively:

$$\begin{aligned} &authorized\_users(R) = \\ &\qquad \{U \in USERS \mid \exists R' \in ROLES.\ (R' \succeq R) \wedge ((U, R') \in UA)\} \\ &authorized\_permissions(R) = \\ &\qquad \{p \in PRMS \mid \exists R' \in ROLES.\ (R \succeq R') \wedge ((p, R') \in PA)\}. \end{aligned}$$

From these definitions, it is straightforward to verify the following two properties:

1. If $R_1 \succeq R_2$, then $authorized\_users(R_1) \subseteq authorized\_users(R_2)$.
2. If $R_1 \succeq R_2$, then $authorized\_permissions(R_2) \subseteq authorized\_permissions(R_1)$.

2.2 Separation of Duty

RBAC also supports constraints such as separation of duty. Static separation of duty provides a way to specify mutually exclusive roles (i.e., roles that should never have authorized users in common). In RBAC, static separation of duty is

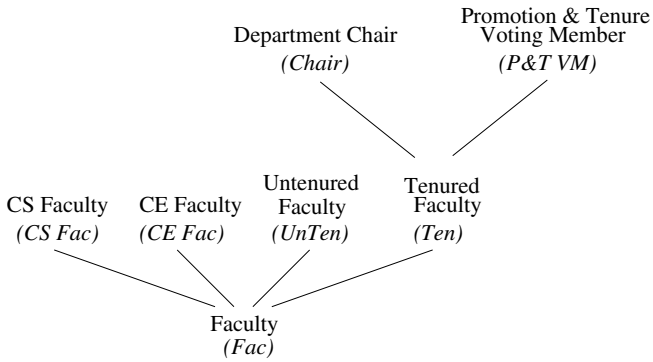

Fig. 1. Role Hierarchy Structure

represented by a set *SSD* of pairs (rs, n), where rs is a set of mutually exclusive roles and $n \geq 2$. When (rs, n) is in *SSD*, no users should be authorized to act in n or more of the roles in rs.

Note that static separation of duty constrains the role hierarchy as well as the user-role assignment *UA*. For example, if a user U is authorized to act in role R_1 and R_1 inherits R_2, U is also authorized to act in role R_2. Thus, both *UA* and $\succeq$ must be checked to ensure that they satisfy the *SSD* constraints.

Dynamic separation of duty constrains the combinations of roles that users may *activate* at any given instant, and is specified by a set *DSD* of pairs similar to *SSD*. When (rs, n) is in *DSD*, a user cannot have n or more roles in rs simultaneously activated. When a user activates a set of roles, the set of roles constitutes a *session*. The function *session_roles*(s) determines the set of activated roles associated with the session s. In an RBAC system, the role-activation monitor denies any role-activation requests that would violate the *DSD* constraints.

2.3 RBAC Example

As an example of an RBAC policy, consider a hypothetical academic department that houses both Computer Science (CS) and Computer Engineering (CE) programs. The department includes both tenured and untenured faculty, and every faculty member is associated with at least one of the two academic programs. In addition, the department has a chairperson and a Promotion & Tenure (P& T) committee. Thus, there are seven relevant roles for this example:

$$ROLES = \{Fac, Ten, UnTen, CS\ Fac, CE\ Fac, Chair, P\&T\ VM\}.$$

Figure 1 provides a Hasse diagram representing a plausible role-inheritance relation for this scenario (e.g., the roles *Chair* and *P&T VM* both inherit *Ten*).

The standard academic situation is that no one can be both tenured and untenured, and hence the roles *Ten* and *UnTen* should be mutually exclusive. Furthermore, the department's bylaws mandate that the chair cannot be a P&T

voting member. These constraints can be represented by the following static separation-of-duty relation:

$$SSD = \{(\{Ten, UnTen\}, 2), (\{P\&T\ VM, Chair\}, 2)\}.$$

Because the roles *Chair* and *P&T VM* both inherit the *Ten* role, these two constraints also prevent untenured faculty from being department chair and from being voting members of the P&T committee.

The department's bylaws also require the P&T Committee to contain a fixed number of representatives from each of the CS and CE programs. Thus, for the purposes of P&T deliberations, no faculty member can simultaneously represent both the CS and CE programs, although she may be associated with both programs. This constraint can be represented by the following dynamic separation-of-duty relation:

$$DSD = \{(\{CS\ Fac, CE\ Fac, P\&T\ VM\}, 3)\}.$$

Thus, no one may simultaneously act as CS faculty, CE faculty, and a P&T voting member, although they may authorized for all three roles and may act in any two of those roles simultaneously.

We have not explicitly given the user-role and permission-role assignments. However, suppose that the permission *read student grade reports* is associated with the faculty role *Fac* (i.e., $(read\ student\ grade\ reports, Fac) \in PA$), and that *Alice* is explicitly assigned to the role *Chair* (i.e., $(Alice, Chair) \in UA$). First of all, note that the *SSD* relation prohibits any user from being authorized for both the *Ten* and *UnTen* roles. Thus, the role hierarchy prevents *Alice* from being assigned to the *UnTen* role, as her assignment to *Chair* also implicitly authorizes her for the *Ten* role. Second, the role-inheritance relation also authorizes *Alice* to act in the role *Fac* ($Alice \in authorized\_users(Fac)$), and hence she is entitled to adopt either the *Fac* or *Chair* roles to *read student grade reports*.

Having described the key concepts of RBAC, we now introduce a modal logic for access control in which RBAC relationships can be described.

3 A Logic for Reasoning About Access Control

The access-control logic of Abadi and colleagues [11,1] incorporates a calculus of principals into a standard multi-agent modal logic. The result is a set of logical rules for manipulating formulas that provides a tool for reasoning about access control, delegation, and trust.

Principals are entities (e.g., people, machines, encryption keys, and processes) that make statements. Principals can be either a simple name (e.g., "*Alice*") or compound principals (e.g., "*Alice and Carol*"). Statements are the things that principals say, such as "read file *foo*" or "*Alice* can read file *foo*."

In this section, we extend the Abadi logic with a few constructs that will allow us to reason about requests in the context of RBAC.

3.1 Syntax

We start out by introducing a collection of principal expressions, ranged over by P and Q. Letting A range over a countable set of simple principal names, the abstract syntax of principal expressions is given as follows:

$$P ::= A \mid P \,\&\, Q \mid P|Q \mid P \ \mathsf{for}_A Q$$

The principal $P \,\&\, Q$ represents a compound principal who makes exactly those statements made by both P and Q. $P|Q$ represents an abstract principal corresponding to principal P quoting principal Q. $P \ \mathsf{for}_A Q$ represents a principal P acting on behalf of principal Q: $P \ \mathsf{for}_A Q$ is syntactic sugar for $P|Q \,\&\, A|Q$, where A is a principal that vouches for P's authorization to make statements on Q's behalf [1].

For the logic itself, we let p range over a countable collection of primitive propositions and define the abstract syntax for the logic as follows:

$$\begin{aligned}\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \supset \varphi_2 \mid \varphi_1 \equiv \varphi_2 \mid P \ \mathsf{says}\ \varphi \mid P \Rightarrow Q \\ \mid P \rightarrowtail_T Q \mid P \ \mathsf{serves}_T^A\ Q \mid P \succcurlyeq_T Q\end{aligned}$$

Here, T ranges over sets of formulas that include only formulas of the forms included on the first line (e.g., not involving $\rightarrowtail_T$, serves_T^A , or $\succcurlyeq_T$); the forms involving T are our extensions.

Primitive propositions are used to represent requests and permissions, while the formula $P \ \mathsf{says}\ \varphi$ represents principal P making the statement φ. In turn, $P \Rightarrow Q$ represents a relationship between principals P and Q through which statements of P can also be attributed to Q. The formula $P \rightarrowtail_T Q$, pronounced as "P is mimicked by Q on T," is our extension to the logic, inspired by Howell and Kotz's *restricted speaks for* relation $\overset{T}{\Rightarrow}$ [9]. This *restricted mimics* formula represents a weaker relation than $P \Rightarrow Q$, in part because only P's statements from the set T can be attributed to Q. Finally, $P \ \mathsf{serves}_T^A\ Q$ (the *restricted serves* relation) and $P \succcurlyeq_T Q$ (the *restricted inherits* relation) are syntactic sugar for $P|Q \rightarrowtail_T A|Q$ and $(P \Rightarrow Q) \wedge (Q \rightarrowtail_T P)$, respectively.

3.2 Semantics

The semantics of the logic is based on Kripke structures. A Kripke structure is a triple $\mathcal{M} = \langle W, I, J\rangle$, where W is a set of possible worlds, I is an interpretation function that maps each primitive proposition to a set of worlds, and J is an interpretation function that maps each primitive principal to a binary relation over W. We extend J to a function $\tilde{J}$ over arbitrary principal expressions as follows:

$$\begin{aligned}\tilde{J}(A) &= J(A)\\ \tilde{J}(P \,\&\, Q) &= \tilde{J}(P) \cup \tilde{J}(Q)\\ \tilde{J}(P|Q) &= \tilde{J}(P) \circ \tilde{J}(Q)\\ &= \{(w_1, w_3) \mid \exists w_2.\ (w_1, w_2) \in \tilde{J}(P) \wedge (w_2, w_3) \in \tilde{J}(Q)\}.\end{aligned}$$

$$
\begin{aligned}
\mathcal{E}_{\mathcal{M}}[\![p]\!] &= I(p) \\
\mathcal{E}_{\mathcal{M}}[\![\neg\varphi]\!] &= W - \mathcal{E}_{\mathcal{M}}[\![\varphi]\!] \\
\mathcal{E}_{\mathcal{M}}[\![\varphi_1 \wedge \varphi_2]\!] &= \mathcal{E}_{\mathcal{M}}[\![\varphi_1]\!] \cap \mathcal{E}_{\mathcal{M}}[\![\varphi_2]\!] \\
\mathcal{E}_{\mathcal{M}}[\![\varphi_1 \vee \varphi_2]\!] &= \mathcal{E}_{\mathcal{M}}[\![\varphi_1]\!] \cup \mathcal{E}_{\mathcal{M}}[\![\varphi_2]\!] \\
\mathcal{E}_{\mathcal{M}}[\![\varphi_1 \supset \varphi_2]\!] &= \mathcal{E}_{\mathcal{M}}[\![\neg\varphi_1]\!] \cup \mathcal{E}_{\mathcal{M}}[\![\varphi_2]\!] \\
\mathcal{E}_{\mathcal{M}}[\![\varphi_1 \equiv \varphi_2]\!] &= \mathcal{E}_{\mathcal{M}}[\![\varphi_1 \supset \varphi_2]\!] \cap \mathcal{E}_{\mathcal{M}}[\![\varphi_2 \supset \varphi_1]\!] \\
\mathcal{E}_{\mathcal{M}}[\![P \ \mathsf{says}\ \varphi]\!] &= \{w \mid \tilde{J}(P)(w) \subseteq \mathcal{E}_{\mathcal{M}}[\![\varphi]\!]\} \\
&= \{w \mid \{w' \mid (w, w') \in \tilde{J}(P)\} \subseteq \mathcal{E}_{\mathcal{M}}[\![\varphi]\!]\} \\
\mathcal{E}_{\mathcal{M}}[\![P \Rightarrow Q]\!] &= \begin{cases} W & \text{if } \tilde{J}(Q) \subseteq \tilde{J}(P) \\ \emptyset & \text{otherwise} \end{cases} \\
\mathcal{E}_{\mathcal{M}}[\![P \rightarrowtail_T Q]\!] &= \begin{cases} W & \text{if } \forall s \in T.\ P \ \mathsf{says}\ s \supset Q \ \mathsf{says}\ s, \\ \emptyset & \text{otherwise} \end{cases} \\
\mathcal{E}_{\mathcal{M}}[\![P \ \mathsf{serves}_T^A\ Q]\!] &= \mathcal{E}_{\mathcal{M}}[\![P|Q \rightarrowtail_T A|Q]\!] \\
\mathcal{E}_{\mathcal{M}}[\![P \succcurlyeq_T Q]\!] &= \mathcal{E}_{\mathcal{M}}[\![(P \Rightarrow Q) \wedge (Q \rightarrowtail_T P)]\!].
\end{aligned}
$$

Fig. 2. The meaning functions $\mathcal{E}_{\mathcal{M}}[\![-]\!]$

We then define a family (indexed by Kripke structures $\mathcal{M}$) of extended meaning functions $\mathcal{E}_{\mathcal{M}}[\![-]\!]$, which map arbitrary formulas to the sets of worlds in which they are considered true. The definition of $\mathcal{E}_{\mathcal{M}}[\![-]\!]$ appears in Figure 2. We write $(\mathcal{M}, w) \models \varphi$ if and only if $w \in \mathcal{E}_{\mathcal{M}}[\![\varphi]\!]$, and we say that $\mathcal{M}$ satisfies φ provided that $(\mathcal{M}, w) \models \varphi$ for all $w \in W$. We say that φ is *valid* if every Kripke structure $\mathcal{M}$ satisfies φ.

3.3 Logical Rules

The Kripke structures provide a precise semantics for the logic, but it is not convenient to reason at that level. Thus, we introduce a collection of logical rules for manipulating logical expressions. These rules, given in Figure 3, are sound with respect to the Kripke semantics: for every formula φ, if φ is derivable (i.e., $\vdash \varphi$), then φ is valid (i.e., satisfied in all Kripke structures).

3.4 Our Extensions to the Access-Control Logic

The original Abadi logic is unable to adequately describe RBAC for two reasons: its notion of roles conflicts with the RBAC concept, and it provides no way to express the role-permission associations. Specifically, the Abadi logic includes a special class of principals called *roles*, and arbitrary principals can adopt roles to make requests (e.g., "(*Alice* as *Fac*) says φ"). However, adopting roles is at the principal's discretion, and the effect is a *reduction* of privileges (e.g., *Alice* as *Fac* has fewer privileges than *Alice* does). In contrast, an RBAC user is granted privileges purely through the adoption of roles, and only when the user has been authorized to adopt a given role. Thus, reasoning about RBAC requires us to

$$\frac{}{\vdash \varphi} \quad \text{if } \varphi \text{ is an instance of a propositional-logic tautology}$$

$$\frac{\vdash \varphi \quad \vdash \varphi \supset \varphi'}{\vdash \varphi'} \qquad \frac{\vdash \varphi}{\vdash P \ \mathsf{says}\ \varphi} \quad \text{(for all } P\text{)}$$

$$\frac{}{\vdash (P \ \mathsf{says}\ (\varphi \supset \varphi')) \supset (P \ \mathsf{says}\ \varphi \supset P \ \mathsf{says}\ \varphi')}$$

$$\frac{}{\vdash \varphi} \quad \text{if } \varphi \text{ a valid formula of the calculus of principals}$$

$$\frac{}{\vdash P \ \mathsf{says}\ (\varphi_1 \wedge \varphi_2) \equiv (P \ \mathsf{says}\ \varphi) \wedge (P \ \mathsf{says}\ \varphi)}$$

$$\frac{}{\vdash (P \mid Q) \ \mathsf{says}\ \varphi \equiv P \ \mathsf{says}\ Q \ \mathsf{says}\ \varphi}$$

$$\frac{}{\vdash (P \Rightarrow Q) \supset ((P \ \mathsf{says}\ \varphi) \supset (Q \ \mathsf{says}\ \varphi))} \text{ (for all } \varphi\text{)}$$

$$\frac{}{\vdash (P \rightarrowtail_T Q) \supset ((P \ \mathsf{says}\ \varphi) \supset (Q \ \mathsf{says}\ \varphi))} \text{ (for all } \varphi \in T\text{)}$$

Fig. 3. Logical rules for the derivability predicate $\vdash$

$$\frac{}{\vdash (P \rightarrowtail_T Q) \wedge (Q \rightarrowtail_T R) \supset (P \rightarrowtail_T R)} \qquad (\rightarrowtail \text{ Trans})$$

$$\frac{}{\vdash (P \rightarrowtail_{T_1} Q) \supset (P \rightarrowtail_{T_2} Q)} \text{ (for all } T_2 \subseteq T_1\text{)} \qquad (\rightarrowtail \text{ Sub})$$

$$\frac{}{\vdash (P \rightarrowtail_T Q) \supset (R|P \rightarrowtail_T R|Q)} \qquad (\rightarrowtail \text{ Mon})$$

$$\frac{}{\vdash (P \ \mathsf{serves}_T^A\ Q) \wedge (P|Q \ \mathsf{says}\ s) \supset (P \ \mathsf{for}_A Q \ \mathsf{says}\ s)} \text{ (for every } s \in T\text{)} \qquad \text{(Role Del)}$$

$$\frac{}{\vdash P \succcurlyeq_T P} \qquad (\succcurlyeq \text{ Ref})$$

$$\frac{}{\vdash (P \succcurlyeq_{T_1} Q) \wedge (Q \succcurlyeq_{T_2} R) \supset (P \succcurlyeq_{T_1 \cap T_2} R)} \qquad (\succcurlyeq \text{ Trans})$$

$$\frac{}{\vdash (Q_1 \succcurlyeq_{T_2} Q_2) \wedge (P \ \mathsf{serves}_{T_1}^A\ Q_1) \supset (P \ \mathsf{serves}_{T_2}^A\ Q_2)} \text{ (for all } T_2 \subseteq T_1\text{)} \qquad \text{(Role Sub)}$$

Fig. 4. Logical rules related to $\rightarrowtail_T$, serves_T^A, and $\succcurlyeq_T$

model role-permission associations, which relate roles (principals) with sets of permissions (sets of statements).

We can model these RBAC notions in our logic using our three extensions: the *restricted mimicked by* relation, the *restricted serves* relation, and the *restricted inherits* relation. We explain how to do so in the next section. For now, we introduce to our logical system some additional rules related to these relations. These rules (see Figure 4) are all sound with respect to the Kripke semantics.

4 Describing RBAC Policies in the Access-Control Logic

When a user U acts in a role R and makes a request q, a reference monitor makes an access-control decision based on UA and PA. The request will be granted if the user has the right to act in role R (i.e., $U \in \mathit{authorized\_users}(R)$) and q is a permission associated with role R (i.e., $q \in \mathit{authorized\_permissions}(R)$).

For the logic to support reasoning about a specific RBAC policy, it must provide ways to express the following components: (1) RBAC entities (e.g., users, roles, permissions), (2) role activation and user requests, and (3) the role-inheritance relationship. We consider these components in turn.

4.1 Describing RBAC Entities

We represent users and roles as principals in the logic, and we represent *permissions* as primitive propositions. *UA* and *PA* are jointly represented in the logic as statements of the form

$$U \ \mathsf{serves}^{RA}_{ap(R)} \ R,$$

where RA represents a role authority that certifies that the user U has the right to act in the role R, and $ap(R)$ is the set of propositions corresponding to the permissions in the set $authorized\_permissions(R)$.[1] Simply put, $U \ \mathsf{serves}^{RA}_{ap(R)} \ R$ indicates that user U is an authorized user of role R and may make requests involving permissions associated with R.

The reference monitor's ultimate decision on whether to grant a request q is based on a series of access-control list (ACL) entries, each of which can be expressed as

$$((U \ \mathsf{for}_{RA} R) \ \mathsf{says} \ q) \supset q,$$

where $U \in authorized\_users(R)$ and $q \in authorized\_permissions(R)$. That is, if the reference monitor can verify that (1) a user U is making the request q while activated in the role R, and (2) q is a permission associated with role R, then the reference monitor will grant the request.

4.2 Describing User Requests

In RBAC, all requests by users are made within the context of a role. The result is that two principals—the user and the role—are involved in all requests.

We use quoting to describe role assertions (e.g., $U|R$) and the says operator to represent the actual requests. For example, a user U asserting role R and making a request q is represented as $U|R \ \mathsf{says} \ q$. Multiple requests can be expressed through conjunction, as in $U|R \ \mathsf{says} \ (q_1 \wedge q_2)$ or $(U|R_1 \ \mathsf{says} \ q_1) \wedge (U|R_2 \ \mathsf{says} \ q_2)$.

Note that the statement $U|R \ \mathsf{says} \ q$ does not guarantee that U is *authorized* for role R: it merely states that U is *claiming* to be acting in role R. There is no danger, however that an inappropriate request will be granted: the ACL entry requires the reference monitor to deduce (via Role Del) that $(U \ \mathsf{for}_{RA} R) \ \mathsf{says} \ q$, which is possible only when U is authorized for role R.

4.3 Describing Role Inheritance

The relationship $R_1 \succeq R_2$ is expressed in the logic by the formula $R_1 \succcurlyeq_{ap(R_2)} R_2$, which is syntactic sugar for $(R_1 \Rightarrow R_2) \wedge (R_2 \rightarrowtail_{ap(R_2)} R_1)$.

[1] Henceforth, we shall blur the distinction between actual permissions and the primitive propositions that are associated with them.

It is important to confirm that this formulation accurately captures all of the important properties of inheritance: reflexivity, transitivity, and the subset relationships between related roles' authorized users and authorized permissions. That is, we must ensure that the logical rules (and thus the logic's semantics) validate the following properties:

- For all roles R, $\vdash R \succcurlyeq_{ap(R)} R$.
  This rule is an instance of the $\succcurlyeq$-reflexivity rule ($\succcurlyeq$ Ref) from Figure 4.
- For all roles R_1, R_2, R_3,

$$\vdash (R_1 \succcurlyeq_{ap(R_2)} R_2 \wedge R_2 \succcurlyeq_{ap(R_3)} R_3) \supset R_1 \succcurlyeq_{ap(R_3)} R_3.$$

  Recall that, whenever $R_2 \succeq R_3$, *authorized_permissions*(R_3) is a subset of *authorized_permissions*(R_2), and thus $ap(R_3) \subseteq ap(R_2)$. Therefore, $ap(R_3) = ap(R_3) \cap ap(R_2)$, and the desired rule is simply an instance of the $\succcurlyeq$-transitivity rule ($\succcurlyeq$ Trans) from Figure 4.
- For all roles R_1 and R_2, users U, and role authorities RA,

$$\vdash (R_1 \succcurlyeq_{ap(R_2)} R_2 \ \wedge \ U \ \mathsf{serves}^{RA}_{ap(R_1)} \ R_1) \supset (U \ \mathsf{serves}^{RA}_{ap(R_2)} \ R_2).$$

  That is, if U is an authorized user of R_1 and R_1 inherits R_2, then U is also an authorized user of R_2. Once again, we rely on the relationship $ap(R_2) \subseteq ap(R_1)$ to see that the desired rule is simply an instance of the role-subsumption (Role Sub) rule from Figure 4.

4.4 Reasoning About Access-Control Decisions

To demonstrate the use of the logic in reasoning about access-control decisions, we return to the example from Section 2. We temporarily ignore the separation-of-duty constraints, and focus on the access-control aspects of the example.

Recall that the permission *read student grade reports* is associated with the role *Fac*: we use *rsg* as the primitive proposition corresponding to this permission. For simplicity, we also assume the permission *rant* (proposition *rt*) is assigned to the *Ten* role; there are no other explicit permission assignments.

Thus, the role hierarchy shown in Figure 1 can be described as follows:

$$(CS\ Fac \succcurlyeq_{\{rsg\}} Fac) \wedge (CE\ Fac \succcurlyeq_{\{rsg\}} Fac) \wedge (UnTen \succcurlyeq_{\{rsg\}} Fac) \wedge$$
$$(Ten \succcurlyeq_{\{rsg\}} Fac) \wedge (Chair \succcurlyeq_{\{rsg,rt\}} Ten) \wedge (P\&T\ VM \succcurlyeq_{\{rsg,rt\}} Ten).$$

Recall that *Alice* is explicitly assigned to the role *Chair*. This fact can be represented in the logic by the statement $Alice\ \mathsf{serves}^{RA}_{\{rsg,rt\}}\ Chair$. This statement, along with the description of the role hierarchy above, provide the basis for reasoning about whether *Alice* should be allowed to read student grade reports.

More specifically, we interpret *Alice*'s attempt to read student grade reports as a statement $Alice|Fac\ \mathsf{says}\ rsg$. Ultimately, the reference monitor must be able to deduce that $(Alice\ \mathsf{for}_{RA} Fac)\ \mathsf{says}\ rsg$, in which case the request will be granted.

Table 1. Mapping from RBAC to Access-Control Logic

RBAC	Access-Control Logic
A permission q is associated with role R	$p \in ap(R)$
User U is authorized to act in role R.	U $\mathsf{serves}^{RA}_{ap(R)}$ R
Role R_1 inherits role R_2 ($R_1 \succeq R_2$)	$R_1 \succcurlyeq_{ap(R_2)} R_2$
User U asserting role R makes a request q.	$U\|R$ says q
User U, acting in authorized role R, makes a request q.	U $\mathsf{for}_{RA} R$ says q

From above, we know that $(\mathit{Chair} \succcurlyeq_{\{rsg,rt\}} \mathit{Ten}) \wedge (\mathit{Ten} \succcurlyeq_{\{rsg\}} \mathit{Fac})$. Role transitivity allows us to conclude $\mathit{Chair} \succcurlyeq_{\{rsg\}} \mathit{Fac}$. Taken together with

$$\mathit{Alice}\ \mathsf{serves}^{RA}_{\{rsg,rt\}}\ \mathit{Chair},$$

Figure 4's role-subsumption rule (Role Sub) lets us deduce $\mathit{Alice}\ \mathsf{serves}^{RA}_{\{rsg\}}\ \mathit{Fac}$.

From $\mathit{Alice}|\mathit{Fac}$ says rsg and $\mathit{Alice}\ \mathsf{serves}^{RA}_{\{rsg\}}\ \mathit{Fac}$, we can use the Figure 4's role-delegation rule (Role Del) to deduce $(\mathit{Alice}\ \mathsf{for}_{RA}\mathit{Fac})$ says rsg as needed. As a result, *Alice*'s request can be granted.

4.5 Summary

Table 1 summarizes how RBAC concepts are translated into formulas of the access-control logic, providing a guideline for describing RBAC policies in the logic. The logical rules in Figure 4 provide the basis for reasoning about access-control decisions. Specifically, to determine whether a request $U|R$ says q should be granted, it suffices to determine whether the statement $(U\ \mathsf{for}_{RA} R)$ says q can be deduced from the logical rules.

5 Formal Specifications of RBAC Constraints

RBAC's separation-of-duty constraints do not directly affect access-control decisions, in that they are not checked at the time a decision is made. Rather, they impose additional restrictions on the initial specification of an RBAC policy. Therefore, we have not incorporated them into our access-control logic. However, it is desirable to be able to verify that a given policy is consistent: its user-role assignment and role hierarchy should not conflict with the stated separation-of-duty constraints.

For this reason, we have formalized RBAC constraints in the Higher-Order Logic (HOL) theorem prover. The result is a tool which one can verify the consistency of RBAC policies. Because the access-control logic has also been implemented and proved sound in HOL, we can easily convert RBAC policies which has been proved consistent in HOL into the access-control logic for reasoning about access-control decisions.

5.1 Static Separation of Duty

The role-inheritance relationship between roles R_1 and R_2 ($R_1 \succeq R_2$) is a partial order and thus reflexive, transitive, and antisymmetric. In RBAC, the role

hierarchy is generally represented pictorially by a Hasse diagram. We implement Hasse diagrams in HOL as a set *HSD* of pairs, with $(R_1, R_2) \in HSD$ precisely when there's an explicit edge between R_1 and R_2 in the Hasse diagram. It is then straightforward to define $\succeq$ as the reflexive, transitive closure of *HSD*, relative to the set *ROLES* of roles:

$$\begin{aligned} rhRel\ HSD\ ROLES = \{(R_1, R_2) \mid (R_1 \in ROLES) \wedge (R_2 \in ROLES) \wedge \\ (RTC\ (CURRY\ HSD)\ R_1\ R_2)\}, \end{aligned}$$

where $(CURRY\ HSD)\ R_1\ R_2$ is equivalent to $(R_1, R_2) \in HSD$ and the predicate *RTC* (defined in HOL's *Relation* theory) identifies the elements in the reflexive, transitive closure of a relation.

For example, the Hasse diagram from Figure 1 can be represented by a set *HSD* as follows:

$$\begin{aligned} HSD = \{&(Chair, Ten), (P\&T\ VM, Ten), (Ten, Fac), \\ &(CS\ Fac, Fac), (CE\ Fac, Fac), (UnTen, Fac)\} \end{aligned}$$

Letting *ROLES* be the set $\{Chair, P\&T\ VM, Ten, UnTen, CS\ Fac, CE\ Fac, Fac\}$, the inheritance relation $\succeq$ is given by:

$$\begin{aligned} &rhRel\ HSD\ ROLES = \\ &\quad HSD \cup \{(R, R) \mid R \in ROLES\} \cup \{(Chair, Fac), (P\&T\ VM, Fac)\}. \end{aligned}$$

The set of users authorized for a role R depends on both the user-role assignments (*UA*) and the inheritance relation $\succeq$; likewise, the set of permissions associated with a role depends on the permission assignments (*PA*) and $\succeq$. Thus, we define predicates *authorized_users* and *authorized_permissions* as follows:

$$\begin{aligned} &authorized\_users\ R\ UA\ HSD\ ROLES = \\ &\quad \{U \mid \exists R'.\ (R \in ROLES)\ \wedge\ ((R', R) \in rhRel\ HSD\ ROLES) \wedge (U, R') \in UA\}, \\ &authorized\_permissions\ R\ PA\ HSD\ ROLES = \\ &\quad \{p \mid \exists R'.\ (R' \in ROLES)\ \wedge\ ((R, R') \in rhRel\ HSD\ ROLES)\ \wedge\ (p, R') \in PA\}. \end{aligned}$$

It is straightforward to prove that, whenever $R_1 \succeq R_2$—that is, when (R_1, R_2) is in *rhRel HSD ROLES*—the following two properties hold:

$$\begin{aligned} &(authorized\_users\ R_1\ UA\ HSD\ ROLES) \subseteq \\ &\qquad (authorized\_users\ R_2\ UA\ HSD\ ROLES) \\ &(authorized\_permissions\ R_2\ PA\ HSD\ ROLES) \subseteq \\ &\qquad (authorized\_permissions\ R_1\ PA\ HSD\ ROLES). \end{aligned}$$

As described in Section 2, static separation-of-duty constraints are given by a set *SSD*: each pair $(rs, n) \in SSD$ represents a constraint to prevent users from being authorized for n or more roles in rs. Because the set of authorized users for a given role depends on both the user assignment *and* the role hierarchy,

we must consider both when determining whether a particular RBAC policy is consistent with its separation-of-duty constraints. The predicate *isConsistent* verifies that a given user assignment *UA* and role hierarchy (given as a Hasse diagram *HSD* and a set of *ROLES*) do not violate the *SSD* constraints:

$$\begin{aligned}
&\forall\, UA\ SSD\ HSD\ ROLES.\ isConsistent\ UA\ SSD\ HSD\ ROLES = \\
&\quad \forall\, rs\ n.\ (rs \subseteq ROLES) \supset (FINITE\ rs) \supset (n \geq 2) \supset (rs, n) \in SSD \supset \\
&\qquad (\forall\, t.\ (t \subseteq rs) \supset (CARD\ t \geq n) \supset \\
&\qquad\quad (\neg\exists\, U.\ \forall r.\ (r \in t) \supset (U \in authorized_users\ r\ UA\ HSD\ ROLES))),
\end{aligned}$$

where $CARD\ t$ returns the number of elements in the finite set t and $FINITE\ s$ returns true if the set s is a finite set.

[4] identifies several properties that should hold of consistent RBAC policies, such as that if two roles are mutually exclusive, then no nonempty role can possibly inherit both of them. We have verified that these properties do hold in our HOL implementation, which provides additional assurance that we have accurately captured the definitions.

5.2 Dynamic Separation of Duty

Dynamic separation of duty imposes constraints on the roles that a user can have *activated* at any given instant. Like static separation of duty, these constraints are expressed as a set *DSD*: each pair $(rs, n) \in DSD$ represents a constraint that prevents a user from activating n or more roles in rs simultaneously.

In other words, if the set of roles associated with a user's session s is a subset of rs, the number of roles in $session\_roles(s)$ must be less than n. The predicate *SessionSatisfies* verifies that a session s satisfies the *DSD* constraints:

$$\begin{aligned}
&\forall\, s\ DSD\ ROLES.\ SessionSatisfies\ s\ DSD\ ROLES = \\
&\qquad \forall\, rs\ n.\ (rs \subseteq ROLES) \wedge (FINITE\ rs) \wedge (n \geq 2) \wedge (CARD\ rs \geq n) \wedge \\
&\qquad\quad ((rs, n) \in DSD) \supset \\
&\qquad\quad (\forall\, t.\ (t \subseteq rs) \wedge (t \subseteq session_roles(s)) \supset (CARD\ t < n)).
\end{aligned}$$

As with static separation of duty, [4] also identified necessary consequences for dynamic separation of duty constraints, such as that, if two roles are mutually exclusive for activation, no session may involve both roles. We have verified that these properties hold of our HOL implementation.

6 Conclusions

Building information systems correctly is difficult—assuring information systems are secure is even more difficult. As the size, scope, and complexity of information systems is ever increasing, designers and verifiers of information systems face an ever more challenging task when assuring security. Many have observed that engineers must design security into systems from the start and that designs must

be provably secure. An implication of this last point is the need for a simple, formal, and rigorous logic for reasoning about access control in a wide variety of forms and situations. Such a logic could be used by designers to reason about access-control decisions in ways that are analogous to how digital designers use propositional logic to reason about digital designs. Our conclusion is that such a logic is possible, based on our experience defining a modal logic capable of specifying and reasoning about access-control policies and decisions that utilize role-based access control (RBAC).

In our logic, user assignments, permission assignments, and role hierarchies are defined within the access-control logic. In so doing, we have soundly united in a single logic the ability to reason about privileges, authority, delegation, credentials, and RBAC. We are currently extending the logic to support the administration of RBAC roles with concepts such as administrative scope [14,3].

The requirement that engineers prove that their designs are correct and secure necessitates the development of automated tools and verification methods. To help meet this need, both the access-control logic and the consistency checks for static and dynamic separation-of-duty constraints are defined as conservative extensions to the logic of the Higher Order Logic (HOL) theorem prover [10]. The HOL extensions provide an executable implementation of the access-control logic, and the inference rules have been verified to be sound. Likewise, verification of static and dynamic separation-of-duty constraints of RBAC policies is also done in HOL. While we do not anticipate that theorem provers such as HOL will be routinely used by practicing engineers, the HOL definitions and theorems are a rigorous and provably correct basis for computer-assisted reasoning tools such as symbolic simulators, rewriting systems, and symbolic calculators. Such tools are accessible and familiar to engineers and do not carry the same burden of formal proof when compared to full-scale theorem proving systems such as HOL.

References

1. Abadi, M., Burrows, M., Lampson, B., Plotkin, G.: A Calculus for Access Control in Distributed Systems. ACM Transactions on Programming Languages and Systems, 15(4) (1993) 706-734
2. Cuppens, F., Demolombe, R.: A Modal Logical Framework for Security Policies. Proceedings of the 10th International Symposium on Foundations of Intelligent Systems, Lecture Notes in Computer Science, Vol.1325. Springer. (1997) 579-589
3. Cramton, J., Loizou, G.: Administrative Scope: A Foundation for Role-Based Administrative Models. ACM Transactions on Information and System Security, 6(2) (2003) 201-231
4. Ferraiolo, D.F., Barkley, J.F., Kuhn, D.R.: A Role-Based Access Control Model and Reference Implementation Within a Corporate Intranet. ACM Transactions on Information and System Security, 2(1) (1999) 34-64
5. Ferraiolo, D., Kuhn, R.: Role-Based Access Control. 15th NIST-NCSC National Computer Security Conference, Gaithersburg, MD (1992) 554-563

6. Ferraiolo, D.F., Sandhu, R.S., Gavrila, S.I., Kuhn, D.R., Chandramouli, R.: Proposed NIST Standard for Role-Based Access Control. ACM Transaction on Information and System Security, 4(3) (2001) 224-274
7. Gordon, M.J.C., Melham, T.F.: Introduction to HOL: A Theorem Proving Environment for Higher Order Logic. Cambridge University Press, New York (1993)
8. Glasgow, J., MacEwen, G., Panangaden P.: A Logic for Reasoning About Security. ACM Transactions on Computer Systems, 10(3) (1992) 226-264
9. Howell, J., Kotz, D.: A Formal Semantics for SPKI. Technical Report TR2000-363, Department of Computer Science, Dartmouth College, Hanover, NH 03755-3510 (2000)
10. International Computer Limited. Higher Order Logic (HOL) Theorem Prover Version 4 (Kananaskis-2) http://hol.sourceforge.net (2004)
11. Lampson, B., Abadi, M., Burrows, M., Wobber, E.: Authentication in Distributed Systems: Theory and Practice. ACM Transactions on Computer Systems, 10(4) (1992) 265-310
12. Older, S., Chin, S.-K.: Building a Rigorous Foundation for Assurance into Information Assurance Education. Proceedings of the 6th National Colloquium for Information Systems Security Education (2002)
13. Older, S., Chin, S.-K.: Using Outcomes-based Assessment as an Assurance Tool for Assurance Education. Journal of Information Warfare, 2(3) (2003) 86-100
14. Sandhu, R., Bhamidipati, V., Munawer, Q.: The ARBAC97 Model for Role-Based Administration of Roles. ACM Transactions on Information and System Security, 2(1) (1999) 105-135