Toward Formal Verification of Role-Based Access Control Policies

Somesh Jha, Ninghui Li, *Senior Member*, *IEEE*, Mahesh Tripunitara,
Qihua Wang, and William H. Winsborough, *Member*, *IEEE Computer Society*

Abstract—Specifying and managing access control policies is a challenging problem. We propose to develop formal verification techniques for access control policies to improve the current state of the art of policy specification and management. In this paper, we formalize classes of security analysis problems in the context of Role-Based Access Control. We show that in general, these problems are **PSPACE**-complete. We also study the factors that contribute to the computational complexity by considering a lattice of various subcases of the problem with different restrictions. We show that several subcases remain **PSPACE**-complete and several further restricted subcases are **NP**-complete, and we identify two subcases that are solvable in polynomial time. We also discuss our experiences and findings from experimentations that use existing formal method tools such as model checking and logic programming for addressing these problems.

Index Terms—Access control, RBAC, formal methods, computational complexity.

1 Introduction

Access control is one of the most fundamental and pervasive security mechanisms in use today. The specification and management of access control policies is a challenging problem, and today's administrators have little tools to assist them. As a result, a large number of security breaches are caused by policy misconfigurations. Administrators are often reluctant to change policy settings, as they do not have confidence in whether the resulting policy configurations indeed enforce the policy objectives. The current state of the art of access control policy specification and management is still "what you specify is what you get but not necessarily what you want." This can be compared to software-hardware development before formal verification techniques [7], [27], [28] were developed and successfully deployed. We believe that formal verification techniques for access control policies can be developed to improve the current state of the art.

In almost all access control systems, there is a need to change the authorization state; for example, users and objects are added and removed, users start sharing resources at one moment and stop such sharing later, and users' job functionalities change. This dynamic aspect makes access control particularly challenging. A fundamental problem that deals with the dynamic aspect of access control is safety analysis, which was first formulated by Harrison et al. [17] for the access matrix model. Safety analysis decides whether undesirable right leakage could occur in future states. Recently, the notion of security analysis, which generalizes safety analysis, has been introduced [25], [26]. A *Security Analysis Problem* (SAP) instance asks whether an access control system preserves security policy invariants (which encode desired security properties) across state changes. Security analysis also allows the explicit specification of trusted principals. This enables one to ask questions such as: Suppose that a set of trusted principals will not initiate any potentially dangerous actions, does a policy invariant hold in all future states? A positive answer provides the assurance that the security of the system depends only on the cooperation of trusted principals.

In this paper (in Section 2), we study the SAP in Role-Based Access Control (RBAC) with the URA97 administrative scheme [37], [38] (we call this problem URA-SAP). We also describe our experiences in building tools for URA-SAP by using model checking and logic programming. Our choice of RBAC as the problem domain is motivated by the fact that RBAC [3], [13], [39] is today's most influential access control model. The last decade has seen an explosion of research in RBAC. Today, most major information technology vendors are offering products that incorporate some form of RBAC. For example, all major DBMS products support RBAC. Microsoft has brought RBAC to the Windows operating systems by introducing the Authorization Manager in Windows Server 2003 [29]. RBAC has also been used in Enterprise Security Management Systems such as the IBM Tivoli Policy Manager [20] and SAM Jupiter [4], [21].

The goal of our work is to develop techniques to help RBAC administrators precisely understand whom they are trusting for maintaining the desirable security properties or, in other words, who will be able to compromise the security of their system.

- *S. Jha is with the Department of Computer Sciences, University of Wisconsin, Madison, WI 53706-1685. E-mail: jha@cs.wisc.edu.*
- *N. Li and Q. Wang are with the Department of Computer Science, Purdue University, West Lafayette, IN 47907-2107. E-mail: ninghui@cs.purdue.edu, wangq@purdue.edu.*
- *M. Tripunitara is with the Security and Privacy Technology Laboratory, Motorola Labs, Schaumburg, IL 60196. E-mail: tripunit@motorola.com.*
- *W.H. Winsborough is with the Department of Computer Science, University of Texas, San Antonio, TX 78205. E-mail: wwinsborough@acm.org.*

Manuscript received 2 June 2006; accepted 10 July 2007; published online 6 Aug. 2007.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-0068-0606.
Digital Object Identifier no. 10.1109/TDSC.2007.70225.

The contributions of this paper are listed as follows:

- In Section 3, we show that in general, URA-SAP is **PSPACE**-complete. We also study the factors that contribute to the computational complexity by considering a lattice of various subcases of the problem with different restrictions. We show that several subcases remain **PSPACE**-complete whereas several further restricted subcases are **NP**-complete, and we identify two subcases that are solvable in polynomial time. We observe that the administrative scheme implemented in Oracle's RBAC system falls into one of the two tractable subcases.
- In Section 4, we compare two approaches for using existing tools to perform URA-SAP and report our findings. One approach is to use model checking (specifically the tool NuSMV [32]), and the other is to use logic programming (specifically the language XSB [15]).

We discuss related work in Section 5 and conclude in Section 6.

2 Problem Definitions

In this section, we give precise problem definitions for SAP. We also describe the URA97 RBAC scheme and present the special cases of SAP for the scheme.

2.1 Access Control Schemes

In existing work on security analysis in access control systems [25], [26], an access control scheme is defined 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-transition rules, and $\vdash: \Gamma \times Q \rightarrow \{true, false\}$ determines whether a query in Q is true or not in a given state in Γ. Each $\psi \in \Psi$ is viewed abstractly as a binary relation on Γ, i.e., $\psi \subseteq \Gamma \times \Gamma$. It determines whether one state can immediately reach another state. Such a definition abstracts a state transition as a binary relation and does not make explicit which principals initiate a particular action to effect a state transition. As a result, to consider multiple SAP instances with the same state but different sets of trusted users, one has to change the state-transition rule, which is unnatural. We now give a definition that makes the initiators explicit, avoiding such problems.

Definition 1 (access control schemes). *An access control scheme is given by a six-tuple $\langle \Gamma, Q, \vdash, \mathcal{A}, \Sigma, \Psi \rangle$, where Γ is a set of states, Q is a set of queries, $\vdash: \Gamma \times Q \rightarrow \{true, false\}$ determines whether a query is true or not in a state, $\mathcal{A}$ is a set of principals, Σ is a set of actions, and Ψ is a set of state-transition rules.*

A state $\gamma \in \Gamma$ *contains all the information necessary to make 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 denied. One may also ask a query that does not correspond to a specific request; for example, one may ask whether every principal that has access to a resource is an employee of an organization. Such queries are useful for understanding the properties of a complex access control system.*

Each action $\sigma \in \Sigma$ is a function mapping Γ to Γ. We write $\sigma(\gamma)$ to denote the state that results from applying the action σ on the state γ. Note that $\sigma(\gamma)$ could be γ; for example, this would happen if the application of the action σ on the state γ fails. Each action σ is associated with a set of principals, denoted by $\mathrm{init}(\sigma)$, *i.e.,* $\mathrm{init}(\sigma) \subseteq \mathcal{A}$. *Principals in* $\mathrm{init}(\sigma)$ *are called the initiators of the action: these are the principals that actively carry out the action σ. In most existing access control schemes, each action is carried out by one initiator, in which case* $\mathrm{init}(\sigma)$ *is a singleton set. When* $\mathrm{init}(\sigma)$ *includes two principals u_1 and u_2, it means that the active participation of u_1 and u_2 is needed to carry out σ.*

Each state-transition rule $\psi \in \Psi$ is given by a subset of Σ, i.e., $\psi \subseteq \Sigma$. The state transition from γ to γ_1 is allowed by ψ (we write $\gamma \rightarrow_\psi \gamma_1$) when there exists an action σ in ψ such that $\sigma(\gamma) = \gamma_1$.

Given an access control scheme $\langle \Gamma, Q, \vdash, \mathcal{A}, \Sigma, \Psi \rangle$, an access control system is specified by a pair (γ, ψ), where $\gamma \in \Gamma$ is the state of the system, and $\psi \in \Psi$ is the state-transition rule that determines which state transitions are allowed.

We say that a set A of principals can take an access control system (γ, ψ) to a state γ_g if principals in A can initiate actions that change the state of the access control system from γ to γ_g; i.e., there exists a sequence of actions $\sigma_1, \sigma_2, \cdots, \sigma_n$ such that the following conditions hold:

1. For each i such that $1 \leq i \leq n$, we have $\sigma_i \in \psi$ and $\mathrm{init}(\sigma_i) \subseteq A$.
2. $\sigma_n(\cdots \sigma_2(\sigma_1(\gamma)) \cdots) = \gamma_g$.

Definition 2 (SAP). *Given an access control scheme $\langle \Gamma, Q, \vdash, \mathcal{A}, \Sigma, \Psi \rangle$, a* SAP instance *is given by a four-tuple $\langle A_T, \gamma, \psi, q \rangle$, where $A_T \subseteq \mathcal{A}$ is a finite set of trusted principals, (γ, ψ) defines an access control system, and $q \in Q$ is a query.*

The answer to the instance is true if principals other than those in A_T can take the access control system (γ, ψ) to a state in which q evaluates to true. That is, this instance asks whether there exists a state γ_g such that principals in the set $\mathcal{A} - A_T$ can take (γ, ψ) to the state γ_g and $\gamma_g \vdash q$.

In an instance of SAP, q typically encodes an unsafe situation that should never occur; that is, $\neg q$ would be a policy invariant that should always hold.

2.2 The URA97 RBAC Scheme

We now define the access control scheme that we study in this paper, i.e., the URA97 RBAC scheme, which is based on the ARBAC97 administrative scheme for RBAC [37], [38]. To our knowledge, ARBAC97 is the first comprehensive and the most influential administrative model for RBAC.

URA97 is one of the three components of ARBAC97 [38]. The other components of ARBAC97 are PRA97 and RRA97, which are 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 in 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 the permission-role assignment and the role hierarchy will use

security analysis and only make those changes that do not violate desirable security properties.

We assume that there are three countable sets: $\mathcal{U}$ (the set of all possible users), $\mathcal{R}$ (the set of all possible roles), and $\mathcal{P}$ (the set of all possible permissions). While the set of all users in any RBAC state is finite, the set of all users that could be added is potentially unbounded. One can think of $\mathcal{U}$ as the set of all possible user identifiers in a system.

States Γ. An RBAC state γ is a six-tuple $\langle UA, PA, RH, CA, CR, CO\rangle$. We call UA, PA, and RH parts of the *basic state*, and CA, CR, and CO parts of the *administrative state*. The basic state is described in the following, and the administrative state is described when we discuss state transitions.

The user assignment relation $UA \subseteq \mathcal{U} \times \mathcal{R}$ associates users with roles, the permission assignment relation $PA \subseteq \mathcal{R} \times \mathcal{P}$ associates roles with permissions, and the role hierarchy relation $RH \subseteq \mathcal{R} \times \mathcal{R}$ is an irreflexive acyclic relation over $\mathcal{R}$. We use $\succeq_{RH}$ to denote the partial order induced by RH, i.e., the transitive and reflexive closure of RH. $r_1 \succeq_{RH} r_2$ means that every user who is authorized for r_1 is also authorized for r_2 and every permission that is associated with r_2 is also associated with r_1.

Given a state γ, each user has a set of roles for which the user is authorized. We formalize this by defining for every state γ a function $\mathrm{authorizedRoles} : \mathcal{U} \to 2^{\mathcal{R}}$:

$$\mathrm{authorizedRoles}(u) = \{r \in \mathcal{R} \mid \exists r_1 \in \mathcal{R}\ [(u, r_1) \in UA \wedge (r_1 \succeq_{RH} r)]\}.$$

When $r \in \mathrm{authorizedRoles}(u)$, we say that the user u is authorized for the role r or, equivalently, u is a member of r. We also define $down(r)$ to be the set of all roles dominated by r, and $up(r)$ to be the set all roles that dominate r as follows:

$$down(r) = \{\ r' \in \mathcal{R} \mid r \succeq_{RH} r'\ \},$$
$$up(r) = \{\ r' \in \mathcal{R} \mid r' \succeq_{RH} r\ \}.$$

State transition: $\mathcal{A}$, Σ, **and** Ψ. We now specify $\mathcal{A}$, Σ, and Ψ, which determine how states may change in the URA97 scheme. $\mathcal{A}$ is defined to be $\mathcal{U}$, the set of all possible users. Σ consists of two kinds of actions: assignment and revocation actions. Whether these actions succeed or not when applied in a state depends on the administrative state of γ, namely, CA, CR, and CO, which we describe as follows:

- The relation $CA \subseteq \mathcal{R} \times C \times 2^{\mathcal{R}}$ determines who can assign users to roles and the preconditions that these users must satisfy. C is the set of conditions, which are expressions formed by using roles, the binary operators $\cap$ and $\cup$, the unary operator $\neg$, and parentheses. A tuple $\langle r_a, c, rset\rangle$ in CA means that members of role r_a can assign any user whose role memberships satisfy the condition c to any role $r \in rset$. For example, $\langle r_0, r_1 \cap r_2 \cap \neg r_3, \{r_4\}\rangle \in CA$ means that a user that is a member of role r_0 is allowed to assign a user that is a member of both r_1 and r_2 but is not a member of r_3 to be a member of r_4.
- The relation $CR \subseteq \mathcal{R} \times 2^{\mathcal{R}}$ determines who can remove users from roles. $\langle r_a, rset\rangle \in CR$ means that the members of role r_a can remove a user from a role $r \in rset$. Unlike relation CA, there is no preconditions in relation CR defined in URA97 [38]. We assume that CA and CR satisfy the property that the administrative roles are not affected by CA and CR. The administrative roles are those that appear in the first component of each tuple in CA or CR. These roles should not appear in the last component of any CA or CR tuple. This condition is satisfied in URA97, which assumes the existence of a set of administrative roles that is disjoint from the set of normal roles.
- CO is a set of mutually exclusive role constraints. Each constraint in CO has the form $\mathrm{smer}\langle\{r_1, \ldots, r_m\}, t\rangle$, where each r_i is a role, and m and t are integers such that $1 < t \leq m$. This constraint forbids a user from being a member of t or more roles in $\{r_1, \ldots, r_m\}$. We say that a set R of roles satisfies a constraint $\mathrm{smer}\langle\{r_1, \ldots, r_m\}, t\rangle$ if and only if $|R \cap \{r_1, \ldots, r_m\}| < t$, where $||$ gives the cardinality of a set.

  For example, $\mathrm{smer}\langle\{r_1, r_2\}, 2\rangle$ means that no user is allowed to be a member of both r_1 and r_2. In an RBAC state γ, if $r_1 \in \mathrm{authorizedRoles}(u)$ for a user u, then an assignment action that assigns the user u to any role in $up(r_2)$ would fail because of the constraint.

Ψ consists of a single state-transition rule ψ, which is a set of actions:

$$\chi = \{assign(u_a, u_t, r_t) \mid u_a, u_t \in \mathcal{U} \wedge r_t \in \mathcal{R}\} \cup \{revoke(u_a, u_t, r_t) \mid u_a, u_t \in \mathcal{U} \wedge r_t \in \mathcal{R}\}.$$

- An assignment action $assign(u_a, u_t, r_t)$ means that the user u_a assigns the user u_t to the role r_t. When this action is applied to an RBAC state γ, it succeeds if and only if the following conditions hold:
  - $(u_t, r_t) \notin UA$, i.e., the user u_t is not yet assigned role r_t.
  - There exists a tuple $\langle r_a, c, rset\rangle \in CA$ such that $r_a \in \mathrm{authorizedRoles}(u_a)$, $\mathrm{authorizedRoles}(u_t)$ satisfies c and $r_t \in rset$.
  - $\mathrm{authorizedRoles}(u_t) \cup down(r_t)$ satisfies every constraint in CO, i.e., the new role memberships of u_t do not violate any constraint.

  The assignment action $assign(u_a, u_t, r_t)$ may succeed, even when u_t is already authorized for r_t indirectly through other roles. For example, even when $(u_t, r_s) \in UA$ and $r_s \succeq_{RH} r_t$, $assign(u_a, u_t, r_t)$ can succeed. The rationale is that these memberships often represent independent relationships. For example, $(u_t, r_s) \in UA$ may represent a shorter term role assignment for u_t because of temporary staff shortages, and $(u_t, r_t) \in UA$ may represent a longer term role assignment. Then, we want to add (u_t, r_t) to UA so that when (u_t, r_s) is removed from UA, u_t is still authorized for r_t.

  When the assignment action is successfully applied to an RBAC state γ, the resulting state γ' differs from γ only in the user-role relation. The result of a successful application is $UA' = UA \cup \{(u_t, r_t)\}$. When

the application is not successful, the state does not change.

- A revocation action is of the form $revoke(u_a, u_t, r_t)$, which means that user u_a revokes user u_t from role r_t. When this action is applied to an RBAC state γ, it succeeds if and only if the following conditions hold:
  - $(u_t, r_t) \in UA$, i.e., user u_t is assigned to role r_t.
  - There exists a tuple $\langle r_a, rset\rangle \in CR$ such that $r_a \in \text{authorizedRoles}(u_a)$ and $r_t \in rset$.

  When the revocation action is successfully applied to an RBAC state γ, the resulting state γ' differs from γ only in the user-role relation. The result of a successful application is $UA' = UA - \{(u_t, r_t)\}$. When the application is not successful, the state does not change.

2.3 SAP in URA97

Definition 3 (URA-SAP). *A URA-SAP instance is given by an RBAC state $\gamma = \langle UA, PA, RH, CA, CR, CO\rangle$, a set $A_T \subseteq \mathcal{U}$ of trusted users, and a query.*

We deliberately leave the syntax for queries unspecified in the above definition. Different kinds of queries may be needed for different policy analyses. The simplest kind is to ask whether a user u is a member of a role r. More sophisticated queries may ask whether a user's role membership satisfies a condition (e.g., $r_1 \cup (r_2 \cap \neg r_3)$) or whether the set of members of one role is a subset of the set of members of another role.

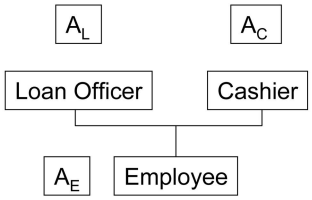
An important observation is that the simplest query that asks whether a user is a member of a role can be used to handle several other kinds of queries. For example, if one wants to know whether the system can reach a state in which u's role membership includes a set $\{r_1, r_2\}$ and excludes $\{r_3\}$, one can add a new user u_a, two new roles r_a and r_t, a user assignment (u_a, r_a), and a new tuple $(r_a, (r_1 \cap r_2 \cap \neg r_3), \{r_t\})$ to CA and use $u \in r_t$ as the query. Similarly, if one wants to know whether the system can go to a state in which u possesses a certain set of permissions, one can compute the role condition that is necessary and sufficient to have the permissions and then translate that into a query about a single role.

Definition 4 (URA-RC-SAP). *A URA-RC-SAP instance is a special case of URA-SAP, in which a query has the form $u \in r$.*

2.4 An Example

Fig. 1 shows a simplified role hierarchy in a bank. This example is inspired by a case study of a commercial bank that appears in the literature [40]. The bank has two functional roles Loan Officer and Cashier apart from the basic Employee role. The bank requires that an employee be a member of exactly one functional role. We quote from [40]: "*Ideally, each employee is assigned to one role.*" This is easily achieved using the mutually exclusive role constraint $\text{smer}\langle\{\text{Loan Officer}, \text{Cashier}\}, 2\rangle$.

In our example, the bank allows employees to be reassigned to a different functional role. However, such a reassignment must involve at least two administrators. This results in a separation of privilege in that no single administrator can change a user's functional role by himself.



$$CA_{bank} = \{\langle \mathsf{A_E}, true, \{\mathsf{Employee}\}\rangle,\ \langle \mathsf{A_L}, \mathsf{Employee}, \{\mathsf{Loan\ Officer}\}\rangle,\ \langle \mathsf{A_C}, \mathsf{Employee}, \{\mathsf{Cashier}\}\rangle\}$$

$$CR_{bank} = \{\langle \mathsf{A_E}, \{\mathsf{Employee}\}\rangle,\ \langle \mathsf{A_L}, \{\mathsf{Loan\ Officer}\}\rangle,\ \langle \mathsf{A_C}, \{\mathsf{Cashier}\}\rangle\}$$

$$CO_{bank} = \{\text{smer}\langle\{\mathsf{Loan\ Officer}, \mathsf{Cashier}\}, 2\rangle\}$$

$$UA_\gamma \supset \{(Alice, \mathsf{A_E}), (Adam, \mathsf{A_L}), (Andy, \mathsf{A_C})\}$$

Fig. 1. A simplified view of a bank, in which there are two functional roles Loan Officer and Cashier, both of which inherit from the Employee role. The role hierarchy RH_{bank} is shown in the figure. A different administrative role is associated with each of the roles. $\mathrm{A_L}$ is for administering Loan Officer, $\mathrm{A_C}$ for Cashier, and $\mathrm{A_E}$ for Employee. This is reflected in the assignment and revocation rules CA_{bank} and CR_{bank}, respectively, for the bank. CO_{bank} is the set of mutually exclusive role constraints. There is only one constraint, i.e., that no user can be both a loan officer and a cashier. The UA_γ in the figure shows that a different administrative user is assigned to each of the administrative roles.

Consider the example query q_{bank} of the form "$Bob \in \text{Cashier}$," where $(Bob, \text{Loan Officer}) \in UA_\gamma$. With this query, we seek to verify that the bank's policy is indeed satisfied for a user Bob that is a loan officer. If we consider a URA-RC-SAP instance with the set of trusted principals $A_T = \{Alice, Adam\}$, we observe that the instance is false. The reason is that for Bob to be assigned to Cashier, he must first be revoked from Loan Officer. Otherwise, the entry in CO_{bank} would be violated. The only administrator that can revoke Bob from Loan Officer is $Adam$. However, as $Adam$ is considered to be a trusted principal in the analysis instance, he cannot initiate any administrative actions.

It turns out in our example that the bank's policy is indeed satisfied for any user, even for one that is not (yet) an employee of the bank. This can be verified by running analysis instances with appropriately instantiated parameters. As an example to demonstrate that a user can indeed be reassigned to a different functional role, if user $Carl$ is a member of Cashier in UA_γ, then a URA-RC-SAP instance with $A_T = \emptyset$ and query "$Carl \in \text{Loan Officer}$" is true, because $Carl$ can first be revoked from Cashier by $Andy$, then assigned to Employee by $Alice$, and finally assigned to Loan Officer by $Adam$. In this case, the cooperation of all three administrators is required.

3 Computational Complexity

In this section, we study the computational complexity of URA-SAP. In particular, we show that URA-RC-SAP is **PSPACE**-complete. The main source of the complexity of SAP is that the state space that needs to be explored is potentially large. We would like to understand how different features in URA97 affect this search space;

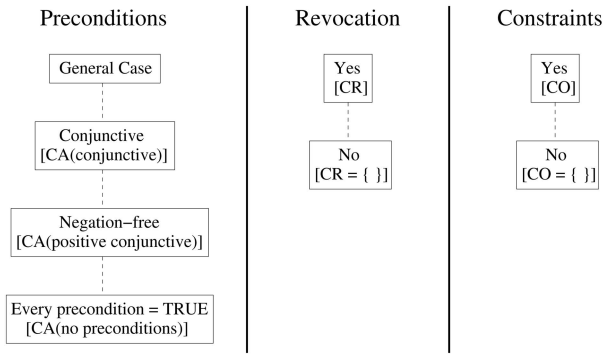


Fig. 2. Possible variations in the features that we consider for the preconditions in CA, revocation, and constraints. A dotted line connects a case with a subcase. For example, negation-free preconditions ($[CA$ (positive conjunctive)$]$) is a subcase of conjunctive preconditions ($[CA$ (conjunctive)$]$). Various combinations based on the three columns are possible. For example, we can consider the analysis problem with negation-free preconditions, with revocation, but without constraints, which corresponds to URA-RC-SAP$[CA$ (positive conjunctive)$, CR, CO = \{\ \}]$.

therefore, we consider special cases of URA-SAP that result from restricting the URA scheme in various ways. Answers to the following questions affect the computational complexity of URA-SAP:

- *What queries are considered?* If queries are allowed to contain conjunctions and disjunctions of roles, then URA-SAP is likely to be intractable. For example, in [26], one can pose a query that asks whether the set of users who satisfy $((r_1 \cup r_2) \cap r_3)$ is always a subset of the set of users that satisfy $((r_1 \cup r_2) \cap (r_2 \cup r_3))$. The intractability results in [26] are consequences of the fact that these sophisticated queries can encode propositional formulas to show **NP**-hardness. In other words, [26] deals with sophisticated queries but very simple state-transition rules.

  In this section, in contrast to the work in [26], we focus on the simplest kind of queries, i.e., whether a user u is a member of a role r, to better understand the complexity caused by state-transition features within URA97. In other words, we consider URA-RC-SAP.
- *Do the preconditions involve only conjunctions?* Each tuple in CA has a precondition. It is conceivable that if the precondition involves arbitrary conjunction, disjunction, and negation of roles, then this could make the problem intractable; however, such a result would be less insightful and of less practical interest. In practical systems, one would not expect the precondition to be a very complicated logical formula. In this paper, we focus on the special case in which each precondition is a conjunction of roles or their negations. We show in the following that for the general case, whether we allow disjunctions in the preconditions or not does not affect the computational complexity.
- *Is negation allowed in preconditions in CA?* When preconditions in CA may contain negation, one needs to consider the revocation of a user's role membership in order to satisfy the precondition and be assigned to a new role.
- *Are SMER constraints allowed; i.e., is $CO = \{\ \}$?* When constraints are allowed, one may need to consider revocations in order to assign a user to a new role.
- *Are revocations allowed, i.e., whether $CR = \{\ \}$?* One may want to consider the special case that role memberships cannot be revoked.

We summarize the variations that we consider in this paper in Fig. 2. The main results of this paper are stated in the following theorem. These results are also summarized in Fig. 3.

Theorem 1. *The computational complexity for URA-RC-SAP and its various subcases are as shown in Fig. 3.*

Some subcases of the problem are not listed in Fig. 3, because they are special cases of the two cases that are known to be in **P**; thus, they are solvable in polynomial

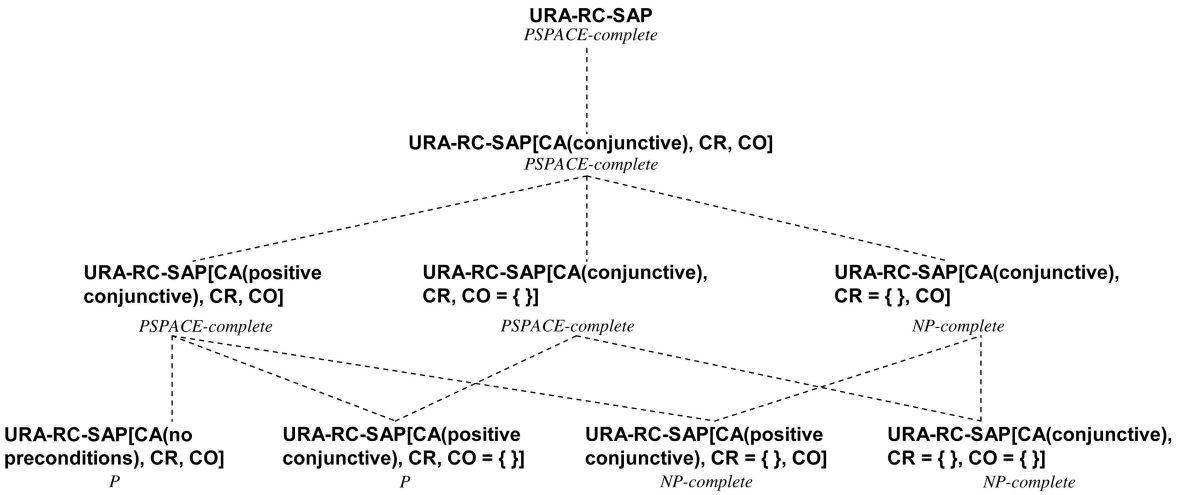


Fig. 3. Summary of computational complexity results for various cases of URA-RC-SAP. A dotted line links a case to a subcase. For example, URA-RC-SAP$[CA$ (positive conjunctive)$, CR, CO]$ is a subcase of URA-RC-SAP$[CA$ (conjunctive)$, CR, CO]$.

time as well. We make several observations from Theorem 1 and Fig. 3.

- Whether we allow only conjunctive preconditions or arbitrary preconditions does not change the computational complexity of URA-RC-SAP in general. The problem is **PSPACE**-complete with or without the conjunctive restriction.
- There are three cases in which the problem's complexity changes from **PSPACE**-complete to **NP**-complete. All three result from making CR empty. The reason is that if CR is empty, then one needs to consider only role assignments. Any role assignment sequence can have a length that is at most polynomial in the size of the problem instance. This makes the problem in **NP**. On the other hand, there may be exponentially many such possible assignment sequences; thus, the problem remains **NP**-complete. However, when CR is not empty, the sequence necessary for entering a user into a role may be of exponential length.
- The effect of nonempty CO is identical to the effect of negation in preconditions of CA from the standpoint of computational complexities. The reason is that the effect of a constraint in CO can be "simulated" using negative preconditions in CA, and vice versa. We use this fact, for example, in the proofs of Lemmas 4 and 7.

The rest of this section proves the results in Theorem 1.

URA-RC-SAP and three subcases are PSPACE-complete. We first show that the general case of URA-RC-SAP is in **PSPACE**. We then show that two subcases URA-RC-SAP$[CA\ (\text{positive conjunctive}), CR, CO]$ and URA-RC-SAP$[CA\ (\text{conjunctive}), CR, CO = \{\ \}]$ are **PSPACE**-hard. These results together prove the four **PSPACE**-completeness result in Theorem 1. In Appendix A, we present background information related to Turing machines (TMs) that we use to establish these results.

Lemma 2. *URA-RC-SAP is in **PSPACE**.*

Proof. Given a URA-RC-SAP instance, let $\gamma = \langle UA, PA, RH, CA, CR, CO\rangle$, let U_T be a set of trusted users, and let $u \in r$ be the query. Notice that the only component of the state that changes is UA. Furthermore, we only need to maintain u's role memberships (the number of roles does not change from the start state). It takes polynomial space to represent u's role memberships. Recall that in URA97, administrative roles (i.e., roles that appear in the first component of a tuple in CA and CR) are not affected by CA and CR. Therefore, we do not need to consider memberships of users other than u, because these role memberships do not affect whether u can be assigned to a role or not. Observe that in order to determine whether u can be added to a role, the precondition is only about u's role memberships.

Observe that if we relax the restriction that administrative roles are not affected by CA and CR, then we need to maintain the role memberships of all users, which can still be done in polynomial space. Therefore, URA-RC-SAP is in **PSPACE**, even without this restriction.

We describe a nondeterministic TM (NDTM) NM to solve this problem. Initially, NM sets the initial state γ_0 to be equal to γ. Given that NM is in state γ_i, it continues its computation as follows:

- If $\gamma_i \vdash q$, NM stops and outputs yes.
- Assume that $\gamma_i \nvdash q$. NM guesses the next state γ_{i+1} that changes user u's role memberships. NM ensures that such a change conforms to CA and CR and satisfies the constraints in CO. NM ensures also that the only users that affect an assign or revoke action are the ones that do not belong to U_T.

The construction given above proves that URA-RC-SAP is in $NSPACE(O(n))$, where n is the space needed to represent the input URA-RC-SAP instance. Using Savitch's theorem, we can conclude that URA-RC-SAP is in $DSPACE(O(n^2))$. □

Lemma 3. *URA-RC-SAP* $[CA\ (\text{positive conjunctive}), CR, CO]$ *is **PSPACE**-hard.*

The proof is given in Appendix B. The proof is by a reduction from the membership problem for linear bounded automata (LBA), which is known to be **PSPACE**-complete. An LBA is a restricted form of a TM. It differs from a TM in that while the tape is initially considered infinite, only a finite contiguous portion whose length is a linear function of the length of the initial input can be accessed by the read/write head.

Lemma 4. *URA-RC-SAP* $[CA\ (\text{conjunctive}), CR, CO = \emptyset]$ *is **PSPACE**-hard.*

The proof of the above lemma is similar to the proof of Lemma 3. The proof of Lemma 3 uses 2-2 SMER constraints. We simulate the use of such constraints in the proof of Lemma 4 by using negative preconditions in CA. For example, if a 2-2 SMER constraint $\text{smer}\langle\{r_1, r_2\}, 2\rangle$ is used in the proof of Lemma 3, we can add $\neg r_1$ to the precondition of each rule that assigns to r_2 and add $\neg r_2$ to the precondition of each rule that assigns to r_1.

URA-RC-SAP $[CA\ (\textbf{conjunctive}), CR = \{\ \}, CO]$ **and its two subcases are NP-complete.** We first demonstrate that disallowing revocations in URA-RC-SAP causes the problem to be in **NP**. We then demonstrate that two subcases are both **NP**-hard. The first subcase is when we disallow negative preconditions in CA. The second subcase is when we disallow constraints. These results together prove the three **NP**-completeness results in Theorem 1.

Lemma 5. *URA-RC-SAP* $[CA\ (\text{conjunctive}), CR = \{\ \}, CO]$ *is in **NP**.*

Proof. We need to demonstrate that if an instance of URA-RC-SAP $[CA\ (\text{conjunctive}), CR = \{\ \}, CO]$ is true, then there exists an evidence of size polynomial in the problem that can be efficiently verified. Let the query q in the problem instance be about user u's membership in role r. As the evidence, we use the shortest state-change sequence from the initial state γ_0 to a state γ_n such that $\gamma_n \vdash q$. Each state change in this sequence is the assignment of u to a role of which he is not yet a member. There can be at most $|R|$ such assignments,

where R is the set of roles in the system. (See the proof of Lemma 2 about why we only need to consider assignment of user u. Also, observe that even if we consider the assignment of all users, the total number of such assignments is still polynomial in the size of the instance.) Therefore, the state-change sequence is of length at most $|R|$, which is polynomial in the input and can certainly be verified in polynomial time. □

Lemma 6. *URA-RC-SAP* $[CA\ (\text{conjunctive}), CR = \{\ \}, CO = \{\ \}]$ *is* ***NP****-hard.*

The proof is in Appendix C. It uses a reduction from the 3SAT problem.

Lemma 7. *URA-RC-SAP* $[CA\ (\text{positive conjunctive}), CR = \{\ \}, CO]$ *is* ***NP****-hard.*

The proof is in Appendix D. This result should not be surprising, given Lemma 6. As discussed earlier, the effects of SMER constraints and negation in preconditions in CA are very similar.

Two subcases that are in P. As we have shown above, either negation in preconditions or SMER constraints is sufficient to make URA-RC-SAP intractable. However, URA-RC-SAP $[CA\ (\text{positive conjunctive}), CR, CO = \{\ \}]$; that is, when neither negations in preconditions nor SMER constraints are allowed, the problem can be solved in linear time. The reason is that to determine whether u can be a member of a role r in some future state, there is no need to consider revocation, as there is no negation in preconditions in CA, and there are no SMER constraints. A straightforward quadratic algorithm is to try each tuple in CA and see whether u can be assigned to more roles. As the number of roles that can be assigned according to CA is bounded by the size of CA, this algorithm takes at most quadratic time. A linear time algorithm can be obtained by reducing this to the Horn-SAT problem, which can be solved in linear time [10]. Each rule in CA can be viewed as a Horn rule; for example, if one such rule says that $r_1 \cap r_2$ is the precondition for r_3, then this can be translated into a Horn clause "$r_3 \longleftarrow r1, r2$." Each initial role membership of the user can be translated into a Horn clause. The query can be translated into a Horn query clause.

Another tractable subcase is URA-RC-SAP $[CA\ (\text{no preconditions}), CR, CO]$, which can be solved in quadratic time. In this subcase, every precondition in CA is "true," but we allow revocations and SMER constraints. The algorithm first checks whether user u is already a member of the role r (where u and r comprise the query). If not, we revoke u from as many roles as possible by using entries from CR. We then check whether there exists an entry in CA that we can exercise to cause u to become a member of r while not violating any entry in CO. If yes, the algorithm returns "true"; otherwise, it returns "false." This is linear in the sizes of CA, CR, and CO.

We observe that in some RBAC schemes in practical systems such as the RBAC scheme in the Oracle database, there is no precondition in role assignment. Security analysis there thus falls under the above tractable case.

4 Tools for Security Analysis

The fact that URA-RC-SAP and several of its subcases are intractable (**PSPACE**-complete or **NP**-complete) means that there exist difficult problem instances. In this section, we describe our experiences by using logic programming and model checking tools for some realistic instances of URA-RC-SAP. Our goals for performing these experiments are twofold. First, we would like to see whether security analysis instances of nontrivial sizes can be solved in reasonable amounts of time. Our experimental results show that the answer is positive. Second, we would like to compare the effectiveness of model checking and logic programming in security analysis. Our results demonstrate that logic programming outperforms model checking in smaller instances; however, model checking appears to scale better than logic programming.

The logic programming approach. Logic programming is a declarative relational style of programming based on first-order logic. A logic program is composed of a set of facts and a number of rules, which specify how new facts can be derived from known ones. We use XSB [15], a Prolog-variant logic programming system developed at the State University of New York, Stony Brook. XSB uses SLG resolution [6], which can correctly evaluate many recursive logic programs that would cause SLD-resolution-based Prolog systems to fail to terminate.

Our implementation is a natural reduction from instances of URA-RC-SAP to logic programs. Recall that an instance of URA-RC-SAP consists of an RBAC state $\langle UA, PA, RH, CA, CR, CO\rangle$ and a query of the form $u \in r$, where $u \in \mathcal{U}$ is a user, and $r \in \mathcal{R}$ is a role. Our logic program defines a predicate over states that is true when the state is reachable. Each tuple in CA and CR is represented as a rule, while RH and CO are incorporated in the rules representing CA entries. The initial role memberships of user u is given as a fact, and the evaluation's goal is to find a state in which the answer to the query $u \in r$ is true.

The model checking approach. Model checking is a technique for determining whether a formal model M of a system satisfies a temporal-logic property p. A model M can be represented as a four-tuple (S, R, s_0, L), where S is a finite set of states, $R \subseteq (S \times S)$ is a transition relation, $s_0 \in S$ is an initial state, and $L : S \rightarrow 2^{AP}$ is a labeling of states with propositional formulas from AP (given a state s, $L(s)$ denotes the atomic propositions in AP that are true in s). We express a safety property p in *Computation Tree Logic* (*CTL*) with the form $\mathbf{AG}f$ (i.e., $p = \mathbf{AG}f$, where f is a formula in propositional logic). ($\mathbf{AG}f$ means that *always globally*, the atomic proposition f is true or, in other words, f is true in every state reachable from the initial state s_0.) If the model M satisfies the property p, a model checker reports true. If M does not satisfy p, a model checker produces a counterexample that shows an execution that leads to a violation of the property. A thorough treatment of model checking is provided in [7].

The model checker that we used is NuSMV [32]. We implemented a program that reads an instance of URA-RC-SAP and then generates an NuSMV program for the instance. Encoding an instance of URA-RC-SAP as a

TABLE 1
Experimental Data on URA-RC-SAP Instances Using XSB and NuSMV

	Man1	Man2	Rand1	Rand2	Rand3	Rand4	Rand5
Num. of Roles	12	16	15	100	40	30	25
Num. of Rules	31	40	45	250	92	88	79
Num. of Rules *(AP)*	22	29	34	20	27	37	57
Transition Length	15	22	1	2	3	2	6
Total States	1.70E+7	1.76E+9	6.60E+8	NA	NA	NA	NA
Reachable States	1.31E+5	2.69E+6	7.54E+5	NA	NA	NA	NA
XSB Runtime	0.55s	14.22s	109.64s	NA	NA	NA	NA
NuSMV Runtime	0.188s	1.78s	2.10s	NA	NA	NA	NA
Total States*(AP)*	1.22E+7	1.29E+9	5.02E+8	1.08E+49	3.40E+20	7.82E+15	4.90E+13
Reachable States *(AP)*	3456	6000	6720	75264	7.31E+5	3.11E+6	3.98E+7
XSB Runtime *(AP)*	0.02s	0.05s	0.06s	0.55s	5.70s	19.94s	NA
NuSMV Runtime *(AP)*	0.11s	0.13s	0.13s	0.94s	1.27s	5.44s	72.94s

Instances with names beginning with Man were manually crafted, while those beginning with Rand were randomly generated. Statistics on the total states and the reachable states is for NuSMV only. Rows marked (AP) present results after preprocessing. NA indicates that, for the NuSMV cases, the program did not finish running within 30 minutes or, for the XSB case, ran out of the memory.

model in NuSMV is straightforward; e.g., states correspond to user assignments to roles, and transitions correspond to rules in CA and CR.

Preprocessing. We observe that given a URA-RC-SAP instance, many rules in CA and CR may be irrelevant to the query. We use a preprocessing stage to remove these rules. Our experimental data shows that preprocessing can be very effective. Given a query $u \in r$ and an RBAC state, our preprocessing does two kinds of pruning:

- *Forward pruning.* We remove rules that will never be successfully executed. We first compute R_{lo}, the set of roles in the initial state that cannot be revoked by rules in CR. We then compute R_{up}, the set of roles that may be assigned to the user u, and A, the set of assignment rules that may be successfully applied. To do this, we initialize R_{up} with I, the initial set of roles that u is a member of, and A with $\emptyset$. For each assignment rule α in CA, if the target role of α (i.e., the last component of α) is not in R_{lo}, the positive precondition of α is satisfied by R_{up}, and the negative precondition of α does not contain any role in R_{lo}, we add the target role of α to R_{up} and add α to A. We repeat this process of iterating through CA until R_{up} does not grow. Only assignment rules in A and revocation rules that revoke roles in R_{up} are kept after the pruning. Letting $|CA|$ be the number of rules in CA, the computation of R_{up} and A requires $O(|CA|^2)$ rule consideration steps, because each pass through CA adds at least one $\alpha \in CA$ to A, and each such α need not be considered thereafter.
- *Backward pruning.* Some roles may be irrelevant to assigning the role r in the query. The backward pruning removes assignment and revocation rules about those roles. We compute two sets of roles: R_{po} is the set of roles on which r positively depends, and R_{ne} is the set of roles on which r negatively depends. We remove assignment rules that assign roles outside R_{po} and revocation rules that revoke roles outside R_{ne}. R_{po} is the smallest set that satisfies the following conditions: 1) $r \in R_{po}$, 2) if $r_p \in R_{po}$, then any role that dominates r_p is also in R_{po}, and 3) if $r_p \in R_{po}$, then any role that appears in the positive precondition of a CA entry assigning to r_p is also in R_{po}. R_{ne} is the smallest set that satisfies the following conditions: 1) if $r_p \in R_{po}$, then any role that appears (or dominates a role that appears) in the negative precondition of a CA entry assigning to r_p is in R_{ne} and 2) if $r_p \in R_{po}$, then any role that is (or dominates a role that is) mutually exclusive with r_p is in R_{ne}.

The preprocessing takes time that is at most cubic in the size of the URA-RC-SAP instance.

Experimental results. We performed experiments by using two kinds of instances. Manually crafted instances are designed to "hide" an unsafe state after a long sequence of transitions. These instances forced the analysis tools to search deep in the state space. Randomly generated instances contain a relatively large number of roles and transition rules.

Experiments were performed on a Windows workstation with an Intel Pentium 4 3 GHz CPU and 512 Mbytes of memory. We tested the performance of NuSMV and XSB on several instances. Table 1 presents results for seven instances, two of which were manually generated, and five were randomly generated.

Experimental results show that the number of rules is a crucial factor in determining the runtime. Therefore, the preprocessing step plays an important role in improving the efficiency of both logic programming and model checking implementations. Many instances such as Rand2 that cannot be solved within 30 minutes without preprocessing are solved within a few seconds with preprocessing.

Both XSB and NuSMV are efficient in cases that have a small number of transition rules. For the example in Section 2.4, XSB uses 0.016 second, and NuSMV uses 0.125 second. When tested on a manually crafted instance Man2 with 16 roles and 40 rules (29 were left after preprocessing), which requires a sequence of at least 22 transitions before reaching an unsafe state, XSB uses 0.05 second, and NuSMV uses 0.13 second. When preprocessing is effective such as instance Rand2 with 100 roles and

250 rules (20 rules left after preprocessing), XSB uses 0.55 second, and NuSMV executes in 0.94 second.

It appears that using XSB does not scale as well as using NuSMV. For example, when it came to the randomly generated instance Rand5 with 25 roles and 79 rules (57 left after preprocessing), XSB ran out of memory after 17 minutes, while NuSMV returned with an answer within 73 seconds. An observation is that the runtime of XSB grows linearly with the number of reachable states.[1] However, NuSMV uses binary decision diagrams (BDDs) to represent its state space, so its runtime depends on the regularity of the state space. A point worth mentioning is that XSB consumes memory quickly. For the instance Rand4 with 30 roles and 88 rules (27 were left after preprocessing), XSB uses more than 400 Mbytes of memory. The high demand on memory impairs the scalability of our XSB program. In contrast, the BDD-based NuSMV requires less memory than XSB.

Both model checking and logic programming have been used in network vulnerability analysis [18], [41]. Recently, Ou et al. [33] have shown that in the context of network vulnerability analysis, logic programming is much more scalable than model checking. Our experimentation data show that for URA-RC-SAP, the scalability of logic programming is worse than model checking. This is because in network vulnerability analysis, one can make the monotonicity assumption; i.e., if an attacker gains a privilege, it never loses it. However, in security analysis, because of negative preconditions and mutual-exclusion constraints, the monotonicity assumption does not hold, and one has to explore the state space.

In real-word large-scale RBAC systems, even though the number of roles in the whole system may be large, we expect that the roles that are relevant for any given query will be only a small portion of all roles. Therefore, we conjecture that our approach of combining preprocessing with existing tools such as NuSMV will be able to handle many queries.

5 Related Work

In their landmark paper [17], Harrison et al. formalized the safety analysis problem in the access matrix model. The problem determines whether a protection system can reach a state in which a particular right is leaked. They show that safety analysis is undecidable in their scheme [17]. Since then, safety analysis has attracted considerable attention in the research community. Safety analysis in monotonic versions of the HRU scheme has been studied in [16]. Jones et al. introduced the Take-Grant scheme [19], in which safety can be decided in linear time. Sandhu introduced the Schematic Protection Model [35], the Extended Schematic Protection Model [1], and the Typed Access Matrix model [36]. Budd [5] and Motwani et al. [30] studied grammatical protection systems. Soshi [43] studied safety analysis in the Dynamic-Typed Access Matrix model. These models all have subcases where safety is decidable. Solworth and Sloan [42] introduced a discretionary access control model in which safety is decidable. This thread of research has produced many new access control schemes but has had limited impact on access control systems used in practice, probably because the proposed schemes are either too simplistic to be useful or too arcane to be usable. In this paper, we focus on policy analysis problems in RBAC, which was invented not for the purpose of safety analysis but for meeting the access control need of real-world applications.

Influential works on RBAC include the pioneering work of Ferraiolo et al. [11], [12] and the widely cited RBAC96 family of formal RBAC models developed by Sandhu et al. [39]. Recently, a standard for RBAC has been proposed and adopted as an ANSI Standard [3], [13]. The administration of RBAC is about controlling who can update the various relations in an RBAC system. The most well-known work on the administration of RBAC is ARBAC97, developed by Sandhu et al. [37], [38]. Recently, Crampton and Loizou [9] have introduced the notion of administrative scope and an RBAC administration scheme based on it.

An administrative scheme, in conjunction with the representation for an RBAC state, naturally lends itself to the safety question in RBAC. The work that is closest to this paper is such works on safety and security analysis in RBAC. Li and Tripunitara [26] studied security analysis for two particular RBAC schemes derived from ARBAC97 [38]—Assignment and Trusted Users (AATU) and Assign and Revocation (AAR)—both of which are subschemes of the URA97 scheme [38]. The main results in [26] are that security analysis in AATU and AAR are intractable (**NP**-hard) in general but can be solved in polynomial time for semistatic queries. The intractability results there are consequences of the fact that a query may be able to encode an arbitrary Boolean formula. The techniques used to establish tractable results were to reduce the problem to security analysis in the RT family of trust management languages [25]. We observe that neither AATU nor AAR allows negative preconditions or constraints. We have shown that URA-RC-SAP and these restrictions are solvable in quadratic time, given direct algorithm for solving them. We point out that even though the role containment queries are special cases of semistatic queries, our two tractable cases do not follow from results in [26], because AATU does not allow revocation, and AAR does not allow trusted users. In essence, the results in [26] deal with very simple state-transition rules but sophisticated queries. In this paper, we consider simple queries but sophisticated state transitions. Since RT [24] is monotonic, it is unclear how the techniques in [26] can be extended to deal with negative preconditions or constraints. Li and Tripunitara [26] explicitly mentioned dealing with negative preconditions and constraints as an open problem.

Koch et al. have proposed an RBAC scheme based on a graph-based formalism [23] and have demonstrated that safety is decidable in a subscheme [22]. However, the decidable fragment of the graph-based formalism [22] does not allow negative application conditions, which are used to specify negative preconditions in assignment rules in the graph-based formalism for RBAC [23]. Therefore, the decidability result applies only to the subcase without

1. The statistics on the total states and the reachable states in Table 1 are actually for NuSMV, but the statistics for XSB should be similar.

negative preconditions or mutual-exclusion constraints. Furthermore, in [22], it has only been shown that safety is decidable in this case. No concrete computational complexity result is given in [22]. The proof shows that the search space is finite; however, searching the space likely takes exponential time. We show that for the case that can be modeled in the decidable fragment, namely, without negative precondition or constraints, URA-RC-SAP is decidable in quadratic time. For cases with negative preconditions and/or constraints, we have given precise computational complexities for them.

Some work related to safety in access control (e.g., [42]) refers to the work by Crampton [8] and Munawer and Sandhu [31] to claim that safety is undecidable in the ARBAC97 scheme. We point out that the undecidability results in Crampton [8] and Munawer and Sandhu [31] are not about the ARBAC97 scheme. The scheme considered by Crampton [8] adds two new features to ARBAC97. One is to allow changes in the CA and CR relations. (Sandhu et al. [38] state specifically that it is assumed that in an ARBAC97 system, these relations are static and may be changed only by (a trusted) chief security officer.) The other is to allow a state-change rule to include an arbitrary command specified using a construct similar to that proposed by Harrison et al. [17]. Such constructs do not exist in ARBAC97. Munawer and Sandhu [31] present a simulation of the Augmented Typed Access Matrix (ATAM) scheme [2] in a particular RBAC scheme that has similar features as those in [8].

Sasturkar et al. [44] also studied policy analysis problems in ARBAC97-based systems. They established a connection between the analysis problem and planning in artificial intelligence. Our work has a number of differences. First, they showed only that the analysis problem is **PSPACE**-complete when revocation rules also have preconditions, which are not in the ARBAC97 model. As the **PSPACE**-hardness result is by a reduction from a planning problem and the reduction requires revocation rules to have preconditions, the results in [44] cannot entail that URA97-RC-SAP is PSPACE-hard. Our proof uses a direct reduction from the membership problem for LBA, and our result that URA-RC-SAP is **PSPACE**-complete is stronger, because it implies the **PSPACE**-completeness result in [44] (proving that the in **PSPACE** part is straightforward). Second, in addition to complexity results, we have also developed tools for such an analysis using model checking tools and logic programming and have experimentally evaluated the two approaches. Third, the use of planning in [44] enables the authors to obtain results on a wider class of problems than those studied in this paper. For example, they also consider special cases where the number of literals in a precondition is limited to a small number.

6 Conclusion and Future Work

We have formalized classes of security analysis in the context of RBAC. We have shown that URA-SAP is **PSPACE**-complete in the general case and that a number of special cases of the problems are **NP**-complete. We have also shown that model checking is a promising approach for solving these problems. In the future, we plan to look at more sophisticated queries and other administration schemes.

Appendix A

Turing Machines

A *TM* is denoted as $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, where Q is the finite set of *states*, Γ is the finite set of allowable *tape symbols*, $B \in \Gamma$ is the *blank* symbol, $\Sigma \subseteq \Gamma - \{B\}$ is the set of *input symbols*, δ is the *next move function* and is a partial function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$, $q_0 \in Q$ is the *start state*, and $F \subseteq Q$ is the set of *final states*. An *NDTM* allows a finite number of choices for its next move; i.e., δ is a function from $Q \times \Gamma$ to the power set of $Q \times \Gamma \times \{L, R\}$. The first definition describes a deterministic TM (DTM).

The *language accepted* by M (denoted by $L(M)$) is the set of words in $\Sigma^\star$ that cause M to enter a final state when placed justified at the left on the tape of M, with M in state q_0, and the tape head of M at the leftmost cell. A language L is accepted by a DTM if and only if it is accepted by an NDTM. A language L is said to be in $DSPACE(S(n))$ if there exists a DTM M accepting L that takes at most $S(n)$ space, where n is the size of the input. Similarly, L is said to be in $NSPACE(S(n))$ if there exists an NDTM M accepting L that takes at most $S(n)$ space. A language L is said to be in **PSPACE** if and only if it is in $DSPACE(p(n))$ for some polynomial p (L is accepted by a DTM that takes polynomial space in the size of the input). We refer the reader to [34] for more details on these and related concepts.

Appendix B

Proof of Lemma 3

We show that URA-RC-SAP is **PSPACE**-hard by a reduction from the membership problem for LBA, which is known to be **PSPACE**-complete. An LBA is a restricted form of a TM. It differs from a TM in that while the tape is initially considered infinite, only a finite contiguous portion whose length is a linear function of the length of the initial input can be accessed by the read/write head.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be any LBA that uses at most $p(n)$ space, where n is the size of the input, and p is a degree-1 polynomial. (We assume that the polynomial $p(n)$ is known. However, it suffices that there exists a polynomial time algorithm for computing $p(n)$, given n.) We construct an RBAC system whose start state is $\gamma = \langle UA, PA, RH, CA, CR, CO \rangle$ and a query $u \in r$ corresponding to the DTM M so that there is an accepting computation in M on an input x if and only if in the RBAC system, there is a sequence of RBAC states from γ (which corresponds to input x) to γ', in which $u \in r$ is true.

In our construction, there are two users: u_0 and u. There is a special role ra (the reader can think of this as the administrative role), and $(u_0, ra) \in UA$. All the assigning and revoking of roles will be performed by u_0. The special user u_0 can remove users from all roles; i.e., CR is equal to $\{\langle ra, R \rangle\}$, where R consists of all the roles introduced in the construction, except for ra.

Encoding TM configurations. For each state $q \in Q$ and $1 \leq i \leq p(n)$, we introduce a role $r_{i,a}$. For each state $q \in Q$,

$1 \leq i \leq p(n)$, and symbol a in Σ, we introduce a role $r_{q,i,a}$. These roles are used to represent the configuration of the TM M. Additional roles will be introduced later to simulate transitions in M. If u is assigned to role $r_{i,a}$, then the ith cell contains a, and the tape head is not on cell i. For $1 \leq i \leq p(n)$, u assigned to role $r_{q,i,a}$ indicates that the tape head is on cell i, M is in state q, and the ith cell contains a. We add the following mutual-exclusion constraints to CO to maintain the integrity of the encoding. We only add a polynomial number of constraints. We use (r_1, r_2) as a shorthand for the mutual-exclusion constraint that a user cannot be a member of both r_1 and r_2 (this is equivalent to $\mathrm{smer}\langle\{r_1, r_2\}\rangle, 2\rangle$ in our earlier notation):

- *M can only be in one state at a time.* We add mutual-exclusion constraints of the form $(r_{q,i,a}, r_{q',i,a})$ for all $q \neq q'$, i in the range $[1, \cdots, p(n)]$, and $a \in \Sigma$.
- *The tape head can only point to one location.* We add mutual-exclusion constraints of the form $(r_{q,i,a}, r_{q,j,a'})$ for all $i \neq j$ and for all q, a, and a'.
- *One location cannot both have the head and not have the head at the same time.* We also add mutual-exclusion constraints of the form $(r_{i,a}, r_{q,i,a})$.
- *Each tape cell can only contain one symbol.* Therefore, we add constraints of the form $(r_{i,a}, r_{i,a'})$ and $(r_{q,i,a}, r_{q,i,a'})$ for all i, $a \neq a'$, and $q \in Q$.

Encoding the initial configuration. Assume that M starts in the initial state q_0, the first n cells of the tape contain $a_1, \cdots, a_n$ (where $a_i \in \Gamma$), the rest of the tape cells consist of blank symbols B, and the tape head points to the first cell. The initial state of the RBAC system has user u in role $r_{q_0,1,a_1}$, r_{i,a_i} for $2 \leq i \leq n$, and $r_{j,B}$ for $n+1 \leq j \leq p(n)$.

Encoding the halting states. We use one role r_{target} to be used in the query and add the following tuples to CA:

- For each i in the range $[1, \cdots, p(n)]$, $a \in \Sigma$, and accepting state q, add $\langle ra, r_{q,i,a}, \{r_{target}\}\rangle$.

This ensures that the TM M enters an accepting state if and only if u can be assigned to r_{target}. The query in the URA-RC-SAP instance that we are constructing is $u \in r_{target}$.

Encoding the next-move function. During each transition in M, two revocations and two assignments need to be done to ensure that the next configuration is correctly represented by the role memberships. We need to make these changes transactional. Therefore, we need to introduce some control roles. We will first present the construction and then explain how a state transition occurs.

We first introduce two roles: r_b and r_c. Initially, u is assigned to r_b but not to r_c. The following tuple is added to CA:

- $\langle ra, r_c, \{r_b\}\rangle$.

Suppose that $\delta(q, a)$ is equal to (q', a', L) (the case when $\delta(q, a) = (q', a, R)$ is similar). The transition $\delta(q, a) = (q', a', L)$ is modeled by doing the following:

- For each $2 \leq i \leq p(n)$, add two roles: $r^{lb}_{i,q',a'}$ and $r^{lc}_{i,q',a'}$. Initially, u is not assigned to any of these roles. We add the following mutual-exclusion constraints:
  - For each $2 \leq i \leq p(n)$, add constraints of the form $(r_b, r^{lc}_{i,q',a'})$.
  - For each $2 \leq i \leq p(n)$, add constraints of the form $(r^{lb}_{i,q',a'}, r_c)$.
- For each $2 \leq i \leq p(n)$ and each $a_\ell \in \Sigma$, add a role r^{ld}_{i,q',a_ℓ}. We add the following mutual-exclusion constraints:
  - For each $a \leq i \leq p(n) - 1$, add constraints of the form $(r^{ld}_{i,q',a'}, r_b)$.
- We then add the following tuples to the relation CA

  Step 1. For all $2 \leq i \leq p(n)$, we add

  $$\langle ra, r_{q,i,a} \cap r_b, \{r^{lb}_{q',i,a'}\}\rangle.$$

  Step 2. For all $2 \leq i \leq p(n)$, we add

  $$\langle ra, r^{lb}_{q',i,a'}, \{r^{lc}_{q',i,a'}\}\rangle.$$

  Step 3. For all $2 \leq i \leq p(n)$, we add

  $$\langle ra, r^{lc}_{q',i,a'}, \{r_{i,a'}\}\rangle.$$

  Step 4. For all $2 \leq i \leq p(n)$, for each $a_\ell \in \Sigma$, we add

  $$\langle ra, r^{lc}_{q',i,a'} \cap r_{i,a'} \cap r_{i-1,a_\ell}, \{r^{ld}_{q',i,a_\ell}\}\rangle.$$

  Step 5. For all $2 \leq i \leq p(n)$, for each $a_\ell \in \Sigma$, we add

  $$\langle ra, r^{ld}_{q',i,a_\ell}, \{r_{q',i-1,a_\ell}\}\rangle.$$

  Step 6. For all $2 \leq i \leq p(n)$, for each $a_\ell \in \Sigma$, we add

  $$\langle ra, r^{ld}_{q',i,a'} \cap r_{q',i-1,a_\ell}, \{r_c\}\rangle.$$

Transitions in the RBAC system. Suppose that at one point of the computation of M, the tape head is at position i, the ith cell contains a, and the current state is q. Then, u is in the role $r_{q,i,a}$. The only way to proceed in the RBAC system is to use the CA tuples added in step 1 to assign u to $r^{lb}_{q',i,a'}$. This can succeed only when u is in r_b. Initially, u is in r_b, and we will show that u can be assigned to r_b after a transition in M has been simulated.

Once u is assigned to r_b and then to $r^{lb}_{q',i,a'}$, the only way to make progress in the RBAC system is to use the CA tuples added in step 2 to assign u to $r^{lc}_{q',i,a'}$; however, because of the constraints, one has to revoke u from r_b first.

Once u is assigned to r_b and then to $r^{lc}_{q',i,a'}$, u can be assigned (using the tuples added in step 3) to $r_{i,a'}$ (after revoking u from $r_{q,i,a}$ first due to the constraints added for tape integrity). To update roles corresponding to the $(i-1)$th cell, let a_ℓ be the symbol on the $(i-1)$th cell, u can be assigned (using the tuples added in step 4) to r^{ld}_{q',i,a_ℓ}. Using the tuples in step 5, u can be assigned to $r_{q',i-1,a_{cell}}$. Finally, after the tape representation roles have been updated, u can be assigned to r_c. Before doing this, u has to be revoked from all roles of the form $r^{lb}_{i,q',a'}$ because of the constraints.

To make the next state transition, u must be assigned to r_b, which requires u to be revoked from all roles of the form $r^{lc}_{i,q',a'}$ and r^{ld}_{i,q',q_ℓ}, clearing all the intermediate roles used in the simulation.

Summary. An *instantaneous description (ID)* of a TM M is given by the contents of the tape, the position of the head, and the state of M. Given two IDs ID_1 and ID_2, we say that $ID_1 \rightarrow_M ID_2$ if ID_2 follows from ID_1 by one move of the M. Given an input x, let $ID_0, \cdots, ID_n$ be a sequence of IDs such that ID_0 corresponds to the input x, $ID_i \rightarrow_M ID_{i+1}$, and ID_n has an accepting state (the sequence is an accepting computation to the string). Let γ_0 be the RBAC state that corresponds to ID_0. Each move of the TM can be emulated by the RBAC scheme in a number of steps. Let this constant be c. Therefore, there exists a sequence of RBAC states $\gamma_0, \cdots, \gamma_{cn}$ such that γ_{ci} encodes ID_i, $\gamma_i \rightarrow \gamma_{i+1}$, and in γ_{cn}, u is in a role $r_{q,i,a}$ such that $q \in F$. Then, u can be assigned to r_{target}. Hence, an accepting computation in M has a corresponding sequence in the RBAC system.

Our discussion of state transitions in the RBAC system above shows that state changes in the RBAC systems correspond to computations. If a user u is assigned role $r_{q,i,a}$ in some state in the RBAC system, there is a corresponding computation in the TM M that reaches state q and has the head on the ith position with the symbol a in the ith position.

Therefore, there is an accepting computation in TM on an input x if and only if there is a sequence of RBAC states from γ (which corresponds to input x) to γ_m, where u is in role r_{target}. This proves the result. □

APPENDIX C

PROOF OF LEMMA 6

Proof. We reduce 3SAT to the URA-RC-SAP, which proves that URA-RC-SAP is NP-hard. Let $f = c_1 \wedge \cdots \wedge c_m$ be an instance of 3SAT and let $p_1, \ldots, p_n$ be the propositional variables in f. We associate a role r_f with the formula f. We now construct an instance of URA-RC-SAP $[CA\ (\text{conjunctive}), CR = \{\ \}, CO = \{\ \}]$ so that a user becomes a member of r_f if and only if f is satisfiable.

In addition to f, $\mathcal{R}$ contains roles $c_1, \ldots, c_m$ corresponding to the clauses and roles $p_1, \ldots, p_n$ corresponding to the propositional variables, plus a role t that will be used for signaling, as will be explained shortly. We construct CA so as to allow u to be assigned to any combination of the p_i's. Once this is done, CA will permit u to be assigned to role t, signaling that a second phase has begun, in which u can be added to role c_j just in case u's assignment to the p_i's represents a truth assignment that makes clause c_j true. More precisely,

$$\begin{aligned} CA = &\{\langle a, \neg t, \{p_1, \ldots, p_n\}\rangle, \langle a, true, \{t\}\rangle, \langle a, c_1 \cap \cdots \cap c_m, f\rangle\} \\ &\cup \{\langle a, t \cap p_i, \{c_j | 1 \le j \le m \wedge p_i \text{ appears positively in } c_j\}\rangle \\ &| 1 \le i \le n\} \cup \{\langle a, t \cap \neg p_i, \{c_j | 1 \le j \le m \wedge p_i \\ &\text{appears negatively in } c_j\}\rangle | 1 \le i \le n\}, \end{aligned}$$

and $CR = \emptyset$.

We now consider an instance of URA-RC-SAP, in which A_T, UA, PA, and RH are empty, and show that it is true if and only if f is satisfiable. If the formula f is satisfiable, it is easy to see that a can add u to role f by first adding u to the role p_i if the propositional variable p_i is true in the solution to f, then adding u to t, and then adding u to each c_j and, finally, to f.

Conversely, if the problem instance is true, then at some point, u must be added to role t. Since u cannot be removed from t, u's assignment to the p_i roles at that time enables u to be added to each role c_j. By defining the propositional variables p_i to be true if and only if the role p_i contains u at that time, we get an assignment that makes at least one literal true in each clause c_j. □

APPENDIX D

PROOF OF LEMMA 7

Proof. We reduce monotone 3SAT to the problem. Monotone 3SAT is a special case of 3SAT, where all literals in a clause are either all positive or all negative. Monotone 3SAT is known to be **NP**-complete [14].

Let $e = c_1 \wedge \ldots \wedge c_l \wedge \overline{c_{l+1}} \wedge \ldots \wedge \overline{c_n}$ be an instance of monotone 3SAT, where $c_1, \ldots, c_l$ are the clauses with only positive literals, and $\overline{c_{l+1}}, \ldots, \overline{c_n}$ are the clauses with only negative literals. Let $p_1, \ldots, p_k$ be all the propositional variables that appear in e, each $c_i = p_{i_1} \vee p_{i_2} \vee p_{i_3}$, and each $\overline{c_j} = \neg p_{j_1} \vee \neg p_{j_2} \vee \neg p_{j_3}$. We produce a corresponding URA-RC-SAP instance for $RBAC_{assign,nonegation}$ as follows:

Corresponding to each propositional variable p_i, we have role r_i. We also have role r_{c_i}, corresponding to each positive clause c_i in e. In addition, we have roles r and a. We assign user u_0 to a, that is, $\langle u_0, a\rangle \in UA$. Role a is an administrative role and appears as the first component of every entry in CA. We first add the tuple $\langle a, r_{c_1} \wedge \ldots \wedge r_{c_l}, r\rangle$ to CA. Corresponding to each positive clause $c_i = p_{i_1} \vee p_{i_2} \vee p_{i_2}$, we add the three entries $\langle a, r_{i_1}, r_{c_i}\rangle, \langle a, r_{i_2}, r_{c_i}\rangle, \langle a, r_{i_3}, r_{c_i}\rangle$ to CA. If c_i has fewer than three literals, then we only add such entries to CA that correspond to the literals in c_i. Clearly, the CA so constructed has no negation in the preconditions of its entries, and each precondition is a conjunction of roles. We capture the negative clauses in e by using entries in CO. For each negative clause $\overline{c_j} = \neg p_{j_1} \vee \neg p_{j_2} \vee \neg p_{j_3}$, we add the constraint $\text{smer}\langle\{r, r_{j_1}, r_{j_2}, r_{j_3}\}, 4\rangle$. If $\overline{c_j} = \neg p_{j_1} \vee \neg p_{j_2}$ (that is, has only two literals), then we add the constraint $\text{smer}\langle\{r, r_{j_1}, r_{j_2}\}, 3\rangle$ to CO, and if $\overline{c_j} = \neg p_{j_1}$ (has only one literal), then we add the constraint $\text{smer}\langle\{r, r_{j_1}\}, 2\rangle$ to CO.

We ensure that user u that is specified in the query q is not a member of r or any r_{c_i} in the start state γ. We now assert that there exists a reachable state in which u is a member of r if and only if e is satisfiable. The reason is that the only way that u can become a member of r is by u_0 successfully exercising the only entry in CA that has r as the target role (the last component of the tuple in CA). This is possible if and only if u already satisfies the role memberships, as specified in the precondition, and assigning u to r does not violate any of the entries in CO. More formally, for the "if" part, assume that e is satisfiable. Then, there is some truth assignment that makes e true. We use the truth assignment as the user-role assignment in γ for u. That is, if the propositional variable p_i is true in the truth assignment that makes e true, then $\langle u, r_i\rangle \in UA$. Now, u_0 will be able to assign u to a role r_{c_i} if and only if $\langle u, r_{i_1}\rangle \in UA$, $\langle u, r_{i_2}\rangle \in UA$, or $\langle u, r_{i_3}\rangle \in UA$, where $c_i = p_{i_1} \vee p_{i_2} \vee p_{i_3}$, and none of the

entries in CO is violated. An entry in CO is violated if and only if u is a member of all roles other than r in the set of the roles in a constraint. This is the case if and only if the corresponding negative clause is false, which is impossible, given the assumption that e is satisfiable.

For the "only if" part, assume that there exists a reachable state in which u is a member of r. We use the role memberships of u in the roles $r_1, \ldots, r_k$ in the start state γ as our truth assignment for e. That is, if $\langle u, r_i \rangle \in UA$, then we set the corresponding propositional variable p_i to be true. Given that u can eventually be assigned to r, we know that u can be assigned to every r_{c_i} in γ. Therefore, each positive clause is true. Furthermore, given that in the final action, we can assign u to r, we know that no entry in CO is violated. Consider the constraint $\mathrm{smer}(\{r, r_{i_1}, r_{i_2}, r_{i_3}\}, 4)$. This constraint would disallow u_0 from assigning u to r if and only if u is already a member of all of r_{i_1}, r_{i_2}, and r_{i_3}. As this is not the case, we know that every negative clause evaluates to true. □

REFERENCES

[1] P. Ammann and R.S. Sandhu, "Safety Analysis for the Extended Schematic Protection Model," *Proc. IEEE Symp. Security and Privacy (S&P '91),* pp. 87-97, May 1991.

[2] P. Ammann and R.S. Sandhu, "Implementing Transaction Control Expressions by Checking for Absence of Access Rights," *Proc. Eighth Ann. Computer Security Applications Conf. (ACSAC '92),* Dec. 1992.

[3] *American National Standard for Information Technology - Role Based Access Control,* ANSI INCITS 359-2004, Am. Nat'l Standards Inst., Feb. 2004.

[4] R. Awischus, "Role-Based Access Control with the Security Administration Manager (SAM)," *Proc. Second ACM Workshop Role-Based Access Control Table of Contents (RBAC '97),* pp. 61-68, 1997.

[5] T. Budd, "Safety in Grammatical Protection Systems," *Int'l J. Computer and Information Sciences,* vol. 12, no. 6, pp. 413-430, 1983.

[6] W. Chen and D.S. Warren, "Tabled Evaluation with Delaying for General Logic Programs," *J. ACM,* vol. 43, no. 1, pp. 20-74, Jan. 1996.

[7] E.M. Clarke, O. Grumberg, and D.A. Peled, *Model Checking.* MIT Press, 2000.

[8] J. Crampton, "Authorizations and Antichains," PhD dissertation, Univ. of London, U.K., 2002.

[9] J. Crampton and G. Loizou, "Administrative Scope: A Foundation for Role-Based Administrative Models," *ACM Trans. Information and System Security,* vol. 6, no. 2, pp. 201-231, May 2003.

[10] W.F. Dowling and J.H. Gallier, "Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae," *J. Logic Programming,* vol. 1, no. 3, pp. 267-284, 1984.

[11] D.F. Ferraiolo, J.A. Cuigini, and D.R. Kuhn, "Role-Based Access Control (RBAC): Features and Motivations," *Proc. 11th Ann. Computer Security Applications Conf. (ACSAC '95),* Dec. 1995.

[12] D.F. Ferraiolo and D.R. Kuhn, "Role-Based Access Control," *Proc. 15th Nat'l Information Systems Security Conf.,* 1992.

[13] D.F. Ferraiolo, R.S. Sandhu, S. Gavrila, D.R. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-Based Access Control," *ACM Trans. Information and Systems Security,* vol. 4, no. 3, pp. 224-274, Aug. 2001.

[14] M.R. Garey and D.J. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., 1979.

[15] T.X.R. Group, *The XSB Programming System,* http://xsb.sourceforge.net/, 2008.

[16] M.A. Harrison and W.L. Ruzzo, "Monotonic Protection Systems," *Foundations of Secure Computation,* R.A. DeMillo, D.P. Dobkin, A.K. Jones, and R.J. Lipton, eds., Academic Press, pp. 461-471, 1978.

[17] M.A. Harrison, W.L. Ruzzo, and J.D. Ullman, "Protection in Operating Systems," *Comm. ACM,* vol. 19, no. 8, pp. 461-471, Aug. 1976.

[18] S. Jha, O. Sheyner, and J.M. Wing, "Two Formal Analysis of Attack Graphs," *Proc. 15th IEEE Computer Security Foundations Workshop (CSFW '02),* June 2002.

[19] A.K. Jones, R.J. Lipton, and L. Snyder, "A Linear Time Algorithm for Deciding Security," *Proc. 17th Ann. IEEE Symp. Foundations of Computer Science (FOCS '76),* pp. 33-41, Oct. 1976.

[20] G. Karjoth, "The Authorization Model of Tivoli Policy Director," *Proc. 17th Ann. Computer Security Applications Conf. (ACSAC '01),* pp. 319-328, Dec. 2001.

[21] A. Kern, "Advanced Features for Enterprise-Wide Role-Based Access Control," *Proc. 18th Ann. Computer Security Applications Conf. (ACSAC '02),* pp. 333-343, Dec. 2002.

[22] M. Koch, L.V. Mancini, and F. Parisi-Presicce, "Decidability of Safety in Graph-Based Models for Access Control," *Proc. Seventh European Symp. Research in Computer Security (ESORICS '02),* pp. 229-243, Oct. 2002.

[23] M. Koch, L.V. Mancini, and F. Parisi-Presicce, "A Graph-Based Formalism for RBAC," *ACM Trans. Information and System Security,* vol. 5, no. 3, pp. 332-365, Aug. 2002.

[24] N. Li, J.C. Mitchell, and W.H. Winsborough, "Design of a Role-Based Trust Management Framework," *Proc. IEEE Symp. Security and Privacy (S&P '02),* pp. 114-130, May 2002.

[25] N. Li, J.C. Mitchell, and W.H. Winsborough, "Beyond Proof-of-Compliance: Security Analysis in Trust Management," *J. ACM,* vol. 52, no. 3, pp. 474-514, A preliminary version appeared in Proc. 2003 IEEE Symp. Security and Privacy (S&P), May 2005.

[26] N. Li and M.V. Tripunitara, "Security Analysis in Role-Based Access Control," *Proc. Ninth ACM Symp. Access Control Models and Technologies (SACMAT '04),* pp. 126-135, June 2004.

[27] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems.* Springer, 1992.

[28] Z. Manna and A. Pnueli, *Temporal Verification of Reactive Systems: Safety.* Springer, 1995.

[29] D. McPherson, *Role-Based Access Control for Multi-Tier Applications Using Authorization Manager,* http://www.microsoft.com/technet/prodtechnol/windowsserver2003/technologies/management/athmanwp.mspx, 2008.

[30] R. Motwani, R. Panigrahy, V.A. Saraswat, and S. Ventkatasubramanian, "On the Decidability of Accessibility Problems," *Proc. 32nd Ann. ACM Symp. Theory of Computing (STOC '00),* extended abstract, pp. 306-315, May 2000.

[31] Q. Munawer and R.S. Sandhu, "Simulation of the Augmented Typed Access Matrix Model (ATAM) Using Roles," *Proc. ACM Int'l Conf. Information and Security (INFOSECU),* 1999.

[32] *NuSMV: A New Symbolic Model Checker,* http://afrodite.itc.it:1024/nusmv/, 2008.

[33] X. Ou, S. Govindavajhala, and A.W. Appel, "MulVAL: A Logic-Based Network Security Analyzer," *Proc. 14th Usenix Security Symp.,* Aug. 2005.

[34] C.H. Papadimitriou, *Computational Complexity.* Addison-Wesley-Longman, 1994.

[35] R.S. Sandhu, "The Schematic Protection Model: Its Definition and Analysis for Acyclic Attenuating Systems," *J. ACM,* vol. 35, no. 2, pp. 404-432, 1988.

[36] R.S. Sandhu, "The Typed Access Matrix Model," *Proc. IEEE Symp. Security and Privacy (S&P),* pp. 122-136, May 1992.

[37] R.S. Sandhu and V. Bhamidipati, "Role-Based Administration of User-Role Assignment: The URA97 Model and Its Oracle Implementation," *J. Computer Security,* vol. 7, 1999.

[38] R.S. Sandhu, V. Bhamidipati, and Q. Munawer, "The ARBAC97 Model for Role-Based Administration of Roles," *ACM Trans. Information and Systems Security,* vol. 2, no. 1, pp. 105-135, Feb. 1999.

[39] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman, "Role-Based Access Control Models," *Computer,* vol. 29, no. 2, pp. 38-47, Feb. 1996.

[40] A. Schaad, J. Moffett, and J. Jacob, "The Role-Based Access Control System of a European Bank: A Case Study and Discussion," *Proc. Sixth ACM Symp. Access Control Models and Technologies (SACMAT '01),* pp. 3-9, 2001.

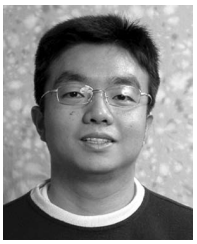
[41] O. Sheyner, J.W. Haines, S. Jha, R. Lippmann, and J.M. Wing, "Automated Generation and Analysis of Attack Graphs," *Proc. IEEE Symp. Security and Privacy (S&P),* 2002.

[42] J.A. Solworth and R.H. Sloan, "A Layered Design of Discretionary Access Controls with Decidable Safety Properties," *Proc. IEEE Symp. Security and Privacy (S&P),* May 2004.
[43] M. Soshi, "Safety Analysis of the Dynamic-Typed Access Matrix Model," *Proc. Sixth European Symp. Research in Computer Security (ESORICS '00),* pp. 106-121, Oct. 2000.
[44] A. Sasturkar, P. Yang, S. Stoller, and C. Ramakrishnan, "Policy Analysis for Administrative Role-Based Access Control," *Proc. 19th Computer Security Foundations Workshop (CSFW '06),* July 2006.

Somesh Jha received the BTech degree in electrical engineering from the Indian Institute of Technology, New Delhi, and the PhD degree in computer science from Carnegie Mellon University in 1996. He is currently an associate professor in the Department of Computer Sciences, University of Wisconsin, Madison. His research interests include the analysis of security protocols, survivability analysis, intrusion detection, formal methods for security, combating malicious code, and, recently, privacy-preserving protocols.

Ninghui Li received the BEng degree in computer science from the University of Science and Technology of China in 1993, and the MSc and PhD degrees in computer science from New York University, in 1998 and 2000, respectively. In 2003, he joined Purdue University, where he is currently an assistant professor in the Department of Computer Science. He was a research associate in the Department of Computer Science, Stanford University. His research interests include security and privacy in information systems, in particular access control. He has worked on projects on trust management, automated trust negotiation, role-based access control, online privacy protection, privacy-preserving data publishing, and operating system access control. He has published more than 60 technical papers in refereed journals and conference proceedings and has served on the program committees of more than three dozen international conferences and workshops. He is a senior member of the IEEE, the IEEE Computer Society, and a member of the ACM and the Usenix.

Mahesh Tripunitara received the PhD degree in computer science from Purdue University, where he specialized in information security. He is currently a principal technical staff in the Security and Privacy Technology Laboratory, Motorola Labs.



Qihua Wang received the BS degree in computer science from the University of Science and Technology of China (USTC) in 2004 and the MS degree from Purdue University in 2007. He is currently working toward the PhD degree in the Department of Computer Science, Purdue University. His research interests include access control policy management, privacy-driven access control in database, and role management in enterprises.

William H. Winsborough received the BA, MS, and PhD degrees from the University of Wisconsin, Madison. He has held several research positions in the academe and the industry. He is currently an associate professor in the Department of Computer Science, University of Texas at San Antonio. His research interests include computer security and privacy, in particular policy-language systems with provable security properties. He is a member of the IEEE Computer Society.