Role Updating for Assignments

Jinwei Hu[†‡], Yan Zhang[‡], Ruixuan Li[†*], and Zhengding Lu[†]
[†]Intelligent and Distributing Computing Laboratory, School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan 430074, China
[‡]Intelligent Systems Laboratory, School of Computing and Mathematics
University of Western Sydney, Sydney 1797, Australia
{jwhu,rxli,zdlu}@hust.edu.cn yan@scm.uws.edu.au

ABSTRACT

The role-based access control (RBAC) has significantly simplified the management of users and permissions in computing systems. In dynamic environments, systems are usually undergoing changes, whereas the associated user-role, role-role and role-permission relations need to be updated accordingly in order to reflect the systems' evolutions. However, such updating process is generally complicated as the resulting system state is expected to meet necessary constraints. This paper presents an approach for assisting administrators with the update task: using this approach, it is possible to check, in an automatic way, whether a required update is achievable or not, and if so, a reference model will be produced. In light of this model, administrators could fulfill the changes to RBAC systems. We propose a formalization of the update approach, investigate its properties, and develop an updating algorithm based on model checking techniques. Our experimental results demonstrate the effectiveness of our approach.

Categories and Subject Descriptors

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

General Terms

Management, Security, Verification, Performance

Keywords

RBAC, Role Updating, Role Engineering, Model Checking

1. INTRODUCTION

Role-based access control (RBAC) is an effective mechanism for simplifying the administration of users and permissions in computing systems [11, 26]. In RBAC systems, users are associated with roles such as manager and employee, and a role in turn is defined as a set of permissions; users acquire permissions via roles. Essentially, an RBAC system manages three relations: user-role relation, role-role relation, and role-permission relation. Owing to RBAC's advantages (e.g., being policy-neutral and the ability to support a wide range of access control requirements) it is widely supported in commercial operating systems and database systems.

However, the administration of large RBAC systems is a complex and challenging task. As organizations undergo changes, the definition of roles (in terms of permissions) may need updating to reflect those changes. We refer to the updating of user-role, role-role, and role-permission relations as *role updating*. Role updating is a key component of role maintenance in role life-cycle [16], and takes a great proportion of the total cost of role maintenance [22].

Despite many convenient RBAC administration models (e.g., [8, 18, 25]) at our disposition, role updating is generally difficult and error-prone, because usually the resulting state is expected to meet various constraints. For example, updating is supposed not to unexpectedly change users' role set or permission set and thus hinder tasks being performed in the system; nor should it assign a student role to a user who is a professor in real-life, even though her permission set remains the same.

Given a high-level update request, the trial-and-error approach is work-intensive, inefficient and, when the request is not achievable at all, very frustrating. An updating strategy, if exists, may involve several administrative actions. The situation becomes worse as the RBAC systems grow large. Hence, a tool facilitating role updating is desired, as is also motivated by the following scenarios.

Repairing. Misconfigurations in access control systems can result in severe consequences [3]. As such, correcting misconfigurations is essential to systems' usability and security. It consists of two parts: identifying misconfigurations and changing access control policies. While the former has been studied, how to change access control policies is still an open problem, even though it may be clear who should implement the changes. As stated in [3]: "How the user elects to modify the policy is orthogonal to our work; it could entail changing an access-control list on a server or ..." Therefore, role updating might be needed under the circumstance that misconfigurations in RBAC systems are detected.

User-permission assignments. In task-based systems, each task is associated with a set of permissions [13].[1] The set of permissions should be assigned to a set of users so that the associated task could be accomplished. In systems where RBAC is deployed, the user-permission assignments are achieved via roles [2]. Considering the variety of tasks, it is likely that any combinations of roles fail to enable exactly the needed user-permission assignments for a task. In this case, to support the task, one may be willing to change the

*Corresponding author

SACMAT'10, June 9–11, 2010, Pittsburgh, Pennsylvania, USA.

[1]Permissions for tasks may be augmented with temporal constraints. One essential problem is to assign permissions.

current RBAC state, to some extent such as with the constraint that previous task assignments are not affected. Furthermore, the permissions required by users and tasks are not static. For one thing, modern enterprises and organizations need to adapt to constantly changing environments and requirements [22]. For another, it could happen that a group U of users are performing a task with a set P of permissions, and that it turns out that an extra permission $p_0 \notin P$ is a must at the late stage of the task; or the administrator may recognize that P contains more privileges than strictly necessary for the task, she may try to revoke redundant permissions from U [13].

To help system security officers (SSOs) understand and manage RBAC policies, various RBAC policy analysis tools have been invented [3, 14, 15, 19, 28, 31, 32, 36]. However, little effort has been devoted to answering what if the current RBAC state fails to meet a requirement. One may consider role updating as an unpleasant experience and thus avoid it. Also, it is because the updating is not straightforward, an automated assistant tool has some appeal.

In this paper, we investigate the role updating problem (RUP), with a bias on the update of role-permission relation, and provide an automated role update tool (Route). Note that Route is not designed to automatically change RBAC policies, but to, given a high-level update request, check whether the request is achievable and, if so, present a reference model, in light of which SSOs could implement changes to RBAC systems. Specifically, we make the following contributions:

- We formally define a basic version of RUP and a notion of *update constraint schema*, which is able to express a wide range of constraints on updates. We show the intractability of RUP.
- We present a set of reductions of RUP. The reductions can not only be used by Route, but also simplify RUP itself.
- We devise an automated tool Route for RUP based on model checking techniques, for which there exist mature tools that have been proven to work well in practice.
- Finally, we undertake a set of experiments, which illustrate the effectiveness and efficiency of Route, considering the use-pattern and frequency of role updating.

The remaining of the paper is organized as follows. Section 2 gives the preliminaries of the RBAC model used in this paper. A motivation example is presented in Section 3. In Section 4, we define RUP, show its complexity, and give the basic approach. The reductions of RUP are presented in Section 5. Experiment results are reported in Section 6. Related work is discussed in Section 7. We conclude in Section 8. In this paper, we write SSOs and administrators interchangeably; and a singleton set $\{s\}$ is sometimes written as s. Except for Theorem 2, proofs are given in [12].

2. PRELIMINARIES

The RBAC model used in this paper is defined as follows. An RBAC state γ is a tuple $\langle \gamma.U, \gamma.R, \gamma.P, \gamma.UA, \gamma.PA \rangle$, where $\gamma.U$ is a set of users, $\gamma.R$ is a set of roles, $\gamma.P$ is a set of permissions, $\gamma.UA \subseteq \gamma.U \times \gamma.R$ is user-role relation, and $\gamma.PA \subseteq \gamma.R \times \gamma.P$ is role-permission relation.

Given a state γ, we define several convenient functions. Function $\mathsf{users}_\gamma : 2^{\gamma.R} \cup 2^{\gamma.P} \mapsto 2^{\gamma.U}$ gives the set of users that roles and permissions are associated with: For $x \subseteq \gamma.R$, $\mathsf{users}_\gamma[x] = \bigcup_{r \in x} \{u \in \gamma.U \mid (u, r) \in \gamma.UA\}$; for $x \subseteq \gamma.P$, $\mathsf{users}_\gamma[x] = \mathsf{users}_\gamma[\mathsf{roles}_\gamma[x]]$.

Function $\mathsf{roles}_\gamma : 2^{\gamma.U} \cup 2^{\gamma.P} \mapsto 2^{\gamma.R}$ returns the set of roles that are related to users and permissions: for $x \subseteq \gamma.U$, $\mathsf{roles}_\gamma[x] = \bigcup_{u \in x} \{r \in \gamma.R \mid u \in \mathsf{users}_\gamma[r]\}$; for $x \subseteq \gamma.P$, $\mathsf{roles}_\gamma[x] = \bigcup_{p \in x} \{r \in \gamma.R \mid (r, p) \in \gamma.PA\}$.

Finally, function $\mathsf{perms}_\gamma : 2^{\gamma.U} \cup 2^{\gamma.R} \mapsto 2^{\gamma.P}$ maps the users and roles to their related permissions: for $x \subseteq \gamma.U$, $\mathsf{perms}_\gamma[x] = \bigcup_{u \in x} \{p \in \gamma.P \mid u \in \mathsf{users}_\gamma[p]\}$; for $x \subseteq \gamma.R$, $\mathsf{perms}_\gamma[x] = \bigcup_{r \in x} \{p \in \gamma.P \mid r \in \mathsf{roles}_\gamma[p]\}$.

Given a set of users U, a set of roles R, and a set of permissions P, we define a set of RBAC states $\Gamma = \{\langle \gamma.U, \gamma.R, \gamma.P, \gamma.UA, \gamma.PA \rangle \mid \gamma.U = U, \gamma.R = R, \gamma.P = P, \gamma.UA \subseteq U \times R, \gamma.PA \subseteq R \times P\}$. We say Γ is the *RBAC state space of* (U, R, P), denoted as $\mathsf{space}(U, R, P)$. Given $\gamma = \langle \gamma.U, \gamma.R, \gamma.P, \gamma.UA, \gamma.PA \rangle \in \mathsf{space}(U, R, P)$, we denote γ as $\langle \gamma.UA, \gamma.PA \rangle$ without explicitly enumerating $\gamma.U$, $\gamma.R$, and $\gamma.P$. We denote the RBAC space that γ resides in as $\mathsf{space}(\gamma)$, where $\mathsf{space}(\gamma) = \mathsf{space}(\gamma.U, \gamma.R, \gamma.P)$.

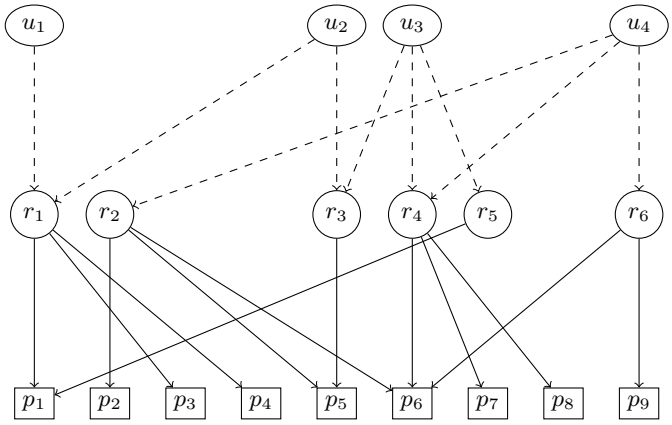


Figure 1: A running example RBAC state γ_{ex}. Users are in ellipses, roles in circles, and permissions in rectangles. The user-role relation $\gamma_{\mathsf{ex}}.UA$ is represented by dashed arrows and the role-permission relation $\gamma_{\mathsf{ex}}.PA$ by solid arrows.

3. A MOTIVATION EXAMPLE

Figure 1 shows an example RBAC state γ_{ex}. Suppose that an SSO needs to update γ_{ex} in response to the following requests.

Repairing. Suppose that a misconfiguration that the user u_2 should have been able to perform p_2 is detected. The SSO is informed of this problem and asked to take measures to grant u_2 the permission p_2. The question is how the SSO may accomplish this, while keeping other users' role sets and permission sets unchanged? It seems overreacting to assign p_2 with r_1 or with r_3, because that would incidentally grant p_2 to u_1 or u_3.

Permission assignments. Suppose that a task `t` requires the permissions $P_{\mathtt{t}} = \{p_5, p_8, p_9\}$. Currently in γ_{ex}, there does not exist a set of roles whose permission set is exactly $P_{\mathtt{t}}$. If `t` is important and all permissions in $P_{\mathtt{t}}$ are indispensable, an update request, that $P_{\mathtt{t}}$ be assigned via roles but with users' role sets and permission sets unchanged, may be issued in hope of enabling task `t`.

This type of requests could also happen in the context of *inter-domain role mappings*: a set of permissions (e.g., $\{p_5, p_8, p_9\}$) are to be shared via roles, but the underlying state could not enable the exact set of permissions [4, 9, 29, 30, 35, 38].

In both cases, the SSO needs to either find a satisfactory update or determine the request is not satisfiable at all. However, even with this toy example, it is not straightforward to make decisions.

4. PROBLEM DEFINITION

4.1 Constraints on updates

Given the current state γ, consider an update request that a set $\mathcal{P}$ of permissions be assigned to users via roles. If new roles can be defined for $\mathcal{P}$, this question becomes trivial. Hence, we look at more difficult role updating problem in RBAC systems where no new roles can be created. We believe this is more common; because new roles should be introduced with discretion, but not merely in response to such permission assignment requests. For any state χ obtained after updating γ, we may want to impose this constraint:

$$\chi.R = \gamma.R. \tag{1}$$

The effects of an update should be confined to a certain set U of users. In other words, updates are supposed to be transparent to users outside U in the sense that their permission sets in χ remain as in γ. In this way, the tasks that are being performed in the system would not be interfered by updating. Put formally, the constraint is:[2]

$$\mathsf{perms}_\chi[u] = \mathsf{perms}_\gamma[u], \quad \text{for all } u \in \gamma.U \backslash U. \tag{2}$$

Additionally, to make updates rational, the following constraint seems necessary.

$$\mathsf{roles}_\chi[u] = \mathsf{roles}_\gamma[u] \quad \text{for all } u \in \gamma.U \backslash U. \tag{3}$$

Constraint (3) keeps users' role sets unchanged. This is motivated by the following observations: (1) User-role assignments are business-driven; users' role memberships are determined by their attributes, jobs, titles, and etc. (2) From the user-experience viewpoint, a staff in a university may be reluctant to take a student role even though she is entitled to the same privileges. Further, a user often activates only a subset of her roles for a task; it would be obtrusive for a professor to activate both the student role and the secretary role to finish what she can do with a professor role before, had an update adjusted her role assignments.

Motivated by these observations, we define a notion of *Update Constraint Schema*, which generalizes constraints (1), (2) and (3).

Definition 1. (Update Constraint Schema - UCS) Given a state γ, a UCS of γ is a tuple $\pi^\gamma = \langle \pi^\gamma.U, \pi^\gamma.th \rangle$, where

1. $\pi^\gamma.U \subseteq \gamma.U$, and
2. $\pi^\gamma.th : \pi^\gamma.U \mapsto 2^{\gamma.P}$ such that, for any $u \in \pi.U$, $\pi^\gamma.th[u] \subseteq \mathsf{perms}_\gamma[u]$ (th is short for *threshold*).

When γ is clear from context, we omit the supscript γ.

When SSOs are specifying π, $\pi.U$ often contains those users for whom the SSOs are not responsible so that the SSOs have to ensure that the potential update does not affect those users (by setting $\pi.th[u] = \mathsf{perms}_\gamma[u]$), and/or those users whose permissions are designated by the SSOs and vary within a range (i.e., the lower bound is $\pi.th[u]$ and the upper bound is $\mathsf{perms}_\gamma[u]$). The set $\gamma.U \backslash \pi.U$ is a set of users whose current role sets and permission sets in γ are neglected by updates; updates may change their role-assignments and permission-assignments.

Definition 2. (π-compatibility) Given γ, π, and an RBAC state $\chi \in \mathsf{space}(\gamma)$, we say χ is π-*compatible* if the following conditions are met.

COND-U-R for any $u \in \pi.U$, $\mathsf{roles}_\chi[u] = \mathsf{roles}_\gamma[u]$, and

COND-U-P for any $u \in \pi.U$, $\pi.th[u] \subseteq \mathsf{perms}_\chi[u] \subseteq \mathsf{perms}_\gamma[u]$.

Note that only RBAC states in $\mathsf{space}(\gamma)$ are considered. That means, we assume that no changes are made to the user set, role set, and permission set. Condition COND-U-R requires that the role assignments of any $u \in \pi.U$ in χ remain as in γ, i.e., $\chi.UA \cap (\pi.U \times \chi.R) = \gamma.UA \cap (\pi.U \times \gamma.R)$. This is exactly the same as (3). COND-U-R also indicates that we focus on the update of role-permission assignments. In accordance with COND-U-P, any $u \in \pi.U$ must satisfy: u's permission set varies from $\pi.th[u]$ to $\mathsf{perms}_\gamma[u]$, both inclusively. This is a generalization of (2). At least, by letting $\pi.th[u] = \mathsf{perms}_\gamma[u]$ so that users permission sets remain the same, SSOs guarantee that tasks assigned to users in $\pi.U$ progress smoothly after update.

Definition 3. An *update request* to γ, $\mathsf{req}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$, comprises the current RBAC state γ, a set of permissions $\mathcal{P} \subseteq \gamma.P$, a UCS π of γ, and a set of roles $\mathcal{T} \subseteq \gamma.R$.

Definition 4. Given $Q = \mathsf{req}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$, we say χ is an update of Q, if χ is π-compatible and there exists a set $R \subseteq \mathcal{T}$ such that $\mathsf{perms}_\chi[R] = \mathcal{P}$. Denote the set of updates of Q as $\mathsf{upd}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$. We say Q is *satisfiable*, if $\mathsf{upd}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle \neq \emptyset$.

Definition 5. (Role Updating Problem - RUP) The RUP is to, given $\mathsf{req}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$, check whether $\mathsf{upd}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle \neq \emptyset$ and, if so, to find one $\chi \in \mathsf{upd}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$.

SSOs express update requests by specifying $\mathsf{req}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$. If it turns out that $\mathsf{upd}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle = \emptyset$, that means the update request cannot be fulfilled without violating π-compatibility. Otherwise, supposing $\chi \in \mathsf{upd}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$ is found, SSOs can make changes to γ in light of χ.

Route might be used when the RBAC state is migrating to an ideal state suggested by role mining techniques; the ideal state may be fine-tuned using Route to adjust assignments. More importantly, Route is able to assist SSOs with administrative works, especially when assigning (or adjusting) permissions to users.

Satisfiability is defined based on π. One most important restriction that π puts is that, only a limited set of permissions may be assigned with a role. Given $\mathsf{req}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$, $r \in \gamma.R$ and $R \subseteq \gamma.R$, let

$$\mathsf{max\text{-}perms}_{\langle \gamma, \pi \rangle}[r] = \bigcap_{u \in (\mathsf{users}_\gamma[r] \cap \pi.U)} \mathsf{perms}_\gamma[u],$$

and $\mathsf{max\text{-}perms}_{\langle \gamma, \pi \rangle}[R] = \bigcup_{r \in R} \mathsf{max\text{-}perms}_{\langle \gamma, \pi \rangle}[r]$. If $\langle \gamma, \pi \rangle$ is clear from context, we omit the subscript $\langle \gamma, \pi \rangle$. $\mathsf{max\text{-}perms}[r]$ (resp., $\mathsf{max\text{-}perms}[R]$) is the maximal set of permissions that r (resp., R) could possibly be assigned with in χ.

PROPOSITION 1. *Given $\mathsf{req}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$, if χ is π-compatible, then, for all $(r, p) \in \chi.PA$, $p \in \mathsf{max\text{-}perms}[r]$.*

Supposing $\chi \in \mathsf{upd}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$, π-compatibility is to restrict the difference between γ and χ, which is measured as below.

Definition 6. Given γ and χ such that $\chi \in \mathsf{space}(\gamma)$, the *difference* between γ and χ is defined as $\mathtt{diff}(\gamma, \chi) = (\gamma.PA \backslash \chi.PA) \cup (\chi.PA \backslash \gamma.PA)$. Note that $\mathtt{diff}(\chi, \gamma) = \mathtt{diff}(\gamma, \chi)$.

4.2 RUP is intractable

THEOREM 2. *Given an update request $\mathsf{req}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$, it is NP-complete to decide its satisfiability.*

[2] Given two sets A and B, $A \backslash B = \{a \in A \mid a \notin B\}$.

PROOF. Since elements in $Q = \mathsf{req}\langle\gamma, \mathcal{P}, \pi, \mathcal{T}\rangle$ are finitely fixed and, for any $\chi \in \mathsf{space}(\gamma)$, it takes polynomial time to check if $\chi \in \mathsf{upd}\langle\gamma, \mathcal{P}, \pi, \mathcal{T}\rangle$, a non-deterministic Turing machine can guess an update χ and verify if $\chi \in \mathsf{upd}\langle\gamma, \mathcal{P}, \pi, \mathcal{T}\rangle$. Hence, the problem is in NP. To show its NP-hardness, we reduce the known NP-complete problem Monotone SAT to our problem.

Monotone SAT: Given a set X of boolean variables and a collection C of clauses over X where each clause contains either only positive literals or only negative literals, is there a truth assignment of X so that $\bigwedge C$ is true? We call a clause with only positive literals a *positive clause*, denoted as c^+ and otherwise a *negative clause*, denoted as c^-.

Given a monotone SAT instance, construct a $\mathsf{req}\langle\gamma, \mathcal{P}, \pi, \mathcal{T}\rangle$ as follows. First we construct γ. For each clause $c^+ \in C$, create a permission p_{c^+}; for each $c^- \in C$, create a permission p_{c^-}. Denote $P^+ = \{p_{c^+} \mid c^+ \in C\}$ and $P^- = \{p_{c^-} \mid c^- \in C\}$. Let $\gamma.P = P^+ \cup P^-$. For each $x \in X$, create a corresponding role r_x. Let $(r_x, p_{c^+}) \in \gamma.PA$ if and only if c^+ contains the literal x and $(r_x, p_{c^-}) \in \gamma.PA$ if and only if c^- contains the literal $\neg x$. For each c^-, create a user u_{c^-} and let $(u_{c^-}, r_x) \in \gamma.UA$ if and only if c^- contains $\neg x$. For each r_x create a user u_x and let $(u_x, r_x) \in \gamma.UA$. Let $\mathcal{P} = P^+$. Now we configure π by letting

- $\pi.U = \gamma.U$, $\mathcal{T} = \gamma.R$,
- for each u_{c^-}, $\pi.th[u_{c^-}] = \mathsf{perms}_\gamma[u_{c^-}]$, and
- for each u_x, $\pi.th[u_x] = \mathsf{perms}_\gamma[r_x] \cap \mathcal{P}$.[3]

We now show that monotone SAT is satisfiable if and only if $\mathsf{upd}\langle\gamma, \mathcal{P}, \pi, \mathcal{T}\rangle \neq \emptyset$.

Suppose that τ is a truth assignment that makes $\bigwedge C$ true. Then an update of $\mathsf{req}\langle\gamma, \mathcal{P}, \pi, \mathcal{T}\rangle$ consists of: removing all (r_x, p) where $\tau(x) = 1$ and $p \in P^-$. Note that $\mathcal{P} = P^+$. Since $\bigwedge C$ is true, all c^+ is true. As a result, for any $p \in \mathcal{P}$, there exists at least one x such that $\tau(x) = 1$ and $(r_x, p) \in \gamma.PA$. Then $\{r_x \mid \tau(x) = 1\}$ in the updating state is a role set whose permission set is exactly $\mathcal{P}$. Since the monotone SAT formula is satisfied by τ, each positive clause c^+ is true under τ; for each $p \in \mathcal{P}$, there must exist at least one r_x such that $(r_x, p) \in \gamma.PA$ and $\tau(x) = 1$; hence after removing all permissions of r_x which are also outside $\mathcal{P}$, p can be assigned to users via r_x in the update. π-compatibility is established as follows. For any user u, if any permission $p \in P^+$ belongs to $\mathsf{perms}_\gamma[u]$, since the update does not involve permissions in P^+, u still acquires that permission in the update. Similarly, conditions regarding any user u_x are also satisfied. For any user u_{c^-}, consider any $p_{c^-} \in \mathsf{perms}_\gamma[u_{c^-}]$. Since c^- is true under τ, there exists at least one x in c^- with $\tau(x) = 0$. From the construction of the update, the update does not affect r_x; therefore, it holds that $p_{c^-} \in \mathsf{perms}_\chi[r_x]$, namely, u_{c^-} is still associated with p_{c^-} via r_x.

On the other hand, suppose that $\chi \in \mathsf{upd}\langle\gamma, \mathcal{P}, \pi, \mathcal{T}\rangle$. Construct a truth assignment τ over X: $\tau(x) = 1$ if and only if $\mathsf{perms}_\chi[r_x] \subseteq \mathcal{P}$, and otherwise, $\tau(x) = 0$. Then τ can make the formula true for the following reasons. Recall that $\mathcal{P} = P^+$. Suppose that $c^+ = x_1 \vee \cdots \vee x_n$ is false under τ; that is, $\tau(x_\ell) = 0$ for $1 \leq \ell \leq n$. In light of the definition of τ, it holds that $\mathsf{perms}_\chi[r_{x_\ell}] \not\subseteq \mathcal{P}$, for $1 \leq \ell \leq n$. Hence, from the fact that $\chi \in \mathsf{upd}\langle\gamma, \mathcal{P}, \pi, \mathcal{T}\rangle$, there exists $r \in \chi.R$ such that $p_{c^+} \in \mathsf{perms}_\chi[r]$, $\mathsf{perms}_\chi[r] \subseteq \mathcal{P}$, and $r \notin \{r_{x_1}, \cdots, r_{x_n}\}$. However, since χ is π-compatible, particularly the effects of users u_x, it must hold that $\mathtt{diff}(\chi, \gamma) \subseteq \gamma.R \times P^-$. But $p_{c^+} \notin \mathsf{perms}_\gamma[r]$ because $r \notin \{r_{x_1}, \cdots, r_{x_n}\}$; as a result, $(r, p_{c^+}) \in \mathtt{diff}(\chi, \gamma)$. Thus, we reach a contradiction. Suppose that $c^- = \neg x_1 \vee \cdots \vee \neg x_n$ is false under τ; that is, $\tau(x_\ell) = 1$ for $1 \leq \ell \leq n$. Then according to the definition of τ, $\mathsf{perms}_\chi[r_{x_\ell}] \subseteq \mathcal{P}$ for $1 \leq \ell \leq n$. Since u_{c^-} is only assigned with roles $\{r_{x_1}, \cdots, r_{x_n}\}$ and $p_{c^-} \notin \mathcal{P}$, it holds that $p_{c^-} \notin \mathsf{perms}_\chi[u_{c^-}]$. However, $p_{c^-} \in \mathsf{perms}_\gamma[u_{c^-}]$, a contradiction with π-compatibility (in terms of COND-U-P for u_{c^-}). □

4.3 A model checking approach

Route leverages model checking techniques [6] for RUP. The basic idea is as follows. Given $\mathsf{req}\langle\gamma, \mathcal{P}, \pi, \mathcal{T}\rangle$, let ϕ denote the statement that a (witness) user could acquire $\mathcal{P}$ via roles in $\mathcal{T}$; we ask if $\neg\phi$ is always true in all π-compatible reachable states from γ. If a positive answer is returned, that means one cannot fulfill $\mathsf{req}\langle\gamma, \mathcal{P}, \pi, \mathcal{T}\rangle$ without violating the constraints on updates specified by π. Otherwise, model checkers would generate a counter-example, from which Route constructs an update.

4.3.1 Formalization

The model checking techniques that Route uses works with *computational tree logic (CTL)* and *Kripke structures* [6]. We first introduce these notions and the *Model Checking Problem (MCP)* considered in this paper; and then translate RUP to MCP.

Let AP be a set of atomic propositions. A *Kripke structure* M is a tuple (S, σ, L) where

1. S is a finite set of states,[4]
2. σ is a binary relation on S (i.e., $\sigma \subseteq S \times S$) which defines the transitions between states, and
3. $L : S \mapsto 2^{AP}$ associates each state $s \in S$ with a set of propositions in AP.

A path in M starting from s is denoted as $H = [s_0, s_1, \cdots, s_i, s_{i+1}, \cdots]$, where $s_0 = s$ and $(s_i, s_{i+1}) \in \sigma$ holds for all $i \geq 0$.

The syntax of CTL is given as: Every atomic proposition $ap \in AP$ is a CTL formula; and if φ_1 and φ_2 are CTL formulas, then so are $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, and $AG\varphi_1$.[5] The formula $AG\varphi$ means that on every computation path φ holds at every state. The semantics of CTL formulas is usually defined with respect to a Kripke structure. Let $M = (S, \sigma, L)$ be a Kripke structure for CTL. Given any $s \in S$, denote a CTL formula φ holds in M at state s as $(M, s) \models \varphi$. The relation $\models$ is defined by structural induction on CTL formulas:

- $(M, s) \models ap$ iff $ap \in L(s)$.
- $(M, s) \models \neg\varphi$ iff $(M, s) \not\models \varphi$.
- $(M, s) \models \varphi_1 \wedge \varphi_2$ iff $(M, s) \models \varphi_1$ and $(M, s) \models \varphi_2$.
- $(M, s) \models \varphi_1 \vee \varphi_2$ iff $(M, s) \models \varphi_1$ or $(M, s) \models \varphi_2$.
- $(M, s) \models AG\varphi$ iff for all paths $H = [s_0, s_1, s_2, \cdots,]$ where $s_0 = s$ and for all $i \geq 0$ $(M, s_i) \models \varphi$.

An MCP can be abstracted as follows [5]. Given (M, ϕ, I), where M is a Kripke structure, ϕ is a CTL formula, and $I \subseteq S$

[3]This means that, for each role, only permissions outside $\mathcal{P}$ could be removed.

[4]Note that these are not RBAC states.

[5]This definition is not complete; but we only use the formula $AG\varphi_1$.

is a set of initial states, the problem is to determine whether $I \subseteq \{s \in S \mid (M, s) \models \varphi\}$.

We now connect RUP with MCP. Given $Q = \mathsf{req}\langle\gamma, \mathcal{P}, \pi, \mathcal{T}\rangle$, we define a set of propositions $\mathsf{prop}(Q) = Pr_1 \cup Pr_2$, where

- $Pr_1 = \{x\text{-}u\text{-}r \mid u \in \gamma.U \wedge r \in \gamma.R\} \cup \{x\text{-}r\text{-}p \mid r \in \gamma.R \wedge p \in \gamma.P\}$, and
- $Pr_2 = \{x\text{-}\mathsf{wu}\text{-}r \mid r \in \mathcal{T}\}$.

The proposition x-r-p is meant to represent the assignment of p to r: x-r-$p = 1$ means r is assigned with p, and otherwise not. The propositions in Pr_1 is used to model the user-role and role-permission assignments in RBAC states. The propositions in Pr_2 denote whether or not a witness user wu is assigned with the roles in $\mathcal{T}$.

Given Q, we construct a Kripke structure $M_Q = (S, \sigma, L)$ as follows.

- $S = \{s_A \mid A \subseteq \mathsf{prop}(Q)\}$,
- L is defined as $L(s_A) = \{ap \in \mathsf{prop}(Q) \mid ap \in A\}$,
- To define σ, we define a mapping $g : S \mapsto \mathsf{space}(\gamma)$: $g(s_A) = \gamma_A = \langle \gamma_A.U, \gamma_A.R, \gamma_A.P, \gamma_A.UA, \gamma_A.PA\rangle$ if and only if
  - $\gamma_A.U = \gamma.U$, $\gamma_A.R = \gamma.R$, $\gamma_A.P = \gamma.P$,
  - $(u, r) \in \gamma_A.UA$ if and only if x-u-$r \in A$, and
  - $(r, p) \in \gamma_A.PA$ if and only if x-r-$p \in A$.

  Then for any $s_A, s_B \in S$, $(s_A, s_B) \in \sigma$ if and only if both $g(s_A)$ and $g(s_B)$ are π-compatible.

To define the initial states of Kripke structure, for any RBAC state γ', let $\mathsf{Kstates}(\gamma') = \{s_A \in S \mid g(s_A) = \gamma'\}$. We let $I_Q = \mathsf{Kstates}(\gamma)$ so that the initial states correspond to the requested RBAC state γ in Q.

Finally, we define ϕ of MCP (M_Q, ϕ, I_Q), i.e., the property that we want to check. Assume that $\mathcal{T} = \{r_1, \cdots, r_t\}$, $\mathcal{P} = \{p_1, \cdots, p_m\}$ and $\gamma.P \backslash \mathcal{P} = \{p_{m+1}, \cdots, p_n\}$. We let ϕ be $AG\neg(\phi_1 \wedge \phi_2)$, where

- $\phi_1 = X\text{-}\mathsf{wu}\text{-}p_1 \wedge \cdots \wedge X\text{-}\mathsf{wu}\text{-}p_m$, and
- $\phi_2 = \neg\left(X\text{-}\mathsf{wu}\text{-}p_{m+1} \vee \cdots \vee X\text{-}\mathsf{wu}\text{-}p_n\right)$.

In turn, for $1 \le \ell \le n$, X-wu-p_ℓ is defined as

$$(x\text{-}\mathsf{wu}\text{-}r_1 \wedge x\text{-}r_1\text{-}p_\ell) \vee (x\text{-}\mathsf{wu}\text{-}r_2 \wedge x\text{-}r_2\text{-}p_\ell) \vee \cdots \vee (x\text{-}\mathsf{wu}\text{-}r_t \wedge x\text{-}r_t\text{-}p_\ell).$$

Intuitively, each X-wu-p_ℓ is testing if the witness user wu has the permission p_ℓ. Then ϕ_1 models if (a) wu can have all permissions in $\mathcal{P}$, whereas ϕ_2 is used to test if (b) wu has no permission in $\gamma.P \backslash \mathcal{P}$. As a result, $AG\neg(\phi_1 \wedge \phi_2)$ is asking if there is no reachable state from the initial states that satisfies both (a) and (b). Formally, we have the following result.

THEOREM 3. *$\mathsf{upd}\langle\gamma, \mathcal{P}, \pi, \mathcal{T}\rangle = \emptyset$ if and only if $I_Q \subseteq \{s \in S \mid (M_Q, s) \models AG\neg(\phi_1 \wedge \phi_2)\}$.*

If $AG\neg(\phi_1 \wedge \phi_2)$ is checked to be true, then there is no reachable state satisfying the requirements (a) and (b); otherwise, NuSMV would generate a counterexample. The counterexample here is a state s_A; then the RBAC state $g(s_A)$ is a desired update for Q.

4.3.2 Implementation in NuSMV

Particularly, Route embeds the model checker NuSMV [5] to search for updates. NuSMV is a modern symbolic model checker, supporting various useful features such as the TRANS constraints. A TRANS constraint defines which next states that the current state may transit into. Namely, the model of the constraint is a set of current/next state pairs. Multiple TRANS constraints are treated as the conjunction of all TRANS constraints. We refer readers to [5] for details of NuSMV.

Encoding states. Recall the condition COND-U-R of π-compatibility, that users' role assignments remain during updates; hence, we do not need to encode the user-role assignments. A set of boolean variables are defined to describe the role-permission assignments. According to Proposition 1, only a vector of variables for $\{r\} \times \mathsf{max\text{-}perms}[r]$ are defined for each $r \in \gamma.R$.

Encoding π. The UCS π mainly defines the set of resulting states that γ may evolve into. Recall that, for a π-compatible RBAC state χ, the condition COND-U-P requires that $\pi.th[u] \subseteq \mathsf{perms}_\chi[u] \subseteq \mathsf{perms}_\gamma[u]$. Since $\mathsf{perms}_\chi[u] = \mathsf{perms}_\chi[\mathsf{roles}_\chi[u]]$ and $\mathsf{roles}_\chi[u] = \mathsf{roles}_\gamma[u]$, this requirement actually puts restrictions on the variables for $\{r\} \times \mathsf{max\text{-}perms}[r]$, for each $r \in \mathsf{roles}_\gamma[u]$.

Route translates this requirement into TRANS constraints in NuSMV. For each $\pi.th[u] \subseteq \mathsf{perms}_\chi[\mathsf{roles}_\gamma[u]] \subseteq \mathsf{perms}_\gamma[u]$, Route constructs two TRANS constraints. The $\pi.th[u] \subseteq \mathsf{perms}_\chi[\mathsf{roles}_\gamma[u]]$ part requires that, for each $p \in \pi.th[u]$, there exists at least one $r \in \mathsf{roles}_\gamma[u]$ such that $(r, p) \in \chi.PA$. Thus, supposing that $\pi.th[u] = \{p_1, \cdots, p_n\}$ and $\mathsf{roles}_\gamma[u] = \{r_1, \cdots, r_m\}$, the following NuSMV constraint is needed.

$$\mathsf{TRANS}\ \mathsf{next}\big((x\text{-}r_1\text{-}p_1 | \cdots | x\text{-}r_m\text{-}p_1) \,\&\, \cdots \,\&\, (x\text{-}r_m\text{-}p_n | \cdots | x\text{-}r_m\text{-}p_n)\big);$$

For $\mathsf{perms}_\chi[\mathsf{roles}_\gamma[u]] \subseteq \mathsf{perms}_\gamma[u]$ part, first observe that $\mathsf{perms}_\chi[\mathsf{roles}_\gamma[u]] \subseteq \mathsf{perms}_\gamma[u]$ if and only if $\mathsf{perms}_\chi[\mathsf{roles}_\gamma[u]] \cap (\mathsf{max\text{-}perms}[\mathsf{roles}_\gamma[u]] \backslash \mathsf{perms}_\gamma[u]) = \emptyset$. Hence, assuming $\mathsf{max\text{-}perms}[\mathsf{roles}_\gamma[u]] \backslash \mathsf{perms}_\gamma[u] = \{q_1, \cdots, q_t\}$, it is required that each x-r_i-q_j be 0 constantly if $q_j \in \mathsf{max\text{-}perms}[r_i]$, for $1 \le i \le m$ and $1 \le j \le t$.

The encoding of $AG\neg(\phi_1 \wedge \phi_2)$ is straightforward. Appendix A provides a counterexample for an example update.

4.4 Example usage of role updating

This section presents several example configurations of $\mathsf{req}\langle\gamma, \mathcal{P}, \pi, \mathcal{T}\rangle$ that SSOs can specify to achieve different update objectives, with respect to γ_{ex} in Figure 1.

Adjusting role sets and permission sets. To repair γ_{ex} so that u_2 can have p_2, the SSO could issue a request $Q_1 = \mathsf{req}\langle\gamma_{\mathsf{ex}}, \mathcal{P}, \pi, \mathcal{T}\rangle$:

- $\mathcal{P} = \mathsf{perms}_{\gamma_{\mathsf{ex}}}[u_2] \cup \{p_2\} = \{p_1, p_2, p_3, p_4, p_5\}$,
- $\pi.U = \gamma_{\mathsf{ex}}.U \backslash \{u_2\} = \{u_1, u_3, u_4\}$ and, for any $u \in \pi.U$, $\pi.th[u] = \mathsf{perms}_{\gamma_{\mathsf{ex}}}[u]$, and
- $\mathcal{T} = \gamma_{\mathsf{ex}}.R = \{r_1, r_2, r_3, r_4, r_5, r_6\}$.

Running with Q_1, Route would suggest a sequence of actions: $s_1 = \langle\ \mathsf{revoke}(p_6, r_2);\ \mathsf{assign}(r_1, \mathsf{wu});\ \mathsf{assign}(r_2, \mathsf{wu});\ \mathsf{assign}(r_3, \mathsf{wu});\ \rangle$, where wu is the witness user not belonging to γ. Then the SSO can follow s_1 to make changes to γ_{ex}: revoke p_6 from r_2 and associate $\{r_1, r_2, r_3\}$ with u_2.

For another example, suppose that the SSO wants to shrink u_3's permission set to $\{p_1, p_5, p_7\}$ and revoke r_3 from u_3. Then $Q_2 = \mathsf{req}\langle\gamma_{\mathsf{ex}}, \mathcal{P}, \pi, \mathcal{T}\rangle$ models this request:

- $\mathcal{P} = \{p_1, p_5, p_7\}$,
- $\pi.U = \gamma_{\mathsf{ex}}.U \backslash \{u_3\} = \{u_1, u_2, u_4\}$ and, for any $u \in \pi.U$, $\pi.th[u] = \mathsf{perms}_{\gamma_{\mathsf{ex}}}[u]$,
- $\mathcal{T} = \mathsf{roles}_{\gamma}[u_3] \backslash \{r_3\} = \{r_4, r_5\}$,

Route would return a sequence of actions: $s_2 = \langle\ \mathsf{assign}(p_5, r_5)$; $\mathsf{revoke}(p_6, r_4)$; $\mathsf{revoke}(p_8, r_4)$; $\mathsf{assign}(p_8, r_6)$; $\mathsf{assign}(r_4, \mathsf{wu})$; $\mathsf{assign}(r_5, \mathsf{wu})\ \rangle$.

If the SSO requests to remove r_3 from u_2 (i.e., assign only r_1 with u_2) but retain u_2's current permission set: $\{p_1, p_3, p_4, p_5\}$, Route would report that the request is not achievable.

Permission assignments. To update for enabling the exact set of permissions $P_{\mathsf{t}} = \{p_5, p_8, p_9\}$, the SSO could set $Q_3 = \mathsf{req}\langle \gamma_{\mathsf{ex}}, \mathcal{P}, \pi, \mathcal{T} \rangle$ as follows:

- $\mathcal{P} = P_{\mathsf{t}}$,
- $\pi.U = \gamma_{\mathsf{ex}}.U = \{u_1, u_2, u_3, u_4\}$ and, for any $u \in \pi.U$, $\pi.th[u] = \mathsf{perms}_{\gamma_{\mathsf{ex}}}[u]$, and
- $\mathcal{T} = \gamma_{\mathsf{ex}}.R = \{r_1, r_2, r_3, r_4, r_5, r_6\}$.

Given Q_3, Route returns the sequence of administrative actions: $s_3 = \langle\ \mathsf{assign}(p_5, r_6)$; $\mathsf{assign}(p_8, r_6)$; $\mathsf{revoke}(p_6, r_6)$; $\mathsf{assign}(r_6, \mathsf{wu})$; $\rangle$.

5. REDUCTIONS

Section 4.3 describes the idea of applying model checking techniques to RUP. However, the execution of the NuSMV program directly transformed from $\mathsf{req}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$ quickly leads to state exploration (and memory crash). In this section, we present a set of reductions for RUP. These reductions are not exclusive to Route but could also benefit other approaches to RUP.

Consider two requests $Q_1 = \mathsf{req}\langle \gamma_1, \mathcal{P}, \pi_1, \mathcal{T} \rangle$ and $Q_2 = \mathsf{req}\langle \gamma_2, \mathcal{P}, \pi_2, \mathcal{T} \rangle$, which share the same $\mathcal{P}$ and $\mathcal{T}$. If it holds that $\mathsf{upd}\langle \gamma_1, \mathcal{P}, \pi_1, \mathcal{T} \rangle \neq \emptyset$ if and only if $\mathsf{upd}\langle \gamma_2, \mathcal{P}, \pi_2, \mathcal{T} \rangle \neq \emptyset$, then Q_1 is satisfiable if and only if so is Q_2. On the other hand, if $\mathsf{upd}\langle \gamma_1, \mathcal{P}, \pi_1, \mathcal{T} \rangle \subseteq \mathsf{upd}\langle \gamma_2, \mathcal{P}, \pi_2, \mathcal{T} \rangle$, then when we find an update χ for Q_1, we also obtain an update for Q_2. Put together, we have the following definition.

Definition 7. Given a request $Q = \mathsf{req}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$ and a set of requests $Q_{set} = \{\mathsf{req}\langle \gamma_1, \mathcal{P}, \pi_1, \mathcal{T} \rangle, \cdots, \mathsf{req}\langle \gamma_m, \mathcal{P}, \pi_m, \mathcal{T} \rangle\}$, we say $Q \hookrightarrow Q_{set}$ if the following two conditions are satisfied.

1. $\bigcup_{1 \leq \ell \leq m} \mathsf{upd}\langle \gamma_\ell, \mathcal{P}, \pi_\ell, \mathcal{T} \rangle \neq \emptyset$ iff $\mathsf{upd}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle \neq \emptyset$.
2. $\bigcup_{1 \leq \ell \leq m} \mathsf{upd}\langle \gamma_\ell, \mathcal{P}, \pi_\ell, \mathcal{T} \rangle \subseteq \mathsf{upd}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$.

It can be seen that if $Q \hookrightarrow Q_{set}$, then we can work with the set Q_{set} instead of Q; we need to find a Q_{set} that is easier to tackle.

5.1 Reduction on core

Observation 1. For π-compatibility, changes can only happen around roles that are related to the permissions in $\mathcal{P}$.

Given $Q = \mathsf{req}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$, we denote the set of roles $\mathsf{roles}_{\gamma}[\mathsf{users}_{\gamma}[\mathcal{P}]]$ as $\mathsf{core}(\gamma, \mathcal{P})$ and call it the *core set* for Q. The reductions center around $\mathsf{core}(\gamma, \mathcal{P})$. The intuition is that: according to π, only users in $\mathsf{users}_{\gamma}[\mathcal{P}]$ can have permissions in $\mathcal{P}$ and thus only their roles may be assigned with permissions in $\mathcal{P}$.

Definition 8. Given a state γ and $R \subset \gamma.R$, we say $\gamma_{\lceil R \rceil}$ *is a filtered state of* γ *by* R, where $\gamma_{\lceil R \rceil}.U = \mathsf{users}_{\gamma}[R]$, $\gamma_{\lceil R \rceil}.R = \mathsf{roles}_{\gamma}[\gamma_{\lceil R \rceil}.U]$, $\gamma_{\lceil R \rceil}.P = \mathsf{perms}_{\gamma}[\gamma_{\lceil R \rceil}.R]$, $\gamma_{\lceil R \rceil}.UA = \gamma.UA \cap (\gamma_{\lceil R \rceil}.U \times \gamma_{\lceil R \rceil}.R)$, and $\gamma_{\lceil R \rceil}.PA = \gamma.UA \cap (\gamma_{\lceil R \rceil}.R \times \gamma_{\lceil R \rceil}.P)$.

Given $\mathsf{req}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$, denote the state $\gamma_{\lceil \mathsf{core}(\gamma, \mathcal{P}) \rceil}$ as core-γ. While γ is filtered with respect to $\mathsf{core}(\gamma, \mathcal{P})$, a new UCS on updates of core-γ is to be constructed as well in a way that those updates to core-γ can be seen as updates to γ. Not surprisingly, this new UCS stems from π.

Definition 9. (Refinement of π) Given $\mathsf{req}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$ and a state γ' obtained by filtering γ with some role set, define a UCS $\pi_{\lceil \gamma' \rceil}$ by letting:

1. $\pi_{\lceil \gamma' \rceil}.U = \pi.U \cap \gamma'.U$, and
2. for any $u \in \pi_{\lceil \gamma' \rceil}.U$, $\pi_{\lceil \gamma' \rceil}.th[u] = \pi.th[u] \backslash \mathsf{perms}_{\gamma}[\mathsf{roles}_{\gamma}[u] \backslash \gamma'.R]$.

The UCS $\pi_{\lceil \gamma' \rceil}$ is a confinement of π to γ'. $\pi_{\lceil \gamma' \rceil}$ only allows changes to γ', which is a filtered state of γ. Hence, all updates allowed by $\pi_{\lceil \gamma' \rceil}$ would not influence users in $\pi.U \backslash \gamma'.U$; the first clause refines the user set. Since $\pi_{\lceil \gamma' \rceil}$ only allows changes to roles in $\gamma'.R$, the permission set of any role in $\gamma.R \backslash \gamma'.R$ would not be changed by any update of γ', and, therefore, nor is the permission set of any role in $\mathsf{roles}_{\gamma}[u] \backslash \gamma'.R$, for each $u \in \pi_{\lceil \gamma' \rceil}.U$.

Let $Q_{\mathsf{core}} = \mathsf{req}\langle \text{core-}\gamma, \mathcal{P}, \pi_{\lceil \text{core-}\gamma \rceil}, \mathcal{T} \rangle$. Proposition 4 corresponds to Observation 1.

PROPOSITION 4. $Q \hookrightarrow Q_{core}$.

5.2 Decomposition

Observation 2. It is sometimes useful to decompose $Q = \mathsf{req}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$ into sub-problems that can be solved separatively.

A collection $\mathcal{C}_{\mathsf{de}} \subset 2^{\gamma.R}$ is a *decomposition* of Q if

1. for any $S, S' \in \mathcal{C}_{\mathsf{de}}$, $\mathcal{P} \subset \mathsf{perms}_{\gamma}[S]$, and either $S = S'$ or $S \not\subset S'$;
2. for any $R \subseteq \gamma.R$, if $R \notin \mathcal{C}_{\mathsf{de}}$, then either $\mathcal{P} \not\subset \mathsf{perms}_{\gamma}[R]$ or there exists $S \in \mathcal{C}_{\mathsf{de}}$ such that $R \subseteq S$.

Given $S \in \mathcal{C}_{\mathsf{de}}$, let $\widehat{S} = \mathsf{roles}_{\gamma}[\mathsf{users}_{\gamma}[S]]$ and denote the state $\gamma_{\lceil \widehat{S} \rceil}$ as $\mathsf{de}_S(\gamma)$. One may decompose Q into a set of requests:

$$Q_{\mathsf{de}} = \{\mathsf{req}\langle \mathsf{de}_S(\gamma), \mathcal{P}, \pi_{\lceil \mathsf{de}_S(\gamma) \rceil}, \mathcal{T} \rangle \mid S \in \mathcal{C}_{\mathsf{de}}\}.$$

Proposition 5 formalizes Observation 2.

PROPOSITION 5. $Q \hookrightarrow Q_{de}$.

Proposition 5 does not reduce the complexity of RUP theoretically. It is likely that there exists $S \in \mathcal{C}_{\mathsf{de}}$ such that $\widehat{S} = \mathsf{core}(\gamma, \mathcal{P})$—in this case, we may still have to work with Q. However, the benefits lie in practice. For one thing, when $\mathsf{upd}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle \neq \emptyset$, we may figure out one $\chi \in \mathsf{upd}\langle \mathsf{de}_S(\gamma), \mathcal{P}, \pi_{\lceil \mathsf{de}_S(\gamma) \rceil}, \mathcal{T} \rangle$ with smaller S, which could be more efficient than working with Q; for another, the incurred changes could be more restricted, for $S \subseteq \mathsf{roles}_{\gamma}[\mathcal{P}]$.

Define $\mathcal{C}^{min}_{\mathsf{de}} = \big\{ S \subseteq \mathsf{roles}_{\gamma}[\mathcal{P}] \mid \mathsf{perms}_{\gamma}[S] \supset \mathcal{P} \wedge \forall S' \subset S : \mathsf{perms}_{\gamma}[S'] \not\supset \mathcal{P} \big\}$.

$\mathcal{C}_{\mathsf{de}}^{min}$ is a decomposition of Q. $\mathcal{C}_{\mathsf{de}}^{min}$ features that each $M \in \mathcal{C}_{\mathsf{de}}$ is a minimal role set whose permission set contains $\mathcal{P}$. Hence, evaluating $\mathsf{req}\langle \mathsf{de}_M(\gamma), \mathcal{P}, \pi_{\lceil \mathsf{de}_M(\gamma) \rceil}, \mathcal{T} \rangle$ for $M \in \mathcal{C}_{\mathsf{de}}^{min}$ might be easier. Unfortunately, computing $\mathcal{C}_{\mathsf{de}}^{min}$ is NP-hard.

In our prototype of Route, priority is given to the linear computing runtime of a decomposition $\mathcal{C}_{\mathsf{de}}^{approx}$, while keeping each member as minimal as possible. Practical systems may use specialized algorithms for $\mathcal{C}_{\mathsf{de}}^{min}$, which is beyond the scope of this paper.

5.3 Removing ignorable roles

Observation 3. For a role r such that $\mathsf{perms}_\gamma[r] \cap \mathcal{P} = \emptyset$, if its permission assignments could not be changed due to π, r would not help with $\mathsf{req}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$ and thus can be ignored by updates.

Given $r \in \gamma.R$, we say r is *ignorable* if $r \notin \mathsf{roles}_\gamma[\mathcal{P}]$ and $\mathsf{perms}_\gamma[r] = \mathsf{max\text{-}perms}[r]$. That is, r is ignorable if it cannot accept any permission assignments other than those that are already assigned with it in γ.

Denote the set of *non-ignorable* roles as R_{nig} and the state obtained by filtering γ with R_{nig} (i.e., $\gamma_{\lceil R_{nig} \rceil}$) as $\mathsf{nig}(\gamma)$. Let $Q = \mathsf{req}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$ and $Q_{\mathsf{nig}} = \mathsf{upd}\langle \mathsf{nig}(\gamma), \mathcal{P}, \pi_{\lceil \mathsf{nig}(\gamma) \rceil}, \mathcal{T} \rangle$. Proposition 6 corresponds to Observation 3.

PROPOSITION 6. $Q \hookrightarrow Q_{nig}$.

5.4 Propagating requested permissions

Observation 4. If complying with π, we may associate permissions in $\mathcal{P}$ with as many roles as possible. In addition, there exists an update that does not remove any assignment (r, p) such that $p \in \mathcal{P}$, should there exist an arbitrary update.

Given $\chi \in \mathsf{upd}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$, $p \in \mathcal{P}$, and $r \in \gamma.R$ such that $\mathsf{perms}_\chi[r] \not\subseteq \mathcal{P}$ and $(r, p) \notin \chi.PA$, if allowed by π, making $(r, p) \in \chi.PA$ would retain χ being an update. Namely, we can propagate permissions in $\mathcal{P}$ among roles and only change the role-permission assignments whose permissions reside outside $\mathcal{P}$.

Define a state $\mathsf{pt}_{\mathcal{P}}(\gamma)$ by letting

- $\mathsf{pt}_{\mathcal{P}}(\gamma).U = \gamma.U$, $\mathsf{pt}_{\mathcal{P}}(\gamma).R = \gamma.R$, $\mathsf{pt}_{\mathcal{P}}(\gamma).P = \gamma.P$, $\mathsf{pt}_{\mathcal{P}}(\gamma).UA = \gamma.UA$, and
- $\mathsf{pt}_{\mathcal{P}}(\gamma).PA = \gamma.PA \cup \{(r, p) \mid r \in \gamma.R \wedge p \in (\mathcal{P} \cap \mathsf{max\text{-}perms}[r])\}$.

PROPOSITION 7. $\mathit{req}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle \hookrightarrow \mathit{req}\langle \mathit{pt}_{\mathcal{P}}(\gamma), \mathcal{P}, \pi, \mathcal{T} \rangle$.

PROPOSITION 8. *If* $\chi_1 \in \mathit{upd}\langle \mathit{pt}_{\mathcal{P}}(\gamma), \mathcal{P}, \pi, \mathcal{T} \rangle$, *then there exists* $\chi_2 \in \mathit{upd}\langle \mathit{pt}_{\mathcal{P}}(\gamma), \mathcal{P}, \pi, \mathcal{T} \rangle$ *such that* $\mathtt{diff}(\mathit{pt}_{\mathcal{P}}(\gamma), \chi_2) \cap \{(r, p) \mid p \in \mathcal{P}\} = \emptyset$.

Proposition 8 means that we can just fix the assignment (r, p) constantly, if $p \in \mathcal{P}$. Together with Proposition 7, we would not miss an update if there exists one.

6. EXPERIMENTS

Figure 2 depicts the prototype of Route. The interface receives SSO's update requests and forwards them to the back-end. Algorithm 1 presents the pseudo code of the back-end. Function `TransNuSMV` translates the request into NuSMV programs; it explores some further simplifications of the problem. The basic intuition is to reduce the number of variables in NuSMV programs in accordance with π-compatibility. `TransNuSMV` also groups together those permissions that can be treated likewise (i.e., either revoke all of them from one role or assign all of them with one role) and retain only one of them.

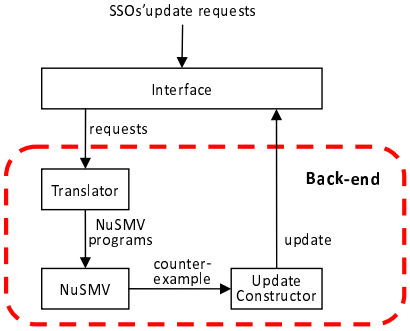


Figure 2: Route prototype.

Algorithm 1: Algorithm of the back-end.

Input: $\mathsf{req}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$
Output: either $\mathsf{upd}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle = \emptyset$ or $\chi \in \mathsf{upd}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$.

```
1  begin
2      Compute core-γ, π_⌈core-γ⌉, and the set C_de^approx;
3      foreach S ∈ C_de^approx do
4          Apply the reductions of "decomposition", "Removing
           ignorable roles" and "Propagating requested
           permissions" to core-γ and π_⌈core-γ⌉ in sequence;
           Denote the obtained state and UCS as γ' and π',
           respectively;
5          TransNuSMV(req⟨γ', P, π', T⟩);
6          Execute the resulting NuSMV program;
7          if a counterexample is returned then
8              Construct χ according to the counterexample;
9              return diff(χ, γ);
10     return upd⟨γ, P, π, T⟩ = ∅;
11 end
```

6.1 Experiments

Experiment Cases. The prototype of Route is implemented in JAVA. A request $Q_{exp} = \mathsf{req}\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$ is randomly generated. In all tests, we let $\mathcal{T} = \gamma.R$ and set π by letting $\pi.U = \gamma.U$ and, for all $u \in \pi.U$, $\pi.th[u] = \mathsf{perms}_\gamma[u]$. The reductions are performed in sequence as shown in Algorithm 1. For each $S \in \mathcal{C}_{\mathsf{de}}^{approx}$, we generate a file for the corresponding NuSMV program (denoted as $\mathsf{prog}(S)$) and a file for NuSMV commands. Then for each NuSMV program, we fork a thread, which executes the NuSMV program via NuSMV's batch mode. The thread is killed if the execution of the NuSMV program exceeds 12 hours. Unlike Algorithm 1, the tests would not finish until all NuSMV programs are generated, executed and returned, even though an update has been found. We record the time for processing each S denoted as $\mathsf{time}(S)$, including both the averaged shared preparation time and its own model checking time. Experiments were performed on a machine with an Intel(R) Core(TM)2 CPU T5500 @ 1.66GHz, and with 2GB of RAM running Microsoft Windows XP Home Edition Service Pack 3.

Synthetic Data Generation. To generate requests, we adapt data-generation algorithms from [34, 37], which is parameterized by the number of users (noU), the number of roles (noR), the number of permissions (noP), the maximum number of roles (noUR) each user may be assigned with, the maximum number of permissions (noRP) each role could possibly be assigned to, and the number of requested permissions (noReqps). The relation $\gamma.UA$ (resp. $\gamma.PA$) is generated by assigning each user (resp. each role) a num-

ber k of roles (resp. permissions) where k is randomly chosen and uniformly distributed between 1 and noUR (resp. noRP). Note that in all tests, no two users have the same role set and no two roles have the same permission set. $\mathcal{P}$ is determined by randomly chosen a number noReqps of permissions from $\gamma.P$.

Time Metric. Given the set of programs of $\mathsf{prog}(\mathcal{C}_{\mathsf{de}}^{approx})$, let prog_f be the set of programs that are checked false, prog_t the set of programs that are checked true, and prog_o the set of other programs that could not finish within time limits or that lead to memory crash. Fortunately, in all our test cases, we did get answers (i.e., either $\mathsf{prog}(\mathcal{C}_{\mathsf{de}}^{approx}) = \mathsf{prog}_t$ and all programs are finished in time or there exist some programs that return with a counterexample prior to time limit). Even though timeouts happen in several cases, there was another $\mathsf{prog}(S)$ whose NuSMV program is efficiently checked false and thus a counterexample is generated, which reveals an update.

The time records shown in Figure 3 are computed as follows.

$$\mathsf{time} = \max\{\mathsf{time}(S) \mid S \in \mathsf{prog}_f\} + \sum_{S \in \mathsf{prog}_t} \mathsf{time}(S). \quad (4)$$

For example, suppose $\mathcal{C}_{\mathsf{de}}^{approx} = \{S_1, S_2, S_3, S_4, S_5\}$ where $\mathsf{prog}(S_1)$ and $\mathsf{prog}(S_5)$ are checked true, checking $\mathsf{prog}(S_2)$ times out, and $\mathsf{prog}(S_3)$ and $\mathsf{prog}(S_4)$ are checked false and counterexamples are generated. Then the computing time is: $\mathsf{time}(S_1) + \mathsf{time}(S_5) + \max\{\mathsf{time}(S_3), \mathsf{time}(S_4)\}$. It is arguably reasonable to ignore the time of prog_o. Take $\mathsf{time}(S_2)$ for instance. In practice, one can easily compare the number of variables in $\mathsf{prog}(S_1)$, $\mathsf{prog}(S_2)$, $\mathsf{prog}(S_3)$, $\mathsf{prog}(S_4)$, and $\mathsf{prog}(S_5)$. Since NuSMV's performance highly depends on the number of variables, one can schedule programs' executions in increasing order by the number of variables and thus put $\mathsf{prog}(S_2)$ at the end of queue. When, for example, $\mathsf{prog}(S_3)$ returns a counterexample, then an update is found and there is no need to execute $\mathsf{prog}(S_2)$.[6] In this case, decomposition appears useful. Hence, we suspect that time in (4) is possibly longest time taken by Route to evaluate $\mathsf{req}\langle\gamma, \mathcal{P}, \pi, \mathcal{T}\rangle$. Since the data set is randomly created, for each configuration of parameters, we ran the test 5 times. The time in Figure 3 is averaged over the 5 runs.

Results. Figure 3 reports the experiment results. The time is in minute and log-scale in Figure 3(e), but is in second and linear in others. Note that for each run, a new instance Q_{exp} was generated each time a configuration was tested. In Figure 3(a), we generated Q_{exp} by fixing "noR=500 noP=2000 noUR=3 noRP=150 noReqps=500" but varying noU. Two longest intervals taken by Route, about 25 minutes, are at "noU=500" and "noU=1500". The main reason for this abnormality is that, in both cases, it happened that $\mathsf{prog}(\mathcal{C}_{\mathsf{de}}^{approx}) = \mathsf{prog}_t$ (i.e., no update exists) in all 5 runs and $|\mathsf{prog}(\mathcal{C}_{\mathsf{de}}^{approx})|$ is quite large, with averagely about 25 and 22 each run, respectively. In other cases, either $\mathsf{prog}_f \neq \emptyset$ (i.e., an update is found) or $|\mathsf{prog}(\mathcal{C}_{\mathsf{de}}^{approx})|$ is small. Generally, RUP is scalable with the number of users. With "noU=1500-2000", there is a notable drop. The observation is that, with larger number of users, there are more constraints on the role-permission assignments and more ignorable roles; thus more NuSMV variables were made constants. This also shows the effectiveness of the reduction of removing ignorable roles.

In Figure 3(b), we generated Q_{exp} by fixing "noU=1500 noP=2000 noUR=3 noRP=150 noReqps=500" but varying noR. As a whole, the time taken was almost polynomial to noR. The reduction of decomposition is useful as the number of roles increases. By decomposition, Route only dealt with requests with a limited number of roles.

In Figure 3(c), we generated Q_{exp} by fixing "noU=1500 noR=500 noP=2000 noUR=3 noRP=150" but varying noReqps. The peak (about 17 minutes) was reached at "noReqps=500-600". The use of "propagating requested permission" reduction saved Route from setting more and more variables, with the increase of noReqps. This explains why the time starts to drop at 700.

Figure 3(d) shows the case where "noU=1500 noR=500 noP=2000 noUR=3 noReqps=500" were fixed and Figure 3(e) the case where "noU=1500 noR=500 noP=2000 noRP=150 noReqps=500" were fixed. While Route dealt quite well with large noRP, the performance of Route with respect to large noUR was relatively poor. One reason might be that, the larger noUR is, the more NuSMV programs were created and checked; and the reductions also took notable time. Even though reductions were performed and many role-permission assignments were set constants according to π, there were still many variables, for $|\mathsf{roles}_\gamma[u]|$ could be quite large (> 5); since constraints are put on the permission assignments of roles in $\mathsf{roles}_\gamma[u]$, the large number of $|\mathsf{roles}_\gamma[u]|$ prevents more variables from being fixed.

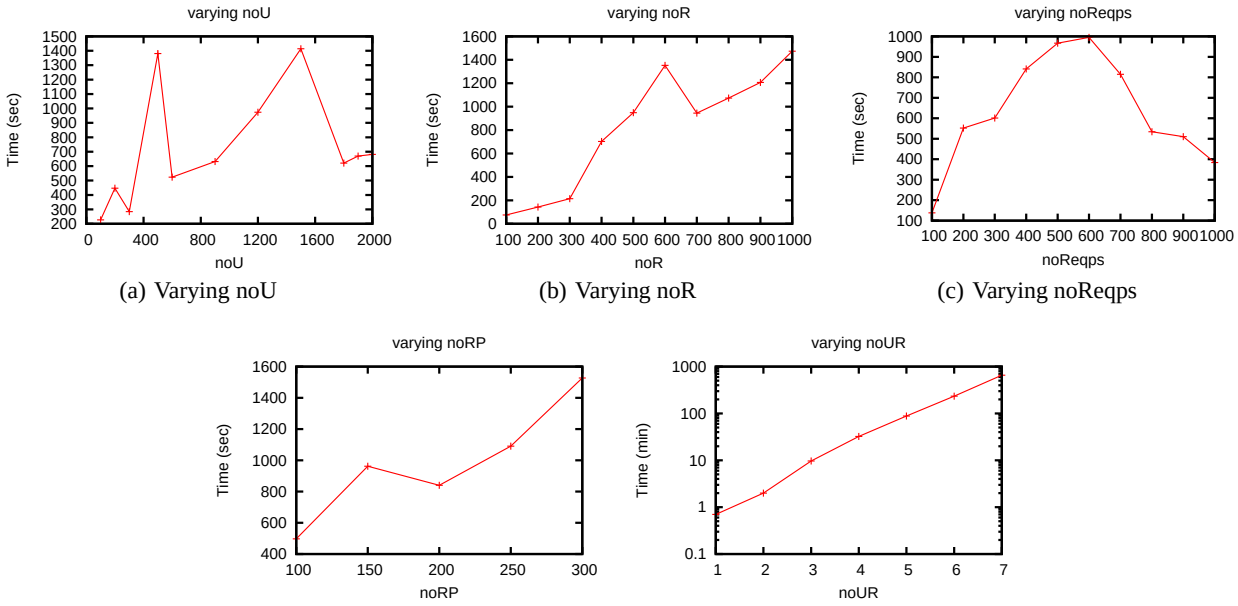
In real-world large-scale RBAC systems, even though noUR could be large (> 5), we expect that only a small portion of users have a number noUR of roles and that the number of roles that are under an SSO's control will be small. Hence, we conjecture Route will be able to handle update requests in these RBAC systems.

7. RELATED WORK

RBAC policy analysis and repair. Many RBAC policy analysis tools (RPATs) are invented to help administrators understand and manage RBAC policies e.g., [3, 14, 19, 31, 32, 36], to name a few. Most (safety) analysis problems in literature basically can be stated as: given the current state γ, a query q, and a state-change rule φ, can γ be taken a state γ' where q evaluates true? If this is the case, one may argue that the steps taking γ to γ' may also be reported to SSOs so that they can follow to make γ'. However, as the objectives are different, we believe this report could hardly be sufficient for RUP. As remarked in [14], "...q typically encodes an unsafe situation that should never occur;..." Hence, RPATs explore every possible sequence of actions, as long as they are allowed by φ, to test if there is such a γ' where q is true; consequently, RPATs do not care what the resulting states look like. On the contrary, Route seeks for a resulting state with expected assignments. In addition, most RPATs focus on user-role assignments; although it is argued that role-permission relation is a dual of user-role relation and might be treated likewise, role-permission relation also deserves its own attention [22], especially in terms of role updating. Finally, Route goes beyond by enabling various constraints on updates, with which SSOs specify requirements on updates.

Role engineering. Recently, much research effort has been devoted to role engineering [10, 7, 20, 33, 34, 37]. Basically speaking, existing role engineering tools (eRETs) consume a predefined user-permission assignments and output a set of user-role assignments and a set of role-permission assignments, taking into account some optimization objectives (e.g., a minimal set of roles) and possibly other concerns such as roles' business meanings, semantics, users' attributes. Taxonomically, Route can be viewed as a role engineering tool, as RUP also deals with these assignments. However, role updating works when RBAC states have been defined and possibly deployed, whereas eRETs usually define roles from scratch. The focuses are also different. Role updating aims to answer SSOs' question whether an update is achievable with respect to update

[6] In practice, one can run all programs in parallel, and stop them when one of them returns a counterexample or all programs return with an answer "true".



(a) Varying noU (b) Varying noR (c) Varying noReqps

(d) Varying noRP (e) Varying noUR

Figure 3: The computing time of evaluating req$\langle \gamma, \mathcal{P}, \pi, \mathcal{T} \rangle$

constraints and, if any, to generate one. On the contrary, eRETs put more emphases on how to define a appropriate set of roles. In the view of role life cycle, Route is for role maintenance, while eRETs help role design. Thus, one may condsider Route as a complement of eRETs; Route can be used to fine-tune the ideal state generated by eRETs.

Model checking in RBAC. Jha et al. [14] presented a transformation from an RBAC policy analysis instance to a NuSMV program by letting states correspond to user-role assignments and transitions correspond to administrative rules. However, their transformation does not fit for Route. The reason is that it is more intuitive and convenient to encode constraints on updates (i.e., π) in NuSMV states rather than in the transitions. This reflects the difference between administrative rules and constraints on updates: administrative rules specify what transitions can be made from each state, whereas constraints on updates put restrictions on the resulting states. Schaad et al. [27] applied model checking techniques to automated analysis of delegation and revocation functionalities, with an emphasis on static and dynamic separation of duty properties. They do not consider the role-permission assignments. Reith et al. [24] applied model checking techniques to the policy analysis of a language RT_0, which can be viewed as a generation of RBAC models considered in this paper. It is unclear how to use their algorithms to tackle RUP though. There are some other works, which applied model checking techniques to RBAC or its variants, such as [1, 21]; but, to our knowledge, Route is the first to use them for RUP.

RBAC updating. Ni et al. [22] studied the role adjustment problem (RAP) in the context of role-based provisioning based on machine-learning algorithms. Though similar, RUP differs from the RAP in several aspects. First, customized constraints on updates are enforced in role updating, whereas it is unclear if these constraints could be supported in RAP. Second, the role updating is request-driven, whereas RAP is a learning process. Specially, SSOs submit a specific update objective to Route, which tries to find the expected update. On the contrary, RAP is supplied by SSOs with provisioning data and output a set of mappings from roles to (new) entitlements. Hence, RAP and Route are both assistant tools for SSOs but with different usage and orientation. Fisler et al. [15] presented a tool to investigate the semantic difference of two RBAC policies (in XACML) and the properties of the difference. They do not consider how to make a different desired state from the current one. Ray [23] studied the problem of real-time update of access control policies, in the context of a database system. The focus was put on the transaction properties. However, RBAC models have important features that deserve consideration when updating.

8. CONCLUSION AND FUTURE WORK

We have studied the RUP problem, presented a set of reductions for RUP, and proposed a role updating tool Route based on model checking techniques. Experiments confirm the effectiveness and efficiency of Route. There are several avenues for future work. Two additional components, role hierarchies and separation of duty (SoD) policies, are also useful in RBAC systems. Their presence complicates the problem. Role hierarchies are important for RBAC systems, as they further mitigate the burden of security administration and maintenance. In the case of SoD policies, enforcing SoD policies is difficult by itself [17]. The interaction between updating and SoD policies poses new challenges. Existing works often assume that role-permission relation is fixed, when considering SoD policies. However, this assumption does not hold from the viewpoint of role updating. Another interesting problem is to update RBAC systems when administrative rules are in position to regulate SSOs' actions.

Acknowledgment

This work is supported by National Natural Science Foundation of China under Grant 60873225, 60773191, 70771043, National High Technology Research and Development Program of China under Grant 2007AA01Z403, and Natural Science Foundation of Hubei Province under Grant 2009CDB298. This project is supported in

part by an Australian Research Council (ARC) Discovery Projects Grant (DP0988396). We thank the anonymous reviewers for their helpful comments.

9. REFERENCES

[1] T. Ahmed and A. R. Tripathi. Static verification of security requirements in role based cscw systems. In *SACMAT'03*, pages 196–203.

[2] D. A. Basin, S. J. Burri, and G. Karjoth. Dynamic enforcement of abstract separation of duty constraints. In *ESORICS*, pages 250–267, 2009.

[3] L. Bauer, S. Garriss, and M. K. Reiter. Detecting and resolving policy misconfigurations in access-control systems. In *SACMAT'08*, pages 185–194.

[4] L. Chen and J. Crampton. Set covering problems in role-based access control. In *ESORICS*, pages 689–704, 2009.

[5] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, LNCS, pages 359–364, 2002.

[6] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[7] A. Colantonio, R. D. Pietro, A. Ocello, and N. V. Verde. A formal framework to elicit roles with business meaning in rbac systems. In *SACMAT'09*, pages 85–94.

[8] J. Crampton. Understanding and developing role-based administrative models. In *CCS*, pages 158 – 167, Alexandria, VA, USA, Nov. 2005. CCS'05.

[9] S. Du and J. B. D. Joshi. Supporting authorization query and inter-domain role mapping in presence of hybrid role hierarchy. In *SACMAT'06*, pages 228–236.

[10] A. Ene, W. G. Horne, N. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan. Fast exact and heuristic methods for role minimization problems. In *SACMAT'08*, pages 1–10.

[11] D. F. Ferraiolo, R. S. Sandhu, S. I. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.

[12] J. Hu, Y. Zhang, R. Li, and Z. Lu. Role updating for assignments. Technical report, Huazhong University of Science and Technology & University of Western Sydney, 2010.

[13] K. Irwin, T. Yu, and W. H. Winsborough. Enforcing security properties in task-based systems. In *SACMAT'08*, pages 41–50.

[14] S. Jha, N. Li, M. Tripunitara, Q. Wang, and W. Winsborough. Towards formal verification of role-based access control policies. *IEEE Trans. Dependable Secur. Comput.*, 5(4):242–255, 2008.

[15] L. M. Kathi Fisler, Shriram Krishnamurthi and M. Tschantz. Verification and change impact analysis of access-control policies. In *ICSE*, May 2005.

[16] A. Kern, M. Kuhlmann, A. Schaad, and J. D. Moffett. Observations on the role life-cycle in the context of enterprise security management. In *SACMAT*, pages 43–51, 2002.

[17] N. Li, Z. Bizri, and M. V. Tripunitara. On mutually-exclusive roles and separation of duty. In *CCS*, pages 42–51, 2004.

[18] N. Li and Z. Mao. Administration in role-based access control. In *ASIACCS*, pages 127–138, 2007.

[19] N. Li and M. V. Tripunitara. Security analysis in role-based access control. In *SACMAT*, pages 126–135, 2004.

[20] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. B. Calo, and J. Lobo. Mining roles with semantic meanings. In *SACMAT*, pages 21–30, 2008.

[21] S. Mondal, S. Sural, and V. Atluri. Towards formal security analysis of gtrbac using timed automata. In *SACMAT'09*, pages 33–42.

[22] Q. Ni, J. Lobo, S. B. Calo, P. Rohatgi, and E. Bertino. Automating role-based provisioning by learning from examples. In *SACMAT*, pages 75–84, 2009.

[23] I. Ray. Applying semantic knowledge to real-time update of access control policies. *IEEE Trans. Knowl. Data Eng.*, 17(6):844–858, 2005.

[24] M. Reith, J. Niu, and W. H. Winsborough. Toward practical analysis for trust management policy. In *ASIACCS '09*, pages 310–321. ACM.

[25] R. S. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for role-based administration of roles. *TISSEC*, 2(1):105–135, 1999.

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

[27] A. Schaad, V. Lotz, and K. Sohr. A model-checking approach to analysing organisational controls in a loan origination process. In *SACMAT'06*, pages 139–149.

[28] A. Schaad and J. D. Moffett. A lightweight approach to specification and analysis of role-based access control extensions. In *SACMAT'02*, pages 13–22.

[29] B. Shafiq, J. Joshi, E. Bertino, and A. Ghafoor. Secure interoperation in a multidomain environment employing rbac policies. *IEEE Trans. Knowl. Data Eng.*, 17(11):1557–1577, 2005.

[30] M. Shehab, E. Bertino, and A. Ghafoor. SERAT: SEcure Role mApping Technique for decentralized secure interoperability. In *SACMAT'05*, pages 159–167.

[31] K. Sohr, M. Drouineaud, G.-J. Ahn, and M. Gogolla. Analyzing and managing role-based access control policies. *Knowledge and Data Engineering, IEEE Transactions on*, 20(7):924–939, July 2008.

[32] S. D. Stoller, P. Yang, C. Ramakrishnan, and M. I. Gofman. Efficient policy analysis for administrative role based access control. In *CCS'07*.

[33] J. Vaidya, V. Atluri, and Q. Guo. The role mining problem: Finding a minimal descriptive set of roles. In *SACMAT*, pages 175–184, 2007.

[34] J. Vaidya, V. Atluri, and J. Warner. Roleminer: mining roles using subset enumeration. In *CCS*, pages 144–153, 2006.

[35] G. T. Wickramaarachchi, W. H. Qardaji, and N. Li. An efficient framework for user authorization queries in rbac systems. In *SACMAT*, pages 23–32, 2009.

[36] W. Xu, M. Shehab, and G.-J. Ahn. Visualization based policy analysis: case study in selinux. In *SACMAT'08*, pages 165–174.

[37] D. Zhang, K. Ramamohanarao, T. Ebringer, and T. Yann. Permission set mining: Discovering practical and useful roles. In *ACSAC*, pages 247–256, 2008.

[38] Y. Zhang and J. B. D. Joshi. Uaq: a framework for user authorization query processing in rbac extended with hybrid hierarchy and constraints. In *SACMAT*, pages 83–92, 2008.

APPENDIX

A. EXAMPLE OUTPUT OF NUSMV

Consider γ_{ex} in Figure 1 and $Q_3 = \mathsf{req}\langle\gamma_{\mathsf{ex}}, \mathcal{P}, \pi, \mathcal{T}\rangle$ as follows:

- $\mathcal{P} = \{p_5, p_8, p_9\}$,
- $\pi.U = \gamma_{\mathsf{ex}}.U = \{u_1, u_2, u_3, u_4\}$ and, for any $u \in \pi.U$, $\pi.th[u] = \mathsf{perms}_{\gamma_{\mathsf{ex}}}[u]$, and
- $\mathcal{T} = \gamma_{\mathsf{ex}}.R = \{r_1, r_2, r_3, r_4, r_5, r_6\}$.

Below is a example running of the NuSMV program translated from Q_3. `State:1.1` describes the RBAC state after the reductions are performed. During the reductions, some assignments are added to γ_{ex}, as indicated by the underlined variables. Note that we denote each assignment as a variable, and fix variables as constants if possible. `State:1.2` only lists the variables whose values have been changed from `State:1.1`.

```
>*** Compilation and copyright information ***
>
>-- specification
     AG !(x-wu-p8 & !x-wu-p6 & !x-wu-p1 &
          x-wu-p9 & !x-wu-p7 & x-wu-p5 & !x-wu-p2)
    is false
>-- as demonstrated by the following execution
    sequence
>Trace Description: CTL Counterexample
>Trace Type: Counterexample
>-> State: 1.1 <-
>  x-wu-r5 = 0  x-wu-r3 = 0  x-wu-r2 = 0
>  x-wu-r6 = 0  x-wu-r4 = 0  x-r5-p6 = 0
>  x-r5-p1 = 1  x-r5-p7 = 0  x-r3-p1 = 0
>  x-r2-p6 = 1  x-r2-p7 = 0  x-r2-p2 = 1
>  x-r6-p6 = 1  x-r6-p7 = 0  x-r6-p2 = 0
>  x-r4-p6 = 1  x-r4-p7 = 1  x-r6-p8 = 1
>  x-r5-p8 = 1  x-r6-p5 = 1  x-r2-p5 = 1
>  x-r4-p8 = 1  x-r5-p5 = 1  x-r2-p8 = 1
>  x-r4-p5 = 1  x-r6-p9 = 1  x-r2-p9 = 1
>  x-r3-p5 = 1  x-r4-p2 = 0  x-r3-p6 = 0
>  x-r4-p1 = 0  x-r3-p7 = 0  x-r4-p9 = 0
>  x-r3-p8 = 0  x-wu-p8 = 0  x-wu-p6 = 0
>  x-wu-p1 = 0  x-wu-p9 = 0  x-wu-p7 = 0
>  x-wu-p5 = 0  x-wu-p2 = 0
>-> Input: 1.2 <-
>-> State: 1.2 <-
>  x-wu-r6 = 1  x-r6-p6 = 0  x-wu-p8 = 1
>  x-wu-p9 = 1  x-wu-p5 = 1
```

From `State:1.1` and `State:1.2`, Route computes χ and outputs $\mathtt{diff}(\chi, \gamma)$ in the form of assign and revoke actions. Note that, as reductant assignments are made by reductions before model checking, one could remove unnecessary changes from χ prior to generating actions.