Solving the user-role reachability problem in ARBAC with role hierarchy

Anh Truong, Dai Hai Ton That

2016 International Conference on Advanced Computing and Applications

Solving the User-role Reachability Problem in ARBAC with Role Hierarchy

Anh Truong
Faculty of Computer Science and Engieering
Ho Chi Minh City University of Technology, Vietnam
Email: anhtt@hcmut.edu.vn

Dai Hai Ton That
CEA LIST Institute, CEA Saclay
Paris Saclay University, France
Email: daihai.tonthat@cea.fr

***Abstract*—Access Control is becoming increasingly important for today's ubiquitous systems since it provides mechanism to prevent sensitive resources in the systems against unauthorized users. In access control models, the administration of access control policies is an important task that raises a crucial analysis problem: if a set of administrators can give a user an unauthorized access permission. We consider the analysis problem in the context of the Administrative Role-Based Access Control (ARBAC), the most widespread administrative model. One of the main assumptions of current analysis techniques is that the role hierarchy is constant and thus can be abstracted away that results in the bad scalability of analysis techniques. In this paper, we introduce three reductions to enable an available analysis technique, namely ASASPXL, to handle the user-role reachability problem with the presence of role hierarchy. An extensive experimentation reports the superiority of our reductions in comparison with the approach used in the literature.**

I. Introduction

Modern information systems contain sensitive information and resources that need to be protected against unauthorized users who want to steal it. The most important mechanism to prevent this is Access Control [1] which is thus becoming increasingly important for today's ubiquitous systems. In general, access control policies protect the resources of the systems by controlling who has permission to access what objects/resources.

Role-Based Access Control (RBAC) [2] is one of the most widely adopted access control models in the real world. In RBAC, access control policies specify which users can be assigned to roles which, in turn, are granted permissions to perform certain operations in the system. Usually, RBAC policies need to be evolved according to the rapidly changing environments and thus, it is demanded to have some mechanisms to control the modification of the policies. Administrative RBAC [3] (ARBAC) is the corresponding widely used administrative model for RBAC policies. The main idea of ARBAC is to provide certain specific users, called administrators, some permissions to execute operations, called administrative actions, to modify the RBAC policies. In fact, permissions to perform administrative actions must be restricted since administrators can only be partially trusted. For instances, some of them may collude to, inadvertently or maliciously, modify the policies (by sequences of administrative actions) so that untrusted users can get sensitive permissions. Thus, automated analysis techniques taking into consideration the effect of all possible sequences of administrative actions to identify the safety issues, i.e. administrative actions generating policies by which a user can acquire permissions that may compromise some security goals, are needed.

Several automated analysis techniques (see, e.g., [4], [5], [6], [7], [8], [9]) have been developed for solving the user-role reachability problem, an instance of the safety issues, in the ARBAC model. One of the the main assumptions of such techniques is that the role hierarchy is constant and can be pre-processed by the approach proposed in [10]. However, this approach results in an exponential number of administrative actions considered in the analysis that negatively affects the performance of analysis tools.

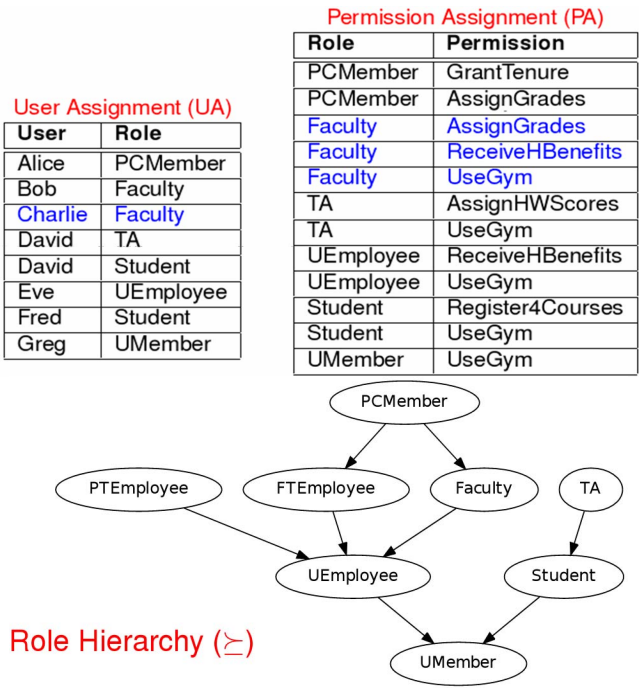
In this paper, we introduce three reductions to enable an available analysis technique, namely ASASPXL, to handle the user-role reachability problem with the presence of role hierarchy. The main idea is to transform an ARBAC system with role hierarchy to an equivalent one without role hierarchy and then use available analysis techniques to analyze the system. The experimental results show that the performance of one of the proposed reductions is superior to the other two and much better than the approach proposed in [10]. The reason for this is that the number of administrative actions considered in the analysis is reduced significantly by using our reductions that allows for the generation of reachability problems with smaller state spaces.

The paper is organized as follows. Section II introduces the RBAC, ARBAC models, and the related analysis problem. Section III briefly introduces the automated analysis tool ASASPXL and the model checking technique underlying it. The three reductions to enable ASASPXL to handle the user-role reachability problem with the presence of role hierarchy are described in Section IV. Section V discusses the dynamic role hierarchy and how to extend our reductions to solve the user-role reachability problem in the context of dynamic role hierarchy. Section VI summarizes our experiments and Section VII concludes the paper.

II. User-Role Reachability Problem

A. Role-Based Access Control

In *Role-Based Access Control (RBAC)*, access decisions are based on the roles that individual users have as part of an organization. The process of defining roles is based on a



User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	Student
Greg	UMember

Role	Permission
PCMember	GrantTenure
PCMember	AssignGrades
Faculty	AssignGrades
Faculty	ReceiveHBenefits
Faculty	UseGym
TA	AssignHWScores
TA	UseGym
UEmployee	ReceiveHBenefits
UEmployee	UseGym
Student	Register4Courses
Student	UseGym
UMember	UseGym

Fig. 1. User and Permission Assignments; and Role Hierarchies

careful analysis of how an organization operates. Permissions are grouped by role name and correspond to various uses of a resource. A permission is restricted to individuals authorized to assume the associated role and represents a unit of control, subject to regulatory constraints within the RBAC model. For example, within a hospital, the role of doctor can include operations to perform diagnosis, prescribe medication, and order laboratory tests; the role of nurse can be limited to a strict subset of the permissions assigned to a doctor such as order laboratory tests.

We formalize a *RBAC policy* as a tuple $(U, R, P, UA, PA, \succeq)$ where U is a set of users, R a set of roles, and P a set of permissions. A binary relation $UA \subseteq U \times R$ represents a user-role assignment and a binary relation $PA \subseteq R \times P$ represents a role-permission assignment. A user-role assignment specifies the roles to which the user has been assigned while a role-permission assignment specifies the permissions that have been granted to a role. A partial order $\succeq$ on R is a role hierarchy of the policy, where $r_1 \succeq r_2$ means that r_1 is *more senior than* r_2 for $r_1, r_2 \in R$, i.e., every permission assigned to r_2 is also available to r_1.

A user u is an *explicit member* of role r when $(u, r) \in UA$ while the user u is an *implicit member* of role r if there exists $r' \in R$ such that $r' \succeq r$ and $(u, r') \in UA$. A user u *has permission* p if there exists a role $r \in R$ such that $(r, p) \in PA$ and u is a (explicit or implicit) member of r.

Example 1 *Consider an RBAC policy describing a department in a university as depicted in Figure 1. The top-left table is the user-role assignment, the top-right is the role-permission assignment, and the bottom is an example of role hierarchies (The role at the tail of an arrow is more senior than the one at the head).*

Let us consider the user Charlie: he is an explicit *member of role Faculty because the tuple* $(Charlie, Faculty)$ *is in the user-role assignment UA. Additionally, role Faculty has been assigned to permissions AssignGrades, ReceiveHBenefits, and UseGym. Thus, Charlie can assign grades, receive benefits and use the gym through the role Faculty.*

Let us consider the role hierarchy: role Faculty is more senior *than role UEmployee (i.e.,* $Faculty \succeq UEmployee$*). Therefore, Charlie is an* implicit *member of role UEmployee, and thus he can also use all permissions assigned to role UEmployee.*

B. Administrative RBAC (ARBAC)

Access control policies need to be maintained according to the evolving needs of the organization. For flexibility and scalability in large distributed systems, several administrators are usually required and there is a need not only to have a consistent policy but also to ensure that the policy is modified by administrators who are allowed to do so.

Several administrative frameworks have been proposed on top of the RBAC model to address these issues. One of the most popular administrative frameworks is Administrative RBAC (ARBAC) [3] that controls how RBAC policies may evolve through administrative actions that update the UA and PA relations (e.g., actions that update UA include assigning or revoking user memberships into roles).

Formalization. Usually, administrators may only update the relation UA while PA and $\succeq$ are assumed constant. This is because a change in PA and/or $\succeq$ implies a change in the organization (see [6] for more detail). From now on, we focus on situations where U and R are finite, P plays no role, and thus, a RBAC policy is a tuple $(U, R, UA, \succeq)$.

Since administrators can be only partially trusted, administration privileges must be limited to selected parts of the RBAC policies, called *administrative domains*. An administrative domain is specified by a *condition* defined as follows:

Definition 1 *A* pre-condition C *is a finite set of expressions of the forms* r *or* $\overline{r}$ *where* $r \in R$.

A user $u \in U$ *satisfies* a pre-condition C with respect to a role hierarchy $\succeq$ if, for each $\ell \in C$, u is a member of r with respect to $\succeq$ when ℓ is r or u is not a member of r with respect to $\succeq$ when ℓ is $\overline{r}$ for $r \in R$. We say that r is a positive role and $\overline{r}$ is a negative role in C. Notice that the role membership must consider both explicit and implicit users (i.e., a user u is a member of role r if u is an explicit or implicit member of r) because of role hierarchy $\succeq$.

Permission to assign users to roles is specified by a ternary relation can_assign containing tuples of the form (C_a, C, r) where C_a and C are pre-conditions, and r a role. Permission to revoke users from roles is specified by a binary relation can_revoke containing tuples of the form (C_a, r) where C_a is a pre-condition and r a role. The relation can_revoke is

only binary because simple pre-conditions are useless when revoking roles (see, e.g., [6]). In both cases, we say that C_a is the *administrative pre-condition*, C is a *(simple) pre-condition*, r is the *target role*, and a user u_a satisfying C_a is the *administrator*. When there exist users satisfying the administrative and the simple (if the case) pre-conditions of an administrative action, the action is *enabled*.

ARBAC transition system. We define an ARBAC transition system as a tuple (α_0, ψ) where α_0 is the *initial* RBAC policy $(U, R, UA_0, \succeq)$ and ψ is the (disjoint) union of the sets of administrative actions can_assign and can_revoke (i.e., $\psi := (can\_assign, can\_revoke)$). A *state* of an ARBAC transition system is a tuple α where α is a RBAC policy. Since the administrative actions depend on and affect only the relations UA and $\succeq$, in the following, we abbreviate a RBAC policy $(U, R, UA, \succeq)$ as $(UA, \succeq)$. We define the effect of executing an administrative action in ψ by defining a binary relation $\rightarrow_\psi$ on the states of the ARBAC system as follows:

Definition 2 *$(UA, \succeq) \rightarrow_\psi (UA', \succeq)$ iff there exist users u_a and u in U such that either:*

- *there exists $(C_a, C, r) \in can\_assign$, u_a satisfies C_a, u satisfies C (i.e. (C_a, C, r) is enabled), and $UA' = UA \cup \{(u, r)\}$ or*
- *there exists $(C_a, r) \in can\_revoke$, u_a satisfies C_a (i.e. (C_a, r) is enabled), and $UA' = UA \setminus \{(u, r)\}$.*

A *run* of the ARBAC transition system (α_0, ψ) is a (possibly infinite) sequence $(UA_0, \succeq), (UA_1, \succeq), ..., (UA_n, \succeq), ...$ of states such that $(UA_i, \succeq) \rightarrow_\psi (UA_{i+1}, \succeq)$ for $i \geq 0$.

Example 2 *Consider the RBAC policy with the UA relation and role hierarchy depicted in Figure 1 and an administrative action $(\{PCMember\}, \{UMember\}, Student) \in can\_assign$, i.e., the administrative pre-condition is $C_a = \{PCMember\}$, the simple pre-condition is $C = \{UMember\}$, and the target role is $Student$.*

User $Alice$ satisfies the pre-condition C_a because $(Alice, PCMember) \in UA$. User Eve satisfies the pre-condition C because he is an explicit member of role $UEmployee$ that is more senior than role $UMember$ (e.g., $(Fred, UEmployee) \in UA$ and $UEmployee \succeq UMember$. As a sequence, the administrative action is enabled.

We can update the current UA to $UA' = UA \cup \{(Eve, Student)\}$ by executing the following instance of the administrative action specified above: administrator $Alice$ (who has role $PCMember$) assigns role $Student$ to user Eve.

C. The User-role Reachability Problem

Normally, policy designers and administrators want to foresee if the interactions among administrative actions, as seen in the Example 2, can lead the system to conflict states violating the security requirements of the organization (e.g., the security requirements forbid a user being assigned to some sensitive roles). Thus, they need to analyze access control policies in order to discover such violation. This problem is called as the user-role reachability problem and is defined as follows.

Definition 3 *A pair (u_g, R_g) is called a* (RBAC) goal *for $u_g \in U$ and R_g a finite set of roles. The cardinality $|R_g|$ of R_g is the* size *of the goal.*

Definition 4 *Given an initial RBAC policy $(UA, \succeq)$, a goal (u_g, R_g), and administrative actions $\psi = (can\_assign, can\_revoke)$; (an instance of) the* **user-role reachability problem***, identified by the tuple $\langle (UA, \succeq), \psi, (u_g, R_g) \rangle$, consists of checking if there exists a finite sequence $(UA_0, \succeq), (UA_1, \succeq), ..., (UA_n, \succeq)$ (for $n \geq 0$) where (i) $(UA_i, \succeq) \rightarrow_\psi (UA_{i+1}, \succeq)$ for each $i = 0, ..., n-1$ and (ii) u_g is a member of each role of R_g in $(UA_n, \succeq)$.*

In real scenario, subtle interactions between administrative actions in real policies may arise that are difficult to be foreseen by policy designers and administrators. Thus, automated analysis techniques are thus of paramount importance to analyze such policies and answer the user-role reachability problem. The analysis techniques we will present in the following will be able to establish this automatically for the problem in ARBAC.

III. Model Checking Modulo Theories and the Reachability Problem

Model Checking Modulo Theories (MCMT). MCMT [11] is a framework to solve reachability problems for infinite state systems that can be represented by transition systems whose set of states and transitions are encoded as constraints in first-order logic. Several systems have been abstracted using such symbolic transition system such as parametrised protocols, sequential programs manipulating arrays, timed system, etc (see again [11] for an overview).

MCMT framework uses a backward reachability procedure that repeatedly computes the so-called pre-images of the set of *goal* states, that is usually obtained by complementing a certain safety property that the system should satisfy. Then, the set of backward reachable states of the system is obtained by taking the union of the pre-images. At each iteration of the procedure, the procedure checks whether the intersection between the set of backward reachable states and the initial set of states is non-empty (i.e., *safety* test) or not (i.e., the *unsafety* of the system: there exists a (finite) sequence of transitions that leads the system from an initial state to one satisfying the goal). Otherwise, when the intersection is empty, the procedure checks if the set of backward reachable states is contained in the set computed at the previous iteration (*fix-point* test) and, if yes, the *safety* of the system (i.e. no (finite) sequence of transitions leads the system from an initial state to one satisfying the goal) is returned. Since sets of states and transitions are represented by first-order constraints, the computation of pre-images reduces to simple symbolic manipulations and testing safety and fix-point to solving a particular class of constraint satisfiability problems, called Satisfiability Modulo Theories (SMT) problems, for which scalable and efficient SMT solvers are currently available (e.g., Z3 [12]).

ASASPXL. In [13], it is studied how the MCMT approach can be used to solve (variants of) the user-role reachability problem in ARBAC transition system with out role hierarchy. On the theoretical side, it is shown that the backward reachability procedure described above decides (variants of) the user-role reachability problem. On the practical side, extensive experiments have shown that an automated tool, called ASASPXL [13], has a good *trade-off* between *scalability* and *expressiveness*. The analysis tool ASASPXL is build on top of MCMT, the first implementation of the MCMT approach [11], that gives some advantage. First, we only need to write a translator from instances of the user-role reachability problem to reachability problems in MCMT input language, a routine programming task. Second, MCMT has been developed and extensively used for the past years. It is thus more robust and offers a higher degree of confidence. Third, we can reuse some features of a better engineered incarnation of the MCMT approach that can be exploited to significantly improve performances. An exhaustive experiment in [13] has shown that ASASPXL is superior to the state-of-the-art analysis tools such as MOHAWK [14], VAC [8], and PMS [9].

The structure of ASASPXL is depicted in Figure 2. It takes as input an instance of the user-role reachability problem and returns `reachable`, when there exists a finite sequence of administrative operations that lead from the initial RBAC policy to one satisfying the goal, and `unreachable` otherwise. To give such results, ASASPXL firstly pre-processes the original user-role reachability problem by module **Heuristics** that helps to refine the original problem to speed up the analysis in the module MCMT (see [13] for more details). Then, it translates the refined user-role reachability problem to the reachability problem in MCMT input language (module **Translator**). Next, ASASPXL invokes the model checker MCMT to verify the reachability of the problem. Finally, according to the answer returned by the model checker (in the data storage **Explored Policies**), ASASPXL refines it and returns `reachable` or `unreachable` as its output (module **Refinement**).

IV. Solving the User-role Reachability Problem with Role Hierarchy

The analysis tool ASASPXL (and all state-of-the-art analysis tools mentioned in Section III) assumes that the role hierarchy relation $\succeq$ is constant and thus can be ignored when solving the user-role reachability problem (As the result, the role membership considers only explicit users instead of both implicit and explicit users as in Section II-B). In the following, we describe three reductions that pre-process away role hierarchies so that (an adapted version of) the technique in ASASPXL can be used to solve user-role reachability problem.

The first reduction, namely $\mathbb{R}_{\mathbb{A}}$, is an adaptation of the approach proposed in [10] and applied to ASASPXL. The second one, namely $\mathbb{R}_{\mathbb{L}}$, proposes a solution that aims to avoid the exponential explosion in size of the user-role reachability problem resulting from the application of $\mathbb{R}_{\mathbb{A}}$. The last one $\mathbb{R}_{\mathbb{M}}$ exploits a feature of the analysis technique inside ASASPXL to atomize chains of the additional administrative actions generated in $\mathbb{R}_{\mathbb{L}}$, thereby reducing significantly the possible interleavings to solve the user-role reachability problems.

A. Abstract Reduction $\mathbb{R}_{\mathbb{A}}$

In [10], the authors propose an approach to analyze ARBAC policies with role hierarchies that applies a preprocess module to abstract away role hierarchy in the original user-role reachability problem and then using an available analysis technique to solve the problem. The main idea of the preprocess module is to replace each action in the original set of administrative actions ψ by a set of additional actions with respect to the hierarchies of roles being present in the pre-conditions of the original action. We adapt this approach to ASASPXL as follows.

Let consider the user-role reachability problem with role hierarchy $\langle (UA, \succeq), \psi, (u_g, R_g)\rangle$ as defined in Section II-C and the following abbreviations:

- $Senior(r)$ stands for a set of all senior roles of r with respect to hierarchy $\succeq$
- $\overline{Senior(r)}$ stands for a set of all senior roles of r with respect to hierarchy $\succeq$ but is written in negative form $\overline{r_i}$ where r_i is a senior role of r
- $Senior(C) = Senior(r_1) \times Senior(r_2) \times ... \times Senior(r_k)$ where $C = \{r_1, r_2, ..., r_k\} \subseteq R$

The abstract reduction $\mathbb{R}_{\mathbb{A}}$ works as follows.

Step 1 Processing negative roles in pre-conditions:
- For each tuple $(C_a, C, r) \in can\_assign$:
  - for each negative role $\overline{r}$ occurring in C_a: replace $\overline{r}$ with $\overline{Senior(r)}$ with respect to role hierarchy $\succeq$.
  - for each negative role $\overline{r}$ occurring in C: replace $\overline{r}$ with $\overline{Senior(r)}$ with respect to role hierarchy $\succeq$.
- For each tuple $(C_a, r) \in can\_revoke$:
  - for each negative role $\overline{r}$ occurring in C_a: replace $\overline{r}$ with $\overline{Senior(r)}$ with respect to role hierarchy $\succeq$

Step 2 Processing positive roles in-preconditions:
- For each $(C_a, C, r) \in can\_assign$:
  - Let $C_a^+ = \{r_{1a}, r_{2a}, ..., r_{ka}\}$ denote the set of all positive roles in C_a and $C^+ = \{r_1, r_2, ..., r_l\}$ denote the set of all positive roles in C. Let C_a^- and C^- denote the set of all negative roles in C_a and C, respectively
  - for each tuple $(r'_{1a}, r'_{2a}, ..., r'_{ka}) \in Senior(C_a^+)$ and tuple $(r'_1, r'_2, ..., r'_l) \in Senior(C^+)$: add to ψ the administrative action $(\{r'_{1a}, r'_{2a}, ..., r'_{ka}\} \cup C_a^-, \{r'_1, r'_2, ..., r'_l\} \cup C^-, r)$
- For each $(C_a, r) \in can\_revoke$:
  - Let $C_a^+ = \{r_{1a}, r_{2a}, ..., r_{ka}\}$ denote the set of all positive roles in C_a and C_a^- denote the set of all negative roles in C_a

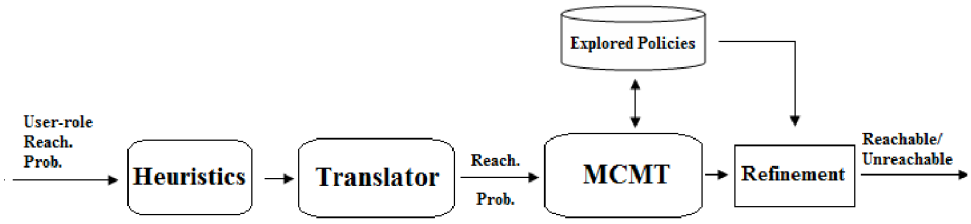


Fig. 2. ASASPXL architecture

- for each tuple $(r'_{1a}, r'_{2a}, ..., r'_{ka}) \in Senior(C_a^+)$: add to ψ the administrative action $(\{r'_{1a}, r'_{2a}, ..., r'_{ka}\} \cup C_a^-, r)$

Step 3 Processing the goal (u_g, R_g):

- For each set $R'_g \in Senior(R_g)$: create a new user-role reachability problem without role hierarchy $\langle (UA), \psi, (u_g, R'_g) \rangle$
- The answer for the original user-role reachability problem is reachable iff one of the new problems $\langle (UA), \psi, (u_g, R'_g) \rangle$ returns reachable

The main idea underlying **Step 1** is to guarantee that a user satisfying a negative role $\overline{r}$ must not be assigned to any senior role of r while **Step 2** exploits all possible associations of positive roles with respect to role hierarchy. We emphasize that solving the original user-role reachability problem with role hierarchy is now equivalent to solving a set of new user-role reachability problems without role hierarchy (**Step 3**). The idea is to ensure that if a role senior to $r_g \in R_g$ (in the goal (u_g, R_g) of the original problem) is reachable, then also r_g is reachable.

Example 3 *Consider an ARBAC system* (α_0, ψ) *where* $\alpha_0 = (UA_0, \succeq)$.

Let $U = \{Alice, Bob, Charlie, David, Eve, Fred, Greg\}$, $R = \{PC(PCMember),\ FA(Faculty),\ TA,\ ST(Student),\ UE(UEmployee),\ UM(UMember), PT(PTEmployee)\}$, *and* $\succeq = \{(TA \succeq ST), (PC \succeq FA), (PC \succeq PT), (UE \succeq UM)\}$. *The goal is* $(Eve, \{ST, UM\})$.

The set ψ *of administrative actions contains:*

$$(\{PC\}, \{UM, \overline{ST}\}, PT) \in can\_assign \quad (1)$$

$$(\{FA\}, ST) \in can\_revoke \quad (2)$$

Consider action (1) (the other actions will be processed in a similar way):

Step 1 *of* $\mathbb{R}_\mathbb{A}$ *transforms action (1)* $\in \psi$ *to the following action:*

$$(\{PC\}, \{UM, \overline{ST}, \overline{TA}\}, PT) \quad (3)$$

by replacing the negative role $\overline{ST}$ *with the set* $\overline{Senior(ST)} = \{\overline{ST}, \overline{TA}\}$.

Step 2 *of* $\mathbb{R}_\mathbb{A}$ *adds the following administrative actions to* ψ*:*

$$(\{PC\}, \{UM, \overline{ST}, \overline{TA}\}, PT) \in can\_assign \quad (4)$$

$$(\{PC\}, \{UE, \overline{ST}, \overline{TA}\}, PT) \in can\_assign \quad (5)$$

Notice how the original role UM *is replaced with its senior* UE *(e.g.,* $(UE \succeq UM)$*) in action (5).*

The user-role reachability problem $\langle (UA_0, \succeq), \psi, (Eve, \{ST, UM\}) \rangle$ *will be transformed to 4 user-role reachability problems without role hierarchy:*

$$\langle (UA_0), \psi, (Eve, \{ST, UM\}) \rangle \quad (6)$$

$$\langle (UA_0), \psi, (Eve, \{TA, UM\}) \rangle \quad (7)$$

$$\langle (UA_0), \psi, (Eve, \{ST, UE\}) \rangle \quad (8)$$

$$\langle (UA_0), \psi, (Eve, \{TA, UE\}) \rangle \quad (9)$$

because of tuples $(TA \succeq ST)$ *and* $(UE \succeq UM)$ *in role hierarchy* $\succeq$ □

B. Linear Reduction $\mathbb{R}_\mathbb{L}$

It is easy to see that the reduction $\mathbb{R}_\mathbb{A}$, in the worst case, results in an exponential number of additional administrative actions in ψ (cf. **Step 2** of $\mathbb{R}_\mathbb{A}$). As shown in [13], the number of administrative actions is the main source of complexity in solving the user-role reachability problem. Moreover, solving a single original user-role reachability problem (with hierarchy) now becomes to solving a set of new user-role reachability problems (user-role reachability analysis for ARBAC policy is PSPACE-complete [6]). The crucial observation to avoid this is the following: if a user is assigned to a role r_1 and $(r_1 \succeq r_2) \in \succeq$, we can assume that the user is also assigned to role r_2. In this case, we say that the user is implicitly assigned to role r_2. This suggests to transform each tuple in the role hierarchy $\succeq$ to a new administrative action of type can_assign_hier (similar to those of type can_assign) such that when a user is assigned to a role r, he can be "implicitly" assigned to any junior role of r by executing the new actions. As we need only one additional action per tuple in role hierarchy $\succeq$, it is easy to see that the number of such actions is linear in the cardinality of $\succeq$.

The effect of explicit and implicit role memberships must be handled carefully. In fact, if a user u is assigned to a role r by a can_assign action (explicit role membership), u then can be implicitly assigned to any junior role of r by executing the can_assign_hier actions mentioned above (implicit role membership). Now, if u is revoked from r by executing a can_revoke action, then the role membership must be handled such that also all the junior roles of r that have been implicitly assigned to u must be revoked. Intuitively, to do this, we need

to keep track of all the junior roles implicitly assigned to every user that is a computationally heavy task. To avoid this problem, we modify the structure of RBAC policies defined in Section II by adding a new relation $UAH \subseteq U \times R$. Now, the relation UA is required to record only explicit role memberships (i.e., those resulting by executing can_assign actions) while the new relation UAH record both the explicit and implicit ones.

Before describing the reduction $\mathbb{R}_{\mathbb{L}}$, we introduce the new administrative action of type can_assign_hier as follows.

Definition 5 *An administrative action of type* can_assign_hier *is of the form* $(r_s \diamond r_j)$ *where* r_s *and* r_j *are roles in* R

Moreover, to handle the effect of explicit and implicit role memberships, we need to modify the relation $\rightarrow_\psi$ defined in Definition 2 in Section II-B as in the following. We note that a state of ARBAC transition system is modified by adding the new relation UAH and removing relation $\succeq$ (since all tuples in $\succeq$ are transformed to can_assign_hier actions):

Definition 6 $(UA, UAH) \rightarrow_\psi (UA', UAH')$ *iff there exist users* u_a *and* u *in* U *such that either:*

- *there exists* $(C_a, C, r) \in$ *can_assign,* u_a *satisfies* C_a*,* u *satisfies* C *(i.e.* (C_a, C, r) *is enabled),* $UA' = UA \cup \{(u, r)\}$*, and* $UAH' = UAH \cup \{(u, r)\}$ *or*
- *there exists* $(C_a, r) \in$ *can_revoke,* u_a *satisfies* C_a *(i.e.* (C_a, r) *is enabled),* $UA' = UA \setminus \{(u, r)\}$*, and* $UAH' = UA'$ *or*
- *there exists* $(r_s, r_j) \in$ *can_assign_hier,* u *satisfies* $\{r_s\}$*, and* $UAH' = UAH \cup \{(u, r_j)\}$.

We note that the satisfiability of a user to a pre-condition is now with respect to relation UAH instead of UA as in Section II-B. We also emphasize that can_assign actions update both UA and UAH while can_assign_hier ones update only UAH. Additionally, can_assign_hier actions do not require an administrator to be executed, it only requires to check that there exists a user u who is member of senior role r_s to add the tuple (u, r_j) to UAH. An action of type can_revoke removes a tuple from UA and then sets UAH to the updated value (i.e., after the removal of the tuple) of UA. The need of resetting UAH to UA after removing a tuple arises from the observation that removing an explicit role membership invalidates all the implicit ones in UAH related to it.

We are ready to define the linear reduction $\mathbb{R}_{\mathbb{L}}$ as follows:

Step 1 Processing negative roles in pre-conditions:
- For each tuple $(C_a, C, r) \in can\_assign$:
  - for each negative role $\overline{r}$ occurring in C_a: replace $\overline{r}$ with $\overline{Senior(r)}$ with respect to role hierarchy $\succeq$.
  - for each negative role $\overline{r}$ occurring in C: replace $\overline{r}$ with $\overline{Senior(r)}$ with respect to role hierarchy $\succeq$.
- For each tuple $(C_a, r) \in can\_revoke$:
  - for each negative role $\overline{r}$ occurring in C_a: replace $\overline{r}$ with $\overline{Senior(r)}$ with respect to role hierarchy $\succeq$

Step 2 Processing tuples in role hierarchy $\succeq$:
- For each tuple $(r_s \succeq r_j) \in \succeq$:
  - add a can_assign_hier action $(r_s \diamond r_j)$ to the set of administrative actions ψ

As argued above, the number of additional actions resulting by applying $\mathbb{R}_{\mathbb{L}}$ is linear to the number of tuples in role hierarchy $\succeq$. Moreover, $\mathbb{R}_{\mathbb{L}}$ does not need to refine the goal in the original user-role reachability problem as in the reduction $\mathbb{R}_{\mathbb{A}}$ (cf. **Step 3** in $\mathbb{R}_{\mathbb{A}}$) and thus, avoid solving a set of (refined) reachability problems.

Example 4 *Consider the ATRBAC system in Example 3 and* $\succeq = \{(TA \succeq ST), (PC \succeq FA), (PC \succeq PT), (UE \succeq UM)\}$.

After using ***Step 1*** *to process negative roles in administrative actions as in Example 3, the reduction* $\mathbb{R}_{\mathbb{L}}$ *adds to the set* ψ *the following* can_assign_hier *actions:*

$$(TA \diamond ST), \tag{10}$$
$$(PC \diamond FA), \tag{11}$$
$$(PC \diamond PT), \tag{12}$$
$$(UE \diamond UM), \tag{13}$$

□

C. Composite Reduction $\mathbb{R}_{\mathbb{C}}$

The reduction $\mathbb{R}_{\mathbb{L}}$ reduces significantly the number of actions added to the set ψ to simulate the role hierarchy $\succeq$. However, the sequences of can_assign_hier actions used to obtain junior roles may negatively affect the performances of analysis tools. In fact, if the depth of role hierarchy $\succeq$, i.e., the longest chain of the form $(r_1 \succeq r_2)$, $(r_2 \succeq r_3)$, ..., $(r_n \succeq r_{n+1})$ for $(r_i \succeq r_{i+1}) \in \succeq$ with $i = 1; ...; n$, is large, we need to execute a long sequence of can_assign_hier actions $(r_1 \diamond r_2)$, $(r_2 \diamond r_3)$, ..., $(r_n \diamond r_{n+1})$ to implicitly assign the user to the most junior role. To avoid this problem, we exploit a feature of the analysis technique underlying the tool ASASPXL to "combine" the affect of the long sequence of can_assign_hier actions into an atomic administrative action. The main idea is to design a new version of action type can_assign_hier, namely $m\_can\_assign\_hier$, such that a user can be assigned to all the junior roles of a given role in one shot.

Definition 7 *An administrative action of type* m_can_assign_hier *is of the form* $(r_s \diamond C)$ *where* r_s *is a role in* R *and* $C \subseteq R$

The relation $\rightarrow_\psi$ is similar to one defined in Definition 6 except the semantic of can_assign_hier actions is replaced by that of $m\_can\_assign\_hier$ as follows.

- there exists $(r_s, C) \in m\_can\_assign\_hier$, u satisfies $\{r_s\}$, and $UAH' = UAH \cup \{(u, r_j) | r_j \in C\}$.

We are now ready to define the composite reduction $\mathbb{R}_{\mathbb{C}}$. In the following, we use $Junior(r) = \{r' \mid r \text{ is more senior than } r'\}$ to denote the set of all junior roles of a given role r:

Step 1 Processing negative roles in pre-conditions:
- For each tuple $(C_a, C, r) \in can\_assign$:
  - for each negative role $\overline{r}$ occurring in C_a: replace $\overline{r}$ with $\overline{Senior(r)}$ with respect to role hierarchy $\succeq$.
  - for each negative role $\overline{r}$ occurring in C: replace $\overline{r}$ with $\overline{Senior(r)}$ with respect to role hierarchy $\succeq$.
- For each tuple $(C_a, r) \in can\_revoke$:
  - for each negative role $\overline{r}$ occurring in C_a: replace $\overline{r}$ with $\overline{Senior(r)}$ with respect to role hierarchy $\succeq$

Step 2 Processing tuples in role hierarchy $\succeq$:
- For each role $r \in R$:
  - If $Junior(r) \neq \emptyset$ then add a $i\_can\_assign\_hier$ action $(r \diamond Junior(r))$ to the set of administrative actions ψ

Example 5 *Consider again the ATRBAC system in Example 3 and* $\succeq = \{(TA \succeq ST), (PC \succeq FA), (PC \succeq PT), (UE \succeq UM)\}$.

The reduction $\mathbb{R}_{\mathbb{C}}$ *adds to the set* ψ *the following* $m\_can\_assign\_hier$ *actions:*

$$(TA \diamond \{ST\}), \tag{14}$$

$$(PC \diamond \{FA, PT\}), \tag{15}$$

$$(UE \diamond \{UM\}), \tag{16}$$

Now, a user assigned to role PC can be implicitly assigned to all junior roles of PC by executing m_can_assign_hier *action (15)* □

V. Discussion

In Section IV, we introduce three reductions to enable available analysis technique to handle the user-role reachability problem with the presence of role hierarchy. The main idea is to transform an ARBAC system with role hierarchy to an equivalent one without role hierarchy and then use available analysis techniques to analyze the system. While proposed reductions is only capable of transforming static role hierarchy, i.e., role hierarchy $\succeq$ is assumed to be constant during the analysis, that is usually happening in real scenario[1], some specific applications may require some modification to the role hierarchy that is also supported by ARBAC framework in [3]. In such applications, role hierarchy $\succeq$ can be modified by administrative actions that add or delete some tuples to/from $\succeq$ (now we also call $\succeq$ as a dynamic role hierarchy). As a result, the analysis techniques need to consider not only the administrative actions modifying the relation UA but also actions modifying the role hierarchy $\succeq$.

In fact, we can use the idea of simulating the effect of role hierarchy in the reductions $\mathbb{R}_{\mathbb{C}}$ and $\mathbb{R}_{\mathbb{L}}$, e.g., using can_assign_hier and $m\_can\_assign\_hier$ actions, to design an approach to solve the user-role reachability problem in the context of dynamic hierarchy as follows: First, we represent all possible role relationships $(r_1 \succeq r_2)$ by using a set of can_assign_hier actions as in $\mathbb{R}_{\mathbb{L}}$. Then, the effect of administrative actions that modify the role hierarchy can be simulated by enabling or disabling the execution of corresponding can_assign_hier actions. For instance, adding a tuple $(r_1 \succeq r_2)$ to role hierarchy will make the can_assign_hier action $(r_1 \diamond r_2)$ enabled and vice versa. Because the role hierarchy now is represented by a set of can_assign_hier actions, it can be abstracted away and thus, it is possible to reuse available analysis techniques such as ASASPXL to solve the user-role reachability problem in the context of dynamic hierarchy. The details of this approach and its implementation is left as our future work.

VI. Implementation and Experiments

We implement three reductions $\mathbb{R}_{\mathbb{A}}$, $\mathbb{R}_{\mathbb{L}}$, and $\mathbb{R}_{\mathbb{C}}$ on top of the analysis tool ASASPXL using Python. Basically, we build a module named **Pre-processing** module containing three submodules $\mathbb{R}_{\mathbb{A}}$, $\mathbb{R}_{\mathbb{L}}$, and $\mathbb{R}_{\mathbb{C}}$ and then put it in front of the module **Heuristics** in the structure of ASASPXL depicted in Figure 2. Now, ASASPXL takes as input an instance of the user-role reachability problem with role hierarchy $\succeq$ and, depending on the reduction used, it forwards the problem to $\mathbb{R}_{\mathbb{A}}$, $\mathbb{R}_{\mathbb{L}}$, or $\mathbb{R}_{\mathbb{C}}$ accordingly to process the role hierarchy. Then, ASASPXL solves the problem by using modules **Heuristics**, **Translator**, **MCMT**, and **Refinement** as described in Section III.

To evaluate the scalability of the three reductions, we generate synthetic benchmark as follows: we use the ARBAC user-role reachability problems from [8] and add randomly generated role hierarchies organized—as suggested in [15]—as lattices with a senior-most and a junior-most role. The benchmark contains policies inspired by a university whose number of roles is 40, the number of administrative actions from 217 to 492, and the cardinality $|\succeq|$ of role hierarchy from 10 to 300. All the experiments were performed on an Intel Core I5 (2.6 GHz) CPU with 4 GB Ram running Ubuntu 11.10.

Table I reports the experimental results of running three reductions $\mathbb{R}_{\mathbb{A}}$, $\mathbb{R}_{\mathbb{L}}$, and $\mathbb{R}_{\mathbb{C}}$ on the benchmark. Column 1 shows the name of the test case, column 2 contains the number of roles, administrative operations, and $|\succeq|$ in the policies. Columns 3, 4, and 5 show the average execution times (in seconds) taken by ASASPXL with using the three reductions $\mathbb{R}_{\mathbb{A}}$, $\mathbb{R}_{\mathbb{L}}$, and $\mathbb{R}_{\mathbb{C}}$, respectively.

The results show the superiority of the last reduction $\mathbb{R}_{\mathbb{C}}$ over the other two. It is also clear that $\mathbb{R}_{\mathbb{A}}$ is the less scalable of the three. The reason comes from the fact that number of additional administrative actions generated by $\mathbb{R}_{\mathbb{L}}$ and $\mathbb{R}_{\mathbb{C}}$

[1]This is because role hierarchy closely reflects the structure of the organizations in which the policies are used, thus, the modifications to role hierarchy should be rare as they imply substantial changes to the organizations themselves.

TABLE I
EXPERIMENTAL RESULTS ON THE BENCHMARK IN [9]

Test case	**# Roles ◇ # Rules ◇ # $\mid \succeq \mid$**	$\mathbb{R}_{\mathbb{A}}$	$\mathbb{R}_{\mathbb{L}}$	$\mathbb{R}_{\mathbb{C}}$
Test 1	40 ◇ 287 ◇ 10	1.25	0.93	**0.28**
Test 2	40 ◇ 217 ◇ 20	2.91	1.17	**0.82**
Test 3	40 ◇ 262 ◇ 50	3.73	1.14	**1.03**
Test 4	40 ◇ 296 ◇ 50	2.51	1.12	**0.97**
Test 5	40 ◇ 480 ◇ 100	8.31	4.27	**1.97**
Test 6	40 ◇ 479 ◇ 150	22.56	11.32	**5.09**
Test 7	40 ◇ 467 ◇ 150	47.22	10.92	**6.12**
Test 8	40 ◇ 484 ◇ 200	51.27	13.65	**5.27**
Test 9	40 ◇ 463 ◇ 250	45.15	19.92	**6.91**
Test 10	40 ◇ 492 ◇ 300	39.92	17.21	**10.62**

is reduced significantly in comparison with that of $\mathbb{R}_{\mathbb{A}}$ as discussed in Section IV-B.

Fig. 3. Performance when varying $\mid \succeq \mid$

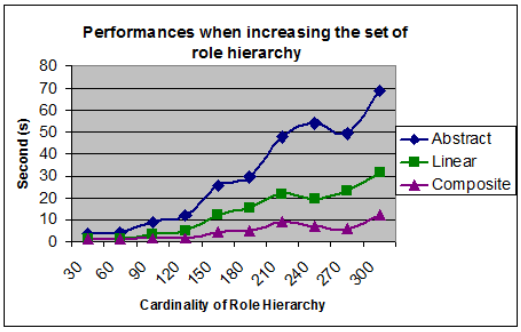


Figure 3 shows behaviour of the three reductions when the cardinality $\mid \succeq \mid$ of temporal role hierarchies increases. To do this experiment, we picked one test case from the benchmark and randomly add hierarchies (with the increase in $\mid \succeq \mid$) to the test case. $\mid \succeq \mid$ varies from 30 to 300 and for each value of it, we generated 15 different role hierarchies and added to the test case. The average time (in seconds) for solving the problems is reported and shown in Figure 3. Blue diamond line reports the behaviour of the abstract reduction $\mathbb{R}_{\mathbb{A}}$; green squares line shows the performance of the linear reduction $\mathbb{R}_{\mathbb{L}}$, and violet triangles line is the performance of the composite reduction $\mathbb{R}_{\mathbb{C}}$. Clearly, $\mathbb{R}_{\mathbb{C}}$ performance is much better than the other two $\mathbb{R}_{\mathbb{L}}$ and $\mathbb{R}_{\mathbb{A}}$.

We emphasize that it is not possible to compare ASASPXL (with the three reductions) with the other state-of-the-art analysis tool since they do not support specific features that we need to build the reductions $\mathbb{R}_{\mathbb{L}}$ and $\mathbb{R}_{\mathbb{C}}$ as discussed in the beginning of Section IV-B and IV-C.

VII. CONCLUSIONS

We have introduced three reductions $\mathbb{R}_{\mathbb{A}}$, $\mathbb{R}_{\mathbb{L}}$, and $\mathbb{R}_{\mathbb{C}}$ that pre-process away role hierarchies so that (an adapted version of) the technique in ASASPXL can be used to solve user-role reachability problem in ARBAC. The experimental results show that the performance of $\mathbb{R}_{\mathbb{C}}$ is better than that of $\mathbb{R}_{\mathbb{L}}$ and much better than $\mathbb{R}_{\mathbb{A}}$. This is because the number of additional administrative actions is reduced significantly by using $\mathbb{R}_{\mathbb{C}}$ and $\mathbb{R}_{\mathbb{L}}$ that allows for the generation of problems with smaller state spaces.

As future work, we plan to study how to extend our approach to cope with the dynamic role hierarchies. In fact, we can use the idea of simulating role hierarchy in the reductions $\mathbb{R}_{\mathbb{C}}$ and $\mathbb{R}_{\mathbb{L}}$, i.e., using can_assign_hier and $m\_can\_assign\_hier$ actions to simulate the affect of role hierarchy, to design an approach in the context of dynamic hierarchy as discussed in Section V.

REFERENCES

[1] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, and P. Samarati, "Access control policies and languages," *Int. Journal of Computational Science and Engineering (IJCSE)*, vol. 3, no. 2, pp. 94–102, 2007.

[2] R. Sandhu, E. Coyne, H. Feinstein, and C. Youmann, "Role-Based Access Control Models," *IEEE Computer*, vol. 2, no. 29, pp. 38–47, 1996.

[3] J. Crampton, "Understanding and developing role-based administrative models," in *Proc. 12th CCS, pages 158–167, ACM Press, 2005*.

[4] N. Li and M. V. Tripunitara, "Security analysis in role-based access control," *ACM TISSEC*, vol. 9, no. 4, pp. 391–420, 2006.

[5] S. Jha, N. Li, M. V. Tripunitara, Q. Wang, and H. Winsborough, "Towards formal verification of role-based access control policies," *IEEE TDSC*, vol. 5, no. 4, pp. 242–255, 2008.

[6] S. D. Stoller, P. Yang, C. Ramakrishnan, and M. I. Gofman, "Efficient policy analysis for administrative role based access control," in *CCS*. ACM Press, 2007.

[7] F. Alberti, A. Armando, and S. Ranise, "ASASP: Automated Symbolic Analysis of Security Policies," in *CADE*, ser. LNCS, vol. 6803. Springer, 2011, pp. 26–34.

[8] A. L. Ferrara, P. Madhusudan, T. L. Nguyen, and G. Parlato, "VAC - Verifier of Administrative Role-based Access Control Policies," in *Proc. of 26th Int'l Conference on Computer Aided Verification (CAV)*. Springer Berlin Heidelberg, 2014, pp. 184–191.

[9] P. Yang, M. Gofman, S. Stoller, and Z. Yang, "Policy Analysis for Administrative Role Based Access Control without Separate Administration," *J. of Computer & Security*, 2014.

[10] A. Sasturkar, P. Yang, S. D. Stoller, and C. Ramakrishnan, "Policy analysis for administrative role based access control," in *CSF*. IEEE Press, Jul. 2006.

[11] S. Ghilardi and S. Ranise, "Backward Reachability of Array-based Systems by SMT solving: Termination and Invariant Synthesis," *In LMCS, Vol. 6, Issue 4*, 2010.

[12] "`http://research.microsoft.com/en-us/um/redmond/projects/z3`."

[13] S. Ranise, T. A. Truong, and A. Armando, "Boosting Model Checking to Analyse Large ARBAC Policies," in *STM'12*, ser. LNCS, vol. 7783, 2012, pp. 273–288.

[14] K. Jayaraman, V. Ganesh, M. Tripunitara, M. Rinard, and S. Chapin, "Automatic Error Finding for Access-Control Policies," in *CCS*. ACM, 2011.

[15] N. Li and Z. Mao, "Administration in Role Based Access Control," in *Proc. ACM Symp. on Information, Computer, and Communication Security (ASIACCS)*, 2007.