COMPUTERS & SECURITY XXX (2010) 1–17

available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Symbolic reachability analysis for parameterized administrative role-based access control☆

Scott D. Stoller [a,*], Ping Yang [b], Mikhail I. Gofman [b], C.R. Ramakrishnan [a]

[a] Dept. of Computer Science, Stony Brook University, Stony Brook, NY 11794-4400, USA
[b] Dept. of Computer Science, Binghamton University, Binghamton, NY 13902, USA

ARTICLE INFO

Article history:
Received 21 April 2010
Received in revised form
1 July 2010
Accepted 25 August 2010

Keywords:
Policy analysis
Policy administration
Role-based access control
RBAC

ABSTRACT

Role-based access control (RBAC) is a widely used access control paradigm. In large organizations, the RBAC policy is managed by multiple administrators. An administrative role-based access control (ARBAC) policy specifies how each administrator may change the RBAC policy. It is often difficult to fully understand the effect of an ARBAC policy by simple inspection, because sequences of changes by different administrators may interact in unexpected ways. ARBAC policy analysis algorithms can help by answering questions, such as user-role reachability, which asks whether a given user can be assigned to given roles by given administrators.

Allowing roles and permissions to have parameters significantly enhances the scalability, flexibility, and expressiveness of ARBAC policies. This paper defines PARBAC, which extends the classic ARBAC97 model to support parameters, proves that user-role reachability analysis for PARBAC is undecidable when parameters may range over infinite types, and presents a semi-decision procedure for reachability analysis of PARBAC. To the best of our knowledge, this is the first analysis algorithm specifically for parameterized ARBAC policies. We evaluate its efficiency by analyzing its parameterized complexity and benchmarking it on case studies and synthetic policies. We also experimentally evaluate the effectiveness of several optimizations.

1. Introduction

Role-based access control (RBAC) (Sandhu et al., 1996) is a widely used access control paradigm. In RBAC, users are assigned to roles, and permissions are granted to roles. Allowing roles and permissions to have parameters significantly enhances scalability: the policies of most large organizations can be expressed more easily and compactly using parameters. For example, consider a policy for a university. To grant different permissions to users (e.g., faculty or students) in different classes or departments, in an RBAC model without parameters, we would need to create a separate role and corresponding permission assignment rules for each course or department, leading to a large and unwieldy policy. In a parameterized RBAC model, this policy can be expressed using a few roles and permissions parameterized by the class identifier or department name. Several parameterized RBAC models have been proposed (e.g., Giuri and Iglio, 1997; Lupu and Sloman, 1997; Bacon et al., 2002; Ge and Osborn, 2004; Li and Mao, 2007).

Administrative role-based access control (ARBAC) refers to administrative policies that specify how an RBAC policy may

☆ This work was supported in part by ONR under Grants N00014-07-1-0928 and N00014-09-1-0651, NSF under Grants CCF-0613913, CNS-0627447, CNS-0831298, and CNS-0855204, and AFOSR under Grant FA0550-09-1-0481.

* *Corresponding author.*
E-mail address: stoller@cs.stonybrook.edu (S.D. Stoller).

0167-4048/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2010.08.002

be changed by each administrator. In ARBAC97, the first comprehensive ARBAC model (Sandhu et al., 1999), ARBAC policies assign users (administrators) to administrative roles, and grant permissions for administrative operations—such as assigning a user to a role—to administrative roles. This supports decentralized policy administration, which is crucial for large organizations, coalitions, etc. Several other ARBAC models were subsequently proposed (e.g., Sandhu and Munawer, 1999; Kern et al., 2003; Crampton and Loizou, 2003; Crampton, 2005; Oh et al., 2006; Li and Mao, 2007).

Allowing administrative roles and administrative permissions to have parameters significantly enhances the scalability and practical applicability of the administrative model. For example, consider the policy that the chair of a department can assign users to committees in that department. In a parameterized ARBAC model, this can be expressed by a single rule, while ARBAC models without parameters would require separate rules for each department and committee. In this paper, we define parameterized RBAC and ARBAC models, by extending the classic ARBAC97 model (Sandhu et al., 1999) with parameters in a fairly straightforward way. We call these models PRBAC and PARBAC, respectively.

While flexible and expressive administrative models are needed to handle the complex policies that can arise in real organizations, they also make it more difficult to ensure that administrative policies accurately capture the author's intentions. It is often difficult to understand the effect of an administrative policy by simple inspection, largely because (without help) people may fail to see the possible effects of sequences of administrative operations by different administrators, and may fail to take into account how the administrative rules interact with role hierarchy. Policy analysis helps system designers and administrators to understand policies, including administrative policies.

This paper focuses on user-role reachability analysis, which answers questions of the form: given an initial PRBAC policy ("state"), a PARBAC policy, a set of administrators, a target user, and a set of roles (called the "goal"), is it possible for those administrators to modify the RBAC policy so that the target user is a member of those roles? Other analysis problems including permission-role reachability, user-permission reachability, availability, role containment (Li and Tripunitara, 2006), and weakest precondition (Stoller et al., 2007) can be solved in a similar manner or by reduction to user-role reachability analysis (Sasturkar et al., 2006; Stoller et al., 2007).

Why are new algorithms needed to solve this problem? If all parameters range over finite types, existing finite-state reachability algorithms for unparameterized ARBAC (e.g., Li and Tripunitara, 2006; Sasturkar et al., 2006; Stoller et al., 2007; Jha et al., 2008) can be used, by instantiating each rule with all combinations of values of its parameters. However, this approach is practical only if the types are small. Realistic policies often involve large types (e.g., Stony Brook University has over 2000 class sections each semester and over 50 departments); symbolic analysis of such policies is much more efficient. Another disadvantage of the finite-state approach is that the analysis results are valid only for the specific types used for instantiation. With symbolic analysis, an infinite type can be used as an abstraction of a finite type, to obtain more general results. This abstraction is conservative in the sense that if the answer to a reachability query (defined in Section 3) is true when the types of some parameters are taken to be finite, then the answer is still true if those types are taken to be infinite. Thus, if symbolic analysis with infinite types says that a goal (of the attackers, i.e., an unsafe state) is unreachable, then that goal is unreachable when the parameters range over any finite types.

This paper shows that user-role reachability analysis for PARBAC is undecidable when parameters may range over infinite types, and it presents the first (to the best of our knowledge) semi-decision procedure for reachability analysis for PARBAC. We define the semantics of PARBAC policies in terms of a straightforward *concrete transition relation*. We then introduce a more complicated *symbolic transition relation* that captures the semantics compactly, efficiently, and exactly using variables and constraints. Our algorithm for user-role reachability has two stages. The first stage performs a goal-directed approximate backward search. The second stage performs an exact forward search limited to transitions identified as useful by the first stage. We also developed several optimizations to the basic algorithm and experimentally evaluated their effectiveness. Although our algorithm is a semi-decision procedure (thus, it may diverge on some problem instances), we show that it is guaranteed to terminate under realistic assumptions about the policy.

We also explore the parameterized complexity (Downey and Fellows, 1995) of user-role reachability for PARBAC and give a fixed-parameter tractability result for it under realistic assumptions about the policies. The idea of parameterized complexity is to identify an aspect of the input that makes the problem computationally difficult, introduce a parameter to measure that aspect of the input, and develop a solution algorithm that may have high complexity in terms of that parameter, but has polynomial complexity in terms of the overall input size when the value of that parameter is fixed. This is called fixed-parameter tractability. Formally, a problem is *fixed-parameter tractable* with respect to parameter k if there exists an algorithm that solves it in $O(f(k) \times n^c)$ time, where f is an arbitrary function (depending only on its argument k), n is the input size, and c is a constant.

Numerous algorithms have been proposed to verify specific classes of infinite-state systems. As discussed in Section 11, to the best of our knowledge, none is suitable for efficient reachability analysis for PARBAC.

In summary, the main contributions of this paper are

- The definition of the symbolic transition graph, which compactly captures the semantics of PARBAC policies and provides the basis for our algorithm,
- A two-stage symbolic algorithm for user-role reachability analysis of PARBAC that terminates under realistic assumptions about the policy,
- A proof that user-role reachability analysis for PARBAC is undecidable.
- A fixed-parameter tractability result for this reachability problem under realistic assumptions about the policy, and
- Experimental results demonstrating the efficiency of our algorithm compared to non-symbolic algorithms and evaluating the effectiveness of several optimizations

Please cite this article in press as: Stoller SD, et al., Symbolic reachability analysis for parameterized administrative role-based access control, Computers & Security (2010), doi:10.1016/j.cose.2010.08.002

We chose ARBAC97 as the basis for our PARBAC model because it is relatively simple while still capturing essential features of realistic administrative policies. We know of only one other parameterized ARBAC model, UARBAC^P (Li and Mao, 2007). UARBAC^P is more sophisticated and flexible than PARBAC, but we believe the work in this paper provides a good foundation for developing practical analysis algorithms for UARBAC^P and other parameterized security policy models.

The rest of this paper is organized as follows. Section 2 defines PRBAC and PARBAC. Section 3 defines user-role reachability for PARBAC. Section 4 defines the symbolic state graph, which provides a foundation for the symbolic analysis algorithm in Section 5. Section 8 discusses extensions and other analysis problems. Case studies and experiments are described in Sections 9 and 10, respectively. Section 11 discusses related work. Section 12 proposes future work and concludes.

2. Parameterized RBAC and parameterized ARBAC

This section formally defines parameterized RBAC (PRBAC) and parameterized ARBAC (PARBAC). The definitions are based on a notion of *role schema*. Each role schema specifies the name of a role and the names of that role's parameters. For brevity, we omit aspects of RBAC and ARBAC related to the user-permission assignment and role hierarchy. Those aspects can be extended with parameters in the same way as aspects related to the user-role assignment. For analysis purposes, hierarchical PRBAC policies can be transformed into non-hierarchical PRBAC policies using an algorithm similar to the one in Sasturkar et al. (2006). PARBAC policies that control the user-permission assignment are structurally similar to PARBAC policies that control the user-role assignment and hence can be analyzed using the same techniques. Analysis of PARBAC policies that control changes to the role hierarchy requires different techniques.

2.1. Parameterized RBAC

The syntax of policies is parameterized by a set *Var* of *variables*, a set $\mathcal{R}$ of *role names*, a set $\mathcal{O}$ of *object names*, a set $\mathcal{P}$ of *parameter names*, and a set *Op* of operation names.

A role schema is a term $\rho(p_1, p_2, \ldots, p_n)$, where $n \geq 0$, $\rho \in \mathcal{R}$ is a role name, and each $p_i \in \mathcal{P}$ is a distinct parameter name. In our basic framework, each parameter ranges over an implicit universal data type that contains an infinite number of data values (constants). Introducing a type system in which each parameter in a role schema ranges over a specified infinite data type has no significant effect on our results, except to add clutter. Allowing finite types requires only a change to the algorithm for checking satisfiability of constraints, as described in Section 4. We implicitly extend our framework with a type system with finite types in some examples.

An *instance* of a role schema $\rho(p_1, p_2, \ldots, p_n)$ has the form $\rho(p_1 = x_1, p_2 = x_2, \ldots, p_n = x_n)$, where each x_i is a data value or a variable. We use identifiers starting with lower-case letters for data values, and identifiers starting with upper-case letters for variables (identifiers starting with upper-case letters are also used for role names, etc.). An instance is *concrete* if it contains no variables. We use r to denote an instance of a role schema, and r_c to denote a concrete instance.

For an instance r, let $schema(r)$ denote the schema of which r is an instance. Let $args(\rho(e_1, \ldots, e_n)) = (e_1, \ldots, e_n)$. For a set RS of role schemas, $inst(RS)$ denotes the set of all instances of RS, and $conc(RS)$ denotes the set of concrete instances of RS. For example, in a policy for a university, the role schema *Student* (*dept*, *cid*) is used for students registered for the course numbered *cid* offered by department *dept*, and the role schema *Student*(*dept*) is used for all students of a specific department. Students taking cs101 are members of the instance *Student* ($dept = cs$, $cid = 101$). We make parameter names explicit to allow overloading; we sometimes omit them for role names that are not overloaded.

A substitution is a mapping from variables to data values and variables. We use θ, σ to denote substitutions. A substitution θ is ground, denoted $ground(\theta)$, if it maps all variables to data values. The application of a substitution θ to an expression e is denoted $e\theta$.

Definition 1. A *parameterized RBAC* (*PRBAC*) *policy* is a tuple $\langle RS, U, UA \rangle$ where

- RS is a finite set of role schemas. U is a finite set of users.
- $UA \subseteq U \times conc(RS)$ is the user-role assignment. $(u, r_c) \in UA$ specifies that user u is a member of r_c.

For example, $(Alan, Student(dept = cs)) \in UA$ specifies that user *Alan* is a member of role $Student(dept = cs)$.

PRBAC policies, as defined above, use only concrete role instances. We could allow variables in PRBAC policies; we do not consider this extension, because our main focus is on administrative policies, defined next.

2.2. Parameterized ARBAC

A PARBAC policy is a tuple $\langle RS, U, URA \rangle$, where RS is a set of role schemas, U is a set of users, and—analogously to ARBAC97—URA is the user-role administration policy. The PARBAC policy defines the transition relation that describes allowed changes to the PRBAC policy.

The user-role administration policy URA controls changes to the user-role assignment. URA consists of two kinds of rules: *can_assign* and *can_revoke*. A *can_assign* rule has the form $can\_assign(r_a, (P, N), r)$, where $r_a \in inst(RS)$ is the administrator's role, $P \subseteq inst(RS)$ is the *positive precondition*, $N \subseteq inst(RS)$ is the *negative precondition*, and $r \in inst(RS)$ is the *target*. The rule means that an administrator in role r_a can add a user to r if the user is a member of all the roles in P and is not a member of any roles in N. In examples, we usually write preconditions as logical formulas; for example, the precondition $(\{r_1, r_2\}, \{r_3\})$ would be written as $r_1 \wedge r_2 \wedge \neg r_3$. For example, the rule *can_assign* (*Dean*(*school* = *engg*), *Prof*(*dept* = *cs*), *Chair*(*dept* = *cs*)) specifies that the Dean of the Engineering School can assign a professor of the CS Department to be the Chair of that Department. For an example that uses variables, the rule *can_assign*(*Chair* (*dept* = *D*), *Faculty*(*dept* = *D*), *AdmissionsCommittee*(*dept* = *D*)) specifies that the Chair of department *D* can assign faculty of that department to the department's admissions committee.

Please cite this article in press as: Stoller SD, et al., Symbolic reachability analysis for parameterized administrative role-based access control, Computers & Security (2010), doi:10.1016/j.cose.2010.08.002

The identity of the administrator performing an action is sometimes relevant, so we introduce a distinguished variable, *Self*, whose value identifies that administrator. For example, *can_assign*(*Faculty*, *Student*, RA(*fac* = *Self*)) specifies that a faculty member can assign a student to be his/her RA.

In negative preconditions, wildcards, denoted by underscore ("_"), may be used as arguments to roles. For example, consider the policy: a department chair can appoint a student as a TA for a course in the same department if the student is not already a TA for any course in any department. This is expressed by the following rule, where the parameter *cid* contains the course number:

$$can\_assign(Chair(dept = D), \neg TA(dept = \_, cid = \_), TA(dept = D, cid = CID))$$

Note that replacing wildcards with fresh variables—for example, changing the negative precondition to $\neg TA(dept = D', cid = CID')$—produces a rule with a much different meaning, namely, that a student can be appointed to $TA(dept = D, cid = CID)$ if there is some instantiation of D' and C' such that the student is not a member of $TA(dept = D', cid = CID')$.

A *can_revoke* rule has the form *can_revoke*(r_a, r). It means that an administrator in role r_a can remove users from role r. We follow ARBAC97 in omitting preconditions from *can_revoke* (Sandhu et al., 1999).

A role schema is an *administrative role schema* if it has an administrative permission, i.e., it appears in the first component of some *can_assign* or *can_revoke* rule. An *administrative role* is an instance of an administrative role schema. The *separate administration restriction* requires that administrative role schemas do not appear in the precondition or target of *can_assign* rules or the target of *can_revoke* rules. We follow ARBAC97 in adopting this restriction. Other work on ARBAC policy analysis, such as Schaad and Moffett (2002), Sasturkar et al. (2006), Li and Tripunitara (2006) also adopts this restriction (or a similar one, in the case of the AAR model in Li and Tripunitara, 2006), with the exception of analysis for the AATU model in Li and Tripunitara (2006), which adopts two other significant restrictions instead. Our analysis algorithm is also applicable to many policies that satisfy a different but related restriction, described in Section 8.

3. User-role reachability

This section defines user-role reachability for PARBAC. For a PRBAC policy γ, let $U(\gamma)$ and $UA(\gamma)$ be the set of users and the user-role assignment in γ, respectively.

Definition 2. A *user-role reachability* query has the form: Given a user u_0, an initial PRBAC policy $\gamma = \langle RS, U, UA\rangle$, a PARBAC user-role administration policy *URA*, a subset *A* of the user-role assignment *UA* containing only administrative roles, and a set g of role instances, can actions by administrators in *A*, acting in the administrative roles to which they are assigned in *A*, and using the administrative permissions granted to those roles by *URA*, transform γ to another PRBAC policy γ' such that, for some substitution θ, u_0 is a member of all roles in the instantiated goal $g\theta$?

Under the separate administration restriction, the user-role reachability problem can be simplified as in Sasturkar et al. (2006). This restriction implies that the transitions allowed by a PARBAC policy do not change the set of tuples containing administrative roles in the user-role assignment *UA*. Hence we can partition *UA* into administrative and non-administrative subsets, corresponding to tuples containing administrative roles and those containing non-administrative roles, respectively. Since the administrative subset does not change, we "factor it out", i.e., we do not include it in the nodes of the concrete state graph, defined below. Moreover, in ARBAC97, each user's role memberships are controlled completely independently of other users' role memberships, so we can perform user-role reachability analysis by tracking only tuples in *UA* that contain the user u_0 mentioned in the reachability query. Thus, the answer to a user-role reachability query can be expressed in terms of a graph whose vertices (states) correspond to sets of non-administrative roles that u_0 is a member of.

The *concrete transition relation* T_c(*URA*, *A*) expresses the semantics of a user-role administration policy *URA*, restricted to administrative actions performed by a user u_A in administrative role r_A such that $(u_A, r_A) \in A$. T_c(URA, A) contains $(s, (\varphi, \theta), s')$ if the rule φ in *URA*, instantiated using substitution θ, allows an administrator u_A acting in role r_A with $(u_A, r_A) \in A$ to perform a role assignment or role revocation that changes the user-role assignment for a user from s to s'. When *URA* and *A* are clear from context, we sometimes write a triple $(s, (\varphi, \theta), s') \in T_c(URA, A)$ as $s \xrightarrow{\varphi,\theta}_c s'$.

Definition 3. The *concrete transition relation* T_c(*URA*, *A*) for a user-role administration policy *URA* and a user-role assignment *A* containing only administrative roles is the smallest relation such that:

- $(s, (\varphi, \theta), s') \in T_c(URA, A)$ if $\varphi = can\_assign(r_a, (P, N), r)$ and $\varphi \in URA$ and θ is a ground substitution such that there exists $(u_A, r_A) \in A$ such that:
  - $r\theta \notin s$ (the role being added is not present in state s),
  - $s' = s \cup \{r\theta\}$,
  - $P\theta \subseteq s$ (the positive preconditions of φ are satisfied in state *s*),
  - $(\forall r_n \in N.\ \forall r \in s.\ r_n \neq_{wc} r)$, where equality considering wildcards is defined by: $r(e_1, \ldots, e_n) =_{wc} r'(e_1', \ldots, e_n')$ if $r = r' \wedge (\forall i \in [1..n].\ e_i = e_i' \vee e_i = \_ \vee e_i' = \_)$ (the negative preconditions of φ are satisfied in state *s*),
  - $r_a\theta = r_A$ (instantiating r_a yields the administrative role in *A* used to perform this role assignment), and
  - $\theta(Self) = u_A$ (θ maps the distinguished variable *Self* to the identity u_A of the administrator performing this role assignment)
- $(s, (\varphi, \theta), s') \in T_c(URA, A)$ if $\varphi = can\_revoke(r_a, r)$ and $\varphi \in URA$ and θ is a ground substitution such that there exists $(u_A, r_A) \in A$ such that:
  - $r\theta \in s$ (the role being revoked is present in state *s*),
  - $s' = s - \{r\theta\}$,
  - $r_a\theta = r_A$, and
  - $\theta(Self) = u_A$

The *concrete state graph* for a user-role reachability query of the form in Definition 2 is the graph created by starting from

the initial user-role assignment for the target user u_0 and using the concrete transition relation to repeatedly add new edges and nodes. For a labeled graph, we use a triple (v, ℓ, v') to represent an edge from v to v' labeled with ℓ.

Definition 4. *The concrete state graph* for a user-role reachability query of the form in Definition 2 is the smallest labeled directed graph (V, E) with vertices V and labeled edges E such that

- $\{r \mid (u_0, r) \in UA(\gamma) \wedge \neg admin(r)\} \in V$, where $admin(r)$ is true if r is an administrative role (the initial state contains all non-administrative roles assigned to the target user u_0 in the initial user-role assignment $UA(\gamma)$).
- $(s_1, \varphi, s_2) \in E$ and $s_2 \in V$ if $s_1 \in V$ and there exists a substitution θ such that $(s_1, (\varphi, \theta), s_2) \in T_c(URA, A)$.

The answer to a user-role reachability query is true if there exists a substitution θ such that the concrete state graph for the query contains a state s with $g\theta \subseteq s$.

Example 1. Consider the following PARBAC policy (for brevity, we do not show the set of users, etc.).

$$RS = \{Chair(dept), Student(dept, cid), TA(dept, cid)\}$$
$$\varphi = can\_assign(Chair(dept = D), \neg Student(dept = D, cid = CID), TA(dept = D, cid = CID))$$

The policy contains no *can_revoke* rules. Consider the query: Can the chair of CS Department assign a user who is initially a member of role $Student(dept = cs, cid = 501)$ to both roles $TA(dept = cs, cid = 101)$ and $TA(dept = cs, cid = 201)$? The answer is yes. For illustrative purposes, suppose the course identifier parameter cid ranges over the set {101, 201, 301, 401, 501}; in this case, the concrete state graph for this query contains 16 states and 32 transitions. If cid ranges over an infinite data type, the concrete state graph is infinite.

These definitions define the semantics of PARBAC policies but do not provide an effective algorithm for reachability analysis: parameters take values from an infinite type, so the concrete state graph is infinite, except for trivial policies. As discussed in Section 1, even if parameters take values from finite types, those types are often large, making construction of the concrete state graph costly or impractical.

4. Symbolic state graph

This section defines symbolic states and symbolic transitions, which are the basis of our symbolic analysis algorithm.

A *symbolic state* is a pair (R, C) where R is a set of role instances (not necessarily concrete), and C is a constraint over variables that appear in R. A *constraint* is the constant *true* or a conjunction of tuple disequalities. A *tuple disequality* has the form $(e_1, \ldots, e_n) \neq (f_1, \ldots, f_n)$, where each e_i and f_i is a constant or a variable. We elide angle brackets around singleton tuples. Note that a conjunction of tuple disequalities is just a more compact notation for a logical combination of single (as opposed to tuple) inequalities, in conjunctive normal form.

For a constraint C, $satisfiable(C)$ is false if C contains a tuple disequality whose left side and right side are the same, and is true otherwise. For example, if C_0 denotes $X \neq cs$, then $satisfiable(C_0)$ is true, and $satisfiable(C_0[X \mapsto cs])$ is false, where $[X \mapsto e]$ denotes the substitution that replaces X with e. As another example, if C_1 denotes $(X, Y) \neq (Z, cs)$ then $satisfiable(C_1)$ and $satisfiable(C_1[X \mapsto Z])$ are true. This satisfiability test is correct when all variables range over infinite data types. Support for finite data types is discussed below.

A symbolic state (R, C) represents the concrete states obtained by instantiating R consistent with C; formally, the meaning of (R, C) is $[\![(R, C)]\!] = \{R\theta \mid ground(\theta) \wedge satisfiable(C\theta)\}$. For example, $(\{Student(dept = D)\}, D \neq cs)$ represents states containing a single instance of *Student* instantiated with any constant other than *cs*.

For a constraint C, $simplify(C)$ returns a new constraint obtained by removing tuple disequalities in which the two tuples have distinct constants in some component (such disequalities are equivalent to *true*, e.g., $(X, cs) \neq (Y, ee)$) and removing components of tuple disequalities that are equal in the two tuples (this yields a logically equivalent disequality, e.g., $(X, Y) \neq (X, Z)$ is replaced with $Y \neq Z$). More formally, $simplify(C)$ is obtained from C as follows:

1. Delete all tuple disequalities $(e_1, \ldots, e_n) \neq (f_1, \ldots, f_n)$ such that for some i, e_i and f_i are distinct constants. If C becomes empty (i.e., all tuple disequalities in it are deleted), then C simplifies to *true*.
2. For each tuple disequality $(e_1, \ldots, e_n) \neq (f_1, \ldots, f_n)$ in C, for each i such that e_i is the same as f_i, or e_i or f_i is a wildcard, delete component i of the tuple disequality. If any tuple disequality becomes empty (i.e., it becomes $() \neq ()$, which is false), then C simplifies to *false*.

For a constraint C and a set $Vars$ of variables, the projection of C on $Vars$, denoted $project(C, Vars)$, is the constraint obtained from C by discarding disequalities that do not affect the satisfying values of variables in $Vars$. Specifically, $project(C, Vars)$ constructs an undirected graph with a vertex for each tuple disequality in C, and with an edge between disequalities d_1 and d_2 if they share a variable (i.e., $vars(d_1) \cap vars(d_2) \neq \emptyset$, where $vars(e)$ is the set of variables that appear in expression e), and discards disequalities that are not reachable in the graph from any vertex d that mentions a variable in $Vars$. For example, $project(X \neq Y \wedge Y \neq Z \wedge U \neq V, \{Z\})$ equals $X \neq Y \wedge Y \neq Z$.

A substitution θ_1 is *more general than* a substitution θ_2, denoted $\theta_2 \preceq_g \theta_1$, if there exists a substitution θ such that $\theta_2 = \theta_1 \circ \theta$, where $\circ$ denotes composition.

The symbolic transition relation needs to replace some variables with *locally fresh* variables, i.e., variables not appearing in the source state of the transition that introduces them. Let $freshSubst(vars_1, vars_2)$ denote a substitution θ that maps variables in $vars_1$ to distinct variables that are not in $vars_2$. Any deterministic method for choosing the fresh variables is fine, e.g., choose the lexicographically smallest variables not in $vars_2$. To simplify the semantics of the graph, after the initial construction, we apply a straightforward, linear-time transformation *mkGloballyFresh* that renames introduced variables so they are globally fresh, i.e., each variable is introduced in at most one state in the graph. For example,

Please cite this article in press as: Stoller SD, et al., Symbolic reachability analysis for parameterized administrative role-based access control, Computers & Security (2010), doi:10.1016/j.cose.2010.08.002

suppose a *can_assign* rule that introduces a locally fresh variable is executed from two states s_1 and s_2, and the variable C is used as a locally fresh variable in the resulting states s_1' and s_2'; then the *mkGloballyFresh* transformation might rename the occurrence of C in s_1' to C_1 (of course, corresponding occurrences of C in states reachable from s_1 are also renamed), and the occurrence of C in s_2' to C_2.

Definition 5. *The symbolic transition relation* $T(URA, A)$ for a user-role administration policy URA and an assignment A of users to administrative roles contains a tuple $((R, C), (\varphi, \theta_f, \theta), (R', C'))$ if execution of rule φ in URA, instantiated with the substitution $\theta \circ \theta_f$, leads from symbolic state (R, C) to symbolic state (R', C'), where θ_f replaces variables in φ with fresh variables, and θ unifies the positive preconditions in φ with roles in R and unifies r_a with an administrative role in A. Formally, $T(URA, A)$ is the least relation such that:

- $((R, C), (\varphi, \theta_f, \theta), (R', C')) \in T(URA, A)$ if $\varphi \in URA$ and $\varphi = can\_assign(r_a, (P, N), r_t)$ and there exist $R_p \subseteq R$, $(u_A, r_A) \in A$ such that
  - $\theta_f = freshSubst(vars(\varphi), vars((R, C)))$
  - θ is $\preceq_g$-maximal among substitutions such that
    - $P\theta_f\theta \subseteq R_p\theta$ (the roles in R_p satisfy the positive preconditions of φ)
    - $range(\theta) \subseteq vars(R_p) \cup Constants$ (θ instantiates variables in the rule φ with constants and with variables in the roles in R used to satisfy the positive preconditions).
    - $r_a\theta_f\theta = r_A$ (instantiating r_a yields the administrative role in A used to perform this role assignment)
    - $\theta(Self) = u_A$ (θ maps the distinguished variable *Self* to the identity u_A of the administrator performing this role assignment)
  - $r_t\theta_f\theta \notin R$ (the role being added is not already in the state)
  - $R' = R\theta \cup \{r_t\theta_f\theta\}$
  - $neg = \wedge_{r_n \in N}$

    $$\wedge_{r \in R \text{ such that } schema(r)=schema(r_n)} args(r\theta) \neq args(r_n\theta_f\theta)$$

    (the negative preconditions of φ are satisfied)
  - $C' = simplify(C\theta \wedge neg)$
  - $satisfiable(C') = true$
- $((R, C), (\varphi, \theta_f, \theta), (R', C')) \in T(URA, A)$ if $\varphi \in URA$ *and* $\varphi = can\_revoke(r_a, r_t)$ and there exist $r \in R$, $\theta_f \in Subst$, $\theta \in Subst$, $(u_A, r_A) \in A$ *such that*
  - $\theta_f = freshSubst(vars(\varphi), vars((R, C)))$
  - θ *is* $\preceq_g$-maximal among substitutions such that
    - $r\theta = r_t\theta_f\theta$ (the role being revoked is in the current state)
    - $range(\theta) \subseteq vars(r) \cup Constants$
    - $r_a\theta_f\theta = r_A$ (instantiating r_a yields the administrative role in A used to perform this role assignment)
    - $\theta(Self) = u_A$ (θ maps the distinguished variable *Self* to the identity u_A of the administrator performing this role revocation)
  - $R' = R\theta \setminus \{r\theta\}$
  - $C_1 = simplify(C\theta)$
  - $satisfiable(C_1) = true$
  - $C' = project(C_1, vars(R'))$

It might seem surprising, at first glance, that in the condition $R' = R\theta \cup \{r_t\theta_f\theta\}$, the substitution θ is applied to the entire state R, not only to the role being added. This reflects the fact that the transition might be possible for only some instances of the symbolic state. For example, consider execution of the rule *can_assign*(*Dean*, *Faculty*(*dept* = *cs*), *ComputingCommittee*) from the symbolic state ({*Faculty*(*dept* = *D*)}, *true*). This leads to the symbolic state ({*Faculty*(*dept* = *cs*), *ComputingCommittee*}, *true*), reflecting that the transition is possible only if the target user is a member of *Faculty*(*dept* = *cs*).

In the condition $R' = R\theta \backslash \{r\theta\}$ for *can_revoke*, the substitution θ is applied to the entire state for similar reasons. This is necessary even though *can_revoke* roles lack preconditions. For example, consider execution of the rule *can_revoke*(*Chair*(*dept* = *D*), *GradAdvisor*(*dept* = *D*)) from symbolic state ({*Faculty*(*dept* = *D*), *GradAdvisor*(*dept* = *D*)}, *true*) with A = {(*charles*, *Chair*(*dept* = *cs*))}. This leads to the symbolic state ({*Faculty*(*dept* = *cs*)}, *true*), reflecting that the transition is possible only if the target user is a member of *Faculty*(*dept* = *cs*).

The condition $range(\theta) \subseteq vars(R_p) \cup Constants$ provides directionality to the unification of the rule's preconditions with roles in the current state: variables in P may be instantiated with variables in R_p, but not *vice versa*. This is still a form of unification, not simply matching of P with R_p, because θ may instantiate variables in R_p with constants in P, as in the above example.

Note that $satisfiable(C')$ is checked separately from the selection of the most-general substitutions θ. This is safe because making θ less general cannot change C' from being unsatisfiable to being satisfiable. Structuring the definition this way (instead of including $satisfiable(C')$ in the inner-most list of conditions checked before selecting the most-general substitution) allows a simpler algorithm to be used to compute the substitution.

Definition 6. The *symbolic state graph* for a user-role reachability query of the form in Definition 2 is a labeled directed graph $mkGloballyFresh(V, E)$, where the set V of vertices and the set E of edges are the smallest sets such that:

- $(\{r \mid (u_0, r) \in UA(\gamma) \wedge \neg admin(r)\}, true) \in V$.
- $((R, C), \varphi, (R', C')) \in E$ and $(R', C') \in V$ if $(R, C) \in V$ and there exist $\theta_f \in Subst$, $\theta \in Subst$, and $((R, C), (\varphi, \theta_f, \theta), (R', C')) \in T(URA, A)$.

Example 2. Consider the construction of the symbolic state graph for the query in Example 1. The initial state is $S_1 = (R_1, C_1) = (\{Student(dept = cs, cid = 501)\}, true)$. From S_1, the *can_assign* rule φ is applied (renaming *D and CID* to fresh variables D' and CID' respectively and then substituting D' with cs). This adds $TA(dept = cs, cid = CID')$ to the state under the constraint $CID' \neq 501$, resulting in a symbolic state $S_2 = (R_2, C_2) = (R_1 \cup \{TA(dept = cs, cid = CID')\}, (CID' \neq 501))$. S_2 represents the four concrete states $\{Student(dept = cs, cid = 501), TA(dept = cs, cid = X)\}$ for $X \in \{101, 201, 301, 401\}$. Similarly, from S_2, rule φ can be applied again (renaming *D* and *CID* to fresh variables D_1' and CID_1' respectively and then substituting D_1' with cs). This leads to the state $S_3 = (R_3, C_3) = (R_2 \cup \{TA(dept = cs, cid = CID_1')\}, C_2 \wedge (CID_1' \neq 501) \wedge (CID_1' \neq CID'))$. Repeating this process results in a symbolic state graph containing 5 states: $S_1, S_2, S_3, S_4 = (R_4, C_4) = (R_3 \cup \{TA(dept = cs, cid = CID_2')\}, C_3 \wedge (CID_2' \neq 501) \wedge (CID_2' \neq CID') \wedge (CID_2' \neq CID_1'))$ and $S_5 = (R_4 \cup \{TA(dept = cs,$

$cid = CID_3'$)}, $C_4 \wedge (CID_3' \neq 501) \wedge (CID_3' \neq CID') \wedge (CID_3' \neq CID_1') \wedge (CID_3' \neq CID_2')$), and 4 transitions: $S_1 \xrightarrow{\varphi} S_2 \xrightarrow{\varphi} S_3 \xrightarrow{\varphi} S_4 \xrightarrow{\varphi} S_5$. If the course identifier parameter cid ranges over the set {101, 201, 301, 401, 501}, this is the complete symbolic state graph: no more symbolic states can be added, because in symbolic states with additional instances of TA, the satisfiability check for the disequality constraint fails, because the answer to the generated graph coloring problem is false. If *cid* ranges over an infinite data types, then the symbolic state graph is infinite, because an infinite number of instances of $TA(dept = cs, cid = CID)$ can be added to the state.

The following theorem says that the symbolic transition relation is an exact abstraction of the concrete transition relation.

Theorem 1. Let A be a user-role assignment containing only administrative roles. Let URA be a user-role administration policy.

1. Every concrete instance of a symbolic transition in $T(URA, A)$ is a concrete transition in $T_c(URA, A)$; more precisely, for all $((R, C), (\varphi, \theta_f, \theta), (R', C')) \in T(URA, A)$, for all ground substitutions θ_c such that $\theta_c \preceq_g \theta$ and such that $C\theta_c \wedge C'\theta_c$ holds, $(R\theta_c, (\varphi, \theta_c \circ \theta \circ \theta_f), R'\theta_c) \in T_c(URA, A)$.
2. Every concrete transition between instances of two symbolic states is represented by a symbolic transition; more precisely, for all symbolic states (R, C) and (R', C'), if there exist $R_c \in [\![(R, C)]\!]$, $R_c' \in [\![(R', C')]\!]$, and a ground substitution θ_c such that $(R_c, (\varphi, \theta_c), R_c') \in T_c(URA, A)$, then there exist substitutions θ_f and θ such that $\theta_c \preceq_g \theta$ and $((R, C), (\varphi, \theta_f, \theta), (R', C')) \in T(URA, A)$.

In the first item in Theorem 1, the condition $\theta_c \preceq_g \theta$ reflects the fact that the symbolic transition requires R_1 to be instantiated consistently with θ.

One might expect a simpler and tighter relationship to hold between the concrete and symbolic transition relations. For example, one might hope to replace both items in Theorem 1 with the biconditional: $((R, C), (\varphi, \theta_f, \theta), (R', C')) \in T(URA, A)$ iff for all ground substitutions θ_c such that $\theta_c \preceq_g \theta$ and such that $C\theta_c \wedge C'\theta_c$ holds, $(R\theta_c, (\varphi, \theta_c \circ \theta \circ \theta_f), R'\theta_c) \in T_c(URA, A)$. However, this relationship does not hold, because of the $\preceq_g$-maximality condition on θ in the definition of the symbolic transition relation. To see this, note that this biconditional would imply that the symbolic transition relation contains all concrete transitions. The $\preceq_g$-maximality condition on θ exists specifically to avoid this. Including concrete transitions in the symbolic transition relation would be sound, but it would be disastrous for efficiency of symbolic analysis.

The following theorem says that the symbolic state graph is an exact abstraction of the concrete state graph. Define the meaning of an edge in the symbolic state graph by

$$[\![(s, \varphi, s')]\!] = \{(s_c, \varphi, s_c') \mid s_c \in [\![s]\!], s_c \in [\![s']\!]\}$$

Theorem 2. Let (V_c, E_c) and (V, E) be the concrete and symbolic state graphs, respectively, for a user-role reachability query of the form in Definition 2.

1. $V_c = \cup_{s \in V} [\![s]\!]$
2. $E_c = \cup_{e \in E} [\![e]\!]$

4.1. *Finite data types*

Our framework can easily be extended to support a type system for parameters of role schemas. If the types may be finite, then we modify the symbolic transition relation to generate disequality constraints that ensure every element of R represents a distinct role instance, and we modify the algorithm for *satisfiable*(C) to check whether there are sufficiently many values of each type to satisfy the disequality constraint. This prevents transitions that introduce unnecessarily many instances of a role schema. For example, if parameter *cid* of role schema *TA*(*cid*) ranges over {101, 201, 301, 401, 501}, then these constraints prevent symbolic transitions to states containing more than 5 instances of *TA*(*cid*).

Specifically, in the definition of the symbolic transition relation, the item $r_t\theta_f\theta \notin R$ is deleted, the item

- $distinct = \wedge_{r \in R \text{ such that } schema(r) = schema(r_t)} args(r\theta) \neq args(r_t\theta_f\theta)$ (the role being added is not already in the state; note that a conjunction with no conjuncts is *true*)

is added, and the definition of C' is changed to

- $C' = simplify(C\theta \wedge neg \wedge distinct)$

satisfiable(C) is false if (1) C contains a tuple disequality whose left side and right side are the same, or (2) one of the following checks fails: for each finite type T, construct a *disequality graph* G_d with a vertex for each constant and variable of type T in C, and with an edge between two vertices if C contains a disequality requiring them to be unequal, and check that the chromatic number of G_d (i.e., the least number of colors needed for a proper vertex coloring of G_d) is less than or equal to the number of values of type T. This is a well-studied NP-complete problem. Many exact algorithms, heuristics, and approximation algorithms have been proposed. G_d is typically small enough that exact algorithms can be used. In the case studies described in Section 10, the abstraction of infinite types is preferable to the use of finite types, but if finite types were used, then for our sample queries, the disequality graph for each symbolic state would contain 3 or fewer nodes.

5. Analysis algorithm

This section presents a symbolic algorithm for user-role reachability analysis of PARBAC policies. The algorithm has two stages. The first stage performs a backward search from the goal toward the initial state. However, some of the enabling conditions of the administrative actions are not checked during the backward search. In other words, this stage constructs an over-approximation of a backward slice (starting from the goal) of the symbolic state graph. The second stage determines which states in that graph are actually reachable, by running an exact forward search from the initial state, but limiting the search based on the results of stage 1. Compared to

Please cite this article in press as: Stoller SD, et al., Symbolic reachability analysis for parameterized administrative role-based access control, Computers & Security (2010), doi:10.1016/j.cose.2010.08.002

a purely forward algorithm, the backward stage improves the algorithm's efficiency by pruning the search space, and improves the algorithm's termination behavior. The overall strategy of using an approximate backward search followed by a forward search is reminiscent of Graphplan (Blum and Furst, 1997), although the details are quite different.

5.1. *First stage*

The graph constructed by the first stage of the algorithm is an over-approximation for two reasons: negative preconditions are ignored, and disequality constraints are ignored. Negative preconditions could, at best, be only partially checked during the first stage, because the symbolic states constructed during the first stage might be *subsets* of the symbolic states that are actually reachable. This is because those states might actually contain additional roles that were needed to satisfy positive preconditions of earlier transitions (i.e., transitions between a state and the initial state); although some of those roles could perhaps be revoked, some of them might not be revocable by the administrative roles in A.

Since negative preconditions cannot be checked completely during the first stage, for simplicity, we do not check them at all during that stage; they are enforced during the second stage. Since disequality constraints are used primarily to enforce negative preconditions, we do not keep track of them during the first stage. Thus, each symbolic state in the *backward symbolic state graph* is simply a set of roles (i.e., role instances). Edges are determined by the *backward symbolic transition relation* T_b. A tuple $(R', (\varphi, \theta_f, \theta), R)$ is in that relation if a backward step from R' to R (i.e., R' is closer to the goal, and R is closer to the initial state) is possible—ignoring negative preconditions—using rule φ with the given substitutions, which are analogous to the substitutions in the forward symbolic transition relation introduced in Section 4. The backward symbolic transition relation considers only role assignment actions; it does not consider revocation, which cannot help satisfy positive preconditions.

Definition 7. The *backward symbolic transition relation* $T_b(URA, A)$ for a user-role administration policy URA and an assignment A of users to administrative roles is the least relation such that:

- $(R', (\varphi, \theta_f, \theta), R) \in T_b(URA, A)$ if $\varphi \in URA$ and $\varphi = can\_assign(r_a, (P, N), r_t)$ and there exist $r_t' \in R'$, $P_1 \subseteq P$, $R_p \subseteq R' \setminus \{r_t'\}$, $(u_A, r_A) \in A$ such that
  - $\theta_f = freshSubst(vars(\varphi), vars(R'))$
  - θ is $\preceq_g$-maximal among substitutions such that
    - $P_1\theta_f\theta \subseteq R_p\theta$ (the positive preconditions in P_1 are satisfied by the roles in R_p; the other preconditions of φ will be added to R, acting as new sub-goals)
    - θ does not map variables in $vars(R')$ to locally fresh variables.
    - $r_a\theta_f\theta = r_A$ (instantiating r_a yields the administrative role in A used to perform this role assignment)
    - $\theta(Self) = u_A$ (θ maps the distinguished variable $Self$ to the identity u_A of the administrator performing this role assignment)
    - $r_t'\theta = r_t\theta_f\theta$ (the role r_t' in R' is the role added by this transition)
  - $R = R'\theta \setminus \{r_t'\theta\} \cup \bigcup_{r \in P \setminus P_1} \{r\theta_f\theta\}$ (the earlier state R contains the roles in R', minus the role added by this transition, plus roles used to satisfy the remaining positive preconditions of φ)
  - $r_t'\theta \notin R$ (the role being added is not present in the earlier state R)

Definition 8. The backward symbolic state graph for a user-role reachability query of the form in Definition 2 is a labeled directed graph $mkGloballyFresh(V, E)$, where the sets V of vertices and E of edges are the smallest sets such that:

- $g \in V$.
- $(R', \varphi, R) \in E$ and $R \in V$ if $R' \in V$ and there exist $\theta_f \in Subst$ and $\theta \in Subst$ such that $(R', (\varphi, \theta_f, \theta), R) \in T_b(URA, A)$.

Example 3. The backward symbolic state graph for the following policy and query is shown in Fig. 1.

$$
\begin{aligned}
&RS = \{r_a, \rho_1(p), \rho_2(p), \rho_3(p)\}\\
&\varphi_1 = can\_assign(r_a, true, \rho_1(p = X))\\
&\varphi_2 = can\_assign(r_a, \rho_1(p = X), \rho_2(p = X))\\
&\varphi_3 = can\_assign(r_a, \rho_2(p = X) \wedge \neg\rho_1(p = X), \rho_3(p = X))\\
&\varphi_4 = can\_revoke(r_a, \rho_1(p = X))\\
&\varphi_5 = can\_revoke(r_a, \rho_3(p = X))\\
&UA(\gamma) = \{(u_a, r_a)\}\\
&A = \{(u_a, r_a)\}\\
&g = \{\rho_1(p = Y), \rho_3(p = Z)\}
\end{aligned}
$$

The definition of the backward symbolic transition relation does not require $\subseteq$-maximality of P_1, so when a backward symbolic transition is possible with a non-empty value of P_1, then backward symbolic transitions corresponding to subsets of P_1 are also possible. This allows positive preconditions to be satisfied using either new role instances or role instances already in the state. For example, for the transition labeled φ_2 from $\{\rho_1(Y), \rho_2(Z)\}$ to $\{\rho_1(Z)\}$, P_1 contains the positive precondition of φ_2 (this is why the size of the symbolic state decreases), while P_1 is empty for the other transition labeled φ_2 from the same source state to $\{\rho_1(Y), \rho_1(Z)\}$.

5.2. *Second stage*

The second stage performs a forward search and maintains a correspondence between states explored by the forward search, called *forward states*, and states explored during the

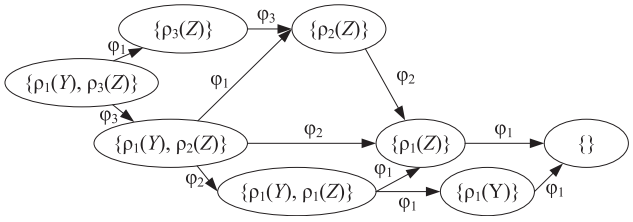

Fig. 1 – Backward symbolic graph for Example 3. An edge from R to R' labeled with φ means $(R', \varphi, R) \in E_b$. The roles all have one parameter, p, whose name is elided, to improve readability; for example, $\rho_1(p = Y)$ is shown as $\rho_1(Y)$.

first stage, called *backward states*. The correspondence is used to limit the forward search to explore only transitions that might be useful for reaching the goal. Specifically, from each forward state (R, C) and each backward state R_b corresponding to it, the (unoptimized) forward algorithm explores (1) all enabled *can_assign* rules φ such that one of the backward states R_b corresponding to (R, C) is the target of an edge labeled with φ in the backward symbolic state graph, and (2) all enabled *can_revoke* rules. The resulting graph is called a *goal-directed forward symbolic state graph*. Its nodes are pairs $((R, C), R_b)$ of a forward state (R, C) and a corresponding backward state R_b.

Definition 9. The *goal-directed forward symbolic state graph* for a user-role reachability query of the form in Definition 2 is a labeled directed graph $mkGloballyFresh(V, E)$, where V and E are the smallest sets satisfying the following conditions, where (V_b, E_b) is the backward symbolic state graph for the query.

- $((UA_0, true), R_b) \in V$ for each $R_b \in V_b$ such that $(\exists \theta \in Subst.\ R_b\theta \subseteq UA_0)$, where UA_0, the initial role assignment for u_0, is $UA_0 = \{r \mid (u_0, r) \in UA\ (\gamma) \wedge \neg admin(r)\}$ (the initial forward state $(UA_0, true)$ is related to backward states that represent subsets of UA_0; intuitively, we use subset, instead of equality, because a backward state is a set of sub-goals, and we just require that the sub-goals are satisfied in the initial state)
- $(((R, C), R_b), \varphi, ((R', C'), R'_b)) \in E$ and $((R', C'), R'_b) \in V$ if $((R, C), R_b) \in V$ and there exist substitutions θ_f and θ such that $((R, C), (\varphi, \theta_f, \theta), (R', C')) \in T(URA, A)$ and either (1) φ is a *can_revoke* rule and $R_b' = R_b$, or (2) φ is a *can_assign* rule and $(R_b', \varphi, R_b) \in E_b$.

A forward state (R, C) *satisfies* goal g with substitution θ if $R\theta \supseteq g\theta$ and $satisfiable(C\theta)$. Let $subst((R, C), g)$ be the set of substitutions θ such that (R, C) satisfies goal g with substitution θ.

The symbolic analysis algorithm can provide a symbolic representation of all reachable instances of the goal: for each reachable forward state (R, C), for each substitution θ in $subst((R, C), g)$ that is $\preceq_g$-maximal in $subst((R, C), g)$, add $((R, C), \theta)$ to the result.

To improve termination and efficiency, the second stage uses a depth-first search limited to explore paths whose projection onto the backward graph is acyclic, except that, as a special case, the self-loops (i.e., an edge from a node to itself) due to *can_revoke* transitions are allowed. In other words, a *can_assign* transition leading to a state $((R, C), R_b)$ is not taken if R_b is the second component of some state already on the search stack. To see that limiting the search in this way is safe, note that a segment of a forward path that projects to a cycle in the backward graph does not help satisfy any positive preconditions, because that is exactly what the backward graph is keeping track of. Thus, the *can_assign* transitions in such a path segment are useless. The path segment might also contain *can_revoke* transitions, which might help satisfy some negative precondition. Since revocation is unconditional, those *can_revoke* transitions do not depend on the *can_assign* transitions and therefore will be explored along another path that does not contain useless *can_assign* transitions.

Example 4. Consider the goal-directed forward symbolic graph for the policy and query in Example 3. The backward state $\emptyset$ corresponds to the initial forward state $(\{\}, true)$. Consider the following path in the backward graph in Fig. 1 (the roles all have one parameter p, whose name is elided to improve readability, as in Fig. 1):

$$\{\rho_1(Z), \rho_3(Z)\} \xrightarrow{\varphi_1} \{\rho_3(Z)\} \xrightarrow{\varphi_3} \{\rho_2(Z)\} \xrightarrow{\varphi_2} \{\rho_1(Z)\} \xrightarrow{\varphi_1} \emptyset.$$

Corresponding to this path in the backward graph, the second stage of the algorithm, without optimizations, constructs the following path in the goal-directed forward graph (space is inserted between the two components of each state to align the second components, which are backward states):

$$\begin{array}{ll} & ((\emptyset, true), \quad \emptyset) \\ \xrightarrow{\varphi_1} & ((\{\rho_1(Z)\}, true), \quad \{\rho_1(Z)\}) \\ \xrightarrow{\varphi_2} & ((\{\rho_1(Z), \rho_2(Z)\}, true), \quad \{\rho_2(Z)\}) \\ \xrightarrow{\varphi_4} & ((\{\rho_2(Z)\}, true), \quad \{\rho_2(Z)\}) \\ \xrightarrow{\varphi_3} & ((\{\rho_2(Z), \rho_3(Z)\}, true) \quad \{\rho_3(Z)\}) \\ \xrightarrow{\varphi_1} & ((\{\rho_1(Y), \rho_2(Z), \rho_3(Z)\}, true), \quad \{\rho_1(Y), \rho_3(Z)\}). \end{array}$$

The last of these states satisfies the goal.

This example illustrates the observation in Section 5.1 that states in the backward graph represent subsets of reachable states: the backward graph contains the state $\{\rho_1(Z), \rho_3(Z)\}$, but all reachable states containing an instance of ρ_3 also contain an instance of ρ_2, because ρ_2 is a positive precondition in the rule φ_3 for adding ρ_3, and ρ_2 cannot be revoked.

5.3. *Termination*

Termination is an issue, because the symbolic state graph may be infinite. For example, each use of the rule *can_assign* $(Chair(dept = D), Faculty(dept = D), Instructor(dept = D, cid = C))$ introduces a fresh variable for the course identifier, so a purely forward symbolic algorithm may add an unbounded number of distinct instances of the Instructor role schema. The backward stage prevents divergence in many cases, but not all. Our algorithm is guaranteed to terminate if either (T1) the policy's positive-precondition dependency graph is acyclic, or (T2) all *can_assign* rules in the policy have at most one positive precondition. The *positive-precondition dependency graph* for a PARBAC policy is a directed graph that contains a vertex for each role schema and contains an edge from r_1 to r_2 if the policy contains a *can_assign* rule with r_1 in the positive precondition and r_2 in the target.

Condition (T1) ensures termination of the backward stage, because it implies the diameter of the backward symbolic state graph is finite; the outdegree of every node in the backward symbolic state graph is also finite, so the graph is finite. Condition (T2) ensures termination of the backward stage, because it implies that every state constructed during the backward search contains at most $|g|$ roles, and there are only a finite number of such states (this is true even though transitions can introduce fresh variables, because two states are equal if they differ only in the names of variables introduced during the search).

Termination of the backward stage ensures termination of the forward stage, because the forward search is limited to exploring paths whose projection onto the backward graph is acyclic.

Please cite this article in press as: Stoller SD, et al., Symbolic reachability analysis for parameterized administrative role-based access control, Computers & Security (2010), doi:10.1016/j.cose.2010.08.002

The policies for both of our case studies satisfy both (T1) and (T2). We expect that most real policies satisfy at least one of them. The positive-precondition dependency graph is often acyclic, because roles in the positive precondition in a *can_assign* rule are often junior in the organizational hierarchy to the target role, and organizational hierarchies are acyclic.

5.4. Optimizations

5.4.1. Slicing

A two-stage policy slicing transformation is applied before analysis. The first stage of slicing computes a simple over-approximation of the set of reachable roles. The second stage of slicing uses this information to eliminate *can_assign* and *can_revoke* rules that are either unreachable, i.e., cannot be executed from any state reachable from the given initial state, or useless, i.e., do not truthify any positive preconditions that help reach the given goal. Slicing can be done in linear time in the size of the problem instance, assuming the arity of roles is bounded by a constant.

The first stage of slicing computes a set QRR of *quasi-reachable roles*. QRR is the least set satisfying the following recursive definition: (1) if r is in the initial state, then r is in QRR; (2) if r is the target of a *can_assign* rule all of whose positive preconditions are unifiable with elements of QRR, then r is in QRR. QRR has the property: every role instance in every reachable concrete state is an instance of (i.e., can be obtained by instantiating) some role in QRR.

The second stage of slicing computes sets CA_u and CR_u of *can_assign* and *can_revoke* rules, respectively, that over-approximate the sets of policy rules that are possibly useful (and reachable), and then discards all rules not in these sets. The set CA_u of useful *can_assign* rules is defined simultaneously with the set R_u of useful roles (i.e., adding these roles to the state might be useful for reaching the goal). A role r is in R_u if r appears in the goal or occurs as a positive precondition of a rule in CA_u. A *can_assign* rule is in CA_u if its target is unifiable with a role in R_u and all of its preconditions are unifiable with roles in QRR. A *can_revoke* rule is in CR_u if its target is unifiable with a negative precondition of a rule in CA_u.

5.4.2. Eager revocation of useless roles

During the second stage of the algorithm, "useless" roles, i.e., roles that will not be needed to satisfy any positive preconditions, are eagerly revoked; this is a kind of partial-order reduction. Specifically, a *can_assign* transition corresponding to a backward transition $(R_b', \varphi, R_b) \in E_b$ is augmented so that, after the role assignment, it also revokes (i.e., removes from the forward state) every revocable role r in the forward state that does not match any element of R_b. We call these roles *useless* in that state, because these roles will not be needed to satisfy any preconditions in the rest of the path to the goal. A role r is *revocable* with respect to a user-role reachability query of the form in Definition 2 if A contains an administrative role with permission to revoke r. A role r_1 *matches* a role r_2 if r_1 and r_2 are instances of the same role schema and, for each parameter p of the schema, either (a) r_1 or r_2 has a variable as the value of p or (b) r_1 and r_2 contain the same constant as the value of p.

Example 5. Consider the same reachability query as in Example 4. With eager revocation of useless roles, the *can_revoke* transition using rule φ_4 would be combined with the preceding *can_assign* transition using φ_2, and if the policy were extended with a *can_revoke* rule for ρ_2, then the transition that uses φ_3 to add $\rho_3(Z)$ would be extended to revoke $\rho_2(Z)$.

5.4.3. Remove α-equivalent useless roles

This optimization detects "redundant" useless roles and removes some of them to eliminate the redundancy. The *local variables* of a role r in a forward state (R, C) are the variables that appear in r and do not appear anywhere else in the forward state. Two roles r_1 and r_2 in a forward state (R, C) are *α-equivalent* if they can be obtained from each other by renaming local variables. If a forward state contains two α-equivalent useless roles, then one of them is removed. This is sound because useless roles are used only to evaluate negative preconditions, and this removal does not affect which negative preconditions are satisfied by the state.

5.4.4. Prune subsumed states

This optimization avoids exploring symbolic states that can only lead to a subset of the states reachable from some symbolic state that has already been explored. We say that a set R' of roles in the forward part of a symbolic state S' *is subsumed by* a set R of roles in the forward part of a symbolic state S if $|R'| = |R|$ and every concrete instance of R' is also a concrete instance of R. We conservatively check subsumption by attempting to construct a 1–1 correspondence between elements of R' and R, and a substitution θ satisfying the disequality constraints, such that for each role r' in R' and the corresponding role r in R, $r' = r\theta$.

During the second stage of the algorithm, let S be a previously explored state, and let S' be a candidate state to explore. If S' agrees with S on the backward state and the useful part of the forward state, and a subset of the useless part of S' is subsumed by the useless part of S, then this optimization suppresses exploration of S'. The subsumption and subset checks both ensure that S' satisfies fewer negative preconditions than S, and the two states satisfy the same positive preconditions, so any goal reachable from S' is also reachable from S.

Note that this is a form of forward subsumption, which explores a state only if it is not subsumed by a previously encountered state. We do not consider backward subsumption, which prevents further exploration from a previously encountered state if it is subsumed by a later state.

5.4.5. Early stopping

If the user wants only one reachable instance of the goal, then the forward search halts as soon as a state satisfying the goal is encountered; we call this *early stopping*.

5.4.6. No revocation of non-negative roles

A role schema r is *non-negative* with respect to a backward state R_b if, for all transitions φ on edges between R_b and g in the backward graph, r does not appear in a negative precondition of φ. After the first stage of the algorithm, we compute the non-negative role schemas for each backward state. During the second stage of the algorithm, from a state $((R, C), R_b)$, *can_revoke* transitions are not explored for roles that are

Please cite this article in press as: Stoller SD, et al., Symbolic reachability analysis for parameterized administrative role-based access control, Computers & Security (2010), doi:10.1016/j.cose.2010.08.002

instances of role schemas that are non-negative with respect to R_b, with the exception that useless roles are still eagerly revoked. Suppressing revocation of non-negative roles is safe because revocation of those roles would not enable any transition on the rest of the path to the goal. Note that useless non-negative roles are still eagerly revoked. This has low cost, because eager revocations are combined with a preceding *can_assign* transition and hence do not add nodes or edges to the graph, and it helps reduce the number of useless roles in the state and thereby reduce the cost of subsumption checks.

6. Undecidability

We show undecidability—more specifically, (strict) semi-decidability—of user-role reachability for PARBAC, based on a semidecidability result for the plan existence problem for STRIPS-style planning. Specifically, we use Theorem 3.10 in Erol et al. (1991a), a technical report containing details of the complexity results summarized in the journal article (Erol et al., 1991b).

The *plan existence problem* is: given a language (of constants, predicate symbols, and function symbols) an initial state, a set of operators, and a goal, determine whether there exists a plan from the initial state to the goal. An operator is characterized by a set of positive preconditions, a set of negative preconditions, an add set (a set of atoms representing facts that the operator adds to the state), and a delete set (a set of atoms representing facts that the operator removes from the state). A problem instance is *function-free* if the language contains no function symbols. A problem instance is *deletion-free* if every operators has an empty delete set.

Theorem 3.10 in Erol et al. (1991a) states that the plan existence problem is semi-decidable if the language is allowed to contain infinitely many constants, even if problem instances are restricted to be deletion-free and function-free (and the initial state is restricted to be finite). The proof in Erol et al. (1991a; Appendix A.4) is by reduction from the halting problem.

To reduce a function-free, deletion-free planning problem to the reachability problem for PARBAC, the main step is to show how the operators in the planning problem can be expressed using *can_assign*. The main difficulty is that an operator may add multiple facts, while a single *can_assign* rule adds only one fact. To separate this aspect from the rest of the reduction, we factor the reduction into two steps: a translation from the planning problem to an extension of PARBAC, called multi-target PARBAC, in which *can_assign* rules may contain multiple target roles, and a translation from multi-target PARBAC to PARBAC. The former translation is completely straightforward. The latter translation introduces new roles that, intuitively, represent sets of simultaneously added roles of the original policy. Preconditions in *can_assign* rules are transformed to take these new roles into account. For example, suppose the policy contains a multi-target *can_assign* rule with target $\{\rho_1(p = X), \rho_2(q = Y)\}$. The translation replaces this rule with a *can_assign* rule with target $\rho_{1,2}(p = X, q = Y)$, where $\rho_{1,2}(p, q)$ is a new role schema. Every *can_assign* rule with $\rho_1(p = X)$ in the positive precondition P is replaced with two rules: one with $\rho_1(p = X)$ in P (i.e., no change), and one with $\rho_{1,2}(p = X, q = Y')$ replacing $\rho_1(p = X)$ in P, where Y' is a fresh variable (i.e., a variable that does not yet appear in the rule). Each negative precondition N containing $\rho_1(p = X)$ is extended to also include $\rho_{1,2}(p = X, q = \_)$, where _ is a wildcard.

More generally, we define a translation that, given a multi-target PARBAC reachability problem instance of the form in Definition 2, and a multi-target rule $\varphi_0 = can\_assign(r_0, (P_0, N_0), \{\rho_1(\vec{e}_1), \ldots, \rho_n(\vec{e}_n)\})$ in URA, where $\vec{e}_i$ is a vector of arguments, produces a set of reachability problem instances (differing only in their goals) in which φ_0 has been replaced with single-target *can_assign* rules, and with the property that the original problem instance is satisfiable (i.e., the goal is reachable) iff one of the generated problem instances is satisfiable. The result of the translation is the problem instance in Fig. 2, where ρ' is a fresh role name, f_p transforms roles used as positive preconditions or goals, f_n transforms roles used as negative preconditions, and f_R transforms *can_assign* rules (the deletion-free restriction implies that there are no *can_revoke* roles). Disjunction in a positive precondition of a rule is a shorthand that is easily expanded by replacing the rule with multiple rules (one for each disjunct). Similarly, disjunction in a goal is a shorthand for multiple goals (hence multiple problem instances).

Repeatedly applying this translation, once for each rule with multiple target roles, reduces a multi-target PARBAC reachability problem instance to a set of PARBAC problem instances.

Harrison et al.'s (1976) classic proof of undecidability of the safety analysis problem for their access control model cannot easily be adapted to our setting for a variety of reasons, e.g., their model allows creation of subjects and objects, while our result applies to reachability analysis for user-role administration (URA) policies, which do not support creation of users or roles. Crampton (2002) defines an administrative model for RBAC (without role parameters) and shows that reachability analysis for it is undecidable. There are several differences between the URA policies considered here and the administrative model considered in Section 5.2 of Crampton (2002),

$$
\begin{aligned}
&\gamma' = \langle RS', U, UA\rangle\\
&RS' = RS \cup \{\rho'(\vec{e}_1, \ldots, \vec{e}_n)\}\\
&A' = A\\
&\varphi_0' = \varphi_0 \text{ with target changed to } \rho'(\vec{e}_1, \ldots, \vec{e}_n)\\
&URA' = \{f_R(\varphi) \mid \varphi \in URA \cup \{\varphi_0'\} \setminus \{\varphi_0\}\}\\
&f_p(\rho(\vec{e})) = \text{let } S = \{i \mid i \in 1..n \wedge \rho_i = \rho\}\\
&\qquad \text{in if } S = \emptyset \text{ then } \rho(\vec{e})\\
&\qquad\quad \text{else } \rho(\vec{e}) \vee \textstyle\bigvee_{i\in S} \rho'(\vec{X}_1, \ldots, \vec{X}_{i-1}, \vec{e}, \vec{X}_{i+1}, \ldots, \vec{X}_n)\\
&\quad \text{where } \vec{X}_j \text{ is a vector of fresh variables,}\\
&\quad \text{with same length as } \vec{e}_j\\
&f_n(\rho(\vec{e})) = \{\rho(\vec{e})\} \cup \textstyle\bigcup_{i\in 1..n \text{ s.t. } \rho_i=\rho} \{\rho'(\_, \ldots, \_, \vec{e}, \_, \ldots, \_)\}\\
&\quad \text{where, in the arguments of } \rho', \text{ there are } i-1 \text{ wildcards}\\
&\quad \text{before } \vec{e} \text{ and } n-i \text{ wildcards after } \vec{e}\\
&f_R(can\_assign(r_a, (P, N), r)) =\\
&\quad can\_assign(r_a, (\{f_p(r) \mid r \in P\}, \textstyle\bigcup_{r\in N} f_n(r)), r)\\
&g' = \{f_p(r) \mid r \in g\}
\end{aligned}
$$

Fig. 2 – Translation to replace a multi-target *can_assign* rule φ_0 with a single-target *can_assign* rule.

Please cite this article in press as: Stoller SD, et al., Symbolic reachability analysis for parameterized administrative role-based access control, Computers & Security (2010), doi:10.1016/j.cose.2010.08.002

e.g., the former supports role parameters, and the latter supports creation of roles. Becker's (2009) semidecidability result for the reachability problem for DynPAL is also based on Theorem 3.10 in Erol et al. (1991a). The main difference from our result is that DynPAL rules may perform multiple updates, so there is no need for an analogue of our reduction from multi-target PARBAC to PARBAC.

7. Fixed-parameter tractability

Expressing the complexity of the optimized backward algorithm as a function of the overall problem size alone is unsatisfactory, because the worst-case complexity with respect to this parameter is exponential, while we expect the typical complexity to be much better. To provide some insight into when and why this is the case, we express the complexity in terms of several metrics that characterize the "difficulty" of the policy. This complexity result applies to policies that satisfy conditions (T1) and (T2) in Section 5.3.

Let G_b denote the backward symbolic state graph for a query. Each backward state R_b in G_b satisfies $|R_b| \leq |g|$, because each backward transition replaces the target role with the positive precondition of the selected *can_assign* rule. Let d_p denote the diameter of the positive-precondition dependency graph. Typically d_p is much smaller than $|RS|$, because it measures the height, not the total size, of an organization's administrative structure. The length of paths in G_b is bounded by $|g|d_p$, because each backward transition decreases the sum of the heights (in the positive-precondition dependency graph) of the schemas of the roles in the backward state.

Let d_t denote the maximum number of *can_assign* rules with the same role schema as a target. The outdegree of a vertex in the backward state graph is bounded by $|g|d_t d_\theta$, where d_θ bounds the number of different successor states that can be reached from a given backward state using a given *can_assign* rule and different substitutions, i.e., it is the maximum, over backward states R'_b in G_b and *can_assign* rules φ in the policy, of $|\{R_b \mid \exists \theta_f, \theta.\ (R'_b, (\varphi, \theta_f, \theta), R_b) \in T_b(URA, A)\}|$. Note that d_θ is bounded by $|R_b'|$ hence by $|g|$, because differences in θ that lead to differences in R_b come from matching the target of φ with different elements of R'_b. Thus, the outdegree of a vertex in the backward state graph is bounded by $|g|^2 d_t$. The number of nodes in a graph with maximum path length ℓ and maximum outdegree d is $O(d^\ell)$. Therefore, the number of backward states is $O((|g|^2 d_t)^{|g|d_p})$.

Let G_f denote the goal-directed forward symbolic state graph for the query. Every node in G_f is reachable by a simple path in G_f. Every simple path in G_f corresponds, by projection onto the second component of each node, to a distinct path in G_b, because (1) every transition in the goal-directed forward symbolic graph corresponds to execution of a backward symbolic transition that changes the second component (i.e., the backward state) in the node, and (2) distinct outgoing transitions from a state in the goal-directed forward symbolic graph must correspond to execution of different *can_assign* transitions hence to execution of different backward symbolic transitions. Furthermore, these paths in G_b contain at most one occurrence of each cycle in G_b, because transitions that go around a cycle in G_b a second time would not add more irrevocable roles or constraints to the corresponding forward states, hence the corresponding fragment of the path in G_f would be a cycle, contradicting the assumption that the path in G_f is simple. Therefore, the number of states in G_f is bounded by the number of paths in G_b that go around each cycle at most once. This is bounded by some function φ of the number of backward states. The time complexity of standard state-graph construction algorithms is polynomial in the size of the input and linear in the size of the output (i.e., the generated state graph). Therefore, the worst-case time complexity of the overall backward algorithm is $O(|I|^c \phi((|g|^2 d_t)^{|g|d_p}))$, for some constant c and some function φ, where $|I|$ is the size of the problem instance (the query). This implies that user-role reachability for queries satisfying (T1) and (T2) is fixed-parameter tractable with respect to $\max(|g|, d_t, d_p)$. For the queries in our case studies, we found $|g| \leq 2$, $d_t \leq 10$, and $d_p \leq 3$.

8. Beyond separate administration

Recall that the preceding algorithms assume separate administration. In Stoller et al. (2007), we presented two approaches to analysis of policies that do not satisfy separate administration. The first approach extends the algorithms to keep track of the user-role assignment for each administrator as well as the target user u_0; this is straightforward but may be computationally expensive. The second approach allows more efficient analysis of policies that satisfy an alternative assumption called *hierarchical role assignment*, which says, roughly, that each administrative role has authority to assign users only to selected roles that are junior to it in the role hierarchy. Both approaches can be adapted for analysis of PARBAC.

9. Case studies

We used PARBAC policies for a university and a health-care facility as case studies. Unparameterized versions of these policies were used as case studies in Stoller et al. (2007); those versions are unrealistic in the sense that they accommodate only one department, one course, one faculty, etc. The parameterized versions accurately handle multiple departments, multiple courses, multiple faculty, etc. Both policies have the following characteristics: (1) the positive-precondition dependency graph is acyclic; (2) every *can_assign* rule has at most one positive precondition; (3) for almost all *can_assign* rules, there is a corresponding *can_revoke* rule, so almost all roles are revocable; and (4) the policy does not satisfy separate administration, but hierarchical role assignment is satisfied for most sets of administrative roles. The policies contain about 3 dozen and 1 dozen *can_assign* rules, respectively.

9.1. *University*

Our PARBAC policy for a university controls assignment of users to student roles and employee roles. It contains 60 role schemas and 35 *can_assign* rules; expanding role hierarchy increases it to 625 rules. Role schemas for students

Please cite this article in press as: Stoller SD, et al., Symbolic reachability analysis for parameterized administrative role-based access control, Computers & Security (2010), doi:10.1016/j.cose.2010.08.002

include *Student*, *Undergrad*, *Undergrad(dept)*, *Undergrad(dept, cid)*, and *RA(fac)*. Role schemas for employees include *Employee*, *Faculty*, *Faculty(dept)*, *Instructor(dept, cid)*, *DeptChair (dept)*, and *Provost*. Role hierarchy relationships include *President* $\geq$ *Provost* $\geq$ *DeanOfAdmissions* $\geq$ *AdmissionsOfficer* $\geq$ *Staff*.

A sample user-role reachability query is: Can an administrative user initially in *DeptChair*(*dept* = *cs*) add a user initially in *Faculty*(*dept* = *ee*) to *QualExamCommittee*(*dept* = *cs*)? The answer is no, because the policy states that members of *DeptChair*(*dept* = *D*) can assign only members of *Faculty* (*dept* = *D*) to *QualExamCommittee*(*dept* = *D*). Another sample reachability query is: Can administrative users in *GradAdmissionOfficer* and *RegistrarOfficer* add a user initially in no roles to *Grad*(*dept* = *cs*, *cid* = 501)? Yes, because a *GradAdmissionOfficer* can first assign the user to *Grad*, and a *RegistrarOfficer* can then assign the user to *Grad*(*dept* = *cs*, *cid* = 501).

9.2. *Health-care facility*

Our second case study is a PARBAC policy for a health-care facility, based on policies in Evered and Bögeholz (2004), Becker (2005). The policy contains 14 *can_assign* rules. Role schemas include *Doctor*, *Doctor(patient)*, *Nurse*, *ReferredDoctor (patient)*, *PrimaryDoctor(patient)*, *Receptionist*, *Manager*, *Medical-Manager*, and *ThirdParty*. Hierarchical role assignment is satisfied for most sets of administrative roles, but not as high a percentage of them as for the university policy.

10. Experimental results

We implemented the symbolic algorithm described in this paper and the forward and backward algorithms for analysis of unparameterized ARBAC in Stoller et al. (2007) using the XSB tabled logic programming system, version 3.1. We refer to the algorithms in Stoller et al. (2007) as *concrete algorithms*. All reported data were obtained on a 2.5 GHz Pentium machine with 4 GB RAM running Linux 2.6.28. The policies used in our experiments are available at http://www.cs.stonybrook.edu/stoller/parbac/.

10.1. *Case studies*

We applied the symbolic algorithm to 5 user-role reachability queries for the university policy and 2 such queries for the health care policy (details are at the above URL). Each query is answered in less than 0.01 s.

10.2. *Performance comparison for parameterized policies*

These experiments evaluate the performance benefit of using symbolic analysis for parameterized policies in which all parameters range over finite types. These experiments compare the performance of the symbolic algorithm applied to a parameterized policy with the performance of the concrete algorithm applied to each unparameterized policy obtained by instantiating the parameterized policy using values from a single finite type, for varying sizes of the type.

To do this for a variety of "realistic" policies, we generate synthetic policies that are structurally similar to our university policy after expansion of role hierarchy and that contain about the same number of *can_assign* rules (namely, 625). The policies contain the same role schemas as the university policy. The number of *can_assign* rules per (target) role schema is chosen randomly following the distribution of rules per role schema in the university policy. The numbers of positive and negative preconditions per rule are chosen in an analogous way. For each rule, role schemas for the positive and negative preconditions are randomly selected and then instantiated based on the following observation about the university policy: in each rule, parameters (in different role schemas) are instantiated with same variable if the parameters have the same name. "Easy" problem instances, for which policy slicing yields an empty policy, are discarded and replaced during policy generation. To generate problem instances with goal size 1, one role is randomly selected as the goal; to generate problem instances with goal size 2, each goal of size 1 is augmented with a randomly selected role; and so on.

Table 1 gives performance data for the algorithms with all optimizations enabled. For the concrete backward algorithm, we report only numbers of nodes and edges in stage 1, because the nodes and edges in stage 2 are a subset of those in stage 1. Each data point is an average over 32 synthetic problem instances. We varied the size $|g|$ of the goal and the size $|T|$ of the finite type. Running time is rounded to the nearest 0.01 s. The running time and memory consumption of the concrete backward algorithm grow quickly as a function of $|T|$. For $|g| = 1$, the symbolic algorithm and the concrete algorithm have similar running times for $|T| \leq 3$, and the symbolic algorithm is faster for $|T| > 3$. For $|g| > 1$, the symbolic algorithm is faster than concrete backward algorithm when $|T| > 1$. For $|g| = 3$, we do not include performance results for the concrete backward algorithm for $|T| = 3$ (or higher), because the concrete algorithm did not terminate within 4 h for some policies. Data for the concrete forward algorithm is omitted from Table 1 because that algorithm runs significantly slower than the other two algorithms when $|T| = 1$ and runs out of memory when $|T| > 1$, because (1) that algorithm is exponential in the number of mixed roles (roles that appear positively in some preconditions and negatively in others) and the number of mixed roles is large (namely, 18), and (2) that algorithm tends to generate states containing unnecessarily many distinct instances of some role schemas.

To illustrate the distribution of running times, Fig. 3 shows the normalized running times of the symbolic algorithm on all the problem instances used for Table 1. For each goal size, the running times are normalized by dividing by the longest running time for that goal size. Observe that the running times do not follow a normal (Gaussian) distribution, and the distribution tends to have outliers on the right side, especially for larger goal sizes.

10.3. *Performance comparison for unparameterized policies*

These experiments evaluate the performance penalty of unnecessarily using symbolic analysis on unparameterized

Please cite this article in press as: Stoller SD, et al., Symbolic reachability analysis for parameterized administrative role-based access control, Computers & Security (2010), doi:10.1016/j.cose.2010.08.002

Table 1 – Running time (s) and memory consumption (MB) on parameterized policies. Node-i/Edge-i: number of nodes and edges generated in Stage i. K and M represent 10^3 and 10^6, respectively.

|g|	Symbolic			|T|	Concrete Backward		
	Time	Mem	Node-1/Edge-1 Node-2/Edge-2		Time	Mem	Node-1/Edge-1
1	0.01	5.92	27/291 21/70	1	0.00	5.45	27/291
				2	0.00	5.73	49/950
				3	0.01	7.06	75/2.2K
				4	0.03	9.10	109/4.4K
				5	0.11	12.4	149/7.7K
				6	0.12	17.2	194/12.5K
				7	0.39	23.9	246/19.1K
2	0.20	14.26	384/8385 362/1337	1	0.05	5.45	370/7.9K
				2	0.41	5.96	1.2K/47K
				3	1.96	7.07	3.0K/178K
				4	8.33	9.10	6.3K/512K
3	7.74	135	826/32K	1	1.79	5.45	3.4K/108.6K
				2	62.94	5.97	20.4K/1.2M

policies, by comparing the performance of the symbolic and concrete algorithms applied to the same unparameterized policies. We used the synthetic unparameterized policies used for Tables 2(a) in Stoller et al. (2007) which vary goal size (running times in Stoller et al., 2007 were obtained with a different implementation of the concrete algorithms, in C++). Table 2 shows the time and space requirements of the three algorithms on the unparameterized policies used for Table 2(a) in Stoller et al. (2007), which contain 32 roles, including 8 mixed roles. The performance of the symbolic algorithm and concrete backward algorithm is similar for these policies. As expected, the time and memory consumption of these algorithms increases quickly with $|g|$, while the cost of the concrete forward algorithm increases slowly with $|g|$.

10.4. *Evaluation of optimizations*

The performance data in Table 3 demonstrates the effect of some of the optimizations for synthetic problem instances with goal size 1. For the first row ("Baseline"), only slicing and early stopping are enabled. We included these two optimizations in the baseline, because without them, the running time is very high (e.g., several hours) for some problem instances. For each of the next four rows, the one specified optimization is also enabled (in addition to slicing and early stopping). For the last row ("Best"), all optimizations are enabled. These experiments use the same synthetic problem instances as in Table 1, except that we omitted one problem instance for which the baseline does not terminate in a reasonable amount of time. The optimizations evaluated in this table affect only stage 2, so the numbers of nodes and edges from stage 1 are not shown.

Observe from the table that all four of the evaluated optimizations reduce the size of the graph generated in stage 2. Eager revocation provides the most dramatic reduction in graph size and running time. The "Eager revocation", "Prune subsumed", "Remove subsumed", and "No revocation of non-negative roles" optimizations reduce the execution time by 99.7%, 94%, 76%, and 75%, respectively.

We draw the following conclusions from our experiments. (1) Slicing and early stopping are very effective in many cases and have low cost. (2) Eager revocation provides a very effective partial-order style reduction, dramatically running time and graph size. It has low per-transition overhead (linear in the size of a state representation). (3) Pruning subsumed states is also effective, but comes at a cost: subsumption checks are expensive. The algorithm we use is $O(N!)$, where N is the number of useless roles in a forward state. Pruning subsumed states works best when combined with removal of α-equivalent useless roles, which helps keep N small. For example, for some problem instances, we saw values of N as high as 13 when "Remove α-equivalent useless roles" was disabled; when it was enabled, we did not see values larger than 3. (4) "No revocation of non-negative roles" eliminates some transitions and hence reduces the size of the graph, but it

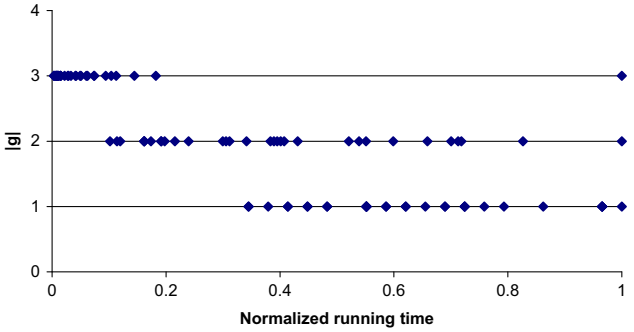

Fig. 3 – Normalized running times for symbolic algorithm.

Table 2 – Running time (s) and memory consumption (MB) on unparameterized policies.

|g|	Symbolic		Concrete Backward		Concrete Forward	
	Time	Mem	Time	Mem	Time	Mem
1	0.00	0.52	0.00	5.12	0.08	12.24
2	0.02	0.93	0.00	5.57	0.09	12.56
3	0.26	17.7	0.15	13.84	0.10	12.71
4	7.06	137.26	4.19	64.60	0.10	12.71
5	119.4	294.78	70.58	236.96	0.10	12.71

Please cite this article in press as: Stoller SD, et al., Symbolic reachability analysis for parameterized administrative role-based access control, Computers & Security (2010), doi:10.1016/j.cose.2010.08.002

Table 3 – Evaluation of optimizations on synthetic problem instances with goal size 1.

Optimization	Time	Mem	Node-2	Edge-2
Baseline	36.49	20.3	4521	6407
Eager revocation	0.12	5.71	18	59
No revoc. nonneg.	8.98	12.92	2234	3520
Prune subsumed	2.17	14.11	2588	3284
Remove subsumed	8.80	10.17	1544	2544
Best	0.01	5.92	21	27

increases the number of roles in the state, potentially making various operations, especially subsumption checks, more expensive. Also, the overhead of computing the set of non-negative roles for each backward state may be significant in some cases.

11. Related work

11.1. *Analysis of unparameterized ARBAC policies*

A significant difference between this paper and prior papers on reachability analysis for ARBAC, including Li and Tripunitara (2006), Sasturkar et al. (2006), Stoller et al. (2007), Jha et al. (2008), is that they consider only policies without parameters, so they are inapplicable to policies with parameters that range over infinite types, and they are inefficient when applied to policies with parameters over finite types that have been eliminated by exhaustive instantiation. Some general ideas in our prior work in Stoller et al. (2007) are also used here (e.g., backward search followed by forward search), but analysis of parameterized policies is significantly more difficult, requiring new algorithms and complexity results: the symbolic transition relations defined here are much more complicated than the concrete ones used in Stoller et al. (2007), the relationship between the two stages of the backward algorithm is different, new optimizations are needed for stage 2, different complexity parameters are used in the fixed-parameter tractability result, etc. Section 10 empirically compares our current work with Stoller et al. (2007).

11.2. *Analysis of other kinds of parameterized systems*

In general, parameterized systems have infinite-state spaces, and the reachability problem for them is undecidable. Many specialized techniques have been developed for verification of various kinds of parameterized systems. There are two broad, overlapping classes of parameterized systems. In the first class, parameters represent the number of components (e.g., processes) in a system. There are numerous sound but incomplete reachability algorithms for such parameterized systems (e.g., Clarke et al., 1997; Emerson and Namjoshi, 1995). In the second class, parameters represent data values that range over infinite or unbounded domains. There are numerous techniques that analyze abstractions of such parameterized systems, giving up either soundness or completeness. In some restricted cases, such as data-independent systems and timed automata, sound and complete algorithms for reachability are known (e.g., Sarna-Starosta and Ramakrishnan, 2003; Alur and Dill, 1994).

We are not aware of an existing symbolic reachability framework that can directly be applied for analysis of PARBAC, because of the following combination of features of the problem: (1) states are described by potentially unbounded sets of parameterized boolean variables corresponding to role membership facts (in other words, the number of roles in a state, hence the number of state variables, is unbounded, as in the first category of parameterized systems described above), (2) the parameterized boolean variables may be used both positively and negatively in preconditions of transitions, (3) the parameters of the boolean variables range over an infinite data type (as in the second category of parameterized systems above), and (4) transitions may introduce an unbounded number of these parameters (i.e., fresh variables) in the symbolic state graph (*cf.* the discussion of termination in Section 5). For example, work on verification of unbounded networks of processes, such as Emerson and Kahlon (2000), is not applicable because it assumes that each process's state ranges over a fixed finite set. Work on verification of data-independent systems, such as Sarna-Starosta and Ramakrishnan (2003), is not applicable because it assumes that fresh variables are not introduced during the search. Work on verification of cryptographic protocols, such as Blanchet and Podelski (2005), allows unbounded numbers of nonces (represented by variables that range over an unbounded domain) but is not applicable because it does not allow these variables to be used in negative preconditions of transitions. Work on verification using inductive assertions, such as Arons et al. (2001), heuristically constructs a candidate inductive invariant ϕ and calls a theorem prover to check whether ϕ is inductive and stronger than the property of interest, but is not applicable for a variety of reasons: the method can prove formulas containing universal but not existential quantifiers, so it can try to prove that a goal is unreachable but not that a goal is reachable (which would require existential quantification over the substitution, as in Definition 2), and if the heuristic method fails to prove that a goal is unreachable, we can draw no conclusions (in particular, we cannot conclude the goal is reachable); also, the heuristic cannot generate invariants that contain existential quantifiers, which are sometimes needed to accurately capture the effects of feature (4) above. Even if some existing parameterized verification framework could be applied, we would still need to define a symbolic transition relation for PARBAC, similar to Definition 5. Also, we are not aware of any fine-grained complexity results or fixed-parameter tractability results for such algorithms.

This paper extends our conference paper Stoller et al. (2009) in several ways. On the theory side, we extended the policy language to allow wildcards in negative preconditions, simplified the statement of Theorem 2, and proved undecidability of user-role reachability analysis for PARBAC. On the algorithm side, we developed three new optimizations: "remove α-equivalent useless roles", "prune subsumed states", and "no revocation of non-negative roles". On the implementation side, we implemented the search stack check, eager revocation, and the three new optimizations, re-ran the previous experiments, and added new experiments that measure the effectiveness of the optimizations. In the experiments reported in Table 3, the new optimizations together reduce the running time by an order of magnitude.

Please cite this article in press as: Stoller SD, et al., Symbolic reachability analysis for parameterized administrative role-based access control, Computers & Security (2010), doi:10.1016/j.cose.2010.08.002

12. Conclusion

This paper shows that user-role reachability analysis for ARBAC with parameters is undecidable, and presents a semi-decision procedure for this problem. The algorithm is symbolic and does not need to consider all instantiations of the parameters. It exploits the structure of PARBAC policies, constructing a backwards graph to prune the search space. We show that the algorithm terminates under realistic assumptions about the policy, and we present a fixed-parameter tractability result. We also developed and evaluated several optimizations. Future work on analysis for PARBAC includes algorithms guaranteed to terminate in more cases, efficient analysis of policies that do not satisfy separate administration or hierarchical role assignment, and analysis of policies that control changes to the role hierarchy. Future work beyond PARBAC includes analysis for trust management frameworks, such as SecPAL (Becker et al., 2007).

Acknowledgment

The authors thank Jason Crampton and the anonymous SACMAT 2009 reviewers for valuable comments and suggestions, and Yogesh Upadhyay for implementing the conversion from hierarchical to non-hierarchical policies.

REFERENCES

Alur R, Dill DL. A theory of timed automata. Theoretical Computer Science 1994;126(2):183–235.

Arons T, Pnueli A, Ruah S, Xu J, Zuck LD. Parameterized verification with automatically computed inductive assertions. In: Proceedings of the 13th international conference on computer aided verification (CAV). Lecture notes in computer science, vol. 2102. Springer; 2001. p. 221–34.

Bacon J, Moody K, Yao W. A model of OASIS role-based access control and its support for active security. ACM Transactions on Information and System Security (TISSEC) 2002;5(4):492–540.

Becker MY. Cassandra: flexible trust management and its application to electronic health records. Ph.D. thesis. University of Cambridge; 2005.

Becker MY. Specification and analysis of dynamic authorisation policies. In: Proc. 22nd IEEE computer security foundations symposium (CSF). IEEE Computer Society Press; 2009. p. 203–17.

Becker MY, Fournet C, Gordon AD. Design and semantics of a decentralized authorization language. In: Proc. 20th IEEE computer security foundations symposium (CSF). IEEE Computer Society Press; 2007. p. 3–15.

Blanchet B, Podelski A. Verification of cryptographic protocols: tagging enforces termination. Theoretical Computer Science 2005;333:67–90.

Blum A, Furst M. Fast planning through planning graph analysis. Artificial Intelligence 1997;90(1–2):281–300.

Clarke E, Grumberg O, Jha S. Verifying parameterized networks. ACM Transactions on Programming Languages and Systems 1997;19(5):726–50.

Crampton J. Authorizations and antichains. Ph.D. thesis. Birbeck College, University of London; 2002.

Crampton J. Understanding and developing role-based administrative models. In: Proc. 12th ACM conference on computer and communications security (CCS). ACM Press; 2005. p. 158–67.

Crampton J, Loizou G. Administrative scope: a foundation for role-based administrative models. ACM Transactions on Information and System Security (TISSEC) 2003;6(2):201–31.

Downey RG, Fellows MR. Fixed-parameter tractability and completeness I: basic results. SIAM Journal on Computing 1995;24(4):873–921.

Emerson EA, Kahlon V. Reducing model checking of the many to the few. In: International conference on automated deduction; 2000. p. 236–54.

Emerson EA, Namjoshi KS. Reasoning about rings. In: Proceedings of the 22nd ACM symposium on principles of programming languages (POPL). ACM Press; 1995. p. 85–94.

Erol K, Nau DS, Subrahmanian VS. Complexity, decidability and undecidability results for domain-independent planning: a detailed analysis. Technical Report CS-TR-2797, UMIACS-TR-91-154, SRC-TR-91-96. Computer Science Department and Institute for Systems Research, University of Maryland; 1991a.

Erol K, Nau DS, Subrahmanian VS. Complexity, decidability and undecidability results for domain-independent planning: a detailed analysis. Artificial Intelligence 1991b;76:75–88.

Evered M, Bögeholz S. A case study in access control requirements for a health information system. In: Second Australasian information security workshop (AISW), vol. 32. Australian Computer Society; 2004. p. 53–61.

Ge M, Osborn S. A design for parameterized roles. Data and applications security XVIII. Status and Prospects; 2004.

Giuri L, Iglio P. Role templates for content-based access control. In: Proc. 2nd ACM workshop on role based access control (RBAC'97). ACM; 1997. p. 153–9.

Harrison MA, Ruzzo WL, Ullman JD. Protection in operating systems. Communications of the ACM 1976;19(8):461–71.

Jha S, Li N, Tripunitara M, Wang Q, Winsborough W. Towards formal verification of role-based access control policies. IEEE Transactions on Dependable and Secure Computing 2008;5(4):242–55.

Kern A, Schaad A, Moffett J. An administration concept for the enterprise role-based access control model. In: Proc. 8th ACM symposium on access control models and technologies (SACMAT). ACM Press; 2003. p. 3–11.

Li N, Mao Z. Administration in role based access control. In: Proc. ACM symposium on information, computer and communications security (ASIACCS); 2007. p. 127–38.

Li N, Tripunitara MV. Security analysis in role-based access control. ACM Transactions on Information and System Security (TISSEC) 2006;9(4):391–420.

Lupu E., Sloman M. Reconciling role based management and role based access control. In: Proc. 2nd ACM workshop on role based access control; 1997. p. 135–41.

Oh S, Sandhu R, Zhang X. An effective role administration model using organization structure. ACM Transactions on Information and System Security (TISSEC) 2006;9(2):113–37.

Sandhu R, Bhamidipati V, Munawer Q. The ARBAC97 model for role-based administration of roles. ACM Transactions on Information and System Security (TISSEC) 1999;2(1):105–35.

Sandhu R, Coyne E, Feinstein H, Youman C. Role-based access control models. IEEE Computer 1996;29(2):38–47.

Sandhu R, Munawer Q. The ARBAC99 model for administration of roles. In: Proceedings of the 18th annual computer security applications conference; 1999. p. 229–38.

Sarna-Starosta B, Ramakrishnan CR. Constraint-based model checking of data-independent systems. In: Proceedings of the 5th international conference on formal engineering methods

(ICFEM). Lecture notes in computer science, vol. 2885. Springer; 2003. p. 579–98.
Sasturkar A, Yang P, Stoller SD, Ramakrishnan CR. Policy analysis for administrative role based access control. In: Proceedings of the 19th IEEE computer security foundations workshop (CSFW). IEEE Computer Society Press; 2006. p. 124–38.
Schaad A, Moffett JD. A lightweight approach to specification and analysis of role-based access control extensions. In: Proc. 7th ACM symposium on access control models and technologies (SACMAT). ACM Press; 2002. p. 13–22.
Stoller SD, Yang P, Gofman M, Ramakrishnan CR. Symbolic reachability analysis for parameterized administrative role based access control. In: Proc. 14th ACM symposium on access control models and technologies (SACMAT). ACM Press; 2009. p. 165–74.
Stoller SD, Yang P, Ramakrishnan CR, Gofman MI. Efficient policy analysis for administrative role based access control. In: Proceedings of the 2007 ACM conference on computer and communication security (CCS). ACM Press; 2007. p. 445–55.

Scott D. Stoller is a professor in the Computer Science Department at Stony Brook University. He received his Bachelor's degree in Physics, summa cum laude, from Princeton University in 1990 and his Ph.D. degree in Computer Science from Cornell University in 1997. He received an NSF CAREER Award in 1999, an ONR Young Investigator Award in 2002, and (with two of his students) the 2005 Haifa Verification Conference Best Paper Award. He is a member of the team that won the NASA Turning Goals Into Reality Award for Engineering Innovation in 2003. He is the author or co-author of over 80 refereed research publications.

Ping Yang is an assistant professor in Computer Science Department at State University of New York at Binghamton. Her research areas are information and systems security, formal methods, and privacy. She holds a Ph.D. from Stony Brook University, an M.E from Chinese Academy of Sciences, and a B.S. from Sun Yat-Sen University.

Mikhail I. Gofman is a Ph.D. student at State University of New York at Binghamton.

C.R. Ramakrishnan is an associate professor in the Computer Science Department at Stony Brook University. He received his Ph.D. in Computer Science from Stony Brook in 1995. He holds M.Sc (Tech.) in Computer Science and M.Sc. (Hons.) in Physics from BITS, Pilani, India. He has been on the faculty in the CS Department at Stony Brook since 1997. His areas of interest include Formal Methods, Logic Programming, Programming Languages, and Security. He is the author or co-author of over 80 refereed research publications.

Please cite this article in press as: Stoller SD, et al., Symbolic reachability analysis for parameterized administrative role-based access control, Computers & Security (2010), doi:10.1016/j.cose.2010.08.002