RBAC-PAT: A Policy Analysis Tool for Role Based Access Control [*]

Mikhail I. Gofman[1], Ruiqi Luo[1], Ayla C. Solomon[2], Yingbin Zhang[1], Ping Yang[1], and Scott D. Stoller[3]

[1] Dept. of Computer Science, Binghamton University, NY 13902, USA
[2] Dept. of Computer Science, Wellesley College, Wellesley, MA 02481, USA
[3] Dept. of Computer Science, Stony Brook University, Stony Brook, NY 11794, USA

Abstract. Role-Based Access Control (RBAC) has been widely used for expressing access control policies. Administrative Role-Based Access Control (ARBAC) specifies how an RBAC policy may be changed by each administrator. Because sequences of changes by different administrators may interact in unintended ways, it is often difficult to fully understand the effect of an ARBAC policy by simple inspection. This paper presents RBAC-PAT, a tool for analyzing RBAC and ARBAC policies, which supports analysis of various properties including reachability, availability, containment, weakest precondition, dead roles, and information flows.

1 Introduction

Role-Based Access Control (RBAC) is widely used for expressing access control policies in areas such as health care and finance. In large organizations, RBAC policies are often managed by multiple administrators with varying authority. An Administrative Role Based Access control (ARBAC) policy specifies how each administrator may change the RBAC policy. Changes by one administrator may interact in unintended ways with changes by other administrators. Consequently, the effect of an ARBAC policy is hard to understand by manual inspection alone.

Policy analysis helps systems designers and administrators understand and debug policies. This paper presents RBAC-PAT, a tool for analyzing various properties of RBAC and ARBAC policies, including (1) *reachability*: e.g., can user u be assigned to role r (called a "goal")? (2) *availability*: e.g., is user u always a member of role r? (3) *role-role containment*: is every member of role r_1 also a member of role r_2? (4) *weakest precondition*: what are the minimal sets of initial roles that enable a user to get added to roles in the goal? (5) *dead roles*: what roles cannot be assigned to any user? and (6) *information flow*: can information flow from object o_1 to object o_2? For properties (1)–(5), the analysis considers all RBAC policies reachable from a given initial RBAC policy by actions allowed by a given ARBAC policy for a given set of administrators.

[*] This work was supported in part by NSF Grants CNS-0831298 and CNS-0627447 and ONR Grant N00014-07-1-0928.

2 Preliminaries

Role Based Access Control. The central notion of RBAC is that users are assigned to appropriate roles, and roles are granted appropriate permissions. Role hierarchy is a partial order on the set of roles. For example, $GradStudent \succeq Student$ means that role $GradStudent$ is senior to role $Student$, *i.e.*, every member of $GradStudent$ is also implicitly a member of $Student$.

Administrative Role Based Access Control. ARBAC97 [2] controls changes to the user-role assignment, the permission-role assignment and the role hierarchy. Authority to assign users to roles and revoke users from roles are specified by the can_assign and can_revoke relations, respectively. For example, $can\_assign(DeptChair, Grad \land \neg RA, TA)$ specifies that the administrative role $DeptChair$ has authority to assign a user who is a member of $Grad$ but not a member of RA to the role TA. A role that appears in a positive precondition, like $Grad$ in this example, is called a *positive role*; similarly, RA is a *negative role* in this example. Authority to assign and revoke permissions is controlled similarly.

3 Tool Description

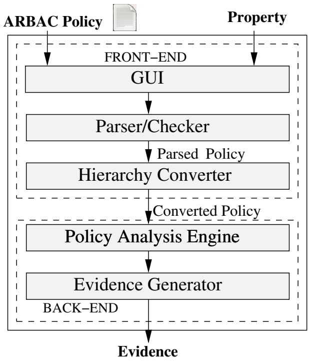

Fig. 1. System architecture.

The architecture of RBAC-PAT is shown in Figure 1. Below, we describe its main components.

3.1 Hierarchy Converter

This components converts hierarchical policies into non-hierarchical policies for analysis [3].

3.2 Policy Analysis Engine

Reachability, availability, role-role containment, and weakest precondition. RBAC-PAT implements algorithms we developed for user-role reachability analysis of ARBAC with and without the separate administration restriction [4]. Separate administration requires that administrative roles and regular roles are disjoint. Our algorithms for the other analysis problems are either similar to these algorithms or reduce the problem to user-role reachability analysis [4, 3]. We developed forward and backward algorithms for user-role reachability with separate administration and analyzed their parameterized complexity. The idea of parameterized complexity is to identify an aspect of the input that makes the problem computationally hard, introduce a parameter k to measure that aspect, and develop an algorithm that may have high complexity in terms of k, but is polynomial in the overall input size when the value of k is fixed. Such an algorithm is said to be *fixed parameter tractable* with respect to k (“FPT w.r.t. k”).

In the forward algorithm, a simple backward slicing transformation eliminates roles and rules irrelevant to the given goal. Next, a *reduced state graph* is constructed; reachability is determined from it. Each node corresponds to an RBAC policy; each edge corresponds to a change allowed by the ARBAC policy. The following reduction is applied: (1) Transitions that revoke non-negative roles or add non-positive roles are prohibited; (2) Transitions that add non-negative roles or revoke non-positive roles are called *invisible* transitions and get combined with a preceding visible transition to form a single composite transition. The forward algorithm is FPT w.r.t. the number of *mixed roles*, i.e., roles that are both positive and negative. This number is usually significantly smaller than the total number of roles. For example, in ARBAC policies we developed for a university and a health care facility, the percentage of mixed roles is less than 25%.

The backward algorithm has two stages. The first stage uses backward search from the goal to construct a directed graph G. Each node in G is a set of roles, and each edge is labeled with a *can_assign* rule and corresponds to a role assignment action allowed by that rule. However, some negative preconditions of *can_assign* transitions cannot be evaluated during the backward search. The second stage is a forward search that annotates G with the additional information needed to check those preconditions, namely, sets of irrevocable roles that might be left in the state by previous transitions. For ARBAC policies with at most one positive precondition per rule, our backward algorithm is FPT w.r.t. the number of irrevocable roles.

We developed a forward algorithm for analysis of ARBAC without separate administration that is FPT w.r.t. the number of mixed roles and the number of users. We also identified a condition called *hierarchical role assignment* that is often satisfied in practice, and we showed that our algorithms that assume separate administration give accurate results for policies satisfying this condition. Informally, the condition is that an administrator cannot assign users to administrative roles that are not junior to his own administrative role.

ARBAC policy analysis problems could be solved using general-purpose finite-state verification tools, but those tools lack the specialized optimizations in our algorithms and would be asymptotically less efficient for some families of policies. A detailed comparison with related work on verification and security policy analysis appears in [4].

RBAC-PAT computes policy statistics, including the numbers of mixed roles and irrevocable roles, and checks whether separate administration and hierarchical role assignment hold. RBAC-PAT uses this information to try to choose the most appropriate analysis algorithms for a given analysis problem. In cases where separate administration restriction is satisfied and it is unclear whether the forward or the backward algorithm will be faster, RBAC-PAT prompts the user to choose between these algorithms.

Dead role analysis. We developed an algorithm to detect *dead roles* in an ARBAC policy, i.e., roles that cannot be assigned to any user. Dead roles might indicate flaws in the policy. A straightforward algorithm for detecting dead roles is: for every user u_i, compute a set R_i of roles that can be assigned to u_i until all roles have been assigned to some user or all users have been considered; roles not in $\bigcup R_i$ are dead. If the policy satisfies separate administration, the following optimizations are applied: (1) a slicing transformation is used to eliminate roles and rules irrelevant to unassigned roles, and only users with distinct sets of initial roles are considered; and (2) at each step, we consider the user that can potentially be assigned to the most currently unassigned roles.

Information flow analysis. Information flow analysis helps administrators understand the information flows allowed by an RBAC policy. Information can flow directly from object o_1 to object o_2 if there exists a user that can read from o_1 and write to o_2. Osborn [1] proposed an algorithm for constructing an information flow graph from an RBAC policy, in which an edge $o_1 \rightarrow o_2$ specifies that information can flow directly from o_1 to o_2. We improve this algorithm by eliminating infeasible intermediate edges, for example, edges resulting from roles that have not been assigned to any users. RBAC-PAT also supports information flow queries such as "can information flow, directly or transitively, from object o_1 to object o_2?"

Evidence generation. RBAC-PAT provides evidence that shows why a property holds or is violated. For example, if the answer to a reachability analysis query is yes, RBAC-PAT provides a sequence of administrative actions that leads to the specified role assignment, and highlights the corresponding ARBAC rules in the policy.

3.3 Case Studies

We developed RBAC and ARBAC policies for a university and a health care facility. Here are some sample properties for the university policy: (1) *User-role reachability*: can a user initially in role $DeptChair$ and a user initially in role $Undergrad$ together assign the latter user to $HonorsStudent$? (2) *Weakest Precondition*: what are the weakest preconditions for an administrator initially in $DeptChair$ to assign a user to $HonorsStudent$? (3) *Role-role containment*: is TA contained in $Grad$? (4) *Information flow query*: can information flow from $GradeBook$ to $DeptReport$? RBAC-PAT terminates in at most 0.19 second for all queries we tried. RBAC-PAT also helped uncover some flaws in the original university policy, for example, a place where we accidentally used $Student$ instead of $Undergrad$ and places where we forgot to take role hierarchy into account, *e.g.*, places where we forgot that $Provost$ inherits from $Staff$. Further, in order to validate our FPT results and explore the practical performance of the algorithms, we applied RBAC-PAT to reachability analysis of hundreds of randomly generated policies [4]. For the policies containing 13 reachable mixed roles and 32 roles, RBAC-PAT generates at most 232320 states and 2900920 transitions, and terminates in 8.6 hours. For the policies containing 5 reachable mixed roles and 500 roles, RBAC-PAT generates at most 510 states and 2550 transitions, and terminates in 155 minutes.

Acknowledgement. We thank C. R. Ramakrishnan, Jian He, Yogesh Upadhyay, Pinki Pasad, and Joel St. John for their contributions to the tool development.

References

1. S. Osborn. Information flow analysis of an RBAC system. In *SACMAT*, pages 163 – 168, 2002.
2. R. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for role-based administration of roles. *TISSEC*, 2(1):105–135, 1999.
3. A. Sasturkar, P. Yang, S. D. Stoller, and C. Ramakrishnan. Policy analysis for administrative role based access control. In *IEEE CSFW*, pages 124–138, 2006.
4. S. Stoller, P. Yang, C. R. Ramakrishnan, and M. Gofman. Efficient policy analysis for administrative role based access control. In *CCS*, pages 445–455, 2007.