Policy Analysis for Self-administrated Role-based Access Control

Anna Lisa Ferrara[1], P. Madhusudan[2], and Gennaro Parlato[3]

[1] University of Bristol, UK
[2] University of Illinois, USA
[3] University of Southampton, UK

Abstract. Current techniques for security analysis of administrative role-based access control (ARBAC) policies restrict themselves to the *separate administration* assumption that essentially separates administrative roles from regular ones. The naive algorithm of tracking all users is all that is known for the security analysis of ARBAC policies without separate administration, and the state space explosion that this results in precludes building effective tools. In contrast, the separate administration assumption greatly simplifies the analysis since it makes it sufficient to track only one user at a time. However, separation limits the expressiveness of the models and restricts modeling distributed administrative control. In this paper, we undertake a fundamental study of analysis of ARBAC policies without the separate administration restriction, and show that analysis algorithms can be built that track only a bounded number of users, where the bound depends only on the number of administrative roles in the system. Using this fundamental insight paves the way for us to design an involved heuristic to further tame the state space explosion in practical systems. Our results are also very effective when applied on policies designed under the separate administration restriction. We implement our techniques and report on experiments conducted on several realistic case studies.

1 Introduction

Role-based access control (RBAC) has emerged in recent years as a simple and effective access control mechanism for large organizations [6, 17]. RBAC is the most popular model for large organizations [15, 1] and can implement a variety of MAC and DAC policies. RBAC is also a popular mechanism in assigning user privileges in computer systems, and is supported in several systems including Microsoft SQL Servers [2], Microsoft Active Directory (AGDLP) [3], SELinux, and Oracle DBMS. It simplifies policy specification and the management of user rights using a two tier management— it groups users into roles and assigns permissions to each role. In any organization, roles can be associated with job functions and hence role-permission assignments are relatively stable, while user-role assignments change quite frequently (e.g., personnel moving across departments, reassignment of duties, etc.). Managing the user-role permissions is significantly easier than managing user rights individually.

Administrative role-based access control (ARBAC) [16, 18] is a policy mechanism for controlling how changes can be made to the RBAC policy by various

administrators. ARBAC policies are generally composed of three sub-policies: one controlling user-role assignment (URA), one controlling permission-role assignment (PRA), and one dealing with role-role assignments (RRA). A set of administrative roles is defined, users are assigned administrative roles, and a URA mechanism specifies when a user in an administrative role can grant or revoke role-assignments to users, including administrative roles as well. (PRA and RRA mechanisms are less common; we will not consider them in this paper.)

Security analysis of ARBAC systems is recognized as an extremely important problem, as an analysis tool can help designers to determine whether their policies meet required security properties. The developers of the administrative systems have an intended security goal that they want to enforce through the policy, and they set up the roles and administrative rules (formalized in ARBAC) to realize these intentions. Even though ARBAC policies are specified using simple administrative rules that intuitively meet the designers' intentions, it is often very difficult to find out subtle behaviours, yielding security breaches. This happens mainly because it is hard to foresee the whole effect of multiple administrative changes and their interaction. Security breaches include privilege escalation (e.g. an employee of a lower rank gaining access to resources meant for a higher rank), violation of separation of duty constraints that model conflict of interest (e.g., a user u simultaneously holding roles r_1 and r_2), etc.

In most cases the security analysis problem for ARBAC system can be phrased as a *role-reachability* problem: given a set of users, can any user in this set gain access to a given role `goal` using the ARBAC policy rules? Such a technical problem seems to be the most useful security question for ARBAC systems. Indeed, almost all interesting security questions can be *reduced* to the above problem (see [10, 22, 4]). Unfortunately, it is very hard to verify security of ARBAC policies precisely. The main source of complexity is that simulating the system to examine the entire set of reachable states causes an explosion in state-space, as it requires tracking *all* users' role-memberships. In a system with $|U|$ users and $|R|$ roles, this means exploring $O((2^{|R|})^{|U|})$ configurations, in the worst case! If the number of users is very small, this can be achieved in practice using model-checking techniques that use symbolic representations of state-spaces (like BDDs), but any realistic scenario would involve thousands of users, making this completely intractable. For those reasons researchers have turned to consider either restricted scenarios or abstraction techniques [22, 9, 4].

One crucial assumption that has been used to tackle the above problem is that of *separate administration* [22]. This essentially assumes that administrative roles and regular roles are disjoint so that administrative operations only affect regular roles and assignment/revocation of administrative permissions are not considered. Such a restriction greatly simplifies the analysis since is then sufficient to consider only the evolution of a *single user* (at a time) as opposed to tracking all users. On the other hand, the separate administration restriction severely limits the expressiveness of the model [22] and precludes the use of frameworks such as SARBAC [5], UARBAC [13] and *Oracle* which do not assume such a limitation. In particular, the separate administration restriction

is increasingly inappropriate in a world where administration is getting more and more distributed. For example, users belonging to regular roles in modern organizations are often administrators of several resources and have the right to grant administrative privileges to other users on these resources (e.g., a user may have administrative powers over access to their rooms, their files, etc., and have the ability to allow administrative access to other users for these resources). In such situations, it is important that administrative permissions are allowed to be granted and revoked. The separate administration restriction precludes the modeling of such scenarios.

Beyond separate administration: While much research has considered the separate administration model, the naive algorithm of tracking all users is all that is known for the security analysis of ARBAC policies without separate administration, and the state space explosion that this results in precludes building effective tools. The only relevant work here that we know of is recent work by Ferrara et al [4] that proposes using abstractions techniques; however, while this is effective in proving correct policies correct, it is not precise and in particular cannot generate attacks when the abstraction fails to prove safety.

Several basic questions of security analysis for policies without separate administration are still open: Does an analysis of these policies necessarily have to track all users? The RBAC model provides a layer of abstraction where users in a system are grouped together into roles; can this be exploited to perform an analysis that is completely independent of the number of users in the system (but dependent on the number of roles)? Finally, can security analysis for ARBAC be made scalable, given that multiple users may need to be tracked?

The goal of this paper is to answer fundamental theoretical questions on the security analysis of general ARBAC systems, and exploiting the insights gained, provide scalable tools to analyze expressive ARBAC policies.

As a **first contribution**, we show that the security analysis of ARBAC systems can be achieved completely independent of the number of users. More precisely, we show that the number of users that an analysis needs to track simultaneously depends only on the number of *administrative* roles (k), and, in fact, it is always sufficient to track $k+1$ users simultaneously. The proof of the theorem is quite non-trivial; a user u may reach a target role just using the administrators currently in the system (in which case, tracking one user would suffice). However, users in certain administrative roles may not exist and the system can evolve to drop administrative privileges of users; further the user u may *collude* with a subset of users, who could become administrators of the right kind and help the user u reach the target role. (When administrators are computer programs that automatically evolve, as is common in some scenarios, this does not even mean collusion, and is just exploitation of these software administrators). The fundamental theorem we prove shows that, however complex these collusions are, tracking $k + 1$ users always suffices.

The *proof* of the fundamental theorem leads to significant insights on the users that need to be tracked by an analysis. As a **second contribution**, we utilize such insights in order to build a trimming procedure that takes as input

an ARBAC system with a target role, and reduces it to an often much smaller ARBAC system, that has significantly smaller sets of users, administrative roles, and rules.

The effectiveness of our procedure is amplified by an *aggressive pruning* algorithm which we propose as our **third contribution** and whose applicability is of independent interest (it is effective also for separated administration scenarios). Static pruning techniques have been previously proposed [10, 22, 4] in order to reduce the state space to explore. The basic idea behind those techniques is that of removing roles and administrative rules from the policies that are immaterial to the reachability of role `goal`.

As a **fourth contribution**, we evaluate our techniques on the policies (without separate assumption) in [19]— these systems are initially populated by thousands of users and the naive precise solution known will track all users and would fail pathetically. Our procedure significantly simplifies the policies considered, reducing the number of effective users that need to be tracked to be very small (often 3 to 4), which then leads us to solve the security problem for them. This result is the first we are aware of that shows that security analysis can be feasible without separate administration restriction.

Moreover, we evaluate our aggressive pruning technique on the realistic policies and test cases considered in [9] (with separate assumption). These policies are considered very complex to analyze given their huge number of roles and rules ranging respectively from 600 to 40k and 1k to 200k. Only under-approximate techniques have been found successful on those policies [9], which of course can find shallow errors but are not complete. We experimentally show that our pruning technique, in contrast to ones known, is extremely effective. Indeed, it reduces most of the aforementioned complex policies to equivalent systems (in terms of role-reachability) having as few roles (rules, resp.) as a single one.

Related Work. Besides the work already cited above, there are a few other works that are related to this paper. Li and Tripunitara [14] have studied the security analysis problem of ARBAC systems and identified fragments and restricted queries that can be solved in polynomial time. Sasturkar et al [20] have showed that the problem is PSPACE-complete, that most restrictions still are NP-hard, and some very restricted cases can be solved in polynomial time. Jha et al. [10] compared the use of model checking and first order logic programming for the security analysis of ARBAC and concluded that model checking is a promising approach for security analysis. Stoller et al [22] identify the *fixed-parameter complexity* of the problem, and show that the problem is tractable if we fix the number of roles. Their techniques are implemented in the RBAC-PAT tool [7]. In more recent work, Stoller et al have extended the ARBAC model to parameterized ARBAC that allows conditions that depend on parameters [21].

2 Preliminaries

Role Based Access Control. An RBAC policy is a tuple $\langle U, R, P, UA, PA, \succ \rangle$ where U, R, and P are finite sets of *users*, *roles*, and *permissions*, respectively,

$UA \subseteq U \times R$ is the *user-role assignment* relation, and $PA \subseteq P \times R$ is the *permission-role assignment* relation. A pair $(u, r) \in UA$ represents the membership of user u to role r, and $(p, r) \in PA$ means that role r has permission p. Roles are related by a *hierarchy relation* defined by the partial order $\succ$. The hierarchy relation allows to inherit permissions by one role from another in the hierarchy: for any two roles $r, r' \in R, r$ inherits all permissions of r' whenever $r \succ r'$. However, in the rest of the paper we restrict ourself to consider only RBAC policies with an empty hierarchy relation. From an analysis point of view it is always possible to transform a hierarchical RBAC system into one without hierarchy that preserves the reachability of its roles (see [20] for the description of such a transformation). Furthermore, we concentrate our analysis on user-role administration. Thus, in the rest of the paper we refer to an RBAC policy as a tuple $\langle U, R, UA \rangle$.

Administrative Role Based Access Control. ARBAC policies [16, 18] describe a model for role-based administration of RBAC. They are composed of three modules: URA user-role administration, PRA permission-role administration, and RRA role-role administration. In this paper we focus on the user-role administration model which is of most practical interest. In practice user-role membership changes are the most frequent [11] when compared with changes in permission-role and role-role relationships. The URA policy describes how the user-role assignment relation UA of an RBAC policy can be modified in the evolution of the system. A central role is played by the set of administrative roles AR: the users in AR (called *administrators*) can assign and/or revoke roles to other users.

The assignment of a user to a role is subject to a *precondition* which depends only on the user's role-memberships. A precondition is a Boolean formula written as a conjunction of literals, where each literal is either in positive form r or in negative form $\neg r$, for some role r in R. To simplify the notation we represent each precondition with two subsets Pos and Neg of R. The set Pos represents the set of all roles the user must be in, as opposed to Neg which is the set of all roles which the user must not belong to.

The permission to assign users to roles is specified by a set of tuples

$$can\_assign \subseteq AR \times 2^R \times 2^R \times R.$$

The meaning of a can-assign tuple $(admin, Pos, Neg, r) \in can\_assign$ is that a member of the administrative role $admin \in AR$ can assign a user whose current role-membership satisfies the precondition (Pos, Neg) to the role $r \in R$. (In the rest of the paper we always assume that $Pos \cap Neg = \emptyset$).

Rules to remove users from roles are defined by a set

$$can\_revoke \subseteq AR \times R.$$

If $(admin, r) \in can\_revoke$ then a member of the administrative role $admin \in AR$ can revoke the membership of a user from role r (regardless the user role-membership). In the rest of the paper we refer to an URA model as a pair $\langle can\_assign, can\_revoke \rangle$.

Separate Administration Restriction. Under the separate administration restriction, the set of administrative roles will never appear as target of a can-assign or can-revoke rule. In other words, it is intrinsically assumed that administrators will never change their membership to administrative roles. We relax the separate administration restriction and allow administrative roles to be part of the set of roles R, i.e., $AR \subseteq R$.

ARBAC Systems. In this section we describe ARBAC systems as state-transition systems, and define the role-reachability problem for them. An ARBAC system is a tuple $\mathcal{S} = \langle U, R, UA, can\_assign, can\_revoke\rangle$ where $\langle U, R, UA\rangle$ is an RBAC policy and $\langle can\_assign, can\_revoke\rangle$ is an URA model over the set of roles R. A *configuration* of $\mathcal{S}$ is any user-role assignment relation $UR \subseteq U \times R$. A configuration UR is *initial* if $UR = UA$. Given two $\mathcal{S}$ configurations UR and UR', there is a *transition* from UR to UR' with rule $m \in (can\_assign \cup can\_revoke)$, denoted $UR \xrightarrow{m} UR'$, if there is an *administrator ad* and an administrative role *admin* with $(ad, admin) \in UR$ and a user $u \in U$, and one of the following holds:
[**can-assign move**] $m = (admin, P, N, r)$, $P \subseteq \{t \,|\, (u,t) \in UR\}$, $N \subseteq R \setminus \{t \,|\, (u,t) \in UR\}$, and $UR' = UR \cup \{(u,r)\}$;
[**can-revoke move**] $m = (admin, r)$, $(u,r) \in UR$, and $UR' = UR \setminus \{(u,r)\}$.
A *run* of $\mathcal{S}$ is any finite sequence of $\mathcal{S}$ transitions $\pi = c_1 \xrightarrow{m_1} c_2 \xrightarrow{m_2} \ldots c_n \xrightarrow{m_n} c_{n+1}$ for some $n \geq 0$, where c_1 is an *initial* configuration of $\mathcal{S}$. An $\mathcal{S}$ configuration c is reachable if c is the last configuration of an $\mathcal{S}$ run.

Definition 1 (Role-reachability Problem). *For any role $r \in R$, r is* reachable *in $\mathcal{S}$ if there is an $\mathcal{S}$ reachable configuration UR such that $(u,r) \in UR$, for some $u \in U$. Given an ARBAC system $\mathcal{S}$ over the set of roles R and a role* `goal` $\in R$, *the* role-reachability problem *asks whether* `goal` *is reachable in $\mathcal{S}$.*

3 Bounding the Number of Users to Track

In this section we show that the role-reachability problem for ARBAC is solvable by tracking at most $k+1$ users, where k is the number of administrative roles.

Theorem 1. *Let $\mathcal{S} = \langle U, R, UA, can\_assign, can\_revoke\rangle$ be an ARBAC system with k administrative roles. If a role* `goal` $\in$ `R` *is reachable in $\mathcal{S}$ then there exists a run of $\mathcal{S}$ in which* `goal` *is reachable and at most $k+1$ users change their role-combination.*

Proof. For any run $\pi = c_1 \xrightarrow{m_1} c_2 \xrightarrow{m_2} \ldots c_n \xrightarrow{m_n} c_{n+1}$ of $\mathcal{S}$, we denote with $\rho(\pi) = admin_1, admin_2, \ldots, admin_n$ the sequence of administrative roles where $admin_j$ is the administrative role used in the j'th transition of π, i.e., for every $j \in [1,n]$, either $m_j = (admin_j, P_j, N_j, t_j)$ or $m_j = (admin_j, t_j)$. A user u is *engaged* in π iff there exists at least a transition in π that changes the role-combination of u. Moreover, u is *essential* in π if, u is engaged and for some $j \in [1,n]$, u is the only user in $admin_j$ in c_j. We denote with $index_\pi(u)$ the greatest $j \in [1,n]$ such that u is the only user in role $admin_j$ in c_j.

We now show that, for each run π of $\mathcal{S}$ in which role `goal` is reachable by a user (say *target*), it is possible to construct another run π' having at most

$k+1$ engaged users. We assume that *target* reaches role `goal`, for the first time, in the last configuration of π. We obtain π' from π by repeatedly applying the following rules.

Simplification rules: Let $\pi_0 = \pi$, and π_i be the run obtained after i steps.

1. If π_i contains an engaged user, but *target*, which is not essential, then pick one of them, say u, and remove from π_i all transitions changing u's role-combination.
2. If all engaged users in π_i are essential, then pick one of them, say u, such that $u \neq target$, and there is a transition m_j changing u's role-combination, where $j \geq \ell$ and $\ell = index_{\pi_i(u)}$. Then, remove from π_i all transitions changing u's role-combination after configuration c_ℓ.

Notice that the simplification process eventually terminates as we reduce the length of the run at each step. Furthermore, a simplification step always produces a run provided π_i is itself a run. The key observation to prove this property is that we always guarantee to leave a user in any administrative role to fire any move in π_i.

To conclude the proof, we show that any of such run π' has at most $k+1$ engaged users. Since each engaged user u in π' (but *target*) is essential and no transition changing u's role-combination after $c_{index_{\pi'}(u)}$ exists, it holds that for any two distinct engaged users u_1 and u_2 in π' (both different from *target*), $admin_{j_1} \neq admin_{j_2}$, with $j_1 = index_{\pi'}(u_1)$ and $j_2 = index_{\pi'}(u_2)$. Thus, the number of engaged users in π' is at most equal to the number of administrative roles in $\mathcal{S}$ plus one (that represents user *target*). □

4 Reducing the Number of Users to Track

Theorem 1 gives an upper-bound on the number of users to track to solve the role-reachability problem. Although the number of users to engage is generally much smaller than the number of users in the system, in practice even tracking few users can be unfeasible, hence it is extremely desirable to reduce as much as possible such a parameter. From a theoretical viewpoint such a bound is tight, but it is unlikely that real world ARBAC instances incur such intricate worst case scenarios. Thus, the main objective of this section is to devise new heuristics to reduce the number of administrative roles, hence the number of users to track.

Our proposal is to provide sufficient conditions to eliminate administrative roles. An administrative role is *immaterial* if there is a user that can belong to that role forever without affecting the reachability of any other role. In particular, we first identify two criteria for an administrative role to be immaterial; then we transform the policy in such a way that immaterial administrative roles become regular ones. For this purpose, we add to the system a fresh administrative role, called *super*, and a new user whose role-membership is the sole role *super*. The first component of any assignment/revocation rule administrated by an immaterial administrative role *admin* is replaced with *super*. This has the effect of making *admin* a regular role (i.e., a role without administrative permissions). More formally, we transform all can-assign moves $(admin, P, N, t) \in can\_assign$

into $(super, P, N, t)$, and similarly each can-revoke rule $(admin, t) \in can\_revoke$ into $(super, t)$. We now present two sufficient conditions which lead to immaterial administrative roles.

The first sufficient condition for immaterial administrative role is as follows. Let *admin* be an administrative role which does not appear in negative form in any precondition, and that initially contains a user, say u. Since the removal of u from *admin* will not allow to fire more rules, we can impose that u will never be removed from *admin* making it immaterial.

For the second condition we resort to Theorem 1. If there are at least $k+2$ users sharing the same role-membership, we can think that one of them will not be engaged in any run, making immaterial all administrative roles which those users are member of. As a side note, it is safe to keep in the system at most $k+1$ users for each role-combination, the remaining ones which we denote as *spare* users can be eliminated.

Notice that, the more users share the same role-combination the more the second condition becomes effective. Therefore, our approach benefits from the use of any technique that may increase the number of users having the same role-combination. For instance, we can employ pruning techniques that transform a system in an equivalent one (in terms of the reachability of role `goal`) by eliminating roles and rules. Pruning techniques have been first introduced in [10] and proved (in some case) useful to reduce the state-space to analyze [10, 22, 4].

```
ReduceAdmin(S, goal)
 Ŝ' ← S;
 do
    Ŝ ← Ŝ';
    Ŝ' ← Pruning( Ŝ', goal );
    Ŝ' ← Immaterial( Ŝ' );
    Ŝ' ← Spare( Ŝ' );
 while (Ŝ ≠ Ŝ');
 return( Ŝ' );
```

Fig. 1. ReduceAdmin.

Fig. 1 shows algorithm ReduceAdmin we propose to eliminate the immaterial administrative roles.

ReduceAdmin takes as input a pair $(\mathcal{S}, \texttt{goal})$, where $\mathcal{S}$ is an ARBAC system and `goal` is a role of $\mathcal{S}$ for which we want to check the reachability. ReduceAdmin returns an ARBAC systems $\widehat{\mathcal{S}}'$ that preserves the role-reachability of `goal` and reduces the number of administrative roles according to the two conditions for immaterial administrative roles given above. We assume that $\mathcal{S}$ has a special administrative role called *super*, containing a user as described above.

The algorithm recursively executes a do-while loop until no further simplification of the system is possible. At each iteration, it first executes a pruning algorithm, that preserves the reachability of `goal` aimed at reducing the number of roles and rules, and then it collapses all immaterial administrative roles into the special role *super*. Finally, spare users are eliminated from the system. It is easy to see that ReduceAdmin eventually terminates, as it shrinks the size of the system at each loop iteration.

Theorem 2 (Correctness of ReduceAdmin). *Let S be an ARBAC system over the set of roles R, $\texttt{goal} \in \texttt{R}$, and $\mathcal{S}' = ReduceAdmin\ (\mathcal{S}, \texttt{goal})$. Then, role* `goal` *is reachable in $\mathcal{S}$ iff it is reachable in $\mathcal{S}'$.*

Our experiments (see Sec. 6), instantiate *Pruning* in the algorithm of Fig. 1 with a novel pruning algorithm that we present in Sec. 5.

5 Aggressive Pruning

In this section we describe a novel pruning algorithm, called *aggressive pruning*, to eliminate roles and rules that are irrelevant to the reachability of role `goal`. We extend previous proposals [10, 22, 4] by identifying six new pruning rules: the first three rules aim at discarding irrelevant roles while the remaining ones identify assignment/revocation rules that can be combined or eliminated.

In the rest of the section we refer to $\mathcal{S}$ as an ARBAC system with a special administrative role *super* which is never removed, and always contains a user whose sole membership is in role *super*. Intuitively, *super* subsumes all administrative roles *admin* for which we can guarantee that it always contains some user ready to perform a rule administered by *admin*. We denote as *persistent* all rules administered by *super*. At each step in the computation we refer to $\widehat{\mathcal{S}} = \langle U, R, UA, \mathit{can\_assign}, \mathit{can\_revoke}\rangle$ as the current pruned ARBAC system derived from $\mathcal{S}$. Below we introduce six pruning rules, called $\mathbf{R_i}$, for $\mathbf{i} \in \{1, \ldots, 6\}$, and denote with $[\![\widehat{\mathcal{S}}]\!]_i$ the ARBAC system resulting from the application of $\mathbf{R_i}$ to $\widehat{\mathcal{S}}$. We now formally define them and argue their correctness.

Removing Irrelevant Roles. A non-administrative role r is *irrelevant*, if each can-assign rule can be fired with any user u, regardless of u's membership to r. An irrelevant role can be eliminated from the system without affecting the reachability of `goal`. The elimination of a set of roles $X \subseteq R$ from $\widehat{\mathcal{S}}$ implies the following changes to the policy: (1) revoke all users-membership from each role in X; (2) remove all assignment/revocation rules having a role in X as target; (3) drop any role in X from each precondition of the remaining can-assign rules.

Formally, let $\mathcal{R}_{ua}(UA, X) = UA \backslash (U \times X)$; $\mathcal{R}_{ca}(\mathit{can\_assign}, X) = \{(\mathit{admin}, P \backslash X, N \backslash X, t) \mid (\mathit{admin}, P, N, t) \in \mathit{can\_assign} \wedge t \notin X)\}$; and $\mathcal{R}_{cr}(\mathit{can\_revoke}, X) = \mathit{can\_revoke} \setminus (R \times X)$. Removing the roles of X from $\widehat{\mathcal{S}}$ results into the system $\mathcal{R}(\widehat{\mathcal{S}}, X) = \langle U, R \setminus X, \mathcal{R}_{ua}(UA, X), \mathcal{R}_{ca}(\mathit{can\_assign}, X), \mathcal{R}_{cr}(\mathit{can\_revoke}, X)\rangle$.

We classify regular roles as non-negative, non-positive, and mixed. A role $r \in (R \backslash AR)$ is *non-positive* (*non-negative*, resp.), if r does not appear in positive (negative, resp.) form in the precondition of any can-assign rule; it is *mixed* if it appears both in positive and negative form in some precondition. We now identify sufficient conditions for a role (different from role `goal`) to be irrelevant in each of those categories.

Remove each regular role $r \in (R \setminus \{\texttt{goal}\})$ from $\widehat{\mathcal{S}}$ such that

$\mathbf{R_1}$: r is **non-positive** and $(\mathit{super}, r) \in \mathit{can\_revoke}$;

$\mathbf{R_2}$: r is **non-negative** and for every $(\mathit{admin}, P, N, t) \in \mathit{can\_assign}$ with $r \in P$, there is $(\mathit{admin}', P', N', r) \in \mathit{can\_assign}$ such that $P' \subseteq P \setminus \{r\}$, $N' \subseteq N \cup \{t\}$, and either $\mathit{admin}' = \mathit{admin}$ or $\mathit{admin}' = \mathit{super}$;

$\mathbf{R_3}$: r is **mixed** and both $\mathbf{R_1}$and $\mathbf{R_2}$ hold.

Fig. 2. Sufficient conditions to remove irrelevant roles.

Non-Positive Roles. A non-positive role r is irrelevant if there is a persistent can-revoke cr with target r. Indeed, since r is a non-positive role, a can-assign ca could not be fired with respect of a user u, only if r is in its precondition and u belongs to r. However, the persistent can-revoke cr can be executed before rule ca, thus enabling ca to be performed regardless of u's membership in r. We translate this property in the following pruning rule: remove the set X of all non-positive roles such that for each $r \in X$, $(super, r) \in can\_revoke$. Then, $[\![\widehat{\mathcal{S}}]\!]_1 = \mathcal{R}(\widehat{\mathcal{S}}, X)$. This pruning rule is summarised in Fig. 2 as $\mathbf{R_1}$.

Non-Negative Roles. Let r be a non-negative role. A can-assign ca could not be fired with respect to a user u, only if r is in the precondition of ca and u does not belong to r. However, if u can be assigned to r while satisfying the precondition of ca (except for u's membership to r), then the ca can be executed regardless of u's membership to r. We translate this property in the following rule: remove the set X of all non-negative roles r such that, for every $(admin, P, N, t) \in can\_assign$ with $r \in P$, there is $(admin', P', N', r) \in can\_assign$ such that $P' \subseteq P \setminus \{r\}$, $N' \subseteq N \cup \{t\}$, and either $admin' = admin$ or $admin' = super$. Then, $[\![\widehat{\mathcal{S}}]\!]_2 = \mathcal{R}(\widehat{\mathcal{S}}, X)$. This rule is summarized in Fig. 2 by $\mathbf{R_2}$.

Mixed Roles. A mixed role r is irrelevant, if it satisfies both conditions that make non-positive and non-negative roles irrelevant. $\mathbf{R_3}$ of Fig. 2 captures the removal of irrelevant mixed roles from $\widehat{\mathcal{S}}$. Then, $[\![\widehat{\mathcal{S}}]\!]_2 = \mathcal{R}(\widehat{\mathcal{S}}, X)$, where X is the set of all mixed roles of $\widehat{\mathcal{S}}$.

Removing Irrelevant Rules. We describe sufficient conditions to get rid of some assignment rules. Specifically, we partition those rules in three categories: (1) *combinable* which refer to pairs of rules that can be merged into a single one; (2) *implied* that identify pairs of rules such that one is subsumed by the other; (3) *non-fireable* corresponding to can-assign rules that cannot appear in any run.

Combinable Rules. Two can-assign rules ca_1, ca_2 can be combined into a single one if (1) they have the same target role, (2) their precondition sets are the same but a single role that appears in positive form in one can-assign rule and in negative form in the other, (3) at each time either there are administrators ready to fire both rules or none of them can be executed. Those rules can be merged in a single one where role r is removed from its precondition. Notice that this operation does not alter the set of reachable configurations since the resulting rule can be fired iff one between ca_1 and ca_2 is fireable. For instance, $(super, \{r_1, r_2\}, \emptyset, r)$ and $(super, \{r_1\}, \{r_2\}, r)$ can be combined in $(super, \{r_1\}, \emptyset, r)$.

Formally, let $ca_1 = (admin_1, P \cup \{r\}, N, t)$ and $ca_2 = (admin_2, P, N \cup \{r\}, t) \in can\_assign$, for some P, N. We define the predicate $Combinable(ca_1, ca_2)$ that holds true if $admin_1 = admin_2$. It is easy to see that if $Combinable(ca_1, ca_2)$ holds then ca_1 and ca_2 are combinable. Then, $[\![\mathcal{S}]\!]_4 = \langle U, R, UA, can\_assign', can\_revoke\rangle$ where $can\_assign' = (can\_assign \setminus \{ca_1 \mid \exists ca_2.\ Combinable(ca_1, ca_2)\}) \cup \{(admin, P, N, t) \mid \exists ca_1 = (admin, P \cup \{r\}, N, t),\ ca_2 = (admin_2, P, N \cup \{r\}, t) \in can\_assign. Combinable(ca_1,\ ca_2)\}$. $\mathbf{R_4}$ of Fig. 3 summarizes this rule.

$\mathbf{R_4}$: Let $ca_1 = (admin_1, P \cup \{r\}, N, t)$, $ca_2 = (admin_2, P, N \cup \{r\}, t) \in can\_assign$ for some P, N, and $admin_1 = admin_2$. Then, replace the **combinable** rules ca_1 and ca_2 with the can-assign role $ca = (admin_1, P, N, t)$.

$\mathbf{R_5}$: If $ca_1 = (a_1, P_1, N_1, r)$, $ca_2 = (a_2, P_2, N_2, r) \in can\_assign$, either $admin_1 = super$ or $admin_1 = admin_2$, $P_1 \subseteq P_2$, and $N_1 \subseteq N_2$, then remove the **implied** rule ca_2 from $\mathcal{S}$.

$\mathbf{R_6}$: Let $ca = (admin, P, N, t) \in can\_assign$ and let $Q = P \cap \{r \mid (u, r) \in UA\}$ such that $\exists i \in [1, |Q|]$ and for every $Z \subseteq Q$ with $|Z| = i$, there is no can-assign rule $(admin', P', N', r) \in can\_assign$ with $r \in Z$, $(P' \cap Q) \subseteq Z$ and $Z \cap N' = \emptyset$, then remove the **non-fireable** rule ca from $\mathcal{S}$.

Fig. 3. Sufficient conditions to remove/combine assignment rules.

Implied Rules. Consider two can-assign rules ca_1, ca_2 with the same target role. Then, ca_1 *implies* ca_2 if for every user u, whenever ca_2 is fireable on u, then also ca_1 is fireable on u. E.g., the rule $(super, \{r_1, r_2\}, \emptyset, r)$ implies $(super, \{r_1, r_2\}, \{r_3\}, r)$. We give a sufficient condition to detect when ca_1 implies ca_2. For every pair of rules $ca_1 = (admin_1, P_1, N_1, r)$ and $ca_2 = (admin_2, P_2, N_2, r)$, we define a predicate $Implies(ca_1, ca_2)$ that holds true iff the following holds: $P_1 \subseteq P_2$, $N_1 \subseteq N_2$, and either $admin_1 = super$ or $admin_1 = admin_2$. It is easy to see that if $Implies(ca_1, ca_2)$ holds, then ca_1 *implies* ca_2. Formally, $[\![\widehat{\mathcal{S}}]\!]_5 = \langle U, R, UA, can\_assign \setminus X, can\_revoke \rangle$, where $X = \{ca' \in can\_assign \mid \exists ca \in can\_assign.\ Implies(ca, ca')\}$. $\mathbf{R_5}$ of Fig. 3 captures this rule.

Non-Fireable Rules. A can-assign rule ca is *non-fireable* if for every run $\pi = c_1 \xrightarrow{m_1} \dots c_n \xrightarrow{m_n} c_{n+1}$ of $\widehat{\mathcal{S}}$, $m_i \neq ca$ for every $i \in [1, n]$. We now give a sufficient condition that allows to detect when a $ca = (admin, P, N, t)$ is non-fireable. Let $Q = P \setminus \{r \mid (u, r) \in UA\}$, that is, the set of P roles that contain no member in the initial configuration of $\widehat{\mathcal{S}}$. Moreover, let *NotFireable* be a predicate over the set of can-assign rules, such that $NotFireable(ca)$ holds true if and only if the following property holds: there exists $i \in [1, |Q|]$ such that for every $Z \subseteq Q$ with $|Z| = i$, there is no can-assign rule $(a', P', N', r) \in can\_assign$ with $r \in Z$, $(P' \cap Q) \subseteq Z$ and $Z \cap N' = \emptyset$.

The following lemma holds:

Lemma 1. *Let $ca \in can\_assign$. If $NotFireable(ca)$ holds true, then can-assign rule ca is non-fireable.*

$[\![\widehat{\mathcal{S}}]\!]_6 = \langle U, R, UA, can\_assign \setminus NF, can\_revoke \rangle$, where NF is the set of all can-assign ca such that $NotFireable(ca)$ holds true. $\mathbf{R_6}$ of Fig. 3 captures this rule.

The correctness of all pruning rules is summarized by the following lemma:

Lemma 2. *Let $\widehat{\mathcal{S}}$ be an ARBAC system. For every $i \in \{1, \dots, 6\}$, (1) $[\![\widehat{\mathcal{S}}]\!]_i$ is an ARBAC system, (2)* `goal` *is reachable in $\widehat{\mathcal{S}}$ iff* `goal` *is reachable in $[\![\widehat{\mathcal{S}}]\!]_i$, and (3) $|[\![\widehat{\mathcal{S}}]\!]_i| < |\widehat{\mathcal{S}}|$ or $|[\![\widehat{\mathcal{S}}]\!]_i| = |\widehat{\mathcal{S}}|$.*

AggressivePruning **Algorithm** takes as input an ARBAC system $\mathcal{S}$ and a role `goal` of $\mathcal{S}$ and returns a system $\mathcal{S}'$ obtained by repeatedly applying the pruning rules $\mathbf{R_i}$, for $\mathbf{i} \in \{1, \dots, 6\}$, along with some existing pruning rules

described in [4, 22]. The algorithm eventually terminates as the application of each pruning rule reduces or leaves unaltered the ARBAC system.

Theorem 3. *Let $\mathcal{S}$ be an ARBAC system and $\mathcal{S}' = AggressivePruning(\mathcal{S}, \texttt{goal})$. Role* `goal` *is reachable in $\mathcal{S}'$ iff* `goal` *is reachable in $\mathcal{S}$.*

6 Experimental Results

We have implemented the procedure ReduceAdmin of Sec. 4 along with AggressivePruning of Sec. 5. Here, we evaluate both procedures on several benchmarks from the literature. The experiments are conducted on a Macbook Pro with an Intel Core i5 2.3 GHz processor and 4GB of RAM. The results of our experiments are reported in Tables 1-3. The tables report the number of roles and rules for each original ARBAC policy. Table 3, in addition reports also the number of administrative roles and the number of users for each policy. All tables report the same information about the original policies, about the policies obtained after the application of our procedures, as well as the time taken.

AggressivePruning **evaluation on a Bank policy.** The first set of experiments are conducted on a policy modeling a bank [9].

	ARBAC Policy		**After Pruning**		
	#roles	#rules	#roles	#rules	Time
1	612	6142	0	0	3.0s
	612	6142	2	1	3.0s
	612	6142	2	1	3.0s
2	612	6142	0	0	2.0s
	612	6142	0	0	2.0s
	612	6142	2	1	2.0s
3	612	6142	468	3285	0.0s
4	612	6142	462	3272	0.1s

Table 1. AggressivePruning on the Bank Policy.

The bank comprises several branches, each consisting of four divisions with five non-managerial roles and two managerial ones. The policy is designed in a way that in any branch a user (non-manager) can be part of at most three non-managerial roles out of five (see [8] for details). The policy has 612 roles and 6142 rules.

The evaluation of our pruning procedure on the bank policy is shown in Table 1 that is divided in four sets of experiments, one for a different security query on the policy. The first query is: *Can any user (non-manager) be assigned to four non-managerial roles in a business division in any of the branches?* The first experiment considers the original policy. After a single iteration the pruning algorithm eliminates some combinable and implied rules and finds that rules assigning users to the role `goal` are non-fireable. For the remaining two experiments, we introduce errors in the policy: we add can-assign rules that allow a (non-manager) user to be part of four non-managerial roles in one of the divisions, respectively, in a single branch (second experiment) and in all branches (third experiment). In both cases, after some iterations that involve all six pruning rules, the simplified system is left with the sole role `goal` (beside the administrative role), and a single can-assign rule: $(admin, \emptyset, \emptyset, \texttt{goal})$ witnessing that `goal` is reachable.

For the second set of experiments the query is: *Can any user (non-manager) be assigned to four non-managerial roles in a business division in all the branches?* As above, the first experiment is done on the original policy, while the other two experiments on the modified policies. In the first and second experiment the

Size Policy		After Aggressive Pruning								
		First Suite			Second Suite			Third Suite		
#roles	#rules	#roles	#rules	Time	#roles	#rules	Time	#roles	#rules	Time
3	15	1	1	0.0s	3	5	0.0s	3	5	0.0s
5	25	1	1	0.0s	1	1	0.0s	1	1	0.0s
20	100	1	1	0.0s	11	26	0.0s	11	26	0.0s
40	200	1	1	0.0s	1	1	0.0s	1	1	0.1s
200	1000	1	1	0.1s	1	1	0.1s	1	1	0.1s
500	2500	1	1	0.1s	1	1	0.1s	1	1	0.2s
4000	20000	1	1	6.0s	1	1	6.0s	1	1	6.3s
20000	80000	1	1	3m24s	1	1	3m32s	1	1	3m20s
30000	120000	1	1	8m14s	1	1	8m33s	1	1	7m47s
40000	200000	1	1	14m50s	1	1	18m7s	1	1	21m1s

Table 2. AggressivePruning on Complex Policies with Separate Administration.

pruning algorithm finds out that `goal` is unreachable (rules assigning users to `goal` are non-fireable), while in the third experiment it returns a system constituted by the sole role `goal` and a single can-assign rule: $(admin, \emptyset, \emptyset, \texttt{goal})$, showing that *goal* is reachable.

The single experiment in the third set considers the query: *Can any user (manager included) retain permissions of four non-managerial roles in a business division in any of the branches?* While the query in the forth set is: *Can any user (manager included) retain permissions of four non-managerial roles in a business division in all of the branches?* Both queries are carried out on the original policy and in both cases the pruning algorithm makes use of all six rules, reducing the system approximatively by a third.

AggressivePruning **evaluation on big policies.** The authors of [9] created three sets of complex test suites capturing the complexity of realistic systems. Each suite comprises ten policies where the number of roles and rules ranges respectively from 3 to 40k and 15 to 200k. Each suite presents different kind of source of complexity. Sources of complexity are parameters such as the size of the policy and type of administrative rules. We refer to [9] for a comprehensive description. We evaluate our pruning on these policies, whose results are summarized in Table 2. In [9] it has been experimentally proven that existing static pruning techniques [10, 22] are ineffective on such complex policies. In contrast, our aggressive pruning is extremely effective making the policies orders of magnitude smaller. In these policies $\mathbf{R_5}$ (implied rules) plays an essential role.

ReduceAdmin **evaluation.** We evaluate ReduceAdmin on two sets of realistic ARBAC policies without separate administration, used in several case studies [22, 7, 9, 21]: a hospital and a university policy [19]. Table 3 summarizes the results of our evaluation. Besides the information of the original and the simplified policy, we also indicate whether the role `goal` is reachable.

The first set of experiments concerns the hospital policy. The first experiment in the table tests that a user is not a member of both the roles *Receptionist* and *Doctor*, to avoid that a user bills the insurance company by falsely claiming to treat a patient. The second one is meant to check that a patient is not his own primary doctor. The third experiment checks that *nurse* and *doctor* roles are disjoint, to avoid that a user has both the permissions of adding progress and private notes. The last experiment tests whether a *doctor* is able to assign a user to the role that groups patients with third party consent.

For the University policy, the first experiment verifies whether a user can be an explicit member of both roles *DeptChair* and *Dean*. The second experiment

	ARBAC Policy					**After** ReduceAdmin					Reach
	name	#roles	#rules	#admin	#users	#roles	#rules	#admin	#users	Time	
1	Hospital	12	25	6	1092	3	5	2	9	0.3s	NO
	Hospital	12	25	6	1092	5	9	3	19	0.0s	NO
	Hospital	12	25	6	1092	3	5	2	9	0.0s	YES
	Hospital	12	25	6	1092	6	8	3	16	0.0s	YES
2	University	32	130	9	943	3	2	1	1	0.2s	NO
	University	32	130	9	943	1	1	1	1	0.2s	YES
	University	32	130	9	943	12	16	2	23	0.2s	NO
	University	32	130	9	943	11	13	2	21	0.2s	YES
	University	32	130	9	943	14	25	3	34	0.2s	YES

Table 3. ReduceAdmin evaluation.

checks if a user may have both privileges of *Dean* and *DeptChair*. The third experiment tests whether a user can be simultaneously an explicit member of both *Undergrad* and *Grad* roles. Finally, the last experiment verifies that a user can be simultaneously in the roles *GradAdmissionsCommitte* and *AdmissionsOfficer*.
Observations: Our techniques allow to significantly reduce, not only the number of roles and rules, but also the number of administrative roles and the amount of distinct user role-combinations. For instance, in the hospital policy, we need to consider at most 19 users out of the initial 1092, each with at most 6 roles, and only 2 or 3 of them need to be tracked! In the first two experiments on the university policy, the pruning reduces the administrative roles from 9 to 1.
These two experiments, particularly benefit from the application of the pruning rule $\mathbf{R_5}$ (implied rules). The pruning rules concerning the removal of positive and negative roles play a significant role for the fourth university experiment and the third and the fourth hospital experiments. Such policy simplifications allowed us to fully check the role-reachability problem for such policies by using off-the-shelf tools. For example, we encoded the reduced policies into Boolean programs and then used Getafix [12] that fully verified them in few seconds. We did the same exercise with the original policies and tried several model-checking tools for finite state systems and none of them could terminate the analysis.

7 Conclusions

We have laid out the foundations of reasoning with ARBAC policies where users self-administer the resources, without recourse to a separate set of administrators. We have identified a small model theorem for analysis of such policies, arguing that tracking a bounded number of users suffices to check the policy. Using this technical insight, we have developed heuristics to reduce an ARBAC system and shown its effectiveness in analyzing real-world policies.

The work reported by us in [4], which presents abstraction techniques aimed at *proving* ARBAC policies correct is complementary to our work here which is a precise analysis that can find security breaches. A combination of these two approaches into a CEGAR scheme would be interesting. The broader view that we suggest is that security analysis of RBAC/ARBAC policies can be solved using model-checking and abstraction techniques commonly used in program verification. Developing such techniques for a larger range of policies beyond RBAC is an interesting future direction to pursue.

Acknowledgements

This research was partially supported by ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO and NSF CCF #1018182.

References

[1] http://en.wikipedia.org/wiki/Role-based_access_control.
[2] http://www.microsoft.com/sqlserver/en/us/default.aspx.
[3] http://www.microsoft.com/it-it/server-cloud/windows-server/active-directory.aspx.
[4] A. L. Ferrara, P. Madhusudan, and G. Parlato. Security analysis of access control policies through program verification. In *CSF*, 113–125. IEEE 2012.
[5] J. Crampton. Understanding and developing role-based administrative models. In *CCS*, 158–167. ACM, 2005.
[6] D. Ferraiolo and R. Kuhn. Role-based access control. In *NCSC*, 554–563, 1992.
[7] M. I. Gofman, R. Luo, A. C. Solomon, Y. Zhang, P. Yang, and S. D. Stoller. RBAC-PAT: A policy analysis tool for role based access control. In *TACAS*, 46–49, 2009.
[8] K. Jayaraman, V. Ganesh, M. Tripunitara, M. C. Rinard, and S. J. Chapin. Arbac policy for a large multi-national bank. http://kjayaram.mysite.syr.edu/mohawk/casestudy.pdf, 2010.
[9] K. Jayaraman, V. Ganesh, M. V. Tripunitara, M. C. Rinard, and S. J. Chapin. Automatic error finding in access-control policies. In *CCS*, 163–174. ACM, 2011.
[10] S. Jha, N. Li, M. V. Tripunitara, Q. Wang, and W. H. Winsborough. Towards formal verification of role-based access control policies. *IEEE Trans. Dependable Sec. Comput.*, 5(4):242–255, 2008.
[11] A. Kern. Advanced features for enterprise-wide role-based access control. In *ACSAC*, 333–342. IEEE, 2002.
[12] S. La Torre, P. Madhusudan, and G. Parlato. Analyzing recursive programs using a fixed-point calculus. In *PLDI*, 211–222. ACM, 2009.
[13] N. Li and Z. Mao. Administration in role-based access control. In *ASIACCS*, 127–138. ACM, 2007.
[14] N. Li and M. V. Tripunitara. Security analysis in role-based access control. In *SACMAT*, 126–135. ACM, 2004.
[15] A. C. O'Connor and R. J. Loomis. http://csrc.nist.gov/groups/SNS/rbac/documents/20101219_RBAC2_Final_Report.pdf.
[16] R. S. Sandhu, V. Bhamidipati, and Q. Munawer. The arbac97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur.*, 2(1):105–135, 1999.
[17] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
[18] R. S. Sandhu and Q. Munawer. The arbac99 model for administration of roles. In *ACSAC*, 229–238. IEEE, 1999.
[19] A. Sasturkar, P. Yang, S. D. Stoller, and C. R. Ramakrishnan. Policy analysis for administrative role based access control. *Tech. Rep., Stony Brook Univ.*, 2006.
[20] A. Sasturkar, P. Yang, S. D. Stoller, and C. R. Ramakrishnan. Policy analysis for administrative role based access control. In *CSFW*, 124–138. IEEE, 2006.
[21] S. D. Stoller, P. Yang, M. I. Gofman, and C. R. Ramakrishnan. Symbolic reachability analysis for parameterized administrative role-based access control. *Computers & Security*, 30(2-3):148–164, 2011.
[22] S. D. Stoller, P. Yang, C. R. Ramakrishnan, and M. I. Gofman. Efficient policy analysis for administrative role based access control. In *CCS*, 445–455. ACM, 2007.