2012 IEEE 25th Computer Security Foundations Symposium

Security Analysis of Role-based Access Control through Program Verification

Anna Lisa Ferrara
University of Bristol, UK
anna.lisa.ferrara@bristol.ac.uk

P. Madhusudan
University of Illinois, USA
madhu@illinois.edu

Gennaro Parlato
University of Southampton, UK
gennaro@ecs.soton.ac.uk

***Abstract*—We propose a novel scheme for proving administrative role-based access control (ARBAC) policies correct with respect to security properties using the powerful abstraction-based tools available for *program verification*. Our scheme uses a combination of abstraction and reduction to program verification to perform security analysis. We convert ARBAC policies to *imperative programs* that simulate the policy abstractly, and then utilize further abstract-interpretation techniques from program analysis to analyze the programs in order to prove the policies secure. We argue that the aggressive set-abstractions and numerical-abstractions we use are natural and appropriate in the access control setting. We implement our scheme using a tool called** VAC **that translates ARBAC policies to imperative programs followed by an interval-based static analysis of the program, and show that we can effectively prove access control policies correct. The salient feature of our approach are the abstraction schemes we develop and the reduction of role-based access control security (which has nothing to do with programs) to program verification problems.**

***Keywords*-Access control; formal methods for security.**

I. Introduction

Role-based access control (RBAC) has emerged as an effective mechanism for administrators to manage the permissions of users in large organizations [4], [21]. RBAC simplifies the administration by defining access control in two stages: *roles* are defined and are associated with sets of permissions, and users are assigned roles. Ease of administration is mainly due to the fact that roles in an organization are more stable as they are associated with job functions, while the dynamic day-to-day changes in the organization deal with user-role associations (personnel change across departments, reassignment of duties, etc.) [18]. RBAC also defines hierarchical relationships amongst roles that allow users of a role to inherit the permissions of roles that are below theirs [21].

Administrative role-based access control (ARBAC) [20], [22] is a policy mechanism for controlling how changes can be made to the RBAC policy by various administrators. ARBAC is appealing in the context of large organizations that have multiple administrators who are given different levels of privileges. In a world where administration is moving towards being highly distributed, ARBAC eases the management of these tasks. ARBAC policies are generally divided into three sub-policy languages: one that controls the reassignment of users to roles (URA) [22], one that controls reassignments of permissions to roles (PRA), and one that controls the role-role assignments, including groups and hierarchies (RRA) [20].

A critical question regarding ARBAC policies is whether they ensure certain security properties. For instance, the policy designer may have put in certain rules to ensure that users cannot hold roles *Recruiter* and *Grievance-Committee* simultaneously, as it may be a conflict-of-interest to do so. While the designer may have crafted some rules in the ARBAC system to ensure this property (for example, allowing users to gain the role *Recruiter* only if they do not already have the role *Grievance-Committee*), she may have unintentionally introduced loopholes in the system. For instance, there may be a rule that gives users with role *Director* the role *Grievance-Committee*, which may have seemed a natural rule at the time it was added, to facilitate directors to see committee meeting minutes. This would allow a director to gain both the conflict-of-interest roles and violate the security intent. Role-hierarchies, multiple administrators, and the administrator roles themselves evolving over time further complicate the problem, and it is very hard for the designer to determine whether security properties really hold in their ARBAC system.

Policy analysis allows policy designers to check whether their policies meet their security goals. Policy analysis gives policy designers and system administrators the ability to examine the logical implications of the policies they have set forth; in particular, it allows them to gauge what security breaches could happen if several untrusted users started exploiting the rules to gain unintended permissions in the system. Despite the importance of this problem, we do not know of any effective tools that are available for policy analysis, even in the ARBAC setting.

Policy analysis that checks if a policy meets the security goals is a classical problem in security. The general problem was phrased by the classical paper by Harrison, Russo, and Ullman [7] using access control matrices and rules that transform them, and it was shown that security analysis is *undecidable*. The field has suffered greatly from this negative result, and has led to the identification of several restricted domains and alternate access control security settings that admit a decidable security analysis problem [15], [25]. In the setting of ARBAC, when the number of users is fixed by the initial state of the system, the problem is decidable,

DOI 10.1109/CSF.2012.28

but suffers from an exponential time dependency on the number of roles, making techniques unscalable [23], [14], [27], [26]. Consequently, to date, there is no effective system that administrators can use for security analysis, *despite the fact that most of these problem instances have simple proofs that they are secure*.

Our philosophy is that undecidability and complexity hardness should not preclude useful tools. The current state of the art in the field of program analysis is an excellent example; scalable program analysis tools have been developed for problems that are inherently undecidable, and provide useful analysis for a large class of realistic programs. For instance, static analysis algorithms (including type systems) used in compilers tackle inherently undecidable problems; however by performing an *abstract* analysis, several properties can be soundly proved about real-world programs in reasonable time [3], [17]. The SLAM/SDV project is another prime example, where device drivers are routinely checked against the Windows API, and proven not to have bugs; this analysis is sound (modulo certain pointer arithmetic aliasing issues) and is based on *counter-example guided predicate abstraction and refinement*, leading to the tool finding simple proofs that the program conforms to the API [2].

The key techniques that have led the path to scalable program analysis is a combination of *abstraction* (sometimes guided through counter-examples) and *efficient fixed-point algorithms*. Our aim, in this paper, is to use the very same techniques to solve the problem of security analysis of ARBAC policies.

Our goal is to extend the program analysis philosophy and even the techniques and tools used in program analysis to bear on the problem of access control policy security. We believe that most ARBAC policies that contain security constraints can be easily proven correct, as administrators designing the policy have such a simple reasoning in mind. We propose effective mechanisms that find these simple proofs, using abstraction and fixed-point computation [3].

The security analysis that we propose for ARBAC (in particular, URA) analyzes the system using several abstractions. The state of the ARBAC system can be accurately modeled by tracking the *number* of users in every possible combination of roles. This is, however, an intractably large system, and we propose abstraction mechanisms to prove properties of this system.

The two main abstraction techniques we propose in this paper are: (a) set-abstractions, that track users' membership in only certain combinations of roles, and (b) numerical abstractions, that abstract the precise number of users in each combination of roles to *interval ranges*. The former reduces the various sets of role-combinations that need to be tracked (which is exponential in the number of roles), while the latter abstracts the precise set of users in each tracked role combination (which leads to using much smaller state-spaces and achieve faster convergence). The main technical contribution of this paper is the design of *precise* abstraction systems that soundly model the two desired abstractions above.

A consequence of using numerical abstractions is that we can handle the ARBAC security analysis problem in a setting with an unbounded number of users, where any number of users are allowed to join or leave the organization.

While we could engineer tools to directly utilize the abstractions we propose, we instead propose a technique to directly utilize the abstraction tools available for program verification. We show that the abstract analysis can be achieved using state-of-the-art tools in program analysis by reducing the security problem to a program analysis problem.

Given an ARBAC system and a security goal, our security analysis works in two phases:

(a) build an imperative program that abstractly simulates the system by tracking only a subset of role-combinations, and

(b) analyze the resulting program using a standard interval-based program analysis tool called INTERPROC [3], [11].

If the program verification phase succeeds in proving that a particular location is unreachable, then this translates to a proof that the ARBAC system meets its security goal.

As far as we know, our work is the first in proposing abstraction based verification for proving security of ARBAC policies. Complexity analysis of checking security (when the number of users is fixed) has been studied [12], [14], [27], and several model-checking techniques and error finding techniques have also been investigated [9] (the latter can find errors but cannot prove that a policy is correct).

Technically, the abstraction techniques we propose are novel, even in the setting of program analysis. The security analysis calls for tracking the number of users in a subset of combinations of roles, and it is non-trivial to build this abstract domain and the abstract transitions that are sound and precise (up to the abstraction). The program that we construct, when tracking the number of users in a particular set of role combinations (say $S_1, \ldots, S_k$, where each S_i is a set of roles), must determine how to simulate whether an administrator can change (grant or deny) the roles of a user in the abstract domain. Our program achieves this using *non-determinism* in a novel way followed by Boolean checks to simulate such a transition, leading to a sound and precise abstraction. Furthermore, the programs are specially crafted so that the subsequent interval abstractions used to analyze the program are meaningful.

In summary, our contributions are:

- The proposal to analyze ARBAC-URA systems for role-reachability using set-abstractions and numerical abstractions.
- The design of an abstract analysis using a transition system that tracks the ARBAC system soundly using

only integers that capture the cardinalities of the sets of users in a subset of role combinations. The design of a sound and precise abstract transformer is technically non-trivial.
- The *realization* of the abstract system above as an imperative non-deterministic while-program manipulating integers, such that the security question reduces to location reachability in the program. The abstract security analysis can then be performed using an interval analysis of the above synthesized program.
- A prototype implementation of the entire proposed methodology that translates ARBAC-URA role-reachability problems to imperative programs, and uses the INTERPROC tool to perform the interval analysis. We also evaluate our tool and show that it is effective in proving several reasonably large ARBAC policies correct according to privilege escalation and separation-of-duty properties, automatically.

Related Work

There is a rich literature on analysis of static and dynamically evolving access control policies. We restrict ourselves in discussing only dynamic security analysis of access control policies, which change over time in accordance with set rules. The classical HRU model is based on access matrices, with ability to add subjects and objects, and which was proved to have an undecidable security analysis problem [7]. Several restricted models have been shown to admit decidable security analysis [16], [19], [25], [13], including the Graham-Denning model [15], [6]. As far as we know, abstraction-based sound but incomplete procedures, as set forth in this paper, are not known even in this general setting.

Turning to analysis of ARBAC, there has been a lot of recent research in finding security errors, finding the complexity of checking security, and finding tractable algorithms for smaller fragments. This work, unlike ours, concentrates on the security analysis problems with a *finite* number of users, which is always decidable. Li and Tripunitara [14] have investigated the security analysis problem for ARBAC-URA schemes and identified fragments (AATU and AAR) and restricted queries that can be solved in polynomial time (for instance negative pre-conditions are completely disallowed in these restricted models). Sasturkar et al [23], [24] study ARBAC security, and show that the problem is PSPACE-complete, that most restrictions still are NP-hard, and some very restricted versions can be solved in polynomial time. Stoller et al follow up this work by identifying the *fixed-parameter complexity* of the problems, and show that the problem is tractable when one fixes the number of roles, and also study restricted versions that are tractable [27], [5]. In more recent work, Stoller et al have extended the ARBAC model to parameterized ARBAC that allows conditions that depend on parameters, and have extended the analysis algorithms to this setting [26]. Recently, Jayaraman et al [9] propose a new under-approximate analysis particularly suited for finding shallow errors in complex ARBAC policies.

Our work differs from the above works in two ways. First, we handle the security problem for unboundedly many users entering and leaving the system, while the above works assume that the set of users is fixed. There are a class of ARBAC policies, called *separate administration* policies, where the set of administrative roles are separate from the set of regular roles, and user membership to administrative roles never changes (ARBAC97 and several other policy languages assume such a restriction). In this setting, there is in fact not much of a difference between handling a fixed number of users vs handling unboundedly many users, as we can show that tracking only *one* regular user is sufficient. However, our setting does not assume policies have separate administration.

A second difference from the above works is that we do not restrict the ARBAC scheme in any way, but rely on *abstraction* based techniques to achieve scalability. We believe that abstraction based techniques, especially when the abstraction can be tuned to the instance, will yield the algorithms that best exploit the simplicity in real-world instances.

Finally, in very recent work, Armando and Ranise [1] have studied the analysis of ARBAC security problems with an unbounded number of users, and showed a decidability result for a fragment of ARBAC policies that can be expressed in a restricted logical framework. However, the algorithms have non-elementary complexity, and the restrictions are awkward with only very restricted negated pre-conditions permitted. Also, in recent work, Jayaraman et al [9] investigate abstraction-refinement techniques for ARBAC analysis, but towards *finding errors* in the policy. In contrast, the mechanism we set forth can be used to *prove* security properties of systems.

II. OVERVIEW

In ARBAC, a set of *administrative roles* are defined, and users with these administrative roles are given the privilege of assigning users to roles, hence giving them new permissions, as well as revoking their roles, removing permissions. In this paper, we restrict ourselves to the ARBAC model URA that allows reassignment of users to roles. We do not consider reassignments of permissions to roles (PRA) and role-hierarchy reconfigurations (RRA). The URA model captures the most interesting changes that happen in an organization day-to-day, where users get reassigned roles; changes of permissions to roles and role-hierarchy changes are much less frequent.

Hiring and retiring: ARBAC with unbounded number of users

In this paper, we formulate security questions in a setting that handles an *unbounded* number of users that can be included into the system, independent of the number of users in the initial configuration of the system. For instance, in an access control system for a university, staff can be hired or retire, and the security question should model this hiring and retiring, and the security properties must be independent of the number of *current* staff members in the university. Our model of ARBAC hence considers the security problem as examining systems where there are an unbounded number of users W, which includes the users in the initial configuration of the system. Users in W that are not in the initial configuration have no roles assigned to them, by default. The security questions that we address ask reachability questions in the system starting from *any* of these infinite initial configurations.

We can then model *hiring* and *retiring* of users in the organization using the ARBAC mechanism itself. We can have a single role called *Org* that all the users in the initial configuration are assigned to. Hiring users can be achieved using rules that hire a user provided they do not have the *Org* role nor any other role assigned to them; retiring of users can be achieved by revoking the *Org* role.

Security of ARBAC policies: The security problem for ARBAC policies relates to the unintended escalation of privileges of users. The developers of the administrative system 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. Security breaches include privilege escalation (e.g. an employee of a lower rank gaining access to resources meant for a higher rank), separation of duty constraints that model conflict of interest (e.g. a user u cannot simultaneously have roles r_1 and r_2), etc.

The security problem we study in this paper is role reachability, formulated as follows:

The Role-reachability problem: *Given an ARBAC policy with unboundedly many users, an initial RBAC configuration of the system, and a target role* `goal`*, is there a reachable configuration of the access-control system where some user is assigned role* `goal`*?*

The above technical problem (formulated in the setting with unboundedly many users) seems to be the most useful security question in the setting of ARBAC-URA; almost all other interesting security questions can be *reduced* to the above problem. For example, the problems of checking whether a particular user u can ever reach a role r, whether any user u can ever have both roles r_1 and r_2, whether any user u can get permission p, etc., can all be reduced to the role-reachability problem (see Section III-D for details).

Tracking ARBAC using cardinalities of sets: One key observation is that ARBAC systems can be modeled using *cardinalities* of a finite set C of sets, where each set in C represents a role combination. Intuitively, the ARBAC system controls the user-role relationships, and hence the *configuration* of a system can be captured by noting down the precise set of users in each combination of roles. A Venn-region is a combination of the form $r_1 \wedge r_2 \wedge \neg r_3 \wedge r_4$, where we choose, for each role r_i, either r_i or $\neg r_i$. A Venn region denotes the set of users that have all the roles r_i mentioned positively in the region and do not have any of the roles r_j mentioned negatively in the region. The configuration of a system is then uniquely described by the set of users in each Venn-region of roles.

It is not hard to see that two users u and u' that have *identical* role-assignments are indistinguishable by the system. This stems from the fact that ARBAC policies, which enable administrators to change roles of users, allow for pre-conditions that check only the user's membership and non-membership in roles. Consequently, the precise identity of the users is immaterial. Hence, instead of modeling the configuration using the precise sets of each Venn region, we can instead model the configuration by only modeling the *cardinality* of each Venn region. The role-reachability problem then reduces to checking whether there is a reachable configuration of the system where the number of users in some Venn-region that positively includes the target role `goal` is greater than zero.

The above modeling of ARBAC using sets and then cardinalities of sets loses absolutely no precision, and accurately models the problem. We are exploiting the crucial fact that RBAC and ARBAC operate at the level of roles, and hence user-identities are not important (which is in fact *the* salient feature of role-based access control).

Abstracting cardinalities: We now come to the first main contribution of this paper: the proposal to use abstraction techniques to reason with ARBAC system security. An access control policy's security crucially depends on the various combinations of roles that can be held by individual users, but almost never depends on the precise number of users that can hold these roles. For instance, it would be highly unnatural if the security of an access control system for a hospital depended on the precise number of doctors present. Consequently, the fact that is most important is whether a *non-zero* number of users can hold a particular combination of roles. In fact, policies in the strict ARBAC-URA standard do not even allow the administrator to set conditions on the precise set of users in a role, making policies that depend on the precise number of users holding role-combinations virtually infeasible (though some amount of 'counting' can be smuggled in by carefully choosing the initial configuration). In our experience, we have never come across any natural scenario where the proof that an ARBAC system is safe depended on the precise number of users in a role-combination.

Our proposal in this paper is to conduct a search for an *abstraction*-based proof that exploits this structure in the instances. We propose that an abstraction-based scheme that tracks the zero/non-zero cardinalities of the various Venn regions is feasible and sufficient to prove real ARBAC systems correct. ARBAC system designers have such a reasoning in their mind that works at this level of abstraction.

Set and cardinality abstractions: The above proposal to abstractly track the number of users in every possible role combination, while sound, is still impractical as it does not scale to large ARBAC policies. There are hence an *exponential* number of Venn regions to track (e.g., 50 roles would lead to 2^{50} Venn regions to track, which is intractable). We hence propose tracking only *certain* combinations of roles in the system, but insist on tracking them soundly. More precisely, we choose $Track$, a subset of Venn regions, by examining the combinations of roles mentioned in the ARBAC policy, and build an abstraction scheme that soundly tracks the system by only tracking the Venn regions in $Track$. By 'soundness', we mean that if the abstracted system cannot reach the target role, then we are assured that the concrete system also cannot reach the target role (the converse does not hold in general, unless $Track$ includes all Venn regions).

Our second main contribution is a sound abstraction of a partial set of Venn regions $Track$. The non-trivial technique here is the design of the abstract transformer that transforms the cardinalities of the partial Venn regions soundly. We propose an abstract transformer that is very precise, by choosing non-deterministically a user that is consistent with the current configuration of the system, and updates the user according to a rule followed by a sound update of the various sets in $Track$. This is the most technical contribution of this paper. This abstraction, furthermore, factors cleanly into program code, as described below.

Reduction to program verification: Our third contribution is the proposal that we can effectively conduct the abstraction-based analysis by reducing the ARBAC role-reachability problem to program analysis. In particular, the modeling of the system by using pure numbers (capturing cardinalities of the partial subset of Venn regions, as sketched above) allows us to model the evolution of an ARBAC system as a simple while-program that manipulates integers. We propose using *abstract-interpretation* techniques in program verification to perform abstract analysis of the ARBAC systems. In particular, we propose to use static analysis that computes interval abstractions for numerical domains, which are extremely scalable, allowing us to effectively analyze the security problems for ARBAC.

Interval abstractions of programs are a classical abstraction framework that track the interval range for each variable at each point in the program, performing widening steps to intervals across loops in order to terminate. The widening procedures hasten the computation of fixed-points by widening a growing interval to extreme values, $+\infty$ or $-\infty$, to reach termination. A slew of techniques for performing interval abstractions are known (including compact representations of intervals, sound abstractions for each operation, widening and narrowing techniques, heuristics for delaying the widening, etc.), and moreover, there are fast tools that have been developed to solve this problem that scale to programs that span hundreds of thousands of lines of code [10], [11].

The reduction to programs that we propose works, more precisely, as follows. Given an ARBAC-URA system, an initial configuration with an unbounded number of users, and a target role `goal`, we follow the procedure below:

- Convert the ARBAC system to a *program* P in an imperative language (similar to C) that tracks the ARBAC system by tracking a subset of Venn regions $Track$, where $Track$ is the set of combinations mentioned in the rules of the system. If the target role `goal` is reachable from the initial configuration in the ARBAC system then the error-configuration will be reachable in the program P. The construction of the program P realizes the precise abstract transformer mentioned above.
- We perform an interval-analysis of the program P to compute the possible intervals that variables can take at various places of the program. In particular, the static analyzer will also compute the various parts of the program that are unreachable using the information in the interval analysis it performs, and mark them.
- If the error position in P is found provably unreachable by the static analysis, we have proved that the ARBAC system is safe (for unbounded number of users entering and leaving the system). If not, we cannot claim there is an error (as the analysis is an over-approximation) and hence report that we could not solve the problem.

Organization of the paper.: The rest of the paper is structured as follows. In the next section, we define RBAC and ARBAC policies, the formal security model with an unbounded number of users, the role-reachability problem, and outline how various other security problems can be reduced to role-reachability. Section IV describes the main technical contribution of the paper, modeling ARBAC systems as integer programs and giving an approximate transformation that defines an abstract transition system that soundly captures the security problem. Section V gives a simple slicing technique that (soundly) removes rules irrelevant to the security question at hand, thus simplifying problems in practice. Section VI discuss a possible choice for a subset of Venn-regions to track. Section VII describes the implementation of our tool VAC and experimental results of our tool on a suite of ARBAC policies queries on various

security questions.

III. Preliminaries

A. Role Based Access Control

In this section we define the RBAC model [4]. Since we do not consider analysis queries involving sessions, we describe a simplification which does not support sessions.

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 $(u, r) \in UA$ means that user u is a member of role r. Similarly, $(p, r) \in PA$ means that members of role r are granted the permission p. Roles are related by a partial order $\succ$, called the *hierarchy relation*. For any two roles $r_1, r_2 \in R$, r_1 inherits permissions of r_2 iff $r_1 \succ r_2$. In the following we will only consider RBAC policies without hierarchy relation. Indeed, we perform the transformation described in [23], [24] to turn a hierarchical policy into a non-hierarchical one before starting our analysis. Thus, in the rest of the paper we refer to an RBAC policy as a tuple $\langle U, R, P, UA, PA \rangle$.

B. Administrative Role Based Access Control

ARBAC97 [20] presents a comprehensive model for role-based administration of RBAC. ARBAC97 has three components: `URA97` user-role administration, `PRA97` permission-role administration, and `RRA97` role-role administration. In this paper we focus on the user-role administration model, which we will refer to as URA.

The URA policy control allows to change the user-role assignment UA by means of assignment/revocation rules carried out by administrators which are organized in a set AR of administrative roles.

Administrators are allowed to change roles of a user according to a precondition which only depends on the user membership and non-membership in roles. A *precondition* is 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. We partition any precondition in two sets denoted Pos and Neg. Such sets, respectively, correspond to the set of roles that appear in positive and negative form in the precondition.

Permission to assign users to roles is specified as:

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

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 make a user whose current role memberships satisfies the precondition (Pos, Neg), a member of $r \in R$. In the rest of the paper we assume that $Pos \cap Neg = \emptyset$.

Permission to revoke users from roles is specified as:

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

The meaning of a tuple $(admin, r) \in can\_revoke$ is that a member of the administrative role $admin \in AR$, can revoke the membership of a user from a role $r \in R$.

In [20] the set of administrative roles AR is disjoint from the set of roles R, such policies are called *separate administration* policies. However, we allow administrative roles to be part of the set of roles R, i.e., $AR \subseteq R$.

C. ARBAC with unbounded users

The URA model does not allow new users to enter the system. Indeed, the set of users U is finite and remains the same along the whole evolution of the system. In this paper we formulate security questions in a setting where an unbounded number of users can enter the system. In order to allow users to join the system and being revoked we define an unbounded set of users W and let the initial configuration of the system be any bounded set of users $\widehat{U} \subseteq W$. Intuitively, $\widehat{U}$ is the set of users that will be involved in the system at any time.

Formally, an *ARBAC with an unbounded world system*, or simply ARBAC system, is a state-transition system defined as: $\mathcal{S} = \langle \mathtt{RBAC}, \mathtt{URA}, W \rangle$ where $\mathtt{RBAC} = \langle U, R, P, UA, PA \rangle$ is an RBAC policy, $\mathtt{URA} = \langle can\_assign, can\_revoke \rangle$ is an URA policy control over the set of roles R, and W is an unbounded set of users.

A *configuration* of $\mathcal{S}$ is any pair $(\widehat{U}, UR)$ where $\widehat{U} \subseteq W$ is a finite set, and $UR \subseteq \widehat{U} \times R$. A configuration is *initial* if $UR = UA$. For any user $u \in \widehat{U}$ and configuration $c = (\widehat{U}, UR)$, we define $conf(c, u) = (P, N)$ where $P = \{r \in R \mid (u, r) \in UR\}$ and $N = R \setminus P$.

Given two configurations $c = (\widehat{U}, UR)$ and $c' = (\widehat{U}, UR')$, there is a *transition* from c to c' with rule $m \in (can\_assign \cup can\_revoke)$, denoted $c \xrightarrow{\tau_m}_{\mathcal{S}} c'$, if there exists an *administrative* user $ad \in \widehat{U}$ with $(ad, admin) \in UR$, a user $u \in \widehat{U}$, and one of the following holds. Let $conf(c, u) = (Pos_u, Neg_u)$.

- [Can_Assign]: $m = (admin, Pos, Neg, r)$, $Pos \subseteq Pos_u$, $Neg \subseteq Neg_u$, and $UR' = UR \cup \{(u, r)\}$;
- [Can_Revoke]: $m = (admin, r)$, $(u, r) \in UR$, and $UR' = UR \setminus \{(u, r)\}$.

A *run* of $\mathcal{S}$ is a sequence of $\mathcal{S}$ configurations $c_0 c_1 \dots c_n$, for some $n \geq 0$, such that c_0 is an initial configuration of $\mathcal{S}$, and for every $i \in \{0, 1, \dots, n-1\}$, $c_i \xrightarrow{\tau_{m_i}}_{\mathcal{S}} c_{i+1}$ for some $m_i \in (can\_assign \cup can\_revoke)$. Furthermore, we say that c_n is a *reachable* configuration of $\mathcal{S}$.

The role-reachability problem: Given an ARBAC system $\mathcal{S}$ over the set of roles R, and a target role $\mathtt{goal} \in R$, the *role-reachability problem* asks whether there is a user set $\widehat{U}$ and a configuration $c = (\widehat{U}, UR)$ that is reachable from the initial configuration $(\widehat{U}, UA)$, such that $(u, \mathtt{goal}) \in UR$, for some user $u \in \widehat{U}$.

D. From security properties to role-reachability

In this section we describe several security properties of interest in the setting of ARBAC-URA and show that they can be reduced to the role-reachability problem. Those properties were first identified by Li and Tripunitara [15]:

Mutual Exclusions: *Will no user be assigned to both roles r_1 and r_2, simultaneously, in all reachable configurations?* This helps ensuring *separation of duty* constraints.

Reduction to role reachability: Add a new role `goal` and a can-assign rule with target `goal` and with the precondition $(r_1 \wedge r_2)$. Check the role-reachability for `goal`.

Bounded Safety: *Will the users (u_1, u_2, u_3) be the only ones that have permissions p_1 and p_2 in any reachable configuration?*

Reduction to role-reachability: Endow the users (u_1, u_2, u_3) with a special role $\widehat{r}$, that is irrevocable. Add a role `goal`. Then, for every minimal subset of roles R' where R' has at most two roles and such that the union of permissions of roles in R' include both p_1 and p_2, add a can-assign rule that has `goal` as target and with the precondition $\neg\widehat{r} \wedge \bigwedge_{r \in R'} r$. Then query whether the role `goal` is reachable.

Availability: *Will a user u have permission p in every reachable configuration?*

Reduction to role-reachability: Endow user u with a special role $\widehat{r}$ which is irrevocable. Then, add a role `goal` and a can-assign rule with target `goal` and precondition $\widehat{r} \wedge \bigwedge_{r \in S} \neg r$, where S is the set of all roles that have the permission p. Check role-reachability for `goal`.

IV. Policies to programs

In this section we describe two translations from ARBAC systems to integer programs. An *integer program* is a program in a simple imperative programming language with only integer variables, the usual control statements (`while`, `if`) and non-determinism, but without function calls. The program that we synthesize for an URA policy will simulate the user-role reassignments that can possibly result due to the policy, but the program will simulate this only at the level of cardinalities of the sets of users in each combination of roles. The resulting programs can be subject to static analysis techniques in order to answer role-reachability questions in the ARBAC system soundly.

The first translation is an exact reduction: we synthesize a program $\mathcal{P}$ with an error configuration for a role-reachability problem on an ARBAC system such that the target role is reachable in the ARBAC system iff the error configuration is reachable in the program $\mathcal{P}$. The main idea is that $\mathcal{P}$ uses an integer variable for each complete Venn region over the set of roles R.

$$
\begin{array}{lcl}
\langle stmt \rangle & ::= & \texttt{skip}; \mid \texttt{assume}(\langle bexpr \rangle); \mid \\
 & & var := \langle nexpr \rangle; \mid var := \star; \mid \\
 & & \texttt{if}(\langle bexpr \rangle)\ \texttt{then}\ \langle stmt \rangle\ \texttt{else}\ \langle stmt \rangle\ \texttt{fi} \mid \\
 & & \texttt{while}\ (\langle bexpr \rangle)\ \texttt{do}\ \langle stmt \rangle\ \texttt{od} \mid \\
 & & \langle stmt \rangle \langle stmt \rangle \\
\langle bexpr \rangle & ::= & \texttt{true} \mid \texttt{false} \mid * \mid \langle nexpr \rangle <= \langle nexpr \rangle \mid \\
 & & \neg\langle bexpr \rangle \mid\langle\ bexpr \rangle \wedge \langle bexpr \rangle \\
\langle nexpr \rangle & ::= & c \mid var \mid -\langle nexpr \rangle \mid \\
 & & \langle nexpr \rangle + \langle nexpr \rangle \mid\langle\ nexpr \rangle - \langle nexpr \rangle \\
 & & \text{where } c \in \mathbb{N}.
\end{array}
$$

Figure 1. Syntax of integer programs.

The precise translation, however, is infeasible in practice, as it produces integer programs of exponential size in the number of roles. When the number of roles is high, the synthesized program $\mathcal{P}$ can be prohibitively large, rendering program verification techniques useless.

Our second translation is a translation that uses set-abstractions. We propose to track only a subset of combinations of roles. This set is assumed to be given as a parameter to the translation, and in practice (see Section VI) we take this to be the combinations mentioned in the can-assign rules in the ARBAC system. Using fewer *tracking* variables to track only a subset of role-combinations loses precision, naturally. Our translation however guarantees soundness: if the synthesized program cannot reach the error configuration, then we are guaranteed that the target role is unreachable in the ARBAC system.

Before presenting the two translations (Section IV-A and Section IV-B), we fix some notation and define integer programs. Throughout this section, we fix an ARBAC system $\mathcal{S}$ over the roles R, with URA policy control $\langle can\_assign, can\_revoke \rangle$.

Venn-regions: A *Venn-region* over the set of roles R is any pair (P, N) with $P, N \subseteq R$ and $P \cap N = \emptyset$. Intuitively, such a role combination stands for the set of users that belong to each role $r \in P$ and do not have any of the roles $r' \in N$. If $P \cup N = R$, then we refer to the role-combination as a *complete* Venn-region.

Given two Venn-regions $(P, N), (P', N')$, (P, N) is said to be *coherent with* (P', N') if $(P \cup P', N \cup N')$ is a Venn-region (intuitively, the two regions do not contradict). Furthermore, (P, N) is *coherently under* (P', N') if $P \subseteq P'$ and $N \subseteq N'$.

Integer programs: Let us fix the syntax of a simple sequential programming language whose variables are of integer type, and with explicit syntax for non-determinism. Integer programs are described by the grammar given in Figure 1.

A program consists of atomic statements combined with sequential composition and control-flow constructs like *conditional* statements and *while-loops*. Atomic statements include *skip*, *assumes*, and *assignments* that assign an (integer)

variable to integer expressions or a nondeterministically chosen value (denoted by *). *Boolean expressions* are built in the standard way, starting from relational comparisons between integer expressions; Boolean expressions can also be the constants true, false, or a non-deterministically chosen value of true or false (denoted by *). Numerical expressions are built from integer constants or variables, and can be combined using standard integer operators.

The semantic of integer programs is the standard one, like C programs. We assume that all variables are initialized to 0. The only non-standard statements are the non deterministic choices, and the *assume* statement. When the program executes an *assume*(b) statement, it evaluates b, and if b is true, it continues executing the program; however, if b evaluates to false, then the program silently terminates (without an error).

A. Precise transformer

In this section we describe a translation $[\![\cdot]\!]_{exact}$ from an ARBAC system $\mathcal{S}$ to an integer program $\mathcal{P}$ that faithfully simulates the evolutions of $\mathcal{S}$. Furthermore, we reduce the role-reachability problem in $\mathcal{S}$ to a reachability problem in the corresponding program $\mathcal{P}$. We do not describe the synthesized program $\mathcal{P}$ formally or in great detail, as this transformation will not feature in the final tool we build; however, a brief account of how this program is constructed will be useful in motivating the abstract program that we construct in the next section.

In the rest of the section, we refer to $\mathcal{P}$ as the program $[\![\mathcal{S}]\!]_{exact}$.

Let V be the set of all *complete* Venn-regions (i.e. Venn-regions (P, N) such that every role $r \in R$ is either in P or in N; i.e. $N = R \setminus P$). Each Venn-region (P, N) in V stands for the set of users whose precise role-membership is P. The program $\mathcal{P}$ will have an integer variable $c_{(P,N)}$, for each Venn-region (P, N) in V, that keeps track of the *cardinality* of the set of users that (P, N) represents.

In the initial configuration, the various integers are initialized according to the initial configuration of the ARBAC system $\mathcal{S}$.

The program $\mathcal{P}$ simulates the evolution of the ARBAC system $\mathcal{S}$ by simulating every *can_assign* and every *can_revoke* rule, choosing to simulate any rule non-deterministically, and does this recursively, forever.

A *can_assign* rule is simulated by appropriately changing the integers representing the various regions. Consider a rule $(admin, Pos, Neg, r) \in can\_assign$. The program $\mathcal{P}$ simulates this rule by:

- Let X be the set of all complete Venn-regions (P, N) such that (P, N) is coherent with (Pos, Neg) and $r \in N$. If $(\Sigma_{(P,N)\in X} \ c_{(P,N)}) > 0$, then the rule can be fired. In this case, pick non-deterministically a region $(P_0, N_0) \in X$ such that $c_{(P_0,N_0)} > 0$. If the rule is not fireable, abort.
- If the rule is determined to be fireable by the above condition, then decrement $c_{(P_0,N_0)}$, and increment $c_{(P_0\cup\{r\},N_0\setminus\{r\})}$.

The *can_revoke* rules are simulated in an analogous manner.

Reachability reduction. We can now show that for any ARBAC system $\mathcal{S}$, and role `goal`, `goal` is reachable from the initial configuration iff the program $\mathcal{P}$ can reach a configuration where some $c_{(P,N)} > 0$, with `goal` $\in P$.

In fact, we can add a statement in $\mathcal{P}$ that checks whether such a configuration is reached, and if so go to an error configuration; hence role-reachability reduces to *program location* reachability.

We can hence verify the security of the ARBAC system by proving correctness of the corresponding program $\mathcal{P}$. We can use any of the variety of techniques for program verification to solve this problem, including abstraction based techniques as we do in this paper.

B. Approximate transformer

Although the translation $[\![\cdot]\!]_{exact}$ above allows us to simulate precisely any execution of an ARBAC system $\mathcal{S}$, the number of variables required to track the system in $\mathcal{P} = [\![\mathcal{S}]\!]_{exact}$ is exponential in the number of roles in R. Thus, when R contains a large number of roles, the size of $\mathcal{P}$ can be prohibitively high and prevents any practical use of static analysis on program.

To circumvent this explosion of variables, we propose an alternative analysis that analyzes the policy using *abstractions*. Rather than track *all* the complete Venn-regions of roles, we propose to track only a subset of Venn-regions (and allow incomplete Venn-regions as well). Given a subset of Venn-regions *Track*, our abstract analysis will *soundly* track the evolution of the ARBAC system–if the role is found unreachable in the abstract analysis, we will be guaranteed that the role is unreachable in the concrete system as well. However, our analysis is an abstraction; if the role is found to be reachable in the abstract analysis, then we are not guaranteed that the role is reachable in the ARBAC system. Consequently, our technique is useful only for proving non-reachability of roles (which is often the more important property, for example in privilege escalation and separation of duty analysis).

We present the abstract analysis in two phases. In the first phase, we set up an abstract transition system that simulates the ARBAC system abstractly using a set of Venn-regions *Track*. The design of this abstract system, especially the updates to the abstract state, is quite involved, and much more intricate than the exact translation described above. We describe the abstraction, and prove its soundness. In the second phase, we implement the abstract transition system in our imperative programming language.

In the rest of the section we assume $Track \subseteq \mathcal{T} = \{(P, N) \mid \ P, N \subseteq R, P \cap N = \emptyset\}$. The set $Track$ could

be initialized as shown in SectionVI.

Phase I. The abstract transition system:
Abstract states of the system are characterized by integer variables $c_{(P,N)}$ for each Venn-region $(P,N) \in \mathit{Track}$, which intuitively stands for the number of users that have all the roles in P and none of the roles in N (their membership in roles outside $P \cup N$ do not matter). Since there can be several concrete configurations corresponding to an abstract state, the update of an abstract state according to a *can_assign* rule is *not* deterministic (unlike the exact translation). Updating the abstract state non-deterministically that takes into account all possible evolutions of concrete configurations corresponding to it, without explicitly enumerating the concrete configurations, and doing it fairly accurately, is hard, and is the main contribution of this section.

Formally, we define the set of *abstract states* of $\mathcal{S}$ with respect to the set Track as $A_{\mathit{Track}} = \{a \mid a : \mathit{Track} \to \mathbb{N}\}$.

With an abuse of notation in the rest of the section we refer to $C = \{UR \mid (\widehat{U}, UR)$ is an $\mathcal{S}$ configuration$\}$ as the set of (concrete) configurations of $\mathcal{S}$. Let us now define an abstraction function α_{Track} that associates each concrete configuration with precisely one abstract state: $\alpha_{\mathit{Track}} : C \to A_{\mathit{Track}}$ is the *abstraction map* defined as:

$$\alpha_{\mathit{Track}}(c) = a \text{ where}$$
$$a((P,N)) = |\{u \in \widehat{U} \mid \forall r \in P.(u,r) \in c, \forall r \in N.(u,r) \notin c\}|.$$

In other words, the map a associates to each pair (P,N) the number of users that are in all roles P and not in any role of N, for every $(P,N) \in \mathit{Track}$.

Building the Abstract transformer. We now describe how to update an abstract state for a *can_assign* rule.

Let $ca \in \mathit{can\_assign}$, and $\tau_{ca} \subseteq C \times C$ be the concrete transition relation of $\mathcal{S}$ on the rule ca. For any ca, we define the *abstract transformer* $\widehat{\tau}_{ca} \subseteq A_{\mathit{Track}} \times A_{\mathit{Track}}$ as follows.

Let $\mathbf{b} = \langle b_r \rangle_{r \in R}$ (respectively, $\mathbf{d} = \langle d_r \rangle_{r \in R}$) denote a tuple of Boolean variables, one for each role. Interpreting the valuation of $\mathbf{b}$ (respectively, $\mathbf{d}$) as the membership status of a user (respectively, administrator) in each role, we can define when a Venn-region (P,N) is coherently under $\mathbf{b}$ (respectively, $\mathbf{d}$) using the following formula:

$$\mathit{coh}((P,N),\mathbf{b}) = \bigwedge_{r \in P} b_r \wedge \bigwedge_{r \in N} \neg b_r$$

Let $ca = (\mathit{admin}, \mathit{Pos}, \mathit{Neg}, t)$. Then, for any $a, a' \in A_{\mathit{Track}}$, $\widehat{\tau}_{ca}(a,a')$ holds iff the following holds:

$$\exists \mathbf{b}.\ (\psi(\mathbf{b}) \wedge \mathit{coh}((\mathit{Pos},\mathit{Neg}),\mathbf{b}) \wedge \varphi_{ca}(\mathbf{b}))$$
$$\wedge \quad \exists \mathbf{d}.\ (\psi(\mathbf{d}) \wedge d_{\mathit{admin}})$$

where

$$\psi(\mathbf{b}) = \bigwedge_{A=(P,N) \in \mathit{Track}} (\mathit{coh}(A,\mathbf{b}) \Rightarrow a(A) > 0)$$

$$\begin{aligned}
\varphi_{ca}(\mathbf{b}) = & \bigwedge_{A=(P,N) \in \mathit{Track}, t \notin P \cup N} a'(A) = a(A) \\
\wedge & \bigwedge_{\substack{A=(P,N) \in \mathit{Track} \\ t \in P}} (\mathit{coh}((P \setminus \{t\}, N), \mathbf{b}) \Rightarrow a'(A) = a(A)+1) \\
\wedge & \bigwedge_{\substack{A=(P,N) \in \mathit{Track} \\ t \in P}} (\neg \mathit{coh}((P \setminus \{t\}, N), \mathbf{b}) \Rightarrow a'(A) = a(A)) \\
\wedge & \bigwedge_{\substack{A=(P,N) \in \mathit{Track} \\ t \in N}} (\mathit{coh}(A,\mathbf{b}) \Rightarrow a'(A) = a(A)-1) \\
\wedge & \bigwedge_{\substack{A=(P,N) \in \mathit{Track} \\ t \in N}} (\neg \mathit{coh}(A,\mathbf{b}) \Rightarrow a'(A) = a(A))
\end{aligned}$$

In the above, the formula expresses that there is an abstract transition from a to a' if there exists some user u whose role-memberships are given by $\mathbf{b}$ that is in accordance with the current configuration (captured by $\psi(\mathbf{b})$) such that the *can_assign* rule is applicable for u (captured by $\mathit{coh}((\mathit{Pos},\mathit{Neg}),\mathbf{b})$) and the updated abstract state reflects the assigning of the role t to this user (captured by $\varphi_{ca}(\mathbf{b})$). Furthermore, there exists an administrator in role admin whose role-memberships are given by $\mathbf{d}$ that is in accordance with his configuration (realized by the formula $\psi(\mathbf{d}) \wedge d_{\mathit{admin}}$).

The formula $\psi(\mathbf{b})$ says that a user with chosen role-membership according to $\mathbf{b}$ is satisfied in some concretization of a. This formula does this check by ensuring that every Venn-region that is tracked and coherent with the user's role-membership has a count greater than 0. The formula $\mathit{coh}((\mathit{Pos},\mathit{Neg}),\mathbf{b})$ checks that the pre-condition of the *can_assign* rule is consistent with the chosen vector $\mathbf{b}$. Finally, the condition $\varphi_{ca}(\mathbf{b})$ checks whether the new abstract state reflects the user u acquiring the role t. This is done by (a) ensuring that the counters for Venn-regions that do not mention t do not change at all, (b) ensuring that a counter with a Venn-region consistent with u's pre-state except for positively including role $\{t\}$ gets incremented, and (c) ensuring that the counter for a Venn-region consistent with u's pre-state but that mentions $\{t\}$ in the negative gets decremented.

We skip the abstract transformer for *can_revoke* rules, which is similar.

For any $a, a' \in A_{\mathit{Track}}$ and any $m \in (\mathit{can\_assign} \cup \mathit{can\_revoke})$, whenever $\widehat{\tau}_{ca}(a,a')$ holds we denote it as $a \xrightarrow{\widehat{\tau}_m}_A a'$.

The lemma below gives the main technical result, namely that the above abstract transformer soundly abstracts the concrete rule. It says that if a configuration c is transformed to c' in the concrete ARBAC system, then there is an abstract transition system from the abstract state corresponding

to c ($\alpha_{Track}(c)$) to the abstract state corresponding to c' ($\alpha_{Track}(c')$).

Lemma 4.1: Let c, c' be two configurations of the ARBAC system $\mathcal{S}$ and $c \xrightarrow{\tau_m}_{\mathcal{S}} c'$. Let $a = \alpha_{Track}(c)$ and $a' = \alpha_{Track}(c')$. Then $a \xrightarrow{\widehat{\tau}_m}_A a'$.

Sketch of proof. Here we give a sketch of the proof when m is a can-assign rule. Let $ca = (admin, Pos, Neg, t) \in can\_assign$. From the hypothesis of the lemma, $c \xrightarrow{\tau_{ac}}_{\mathcal{S}} c'$ and hence there must exist (1) a user $u \in \widehat{U}$ such that $conf(c, u) = (Pos_u, Neg_u)$ is coherent under (Pos, Neg) and $c' = c \cup \{(u, t)\}$, and (2) an administrator ad such that $(ad, admin) \in c$.

Let $conf(c, ad) = (Pos_{ad}, Neg_{ad})$. Now by picking $\langle b_r \rangle_{r \in R}$ (respectively, $\langle d_r \rangle_{r \in R}$) such that $b_r = true$ (respectively, $d_r = true$) iff $r \in Pos_u$ (respectively, $r \in Pos_{ad}$), it is direct to prove that $(\psi(\mathbf{b}) \wedge coh((Pos, Neg), \mathbf{b}) \wedge \varphi_{ca}(\mathbf{b})) \wedge (\psi(\mathbf{d}) \wedge d_{admin})$ holds true, and hence $a \xrightarrow{\widehat{\tau}_m}_A a'$.

Phase II. Integer program simulation:
The sound abstract transformer described above can be translated into an integer program. We create a program $\mathcal{P}$ that has one integer variable $c_{(P,N)}$ for each $(P, N) \in Track$. Hence the abstract state is captured by the valuation of these variables in the program.

To construct the program $\mathcal{P}$, we have a recursive loop, where we simulate non-deterministically the abstract transitions. This essentially calls for translating the abstract transition relation described above in terms of a program. Though the formulas above seem complex, this is not hard to achieve. We sketch the main points below:

- The existential quantifier over variables $\mathbf{b}$ is simulated in the program by having a set of Boolean variables $\mathbf{b}$, and by assigning them non-deterministically a truth value. This is done with a sequence of statements of the form $b_r := *;$ `assume`$((b_r = 0) \vee (b_r = 1));$. The same is done for the variables $\mathbf{d}$.
- The program then checks whether $\mathbf{b}$ is consistent with the current state by executing an assume statement on the formula $\psi(\mathbf{b})$.
- The program then checks whether the *can_assign* rule is applicable for $\mathbf{b}$ by executing an assume statement on the formula $coh((Pos, Neg), \mathbf{b})$.
- Finally, the program updates the current state of the variables $c_{(P,N)}$ by checking the various subformulas on the left-hand side of the conjuncts in φ_{ca}, and incrementing, decrementing, or keeping the same value of the various counters, as appropriate.

We skip the formal program corresponding to the abstract transitions, as it is fairly straightforward.

For example, consider the scenario where we have three roles $\{A, B, C\}$, and assume that $Track = \{(\{A, B\}, \{C\}), (\{B, C\}, \emptyset)\}$. Consider the can-assign rule $(B, \{A\}, \{B\}, C)$. Then the program that does the abstract update of the variables is given by:

```
assume(        // Implementation of ψ(b)
          ((b_A ∧ b_B ∧ ¬b_C) ⇒ c_({A,B},C) > 0)
        ∧ ((b_B ∧ b_C) ⇒ c_({B,C},∅) > 0)
        ∧ (b_A ∧ ¬b_B)   // Impl. of coh((Pos, Neg), b)
              // Implementation of ψ(d) ∧ d_admin
          ((d_A ∧ d_B ∧ ¬d_c) ⇒ c_({A,B},C) > 0)
        ∧ ((d_B ∧ d_C) ⇒ c_({B,C},∅) > 0)) ∧ d_B);
   // Implementation of φ_ca(b)
if (b_B) then c_({B,C},∅) := c_({B,C},∅) + 1; fi
if (b_A ∧ b_B ∧ ¬b_C) then c_({A,B},C) := c_({A,B},C) − 1; fi
```

Over-approximation guarantee: From the fact that the abstract transition system is an over-approximation of the ARBAC system, and from the fact that the program defined above realizes the abstract transition system, we conclude the following soundness theorem: (below, $[\![\mathcal{S}, Track]\!]_{approx}$ denotes the abstract program constructed above).

Theorem 4.2: (OVER-APPROXIMATE SIMULATION)
Let $\mathcal{S}$ be an ARBAC system over the set of roles R, $t \in R$, $Track \subseteq \mathcal{T} = \{(P, N) \mid P, N \subseteq R, P \cap N = \emptyset\}$ with $(\{t\}, \emptyset) \in Track$, and $\mathcal{P} = [\![\mathcal{S}, Track]\!]_{approx}$.
Then, the role t is reachable in $\mathcal{S}$ if a program state is reachable in $\mathcal{P}$ with variable $c_{(\{t\},\emptyset)} > 0$.

V. SLICING

In this section we give a procedure to simplify ARBAC role-reachability problems by safely removing a set of roles and the rules that involve them. Intuitively, a role is *uninteresting* if it is not acquired by a user starting from the initial state and reaching the target role. Finding the precise set of uninteresting roles is hard, but we do a quick under-approximation of this set using simple slicing, where we track only individual roles (as opposed to combinations of roles). Removing these roles is safe and simplifies the reachability problem. A slicing technique similar to our can be found in [12].

Our procedure works by repeating two phases, until a fixed point is reached. Each phase itself is a fixed point computation followed by a simplification of the system.

In the forward phase we compute an over-approximation of reachable goals of $\mathcal{S}$ as the fixed point of the following set of equations. Let $Init$ be the initial configuration of $\mathcal{S}$. Then, $S_0 = \{r \mid \exists u.(u, r) \in Init\}$, and for $i > 0$, $S_i = S_{i-1} \cup \{r \mid (admin, P, N, r) \in can\_assign, P \cup \{admin\} \subseteq S_{i-1}\}$. In other words, we start with S_0 as the set of all roles containing at least one user in the initial configuration of $\mathcal{S}$, and for each $i > 0$, we add in S_i the targets of all can-assign roles ca such that the roles in the positive precondition and the administrative role of ca are already in S_{i-1}. It is easy

to see that the fixed point reached will be a set S^* which is an over-approximation of the set of reachable roles. We then simplify the system $\mathcal{S}$ by (a) remove all *can_assign* rules which mention in the positive precondition any role in $R \setminus S^*$ or have a role in $R \setminus S^*$ as the target, (b) remove all *can_revoke* rules referring to any role in $R \setminus S^*$, (c) remove the roles $R \setminus S^*$ from the negative preconditions of all the rules, and (d) remove the roles $R \setminus S^*$ from the initial RBAC system.

In the backward phase, we start with a system $\mathcal{S}$ (obtained as the result of the above phase), and compute the fixed point of the equations: $S'_0 = \{\texttt{goal}\}$, and for $i > 0$, $S'_i = S'_{i-1} \cup \bigcup_{(admin,P,N,r) \in can\_assign, r \in S'_{i-1}} (P \cup N)$. This process will also reach a fixed-point, S'^*. $R \setminus S'^*$ are useless roles, as membership or non-membership in these roles are not important to reach the *target* role `goal`. We then simplify the system $\mathcal{S}$ by (a) remove all *can_assign* rules which have in the positive precondition a role in S'^* or have as target a role in S'^*, (b) remove all *can_revoke* rules referring to any role in S'^*, (c) remove the roles S'^* from the negative preconditions of all the rules, and (d) remove the roles S'^* from the initial ARBAC system.

We continue the above two phases till the system stabilizes and reaches a fixed-point. This system $\mathcal{S}'$ is the slicing of the original system $\mathcal{S}$.

It is not hard to see that the slicing above preserves the reachability of the role `goal`. Hence,

Theorem 5.1: Let $\mathcal{S}$ be a ARBAC system and $\mathcal{S}'$ be the slicing of $\mathcal{S}$ w.r.t. the role `goal`. Then the role `goal` is reachable in $\mathcal{S}$ iff `goal` is reachable in $\mathcal{S}'$.

VI. Choosing the Venn-regions to track

The set of Venn-regions $Track$ should be small enough for the analysis to be practical while preserving a certain level of accuracy to make the analysis effective. We notice that in order to assign a user to a role (i.e., firing a can-assign rule), it is significant for accuracy to check the existence of either an administrator enabled to fire the rule and that of a user whose configuration is consistent with the precondition of the can-assign rule. Thus, we choose $Track$ according to the *preconditions* of the can-assign rules. More in details, $Track$ contains the Venn-regions (Pos, Neg) and $(admin, \emptyset)$ for each $(admin, Pos, Neg, r) \in can\_assign$, $(admin, \emptyset)$ for each $(admin, r) \in can\_revoke$ and $(goal, \emptyset)$.

Such a choice requires to track a linear number of role-combinations (linear in the number of can-assign rules), as opposed to the exponential number of role-combinations deriving by all complete Venn-regions which would make the approach impractical for even medium-sized policies. Intuitively, the set $Track$ chosen provides a certain minimal level of precision. Clearly, we can prove only empirically that such a *minimal* choice is effective, which we do in the next section.

VII. Evaluation

We implemented a prototype tool, VAC (Verifier of Access Control), that realizes the abstract analysis set forth in this paper to verify ARBAC policies for role-reachability. VAC is available at the website `http://users.ecs.soton.ac.uk/gp4/VAC.html`.

Given an ARBAC system, VAC first executes the slicing phase, as described in Section V, to reduce the number of roles and rules of the system, if possible. Then, it translates the obtained ARBAC system into an integer program instrumented for the INTERPROC analyzer, according to the approximation described in Section IV-B. We parameterize the translation by choosing the set of Venn-regions as explained in SectionVI.

Finally, VAC uses the INTERPROC static analyzer to perform the interval analysis, and checks whether the program can reach the particular program location corresponding to the target role being reached (INTERPROC marks unreachable locations with $\bot$).

Our experiments are conducted on three sets of realistic ARBAC policies. The first two policies refer to a hospital policy and a university policy used in several case studies in the literature [27], [5], while the third one describes an ARBAC policy for large bank comprising a number of identically structured branches [8], [9]. In the bank system, each branch consists of four divisions, each having five non-managerial roles and two managerial ones. In the following we will refer to the ARBAC policy of the bank with i branches as Bank$_i$. In all the policies we allow an unbounded number of users that can be recruited into any of the individual roles of the system (and then acquire more roles according to the rules).

In all the experiments the input consists of an ARBAC policy and a role-reachability question. We consider Privilege Escalation Properties (PEP), (e.g. an employee of a lower rank gaining access to resources meant for a higher rank), and Separation of Duty (SoD) constraints that model conflict of interest.

Table I summarizes our experiment results. We report, for an ARBAC policy and role-reachability query, the number of roles and rules in the system, the number of roles and rules after slicing, the size of the synthesized abstract program and the time it took to synthesize it, the time taken by INTERPROC to do the analysis, the total time taken, and whether the role was found unreachable (safe).

The experiments were conducted on an Intel Xeon Quad-Core 2.33GHz machine with 8Gb RAM.

Table I is divided into five sets of experiments as follows. The first set of experiments were executed on the hospital policy. The first experiment checks that a patient cannot have privileges of his own primary doctor! The second one tests that a user cannot be a member of both the roles receptionist and doctor, thus avoiding fraud by preventing the user to

	ARBAC Policy	Property			After Slicing		VAC				
			#roles	#rules	#roles	#rules	LOC of synthesized program	Time taken to transform	INTERPROC Analysis Time	Total Time	Result
1	Hospital	PEP	12	24	3	4	73	0.3s	0s	0.3s	safe
	Hospital	SoD	12	24	5	8	134	0.3s	0.01s	0.3s	safe
2	University	PEP	32	132	5	9	162	0.5s	0.1s	0.6s	safe
	University	SoD	32	132	13	37	541	0.6s	0.2s	0.8s	safe
	University	SoD	32	132	15	43	535	0.6s	0.2s	0.8s	safe
3	$Bank_1$	SoD	34	593	34	593	13,356	7s	44s	51s	safe
	$Bank_2$	SoD	68	1186	68	1186	26,684	9s	3m 02s	3m 11s	safe
	$Bank_3$	SoD	102	1779	102	1779	40,012	11s	7m 08s	7m 19s	safe
	$Bank_4$	SoD	136	2372	136	2372	53,340	11s	13m 16s	13m 27s	safe
4	$Bank_2$	SoD	68	1186	68	1186	26,705	9s	3m 03s	3m 12s	safe
	$Bank_3$	SoD	102	1779	102	1779	40,045	10s	7m 08s	7m 18s	safe
	$Bank_4$	SoD	136	2372	136	2372	53,383	9s	13m 15s	13m 24s	safe
5	University	SoD	32	132	17	53	698	0.6s	0.3s	0.9s	-
	$Bank_1$	SoD	34	593	34	593	16,577	4s	1m 31s	1m 35s	-

Table I
EXPERIMENTAL RESULTS.

falsely claim to treat a patient and billing the insurance company.

The second set of experiments were performed on the university policy and consider both privilege escalation properties and separation of duties. The first experiment verifies whether a department chair may gain the privileges of a Dean, the second asks whether a student may be simultaneously a grad student and an undergraduate, and the third one checks separation of duties between an admission officer and a graduate admission officer.

In the third set, experiments were conducted on $Bank_i$ policies (the bank policy on i branches). They consider the query: *can a user be assigned to four non-managerial roles in a business division in any of the i branches?* The fourth set is also on the Bank policies, asking: *can a user be assigned to four non-managerial roles in a business division in* all *the i branches?*

The final set of experiments were performed for queries where the role is reachable (i.e. unsafe). The first query tests whether a user may belong both to undergraduate and honors student roles in the university system. The second tests whether a user may belong to two non-managerial roles in the same division of the bank case study.

Observations: The experiments clearly show that our abstraction schemes and usage of program analysis tools scale to verify the security of large ARBAC policies. Note that these policies are quite large, and given that the computational complexity of precise checking is exponential in the number of roles (which are in the hundreds in these benchmarks), precise checking will not scale to handle them. The abstraction-based approach, by keeping track of only a subset of combinations of roles, and further by performing numerical interval-based abstraction on those, is able to prove the policies safe. We consider this remarkable given that the tool proves the policies entirely correct, in the presence of an unbounded number of users.

While the static slicing does help in some experiments, it was not helpful on the *Bank* examples as all roles and rules were relevant for the target. In general, the time taken to synthesize programs was negligible, despite creating programs that span tens of thousands of lines of code! Furthermore, INTERPROC performs extremely well over these programs, as it is a scalable interval-based analysis, and proves safety in reasonable time.

On the experiments on policies where the target role is reachable, our tool obviously failed to find a safety proof (as our techniques are sound), and quickly reached unsafe fixed-points.

We believe our technique shows promise in being the foundations for a versatile tool for verification of security of role-based access control policies.

VIII. CONCLUSIONS AND FUTURE WORK

The work set forth in this paper solves the security problem for role-reachability in ARBAC systems using abstract analysis. Furthermore, it achieves the abstract analysis by reducing the problem to a program analysis problem, using a novel set abstraction scheme. The resulting program can then be analyzed using abstract numerical domains, exploiting existing efficient fixed-point algorithms developed in this domain. We believe that an abstraction-based approach to access control verification paves the way to exploiting the

simplicity in the instances, avoiding the pitfalls of undecidability and complexity hardness for this problem.

An important future extension of this work would be a *counter-example guided* abstraction scheme. Intuitively, instead of choosing the role combinations statically, we can let the combinations be chosen using iterative abstractions, where the abstraction is refined using concrete counter-examples obtained in each iteration (similar to such CEGAR schemes in program verification [2]). This involves several technical challenges: to generate concrete counter-examples from abstract interval abstractions, to learn role-combinations from them that avoid the counter-example, and incorporating those into a refinement scheme. Such a technique would be valuable as it will more closely exploit the simplicity of the instance, and give administrators concrete attacks that can happen in their system.

Acknowledgements: This research was partially supported by NSF CCF #1018182 and NSF CAREER #0747041.

REFERENCES

[1] A. Armando and S. Ranise. Automated symbolic analysis of arbac-policies. In *STM*, volume 6710 of *LNCS*, pages 17–34. Springer, 2010.

[2] T. Ball, B. Cook, V. Levin, and S. K. Rajamani. Slam and static driver verifier: Technology transfer of formal methods inside microsoft. In *IFM*, volume 2999 of *LNCS*, pages 1–20. Springer, 2004.

[3] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977.

[4] D. Ferraiolo and R. Kuhn. Role-based access control. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[5] 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*, volume 5505 of *LNCS*, pages 46–49. Springer, 2009.

[6] G. S. Graham and P. J. Denning. Protection: principles and practice. In *Proc. of the AFIPS Spring Joint Computer Conference*, pages 417–429, 1972.

[7] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. On protection in operating system. In *SOSP*, pages 14–24, 1975.

[8] K. Jayaraman, V. Ganesh, M. Tripunitara, M. C. Rinard, and S. J. Chapin. Arbac policy for a large multi-national bank. In *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*, pages 163–174. ACM, 2011.

[10] B. Jeannet and A. Miné. Apron: A library of numerical abstract domains for static analysis. In *CAV*, volume 5643 of *LNCS*, pages 661–667. Springer, 2009.

[11] B. Jeannet, et al. The interproc analyzer. *http://pop-art.inrialpes.fr/interproc/interprocweb.cgi*.

[12] 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.

[13] N. Li, J. C. Mitchell, and W. H. Winsborough. Beyond proof-of-compliance: security analysis in trust management. *J. ACM*, 52(3):474–514, 2005.

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

[15] N. Li and M. V. Tripunitara. On safety in discretionary access control. In *IEEE Symp. on S&P*, pages 96–109. IEEE Computer Society, 2005.

[16] R. J. Lipton and L. Snyder. A linear time algorithm for deciding subject security. *J. ACM*, 24(3):455–464, 1977.

[17] Muchnick and Steven S. *Advanced compiler design and implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[18] A.C. O'Connor and R.J. Loomis. Economic analysis of role-based access control. In *Final Report. RTI International, Project Number 0211876*, 2010.

[19] R. S. Sandhu. The typed access matrix model. In *IEEE Symp. on S&P*, pages 122–136, 1992.

[20] 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.

[21] 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.

[22] R. S. Sandhu and Q. Munawer. The arbac99 model for administration of roles. In *ACSAC*, pages 229–238. IEEE Computer Society, 1999.

[23] A. Sasturkar, P. Yang, S. D. Stoller, and C. R. Ramakrishnan. Policy analysis for administrative role based access control. In *CSFW*, pages 124–138. IEEE Computer Society, 2006.

[24] A. Sasturkar, P. Yang, S. D. Stoller, and C. R. Ramakrishnan. Policy analysis for administrative role-based access control. *Theor. Comput. Sci.*, 412(44):6208–6234, 2011.

[25] J. A. Solworth and R. H. Sloan. A layered design of discretionary access controls with decidable safety properties. In *IEEE Symp. on S&P*, pages 56–. IEEE Computer Society, 2004.

[26] 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.

[27] S. D. Stoller, Ping Yang, C. R. Ramakrishnan, and M. I. Gofman. Efficient policy analysis for administrative role based access control. In *CCS*, pages 445–455. ACM, 2007.