Efficient Symbolic Automated Analysis of Administrative Attribute-based RBAC-Policies

Francesco Alberti
Università della Svizzera Italiana, Lugano
francesco.alberti@usi.ch

Alessandro Armando
Università di Genova and FBK-Irst, Trento (Italy)
armando@fbk.eu

Silvio Ranise
Security and Trust Unit, FBK-Irst, Trento (Italy)
ranise@fbk.eu

ABSTRACT

Automated techniques for the security analysis of Role-Based Access Control (RBAC) access control policies are crucial for their design and maintenance. The definition of administrative domains by means of attributes attached to users makes the RBAC model easier to use in real scenarios but complicates the development of security analysis techniques, that should be able to modularly reason about a wide range of attribute domains. In this paper, we describe an automated symbolic security analysis technique for administrative attribute-based RBAC policies. A class of formulae of first-order logic is used as an adequate symbolic representation for the policies and their administrative actions. State-of-the-art automated theorem proving techniques are used (off-the-shelf) to mechanize the security analysis procedure. Besides discussing the assumptions for the effectiveness and termination of the procedure, we demonstrate its efficiency through an extensive empirical evaluation.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Software/Program Verification

General Terms

Security, Automated Verification

Keywords

Access Control, Policy, Symbolic Model Checking

1. INTRODUCTION

Access control is one of the key ingredients to ensure the security of distributed systems where several users may perform actions on shared resources. To guarantee the flexibility and scalability of these systems, access control is managed by several security officers that may delegate permissions to other users. In this context, security analysis is critical for the design and maintenance of access control policies.

ASIACCS '11, March 22–24, 2011, Hong Kong, China.

In order to analyze access control policies independently of the mechanisms used to enforce them, the current practice is to clearly separate these two levels and use a security model with a suitable specification language to design a set of access control policies. Role Based Access Control (RBAC) [20] is one of the most popular security models for access policies which regulates access by assigning users to roles which, in turn, are granted permissions to perform certain operations. On top of RBAC, several administrative models have been proposed to address the challenges posed by administrating large decentralized RBAC policies (see, e.g., [17] for an in-depth discussion of this and related problems). The various proposals differ for a variety of aspects but one of the most important is how *administrative domains* are specified; i.e. when a security officer delegates (part of his) administrative permissions to a (partially) trusted administrator, such permissions should be limited to a portion of the RBAC state. The idea underlying ARBAC (see, e.g., [8]) is to use again RBAC (in particular, role hierarchies) to specify administrative domains. However, it has been observed [16] that this is problematic in real world scenarios since administrative domains mirror the organization structure while roles are based on job functions and these two notions are often quite different. To alleviate these problems, several alternatives have been put forward (see [17] for an overview). As a consequence, automated security analysis techniques are stretched between *expressiveness*, to accommodate the various specifications of the administrative domains, and *scalability*, to permit security analysis of large (real world) policies.

The main contribution of this paper is a security analysis technique based on symbolic model checking that allows us to address the issues of expressiveness and scalability discussed above. We do this by generalizing the notion of ARBAC policy introduced in [24] so as to permit the uniform mechanization of security analysis techniques for expressive specifications of administrative access control policies. We derive these techniques from a recently proposed symbolic model checking method for infinite state systems [11]. We detail the derivation in three steps. First, we introduce a translation from attribute RBAC policies and their administrative actions to a symbolic representation by means of formulae in a decidable fragment of first-order logic, called Bernays-Shönfinkel-Ramsey (BSR) class (see, e.g., [19]). Second, we explain how to mechanize a procedure to solve a reachability problem by using the symbolic representation previously introduced. Third, we show how to encode an interesting problem for security analysis, called

the user-role reachability problem, as a symbolic reachability problem defined in the previous step. The symbolic procedure uniformly accommodates the several different specifications of administrative domains that can be symbolically represented by formulae in the BSR class. This turns out a reasonable hypothesis for many practical applications. The termination of the analysis is guaranteed under the same hypothesis of representability of the administrative domains in the BSR class and follows from results in [11].

A second contribution of the paper is a discussion of the pragmatics of implementing the symbolic reachability procedure and a report of the experimental results with a prototype on a set of significant benchmarks which clearly demonstrate the scalability of the proposed technique. In fact, our findings show that our prototype scales better on the set of problems for ARBAC policies considered difficult in [24] compared to the state-of-the-art tool described in [24]. Since our tool can handle more expressive specification of administrative RBAC policies than that of [24], we propose a new set of synthetic benchmarks derived from those in [24] that the tool in [24] cannot handle because of the presence of an applicability condition on an attribute in the administrative actions. We believe that the new benchmarks are an interesting contribution *per se* that will stimulate the development of new automated analysis techniques.

The plan of the paper is as follows. Section 2 introduces the notion of attribute RBAC policies and the related user-role reachability problem. Section 3 provides a quick overview of symbolic backward reachability and of the requirements for its mechanization. Section 4 shows that BSR formulae are an adequate symbolic representation for attribute RBAC policies. Section 5 describes our implementation of symbolic reachability and reports on an extensive (comparative) evaluation on a set of benchmarks. Section 6 compares our approach with related work and concludes.

2. ADMINISTRATIVE RBAC

We generalize the RBAC model of [24] by allowing administrative domains to be defined in terms of roles and attributes. Let U, R, and P be (countably infinite) sets of users, roles, and permissions, respectively; let $An = \{\alpha_1, ..., \alpha_n\}$ be a finite set of attribute names, V_i be a (countably infinite) set of elements from which α_i takes values ($i \in \{1, ..., n\}$), $Rel_1, ..., Rel_k$ be relations over $V = \bigcup_{i=1}^{n} V_i$, and $\mapsto \subseteq U \times An \times V$ be a relation associating a user and an attribute name with a value. (The relation $\mapsto$ is not required to be functional, although in many cases is so, since the policy analysis techniques described in the following do not exploit this assumption.) We write $u.\alpha_i \mapsto v$ for $(u, \alpha_i, v) \in \mapsto$, where $v \in V_i$ and call $u.\alpha_i$ an *attribute*, for $i \in \{1, ..., n\}$. An *attribute RBAC policy* is a tuple $\langle U, An, V_1, ..., V_n, \mapsto, R, P, UA, PA, \succeq, Rel_1, ..., Rel_k\rangle$, where $UA \subseteq U \times R$ is the *user-role assignment* relation, $PA \subseteq P \times R$ is the *permission assignment* relation; and $\succeq \subseteq R \times R$ is a partial order on roles. (If $An = \emptyset$, then this notion reduces to that of hierarchic RBAC policy as defined, e.g., in [24].) Let r_1, r_2 be roles in R. If $r_1 \succeq r_2$, then we say that r_1 is *more senior than* r_2. If $(u, r) \in UA$, then u is an *explicit member* of UA and if $(u, r') \in UA$ for some $r' \succeq r$ and $r' \neq r$, then u is an *implicit member* of UA. A user u is a *member* of role r if u is an explicit or an implicit member of r.

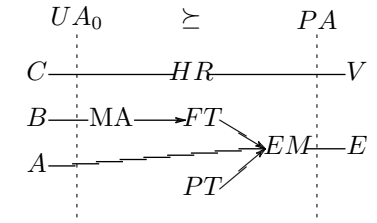


Figure 1: Graphical representation of ρ_0 in Ex. 1

Example 1. Let $U = \{A, B, C\}$, $P = \{E, V\}$, $R = \{HR, MA, FT, PT, EM\}$, $PA = \{(V, HR), (E, EM))\}$, and $\succeq$ is the least partial order containing $\{(MA, FT), (FT, EM), (PT, EM)\}$. An example of hierarchic RBAC policy [24] is $\rho_0 = \langle U, R, P, UA_0, PA, \succeq\rangle$ where $UA_0 = \{(A, EM), (B, MA), (C, HR)\}$ (see Fig. 1 for a graphical representation). For example, B is an implicit member of role EM because MA is more senior than EM (as both (MA, FT) and (FT, EM) are in $\succeq$) and A is an explicit member of role MA. We can extend this by considering an attribute name *age* taking values over the set $\mathbb{N}$ of the natural numbers endowed with the standard 'greater than' relation $>$. An example of attribute RBAC policy is $\rho_0^a = \langle U, \{age\}, \mathbb{N}, \mapsto, R, P, UA_0, PA, \succeq, >\rangle$ where UA_0 is as above, $A.age \mapsto 25$, $B.age \mapsto 45$, and $C.age \mapsto 38$. □

Along the lines of [24], the administrative policies control changes to the user-role assignment relation UA only, i.e. all the other sets and relations are constant. Preliminarily, we define pre-conditions. A *role literal* is an expression of the forms r or $\overline{r}$ for $r \in R$. An *attribute literal* is an expression of the forms $\mathsf{Rel_j}(\alpha_{i_1}, ..., \alpha_{i_r})$ or $\overline{\mathsf{Rel_j}}(\alpha_{i_1}, ..., \alpha_{i_r})$, where $\mathsf{Rel_j}$ is the *name of the relation* $Rel_j \subseteq (V_{i_1} \times \cdots \times V_{i_r})$, $j \in \{1, ..., k\}$, and $i_1, ..., i_r \in \{1, ..., n\}$. An *RBAC pre-condition* is a finite set of role literals. An *administrative pre-condition* is a finite set of role or attribute literals. In the rest of this section, let $\rho = \langle U, An, V_1, ..., V_n, \mapsto, R, P, UA, PA, \succeq, Rel_1, ..., Rel_k\rangle$. The *user u satisfies the administrative pre-condition C in the attribute RBAC policy ρ* iff for each literal $\ell \in C$, we have that (a) if ℓ is r, then u is a member of the role r, (b) if ℓ is $\overline{r}$, then u is not a member of the role r, (c) if ℓ is $\mathsf{Rel_j}(\alpha_{i_1}, ..., \alpha_{i_r})$, then $u.\alpha_{i_1} \mapsto v_{i_1}, ..., u.\alpha_{i_r} \mapsto v_{i_r}$, $(v_{i_1}, ..., v_{i_r}) \in Rel_j$, and $v_{i_1} \in V_{i_1}, ..., v_{i_r} \in V_{i_r}$, and (d) if ℓ is $\overline{\mathsf{Rel_j}}(\alpha_{i_1}, ..., \alpha_{i_r})$, then $u.\alpha_{i_1} \mapsto v_{i_1}, ..., u.\alpha_{i_r} \mapsto v_{i_r}$, $(v_{i_1}, ..., v_{i_r}) \notin Rel_j$, and $v_{i_1} \in V_{i_1}, ..., v_{i_r} \in V_{i_r}$. Satisfaction for RBAC pre-conditions is defined similarly, i.e. we consider just cases (a) and (b) above.

Permission to assign users to roles is specified by the ternary relation *can_assign* containing tuples of the form (C_a, C, r) such that a user $u \in U$ is assigned to the role $r \in R$ (i.e. it is added as member of that role) by an administrator $u_a \in U$ in the RBAC policy ρ iff u_a satisfies C_a and u satisfies C. The execution of this action updates ρ to $\rho' = \langle U, An, V_1, ..., V_n, \mapsto, R, P, UA \cup \{(u, r)\}, PA, \succeq, Rel_1, ..., Rel_k\rangle$. Permission to revoke users from roles is specified by the binary relation *can_revoke* containing tuples of the form (C_a, r) such that a user $u \in U$ is revoked from role $r \in R$ (i.e. it is removed from being a member of that role) by an administrator $u_a \in U$ in the RBAC policy ρ iff u_a satisfies C (we follow [24] in omitting RBAC conditions when specifying role revocation). The execution of this action updates ρ to $\rho' = \langle U, An, V_1, ..., V_n, \mapsto, R, P, UA \setminus$

$\{(u,r)\}, PA, \succeq, Rel_1, ..., Rel_k\rangle$. The role r to which users are assigned or removed (namely, the third component of the tuples in the relation *can_assign* and the second component of the tuples in the relation *can_revoke*) is the *target* role. The user u_a satisfying the administrative pre-condition of an action is the *administrator* of that action. An *administrative RBAC policy* is a pair $\delta = \langle can\_assign, can\_revoke\rangle$. If no attribute literals are used in the administrative pre-conditions, then it is easy to see that the definitions above reduce to those in [24] for the ARBAC framework.

EXAMPLE 2. Let us consider the following two administrative actions: $(\{HR, age > 35\}, \{EM, \overline{FT}\}, PT)$ is in *can_assign* and $(\{MA, age > 50\}, FT)$ is in *can_revoke*. In the attribute RBAC policy ρ_0^a of Example 1, user C can become the administrator of the *can_assign* action since it belongs to role HR and its age is $38(> 35)$, and user A satisfies the RBAC pre-condition since it is an explicit member of EM and is not a member of role FT. Hence, the result of applying the action to ρ_0^a is the following attribute RBAC policy: $\rho_1^a = \langle U, \{age\}, \mathbb{N}, \mapsto, R, P, UA_0 \cup \{(A, PT)\}, PA, \succeq, >\rangle$. The action *can_revoke* cannot be applied neither in ρ_0^a nor in ρ_1^a since user B is the only one who is a member of role MA but is younger than 50. □

An administrative RBAC policy δ induces in the obvious way a transition relation $\rightarrow_\delta$ between pairs of RBAC policies. We denote the reflexive-transitive closure of $\rightarrow_\delta$ with $\rightarrow_\delta^*$. If ρ and ρ' are two distinct attribute RBAC policies such that $\rho \rightarrow_\delta^* \rho'$, then there exists $\rho_1, ..., \rho_n$ $(n \geq 1)$ attribute RBAC policies such that $\rho_1 = \rho$, $\rho_i \rightarrow_\delta \rho_{i+1}$ for $i = 1, ..., n-1$, and $\rho_n = \rho'$. The *set of administrators involved in the sequence of transitions* $\rho \rightarrow_\delta^* \rho'$ is formed by the $n-1$ administrators of each action $\rho_i \rightarrow_\delta \rho_{i+1}$, for $i = 1, ..., n-1$; when $\rho = \rho'$, the set of administrators is empty. The *user-role reachability problem* is stated as follows (it is a generalization to attribute RBAC policies of the definition in [24]): let $\rho_0 = \langle U, An, V_1, ..., V_n, \mapsto, R, P, UA_0, PA, \succeq, Rel_1, ..., Rel_k\rangle$ be an attribute RBAC policy, δ an administrative RBAC policy, $U_0 \subseteq U$, $u_g \in U$ be the *goal user*, and $R_g \subseteq R$ be a finite set of *goal roles*, does there exist a RBAC policy $\rho_1 = \langle U, An, V_1, ..., V_n, \mapsto, R, P, UA_1, PA, \succeq, Rel_1, ..., Rel_k\rangle$ such that $\rho_0 \rightarrow_\delta^* \rho_1$, u_g is a member of each role in R_g, and all the administrators involved in $\rho_0 \rightarrow_\delta^* \rho_1$ are in U_0 (intuitively, U_0 is the set of untrusted users). The cardinality of the set R_g of goal roles is called the *size of the goal*.

3. SYMBOLIC MODEL CHECKING

We review a symbolic model checking procedure for solving the reachability problem for a system S and a set G of goal states, i.e. the problem of checking if there exists a sequence of transitions leading S from one of its initial states I to a goal state in G. Instances of this procedure have been used successfully for finite state model checking based on binary decision diagrams (see, e.g., [7]) and for infinite state model checking by using fragments of first-order logic (see, e.g., [11]).

Formally, we use many-sorted first-order logic with equality (see, e.g., [9]). We model the system S abstractly using logical formulas. Let V be a set of state variables corresponding to individual variables in the system S. A *state formula* is a formula $\varphi(V)$ where only the symbols from V may occur free. In addition to the symbols in V, a state formula may also contain interpreted symbols like the 'more senior than' relation $\succeq$. The interpretations of such symbols are identified by associating to the system S a *theory*, i.e.—according to [5]—a pair $\mathcal{T} = (\Sigma, \mathcal{C})$ where Σ is a set of symbols with their arity, called *signature*, and $\mathcal{C}$ is a class of first-order structures, called the *models* of $\mathcal{T}$. Usually, $\mathcal{C}$ is defined by a set Ax of sentences (i.e. formulae with no free variables) as follows: $Mod(Ax) = \{M | M \models \varphi \text{ for every } \varphi \in Ax\}$, where $\models$ is the standard satisfaction relation in first-order logic [9]. When $\mathcal{T} = (\Sigma, Mod(Ax))$, we say that Ax is the set of *axioms* of $\mathcal{T}$. For example, the 'more senior than' relation $\succeq$ is formalized by the theory $\mathcal{T}_{po} = (\Sigma_{po}, Mod(Ax_{po}))$ of partial orders, where Σ_{po} contains the sort symbol *Role* and the binary predicate symbol $\succeq$: *Role* × *Role* (written infix), and Ax_{po} is the finite set of sentences $\forall r.(r \succeq r)$, $\forall r_1, r_2, r_3.((r_1 \succeq r_2 \wedge r_2 \succeq r_3) \Rightarrow r_1 \succeq r_3)$, and $\forall r_1, r_2.((r_1 \succeq r_2 \wedge r_2 \succeq r_1) \Rightarrow r_1 = r_2)$. A *transition* is a formula $T(V, V')$ where V' denotes the values of the state variables after the execution of the transition while V those immediately before. A *transition system* is a tuple $(V, I(V), T(V, V'))$ where V is the set of state variables, $I(V)$ is a state formula denoting the set of initial states, and $T(V, V')$ denote the transitions. In the following, we assume that a transition system is associated with a theory describing the interpretations of the symbols which occur in I and T and are not in $V \cup V'$.

EXAMPLE 3. We now sketch how it is possible to formalize the administrative attribute RBAC policies introduced in Section 2 as transition systems. Since the administrative actions can modify only the relation UA, we assume $V = \{ua\}$. It is possible to specify the partial order of Example 1, by adding to the theory $\mathcal{T}_{po}$ introduced above, the set $K := \{HR, MA, FT, PT, EM\}$ of constants of sort *Role* and the following sentences to its axioms: $\bigwedge_{c \in K, d \in K \setminus \{c\}} c \neq d$, $\forall r. \bigvee_{c \in K} r = c$, $MA \succeq FT$, $FT \succeq EM$, $PT \succeq EM$. Intuitively, the first two sentences constrains the interpretations of *Role* to contain exactly 6 elements while the remaining ones restrict the interpretation of $\succeq$ to be the partial order depicted in Figure 1. We are now ready to write the state formula describing the relation UA_0 of Example 1:

$$\forall u, r.(ua(u,r) \quad \Leftrightarrow \quad \left(\begin{array}{ll} (u = A \wedge r = EM) & \vee \\ (u = B \wedge r = MA) & \vee \\ (u = C \wedge r = HR) & \end{array} \right)), \quad (1)$$

which can be seen as an instance of the formula $I(V)$ describing the set of initial states. In order to fully describe the administrative attribute RBAC policies of Examples 1 and 2, we need to furtherly extend the theory associated to our transition system. So, we introduce the sort *Nat* with the numerals $0, 1, 2, ...$ as constants of this sort, the predicate symbol $>$: *Nat* × *Nat* (written infix), and the predicate symbol age : *User* × *Nat*.[1] The set of axioms are extended with the sentences to formalize the fact that $>$ is a (strict) total order, that the numerals are ordered in the obvious way (e.g., $38 > 25$ and $45 > 38$), and the sentence

$$\forall u, n.(age(u,n) \quad \Leftrightarrow \quad \left(\begin{array}{ll} (u = A \wedge n = 25) & \vee \\ (u = B \wedge n = 45) & \vee \\ (u = C \wedge n = 38) & \end{array} \right)) \quad (2)$$

[1]Strictly speaking, we should introduce a ternary relation to represent $\mapsto$. Since the set of attribute names is assumed to be finite, we can equivalently introduce a binary predicate symbol for each attribute name as we do here.

to describe the values of the attribute *age* given in Example 1. At this point, we have all the ingredients to formalize the *can_assign* action of Example 2:

$\exists u_a, r, n.(ua(u_a, r) \wedge r \succeq HR \wedge age(u_a, n) \wedge n > 35) \wedge$
$\exists u_1, r_1.(ua(u_1, r_1) \wedge r_1 \succeq EM \wedge \neg\exists r_2.(ua(u_1, r_2) \wedge r_2 \succeq FT) \wedge$
$\forall x, y.(ua'(x, y) \Leftrightarrow ((x = u_1 \wedge y = PT) \vee ua(x, y))))$

The first line of the formula corresponds to the administrative pre-condition already considered in Example 4, the second line is the RBAC pre-condition saying that there exists a user u_1 who is a member of role EM and not a member of role FT, and the last line specifies the effect of the action on the user assignment relation, requiring that the pair consisting of the user u_1 and the role PT should be added to it. The *can_revoke* action can be formalized similarly. It is sufficient to take the disjunction of the resulting formulae to build the transition formula $T(ua, ua')$. □

We now describe a procedure to solve the reachability problem based on the symbolic representation introduced above (it will be used to solve the user-role reachability problem, defined at the end of Section 2). Preliminarily, we recall the notion of satisfiability and validity modulo theories (see, e.g., [5]). A formula φ is *satisfiable modulo the theory* $\mathcal{T} = (\Sigma, \mathcal{C})$ iff there exists a structure $M \in \mathcal{C}$ such that $M \models \exists x_1, ..., x_n.\varphi$, where $x_1, ..., x_n$ are the variables that occur free in φ. A formula φ is *valid modulo the theory* $\mathcal{T} = (\Sigma, \mathcal{C})$ iff $\neg\varphi$ is unsatisfiable modulo $\mathcal{T}$.

Example 4. To illustrate the importance of satisfiability modulo theories, we show how to formally check that user C can become the administrator of the *can_assign* action in Example 2. Recall the first line of the formula in Example 3, namely $C_a(u_a, n) := ua(u_a, HR) \wedge age(u_a, n) \wedge n > 35$ corresponding to the first component of the *can_assign* action formalizing the *can_assign* action $(\{HR, age > 35\}, \{EM, \overline{FT}\}, PT)$ in Example 2. It is possible to see that $C_a(u_a, n)$ is satisfiable modulo the theory of Example 3, and that C is the witness of the variable u_a in $C_a(u_a, n)$. □

The procedure to solve the reachability problem is based on *backward reachability*, a method which iteratively computes the symbolic representations of the set $R(V)$ of states from which it is possible to reach a goal state in $G(V)$, by applying—finitely many times—the transition $T(V, V')$. Formally, $R(V)$ is defined by the following (possibly infinite) sequence of formulae: $R_0 := G(V)$ and

$R_{i+1}(V) := R_i(V) \vee \exists V'.(R_i(V') \wedge T(V, V'))$ for $i \geq 0$.

In order to stop computing formulae in the sequence, there are two criteria. First, we can check whether $R_n(V) \wedge I(V)$ is satisfiable modulo the theory $\mathcal{T}$ associated to the transition system $(V, I(V), T(V, V'))$: in this case, there exists a finite sequence of the transitions in T that leads the system from an initial state in I to a state in G. Second, we can check whether $R_{n+1}(V) \Rightarrow R_n(V)$ is valid modulo $\mathcal{T}$: the sequence $R_0, R_1, ..., R_n, R_{n+1}$ reaches a *fix-point* at $n + 1$. Following [12], to make the procedure effective, there should exist two classes $\mathcal{E}$ and $\mathcal{U}$ of formulae of the theory $\mathcal{T}$ such that (R1) G is in $\mathcal{E}$ and it is possible to effectively find a formula in $\mathcal{E}$ which is logically equivalent to $\exists V'.(R_i(V') \wedge T(V, V'))$, called the *pre-image* of R_i, (R2) I is in $\mathcal{U}$ and it is possible to effectively check the satisfiability of the conjunction of a formula in $\mathcal{E}$ and a formula in $\mathcal{U}$ modulo $\mathcal{T}$, i.e. it is decidable to check the satisfiability of $R_i \wedge I$ modulo $\mathcal{T}$, (R3) it is possible to effectively check the validity modulo $\mathcal{T}$ of any pair of formulae in $\mathcal{E}$, i.e. it is decidable to check the validity of $R_{i+1} \Rightarrow R_i$ modulo $\mathcal{T}$. If these requirements are met, we say that the theory $\mathcal{T}$ with the classes of formulae $\mathcal{E}$ and $\mathcal{U}$ is *adequate for symbolic model checking*.

Although the backward reachability procedure can be made effective, its termination is not guaranteed:it may be the case that, $R_i(V) \wedge I(V)$ is unsatisfiable and $R_{i+1}(V) \Rightarrow R_i(V)$ is not valid, for every $i \geq 0$. This is related to the definability problem in first-order logic (see, e.g., [9]): the formulae in the sequence $R_0, R_1, ...$ may not express the set of reachable states of the system but only some approximation. In other words, the backward reachability procedure is, in general, a semi-decision procedure for the reachability problem. We will see that for RBAC policies, termination is guaranteed.

	Σ_{RBAC}
U, R, P	$User, Role, Permission$
$V_1, ..., V_n$	$Val_1, ..., Val_n$
$\alpha \in An$	$\alpha : User \times Val_i$
PA	$pa : Role \times Permission$
$\succeq$	$\succeq$: $Role \times Role$
$Rel_j \subseteq V_{j_1} \times \cdots \times V_{j_r}$	$\mathsf{Rel}_j : Val_{j_1} \times \cdots \times Val_{j_r}$

Table 1: Formalization of attribute RBAC: syntax

4. SECURITY ANALYSIS: THEORY

We now explain how to automatically translate attribute RBAC policies and the administrative actions *can_assign* and *can_revoke* to an adequate theory with two classes of first-order formulae. We recall the definition of the *Bernays-Shönfinkel-Ramsey* (BSR) class (see, e.g., [19]) which contains formulae of the form $\exists x_1, ..., x_n.\forall y_1, ..., y_m.\varphi$, where $\{x_1, ..., x_n\}$ and $\{y_1, ..., y_m\}$ are disjoint sets of variables (for $n \geq 0$ and $m \geq 0$) and φ is a quantifier-free formula (i.e. a formula obtained by arbitrary Boolean combinations of first-order atoms) where at most the variables in $\{x_1, ..., x_n, y_1, ..., y_m\}$ may occur and no function symbols of arity greater than or equal to 1 is allowed. When $n = 0$ ($m = 0$), the existential (universal, resp.) prefix is dropped and the resulting formulae are called *universal* (*existential*, resp.). Both satisfiability and validity of BSR formulae is decidable (see, e.g., [19]). We will write $BSR(\Sigma)$ for the set of BSR formulae built over the signature Σ and we will tacitly assume that Σ contains only constant and predicate symbols but no function symbols. If $\mathcal{T} = (\Sigma, Mod(Ax))$ and Ax are $BSR(\Sigma)$, then the satisfiability of a $BSR(\Sigma)$ formula ψ modulo $\mathcal{T}$ is decidable since we can transform it to the problem of checking the satisfiability of the conjunction of the formulae in Ax with ψ, which is a formula in BSR and hence its satisfiability is decidable.

4.1 A theory for attribute RBAC policies

Let $\rho = \langle U, An, V_1, ..., V_n, \mapsto, R, P, UA, PA, \succeq, Rel_1, ..., Rel_k \rangle$ be a given attribute RBAC policy. We define the signature Σ_{RBAC} to contain the symbols identified in the second column of Table 1; in the corresponding row, the first column shows the associated elements of the attribute RBAC policy ρ. Notice that we do not introduce a ternary predicate sym-

bol for $\mapsto$ and instead use binary predicate symbols corresponding to each attribute in An (see also Example 3). Furthermore, we assume that Σ_{RBAC} also contains countably many constants of sort *User*, *Role*, *Permission*, and Val_i to represent the elements in U, R, P, and V_i (for $i = 1, ..., n$), respectively. (Σ_{RBAC} contains no function symbols.)

Let Ax_{RBAC} be a set of axioms containing (a) a set Ax_{dom} of (possibly countably many) $BSR(\Sigma_{RBAC})$ formulae describing the algebraic structure of U, R, P, and V_i ($i \in \{1, ..., n\}$), (b) the axioms of $\mathcal{T}_{po}$ in Section 3 with a (finite) set of ground atoms corresponding to the hierarchy of roles in ρ (see, e.g., Example 3) formalizing the 'more senior than' relation, and (c) a (finite) set Ax_{rel} of $BSR(\Sigma_{RBAC})$ formulae defining the relations $Rel_1, ..., Rel_k$ of ρ, i.e. for each Rel_j, we have that $(e_1, ..., e_{n_j}) \in Rel_j$ iff $\mathsf{Rel}_j(\tilde{e}_1, ..., \tilde{e}_{n_j})$ is satisfiable modulo the theory $\mathcal{T}_{RBAC} := (\Sigma_{RBAC}, Mod(Ax_{RBAC}))$, where $\tilde{e}_1, ..., \tilde{e}_{n_j}$ are the constants representing the elements $e_1, ..., e_{n_j}$ (e.g., numerals representing natural numbers as in Example 3) and $n_j \geq 1$ is the arity of the relation Rel_j. In many cases, the set Ax_{dom} (point (a) above) is either empty and in this case the sets U, R, P, and V_i are infinite or it is formalized by the following sentences: $\bigwedge_{1 \leq i < j \leq n} c_i \neq c_j$ and $\forall x.(x = c_1 \vee \cdots \vee x = c_n)$, where x (c_j) is a variable (constant, resp.) of sort $S \in \{User, Role, Permission, Val_1, ..., Val_n\}$ (as illustrated in Example 3 for roles).

4.2 Adequacy

We show that the theory $\mathcal{T}_{RBAC}$ with its existential and universal formulae are adequate for symbolic model checking by showing that requirements (R1), (R2), and (R3) of Section 3 are all satisfied. We assume that our transition systems have just one state variable $ua : User \times Role$.

Requirement (R1). The goal of a user-role reachability problem (namely, "there should exist a user who is member of each role in the set $R_g := \{\mathsf{r}_1, ..., \mathsf{r}_k\}$ of goal roles") can be represented by the following existential formula in $BSR(\Sigma_{RBAC})$:

$$\exists u, r_1, ..., r_k. \left(\begin{array}{l} ua(u, r_1) \wedge \cdots \wedge ua(u, r_n) \quad \wedge \\ r_1 \succeq \tilde{\mathsf{r}}_1 \wedge \cdots \wedge r_k \succeq \tilde{\mathsf{r}}_k \end{array} \right) \tag{3}$$

where $\tilde{\mathsf{r}}_j$ is the constant representing the role $\tilde{\mathsf{r}}_j$ for $j = 1, ...k$, and k is the size of the goal. The closure under pre-image computation of the existential class of formulae in $BSR(\Sigma_{RBAC})$ will be considered below when we explain how to translate the administrative actions *can_assign* and *can_revoke* of Section 2 to transition formulae; for the moment, we assume this to be possible.

Requirement (R2). First, the initial user assignment relation can be represented by suitable universal formulae since it can be seen as a database whose content can be specified by a suitable $BSR(\Sigma_{RBAC})$ as it is well-known in the logic approach to databases (see, e.g., [10]) and is illustrated by formula (1) in Example 3. Second, it is always possible to transform a conjunction of an existential with a universal formula to a $BSR(\Sigma_{RBAC})$ formula. Thus, we are entitled to conclude the decidability of the satisfiability (modulo $\mathcal{T}_{RBAC}$) of conjunction of existential and universal formulae.

Requirement (R3). It is possible to reduce the validity (modulo $\mathcal{T}_{RBAC}$) of implications of the form $R_{i+1} \Rightarrow R_i$ to the satisfiability (again, modulo $\mathcal{T}_{RBAC}$) of $R_{i+1} \wedge \neg R_i$, i.e. a conjunction of an existential R_{i+1} and a universal formula $\neg R_i$ (being the negation of the existential formula R_i), whose decidability has been shown for requirement (R2) above.

To conclude the adequacy of $\mathcal{T}_{RBAC}$ with its existential and universal classes, we must discharge the assumption concerning requirement (R2), namely that it is possible to compute an existential formula which is logically equivalent to $\exists ua'.(R(ua') \wedge T(ua, ua'))$. To do this, we show how to symbolically represent *can_assign* and *can_revoke* actions. Preliminarily, we introduce the translation of Figure 2 (left) mapping role and attribute literals (recall their definition in Section 2) to first-order formulae; the mapping is extended to a finite set C of literals in the obvious way: $[[C]]_u := \bigwedge_{\ell \in C} [\ell]_u$. The formula representing the *can_assign* action (C_a, C, r) is (4) and that for the *can_revoke* action (C_a, r) is (5), see right of Figure 2. With m_{U_0} we denote the formula of $BSR(\Sigma_{RBAC})$ defining the set U_0 of (administrative) users specified in a user-role reachability problem (recall the definition at the end of Section 2). For example, if U_0 is a finite set of users, then $\forall u.(m_{U_0}(u) \Leftrightarrow \bigvee_{c \in U_0} u = c)$; if $U_0 = U$, then m_{U_0} is assumed to be logically equivalent to true and is omitted from (4) and (5), as it is the case in the examples of the paper, for the sake of simplicity.

Example 5. Let us consider again Example 2. The translation of the *can_assign* action $(\{HR,\ age > 35, \{EM, \overline{FT}\}, PT)$ is the following (after some simple logical manipulations):

$$\exists u_a, u, v, r_a, r_1. \left(\begin{array}{l} ua(u_a, r_a) \wedge r_a \succeq HR \wedge age(u_a, v) \wedge v > 35 \wedge \\ ua(u, r_1) \wedge r_1 \succeq EM \wedge \forall r_2.(r_2 \succeq FT \Rightarrow \neg ua(u, r_2)) \wedge \\ \forall x, y.(ua'(x, y) \Leftrightarrow Upd^{x,y}_{+}(ua, u, PT)) \end{array} \right),$$

which is logically equivalent to the formula in Example 3. Translating the *can_revoke* action $(\{MA,\ age > 50\}, FT)$ yields

$$\exists u_a, u, v, r_a. \left(\begin{array}{l} ua(u_a, r_a) \wedge r_a \succeq MA \wedge age(u_a, v) \wedge v > 50 \wedge \\ \forall x, y.(ua'(x, y) \Leftrightarrow Upd^{x,y}_{-}(ua, u, FT)) \end{array} \right). \quad \square$$

If we allow negative role and attribute literals (i.e. those giving rise to a universal quantifier in the left part of Figure 2), then it is certainly not possible to satisfy requirement (R2) because universal quantification is necessary to express the fact that a user should not be an implicit member of a certain role as the following example illustrates.

Example 6. Let us compute $\exists ua'.(R(ua') \wedge T(ua, ua'))$ where $R(ua) := \exists u_3, r_3.(ua(u_3, r_3) \wedge r_3 \succeq PT)$ and T is the translation of the *can_assign* action in Example 5 as computed in Example 5. After routine logical manipulations, we derive the formula:

$$\begin{array}{l} \exists ua'.(\forall x, y.(ua'(x, y) \Leftrightarrow Upd^{x,y}_{+}(ua, u, PT))) \wedge \\ \exists u_a, u, v, r_a, r_1, u_2, r_2. \left(\begin{array}{l} Upd^{u_3,r_3}_{+}(ua, u, PT) \wedge r_2 \succeq PT \wedge \\ ua(u_a, r_a) \wedge r_a \succeq HR \wedge age(u_a, v) \wedge v > 35 \wedge \\ ua(u, r_1) \wedge r_1 \succeq EM \wedge \forall r_2.(r_2 \succeq FT \Rightarrow \neg ua(u, r_2)) \end{array} \right) \end{array}$$

whose first conjunct can be dropped being always valid (a tautology is obtained by substituting $Upd^{x,y}_{+}(ua, u, PT)$ to ua') and we are left with a formula in BSR ($Upd^{u_3,r_3}_{+}(ua, u, PT)$ abbreviates a quantifier-free formula) with a non-empty universal quantifier prefix. $\square$

To overcome this problem, we propose the following alternative mapping for negative role literals:

$$[\overline{r}]_u \quad := \quad \exists r'.(\neg ua(u, r') \wedge r' = \tilde{r}),$$

$$[r]_u := \exists r'.(ua(u,r') \wedge \tilde{r} \succeq r') \quad [\overline{r}]_u := \forall r'.(\tilde{r} \succeq r' \Rightarrow \neg ua(u,r'))$$

$$[Rel_j(\alpha_{i_1},...,\alpha_{i_r})]_u := \exists v_1,...,v_r. \left(\begin{array}{l} \alpha(u,v_1) \wedge \cdots \wedge \alpha(u,v_n) \wedge \\ \mathsf{Rel}_j(v_1,...,v_r) \end{array} \right)$$

$$[\overline{Rel}_j(\alpha_{i_1},...,\alpha_{i_r})]_u := \forall v_1,...,v_r. \left(\begin{array}{l} (\alpha(u,v_1) \wedge \cdots \wedge \alpha(u,v_n)) \\ \Rightarrow \neg \mathsf{Rel}_j(v_1,...,v_r) \end{array} \right)$$

$$\exists u_a, u. \left(\begin{array}{l} [[C]]_{u_a} \wedge [[C]]_u \wedge m_{U_0}(u_a) \wedge \\ \forall x, y.(ua'(x,y) \Leftrightarrow Upd_+^{x,y}(ua,u,\tilde{r})) \end{array} \right) \quad (4)$$

$$\exists u_a, u. \left(\begin{array}{l} [[C]]_{u_a} \wedge m_{U_0}(u_a) \wedge \\ \forall x, y.(ua'(x,y) \Leftrightarrow Upd_-^{x,y}(ua,u,\tilde{r})) \end{array} \right) \quad (5)$$

Legenda: $\tilde{x}$ denotes the constant representing the element x, and $Upd_+^{x,y}(ua,u,\tilde{r})$ and $Upd_-^{x,y}(ua,u,\tilde{r})$ stand for $((x = u \wedge y = \tilde{r}) \vee ua(x,y))$ and $(\neg(x = u \wedge y = \tilde{r}) \wedge ua(x,y))$, respectively.

Figure 2: Symbolic representation of administrative actions

which amounts to ignore the role hierarchy when considering the negation or, equivalently, to require that the user u is not an explicit member of role r (although u can be an implicit member of the role). We use *explicit negation* (*implicit negation*, resp.) when translating negative role literals ignoring (taking into account, resp.) the role hierarchy. A similar observation holds also for negative attribute literals which introduce universal quantifiers over the values of the attributes (see left of Figure 2). However, if the axioms in Ax_{rel} allows us to effectively find a BSR formula which is logically equivalent to the negation of each atomic formula of the form $\neg\mathsf{Rel}_j(v_1,...,v_{n_j})$ or, equivalently, the complement Rel_j is definable in the *attribute theory*, i.e. the subtheory of $\mathcal{T}_{RBAC}$ whose axioms are in Ax_{dom}, then we can only use positive attribute literals and no universal quantifiers will be generated during translations. In this case, we say that the attribute theory is *closed under negation*.

EXAMPLE 7. Let us consider the linear order $>$ over elements of sort *Nat* in Example 3. It is possible to obtain closure under negation by extending the signature with the predicate symbol $\leq$: $Nat \times Nat$ denoting a weak version of the linear order $>$ (which can be axiomatized by universal sentences). In fact, it is easy to show that $\neg x > y$ is equivalent to $x \leq y$. Hence, we can replace each negative attribute literal with a positive one. □

We are now ready to prove the following result.

FACT 1. Let $\mathcal{T}_{RBAC}$ be the theory associated to a transition system (ua, I, T) obtained by the modified version of the translation in Figure 2 where explicit negation is used. Furthermore, let the attribute theory of $\mathcal{T}_{RBAC}$ be closed under negation. If R is an existential formula, then $\exists ua'.(R(ua') \wedge T(ua,ua'))$ is logically equivalent to an effectively computable existential formula.

The proof of this fact consists of simple logical manipulations (similar to those in Example 6) and is omitted for lack of space. This implies that (under reasonable) assumptions also the second part of requirement (R2) can be satisfied.

4.3 Automated security analysis

We now explain how to use the symbolic reachability procedure of Section 3 to solve instances of the user-role reachability problem defined at the end of Section 2.

Let ρ_0 be an attribute RBAC policy, δ an administrative RBAC policy, $U_0 \subseteq U$, $u_g \in U$ be a goal user, and $R_g \subseteq R$ be a finite set of goal roles. We want to answer the question: does there exist a RBAC policy ρ_1 such that $\rho_0 \rightarrow_\delta^* \rho_1$, u_g is a member of each role in R_g, and all the administrators involved in $\rho_0 \rightarrow_\delta^* \rho_1$ are in U_0? First, we create the theory $\mathcal{T}_{RBAC}$ as explained in Section 4.1. Second, we build the formulae (i) $I(ua)$ representing ρ_0, (ii) m_{U_0} defining the set U_0, (iii) G for "there exists a user u_g who is a member of each role in R_g," and (iv) the disjunction $T(ua,ua')$ of the formulae representing the actions of the administrative RBAC policy in δ, as described in Section 4.2. Third, we run the backward reachability procedure of Section 3 on the transition system $(ua, I(ua), T(ua,ua'))$ and goal $G(ua)$. If it terminates at the n-th iteration because $R_n \wedge I$ is satisfiable (modulo $\mathcal{T}_{RBAC}$), then we conclude that the instance of the user-role reachability problems is solvable. Otherwise, i.e. the procedure terminates at the n-th iteration because $R_n \Rightarrow R_{n-1}$ is valid and $R_n \wedge I$ is unsatisfiable (both modulo $\mathcal{T}_{RBAC}$), we conclude that the instance of the user-role reachability problems is not solvable.

By using the results in [12], it is possible to show that the backward reachability procedure always terminates under the same hypotheses for which we have shown the adequacy of $\mathcal{T}_{RBAC}$ with its existential and universal fragments. As a consequence, we have a push-button technique to solve security problems for administrative RBAC policies.

To evaluate the practical value of the proposed technique, an experimental evaluation is mandatory because, when terminating, the procedure of Section 3 may have non-elementary complexity (see, e.g., [11]). This is done in the next section, after a discussion of how to tune the symbolic reachability procedure for the security analysis of attribute RBAC policies.

5. SECURITY ANALYSIS: PRACTICE

A client-server architecture is the most obvious choice to implement the backward reachability procedure of Section 3. The client computes pre-images and generates the proof obligations required to test for fix-point or the non-empty intersection with the initial set of states. The server performs the checks for satisfiability modulo $\mathcal{T}_{RBAC}$ and can be implemented by using state-of-the-art automated deduction systems such as Automated Theorem Provers (ATPs) or Satisfiability Modulo Theories (SMT) solvers. Although these tools are quite powerful, preliminary experiments have shown that the formulae to be checked for satisfiability generated by the client quickly become very large and are not easily solved by available state-of-the-art tools. A closer look at the formulae reveals that they can be greatly simplified.

5.1 Detection of redundancies

We study how to simplify the existential formulae which are logically equivalent to $\exists ua'.(R(ua') \wedge T(ua,ua'))$. Recall that T is of the form $\bigvee t_i(ua,ua')$ where each t_i is of the form (4) or (5), corresponding to the translation of *can_assign* and *can_revoke* actions (see Figure 2). Since $\exists ua'.(R(ua') \wedge \bigvee t_i(ua,ua'))$ is logically equivalent to

$\bigvee_i \exists ua'.(R(ua') \wedge \bigvee t_i(ua, ua'))$, if we could show that some of its disjuncts are false, we could obtain a more compact formula. We would like to have a computationally cheap sufficient condition to drop a disjunct. To this end, let us consider the case where t_i is of the form (4), i.e.

$$\exists u_a, u.(Cond(u_a, u) \wedge \forall x, y.(ua'(x, y) \Leftrightarrow Upd^{x,y}_{+}(ua, u, \tilde{r}))),$$

where $Cond(u_a, u)$ abbreviates $[[C]]_{u_a} \wedge [[C]]_u \wedge m_{U_0}(u_a)$ (see Figure 2). Without loss of generality, let us assume that R is an existential formula of the form $\exists x_1, ..., x_m.(\ell_1 \wedge \cdots \wedge \ell_k)$ where ℓ_j is a literal for $j = 1, ..., k$; let us call this sub-class of existential formulae as $\exists^+$*-formulae*. Notice that (3) is an $\exists^+$-formula. For concreteness, let us assume that $R(ua) := \exists u_1, r_1.(ua(u_1, r_1) \wedge r_1 \succeq \tilde{r}_1)$ and compute its pre-image with respect to the *can_assign* action considered above:

$$\exists ua'. \left(\begin{array}{l} \exists u_1, r_1.(ua'(u_1, r_1) \wedge r_1 \succeq \tilde{r}_1) \wedge \\ \exists u_a, u. \left(\begin{array}{l} Cond(u_a, u) \wedge \\ \forall x, y.(ua'(x, y) \Leftrightarrow Upd^{x,y}_{+}(ua, u, \tilde{r})) \end{array} \right) \end{array} \right),$$

which simplifies to

$$\begin{array}{l} \exists u_a, u.(Cond(u_a, u) \wedge \tilde{r} \succeq \tilde{r}_1) \vee \\ \exists u_a, u, u_1, r_1.(Cond(u_a, u) \wedge r_1 \succeq \tilde{r}_1 \wedge ua(u_1, r_1)), \end{array}$$

by recalling that $Upd^{u_1,r_1}_{+}(ua, u, \tilde{r}_1)$ abbreviates $(u_1 = u \wedge r_1 = \tilde{r}_1) \vee ua(u_1, r_1)$. If it is not the case that $\tilde{r} \succeq \tilde{r}_1$ holds, then we can delete the first disjunct above. Furthermore, the second disjunct can be rewritten as $R(ua) \wedge \exists u_a, u.Cond(u_a, u)$ and this makes the fix-point vacuously true since the implication $(R(ua) \wedge \exists u_a, u.Cond(u_a, u)) \Rightarrow R(ua)$ is a tautology.

PROPERTY 1. Let $R(ua)$ be an $\exists^+$-formula of the form (3) and $t_i(ua, ua')$ is the form (4), whose update is $Upd^{\cdot,\cdot}_{+}(ua, u, \tilde{r})$. Then, if $\tilde{r} \succeq \tilde{r}_1 \wedge \cdots \wedge \tilde{r} \succeq \tilde{r}_k$ is unsatisfiable modulo $\mathcal{T}_{RBAC}$, then $\exists ua'.(R(ua') \wedge t(ua.ua')) \Rightarrow R(ua)$ is valid modulo $\mathcal{T}_{RBAC}$.

This tells us that we can simplify $\bigvee_i \exists ua'.(R(ua') \wedge t_i(ua, ua'))$ by dropping those disjunct where the update of t_i (obtained by translating a *can_assign*) does not contain a role which is more senior of (at least) one role in R. Checking the unsatisfiability of $\tilde{r} \succeq \tilde{r}_1 \wedge \cdots \wedge \tilde{r} \succeq \tilde{r}_k$ modulo $\mathcal{T}_{RBAC}$ can be done easily in practically relevant situations (e.g., when the number of roles is bounded). Similar (dual) observations can be made when we consider *can_revoke* actions and negative occurrences of literals containing ua are allowed in R.

5.2 Efficient elimination of redundancies

The application of Property 1 requires R (in $\exists ua'.(R(ua') \wedge T(ua, ua'))$) to be an $\exists^+$-formula. Although this is without loss of generality, it may be computationally very expensive since transforming formulae to disjunctive normal form can lead to an exponential blow-up. Fortunately, we can avoid this problem by making the following two observations. First, formula (3) representing the goal is already an $\exists^+$-formulae. Second, we can refine the symbolic backward reachability procedure by using an appropriate data structure that stores the $\exists^+$-formulae required to apply Property 1 so as to avoid the frequent conversion into disjunctive normal form. (By abusing notation, we regard the formula T as a finite set of formulae of the forms (4) and (5).)

DEFINITION 1. A *reachability tree* associated to a transition system (ua, I, T) and an existential formula $G(ua)$ of the form (3) is a labelled tree such that (I) the root node is labelled by G, and (II) for each node ν in the tree labelled by an $\exists^+$-formula R_ν, there exist children $\nu_1, ..., \nu_k$ of ν labelled by $\exists^+$-formulae $R_{\nu_1}, ..., R_{\nu_k}$, respectively, such that (II.a) R_{ν_j} is satisfiable (modulo T_{RBAC}), (II.b) $R_{\nu_1} \vee \cdots \vee R_{\nu_k}$ is logically equivalent (modulo T_{RBAC}) to $\exists ua'.(R(ua') \wedge t(ua, ua'))$ where t is a formula in T, and (II.c) t labels each edge from ν_j to ν, for $j = 1, ..., k$.

It is easy to show that the disjunction of the $\exists^+$-formulae labelling all the nodes in a reachability tree of depth n is equivalent to R_n as defined in Section 3. Interleaving satisfiability checking modulo $\mathcal{T}_{RBAC}$ (item (II.a) of Definition 1) and the application of Property 1 allow us to eliminate redundancies and to create only those nodes labelled by satisfiable (modulo $\mathcal{T}_{RBAC}$) $\exists^+$-formulae. Since each edge in the reachability tree is labelled by a transition in T (item (II.c) of Definition 1), it is trivial to compute the sequence of transitions leading the system from an initial state to a goal state by collecting the labels of the edges in the path from the node ν labelled by the $\exists^+$-formula R_ν such that $R_\nu \wedge I$ is satisfiable modulo $\mathcal{T}_{RBAC}$ to the root node.

While the reachability tree provides the right data structure to eliminate redundancies and to perform the check for intersection with the initial states, it seems that the fix-point checks become more difficult to implement. Fortunately, this is not the case as the following observations show. Consider a node ν in the reachability tree associated to a transition system (ua, I, T) and an $\exists^+$-formula G of the form (3). If each children ν' of ν is labelled by an $\exists^+$-formula $R_{\nu'}$ such that $R_{\nu'} \Rightarrow R_\nu$ is valid modulo $\mathcal{T}_{RBAC}$, then we say that R_ν is a *local fix-point* and stop adding children to its children. When each $\exists^+$-formula labelling a node in the fringe of a reachability tree is a local fix-point, then we say that a *global fix-point* has been reached and stop expanding the reachability tree: this is equivalent to detecting a fix-point as defined in Section 3.

5.3 A refinement of backward reachability

To design a practical version of the symbolic reachability procedure using the notion of reachability tree, we observe that such a data structure does not need to be constructed explicitly. Instead, we can visit the tree on-the-fly by simply maintaining two sets of $\exists^+$-formulae labelling the nodes of the reachability tree: those 'to be visited' (TBV) and those 'already visited' (AV), while performing the various satisfiability checks. The pseudo-code of the procedure is in Figure 3. The function SMC takes as input a transition system (ua, I, T)—where I and T are obtained as described in Section 4.2—and an $\exists^+$-formula G of the form (3). Initially, SMC initializes the set TBV to the singleton set containing G and AV to the empty set. The goal G is decorated with the empty sequence ϵ of formulae representing transitions. In general, each $\exists^+$-formula ν is decorated by the sequence σ of transitions from T (in symbols, ν^σ) that (backward) applied to φ yields a formula which is satisfiable when conjoined with G or, equivalently, σ is the sequence of transitions that leads the system from a state in ν to a state in G. (When the sequence σ is unimportant, we write ν^*.) The main loop of the procedure (lines 3–16) is entered if the set of formulae to be visited is non-empty. At each iteration, the following steps are taken. A formula from TBV is selected

```
1  function SMC(I, T, G)
2  TBV ⟵ {G^ϵ}; AV ⟵ ∅;
3  while TBV ≠ ∅ do
4    ρ^σ ⟵ select(TBV); TBV ⟵ TBV \ {ρ^σ};
5    if check(ρ, {¬ν | ν* ∈ AV}) = unsat
6    then AV ⟵ {ρ^σ} ∪ AV;
7    for each t ∈ T do
8      if compatible(t, ρ) then P ⟵ Pre(t, ρ^σ) else P ⟵ ∅;
9      for each π^{t,σ} ∈ P do
10       if subsume(π, {ν|ν* ∈ AV) then P ⟵ P \ {π^{t,σ}};
11       else if check(π, {I}) = sat
12            then return "goal reached via t, σ"
13     end
14     TBV ⟵ P ∪ TBV;
15   end
16 end
17 return "goal unreachable"
```

Figure 3: Backward reachability as the on-the-fly visit of a reachability tree

(line 4) according to the following observation: the higher the number of occurrences of literals with *ua* (for goals, this is equivalent to their size), the higher is the number of $\exists^+$-formulae whose disjunction is equivalent to the pre-image of the formula with respect to a formula in T. So, *select* picks the $\exists^+$-formula ρ with the lowest number of occurrences of *ua* among those in TBV. (For similar reasons, the set T is kept ordered by increasing number of occurrences of *ua* in the pre-conditions of the formulae representing transitions.) At line 5, we check if the selected formula ρ is a local fix-point. To this end, we need to check the validity (modulo $\mathcal{T}_{RBAC}$) of $\rho \Rightarrow \bigvee_{\nu \in AV} \nu$. By refutation, this is equivalent to checking the unsatisfiability (modulo $\mathcal{T}_{RBAC}$) of $\rho \wedge \bigvee_{\nu \in AV} \neg\nu$. This is done by the invocation of the function *check* at line 5. The function performs the satisfiability check incrementally by considering the formulae in AV in reverse chronological order because we have seen that formulae recently added to AV are more likely to contribute to unsatisfiability. If unsatisfiability is detected, we insert the formula ρ into AV as it is a local fix-point (line 6); otherwise, the transitions in T must be applied to ρ (lines 7–15). The function *compatible* (line 8) implements the test of redundancy of Property 1. When the test is successful, we consider another transition in T if any (this is done by setting P to the empty set so that the inner loop, lines 9–13, is never executed). Otherwise, we apply t to ρ^σ by invoking the function Pre (line 8) which implements the symbolic manipulations to derive a set (intended disjunctively) of logically equivalent $\exists^+$-formulae (according to Fact 1). These formulae are stored in the set P, each one decorated with the sequence of transitions t, σ (i.e. the last transition applied to π is added to the front of the sequence of transitions that allowed for the derivation of ρ from G). The content of P is filtered (lines 9–13) so as to eliminate redundant formulae: by invoking the function *subsume*, it is first checked that $\pi \in P$ is satisfiable (modulo $\mathcal{T}_{RBAC}$) and then that π does not imply a formula in AV (i.e. we check the validity modulo $\mathcal{T}_{RBAC}$ of $\rho \Rightarrow \nu$ for $\nu^* \in AV$).[2] If this is the case, then π is a local fix-point and can be deleted from P; otherwise, it is checked if $\pi \wedge I$ is satisfiable and if so, *SMC* returns by reporting that the goal G is reachable by the sequence of transitions labelling π (line 12). After P is filtered out, its content is added to TBV (line 14) and the main loop of the procedure is restarted. If there are no more formulae in TBV and the test of the conditional at line 12 has never been true, then *SMC* returns by saying that the goal G is unreachable since a global fix-point has been detected.

Using ATPs and SMT solvers. There are several ways to implement the function *check* for incremental satisfiability checking. On the one hand, SMT solvers natively support (stack-wise) incrementality but only approximate satisfiability checks for BSR by using instantiation based methods (unfortunately, the complete SMT procedure described in [19] has not yet been integrated in the available version of the Z3 SMT solver [1]). On the other hand, ATPs are refutation complete and performs very well on formulae belonging to the BSR class, e.g., SPASS [13] or the iProver [2], but do not support incremental satisfiability checks. To find the right trade-off between efficiency and completeness, we decided to use a hierarchic combination of SMT solvers and ATPs. First, we invoke the Z3 SMT solver: if Z3 returns 'unknown,' we invoke either SPASS or the iProver among the several available ATPs. Since we use the new SMT-LIB standard [5] to communicate with Z3 and the TPTP format [3] to invoke the ATPs, other SMT solvers and provers can be readily plugged in our tool as new advances in both fields become available.

5.4 Sub-goaling

As pointed out in [24], the size of the goal (see the definition at the end of Section 2) is critical for the complexity of the user-role reachability problem. Experiments with a preliminary version of our tool confirmed this result: it did not scale up even to modest values of this parameter. In order to circumvent the problem, we incorporated a *divide et impera* strategy. The idea is to generate a user-goal reachability problem for each role in R_g, say that the size of the goal is k. If the procedure *SMC* returns with "goal unreachable" on (at least) one of the k problems, we are entitled to conclude that the more general problem is also unsolvable. Otherwise, if *SMC* returned with "goal reachable via σ_j" for each $j = 1, ..., k$, then we create a new problem with the original goal but with only the transitions in $T' = \bigcup_{j=1}^{k} \sigma_j$ (by abuse of notation, we consider sequences as sets here) and we run again the procedure *SMC*. The last call to the procedure is done for checking if the transitions in T' are sufficient to solve the reachability problem. In practice, we have observed that the cardinality of the set T' obtained in this way is much smaller than that of the original set T of transitions, and this permits substantial savings in the execution time. If the last call to *SMC* is not successful (because some transitions interfere in unexpected ways), we can iterate again the process by selecting some other solutions (if any) to one or more of the k sub-problems and try to solve again the original problem.

5.5 Experiments

We briefly discuss our experiments with a prototype of the techniques described above that we call `ASASP` for Automatic

[2]Indeed, this test can be seen as a dramatically simplified version of the local fix-point check at line 5. The observation here is that we want to perform a computationally cheap test to reduce the number of formulae to be inserted in TBV.

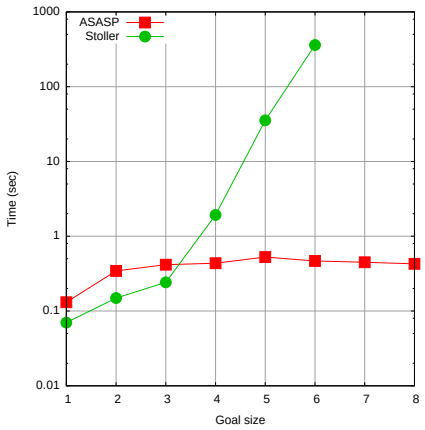


Figure 4: Timings on benchmark class three of [24]

Symbolic Analysis of Security Policies. We consider three classes of problems: (a) the synthetic benchmarks described in [24] and available on the web at [4] for the ARBAC model without hierarchy, (b) the same problems considered in (a) augmented with randomly generated role hierarchies, and (c) a new set of synthetic benchmarks—derived from (b)—for the administration of a simple instance of the attribute RBAC policies introduced in this paper. All the experiments were conducted on an Intel(R) Core(TM)2 Duo CPU T5870, 2 GHz, 3 GB RAM, running Linux Debian 2.6.32. The set of benchmarks, the executable of `ASASP`, and the experimental results are available at `http://st.fbk.eu/SilvioRanise`.

(a) ARBAC without role hierarchy. In [4], there are five classes of randomly generated benchmarks for ARBAC policies *without role hierarchy* (for a detailed description, the reader is pointed to [24, 4]). The first and second classes of benchmarks were used to evaluate a forward search algorithm described in [24] and are too easy for backward procedures which immediately realize that the goal is never reachable as the goal role is different from all the target roles of the administrative actions. The fourth and fifth classes of benchmarks are also easily solved by both `ASASP` and `Stoller`, with the former slightly slower than the latter because of the overhead of invoking SMT solvers or ATPs instead of the *ad hoc* techniques used in [24]. The most interesting class of problems is the third, which was used to evaluate the scalability of the backward reachability algorithm of [24] with respect to increasing values of the goal size, along the lines of the parametrised complexity result in [24]. Figure 4 shows the plot of the median time (logarithmic scale) to solve 32 problems of `ASASP` and `Stoller` for increasing values of the goal size (up to 8); the time out was set to $1,800$ sec. It is clear that the behavior of `Stoller` grows quickly as the function of the size of the goal. For goal size larger than 6, we were no more able to report the median value as `Stoller` solved less than 50% of the instance problems in the given time-out. Up to goal size 4, `Stoller` was able to solve 100% of the problem instances, for a value of 5 the percentage of success goes down to around 73%, for 6 goes further down to 60%, and for 7 and 8 reduces to 37% and 33%, respectively. Instead, `ASASP` shows a much better behavior, solving the problem instances for goal sizes $1, 2$, and 3 with a slight overhead with respect to `Stoller` and outperforms the latter for goal sizes bigger than 3; in particular, `ASASP` can solve 100% of the problem instances up to a goal size of 7 and 90% for a goal size of 8. Notice the "cut-off effect" for goal size larger than 5 when problem instances become over-constrained (as it is unlikely that more and more goal roles are reachable). The key to obtain this nice asymptotic behavior was the sub-goaling technique described in Section 5.4.

(b) ARBAC with role hierarchy. We extend the benchmark problems considered above with randomly generated hierarchies parametrised with respect to the shape (lattice, inverted tree, or layered [17]) and the maximal length of the transitivity chains (called the *depth* of the hierarchy). In order to enable `Stoller` to process the resulting problem instances, we pre-processed the problem by implementing the translation in [21] to compile away the hierarchy, taking into account the explicit negation assumption of Section 4.2 to make the results of the analysis compatible with those of our tool. For `ASASP`, we considered the instance problems with the hierarchy axiomatized as a partial order (recall Section 4.2) and those with the hierarchy compiled away. It turns out that only the depth of the hierarchy is significant while the performances of the tools are insensitive to the shape. Each plot in Figure 5(a) shows the median time (logarithmic scale) of `Stoller` and of `ASASP` (both with axiomatization and compilation of the role hierarchy) against ten increasing values of the depth of the role hierarchy for a fixed value of the goal size (the time-out was set to 600 sec). Similarly to the results in Figure 4, `ASASP` scales much better than `Stoller` with respect to the size of the goal: for increasing values of the depth of the role hierarchy, this behavior is amplified. For `ASASP`, using a symbolic representation of the role hierarchy is beneficial when compared with the performances on the same problem instances with the hierarchy compiled away. The reason is that the technique to compile away the hierarchy may give exponentially larger problems (as observed in [21]) that are more difficult to solve. To confirm this, we remark that while `ASASP` can solve all problem instances in the given time-out, the percentage of success of `Stoller` degrades with increasing depth: from 90% at depth 2 down to 66% at depth 30.

(c) Adding attributes. We furtherly extend the benchmark problems considered above by adding an attribute for each user, which may be seen as a quantitative measure of his skills to perform administrative actions. The attribute values are natural numbers which are compared by the usual total order relation $>$; this can be easily axiomatized by BSR formulae. To generate new benchmark sets, we have created a program taking as input a set of ARBAC problems in the format of [4] and some other parameters to randomly add to the pre-conditions of the actions suitable constraints about the value of the attribute of the administrator. For each administrative action in the benchmark problems of the third class of [4], with a probability of 0.75, we add the constraint that the value of the attribute should be $>$ than a threshold value, again randomly selected (with a uniform distribution) among 5 distinct values. For these problems, we also considered the ten hierarchies of roles that we have generated for the previous set of benchmarks. Each plot in Figure 5(b) shows the median time (logarithmic scale) of `ASASP` (both with axiomatization and compilation of the role hierarchy) against 10 increasing values of the depth of the role hierarchy for a fixed value of the goal size (the time-out was 600 sec). `Stoller` cannot handle this type of problems at all.

Surprisingly (with respect to what we have observed for the previous set of benchmarks), compiling away the role hierarchy pays off. This is probably due to the fact that the automatic modes of the provers are not "smart" enough to handle the problems generated by our tool.

6. RELATED WORK AND DISCUSSION

Logic Programming (LP) has been extensively used to provide a basis for the security analysis of access control and trust management policies (see, e.g., [18]). The main drawback of using LP is the limited support for negation in the applicability conditions of (administrative) actions. Our technique supports a form of negation (see Section 4.3) which, when combined with the technique in [21] to compile away the role hierarchy (in case, the number of roles is bounded), allows us to generalize the decidability results for security analysis problems obtained in [18].

Planning techniques have also been adapted to design automated analysis for security of access control policies [21, 24, 23]. They support negation although, for efficiency, only one negative literal is allowed in the pre-conditions of administrative actions. Besides the theoretical results, [24] presents an extensive experimental evaluation of security analysis techniques on a set of synthetic benchmarks that we found very useful for developing our tool and the (comparative) experimental evaluation in Section 5.5, which clearly demonstrates the scalability of the automated analysis technique proposed in this paper. We have contributed two new classes of benchmarks as we believe that the availability of a library of problems will help the development and comparison of new techniques for the security analysis of RBAC policies.

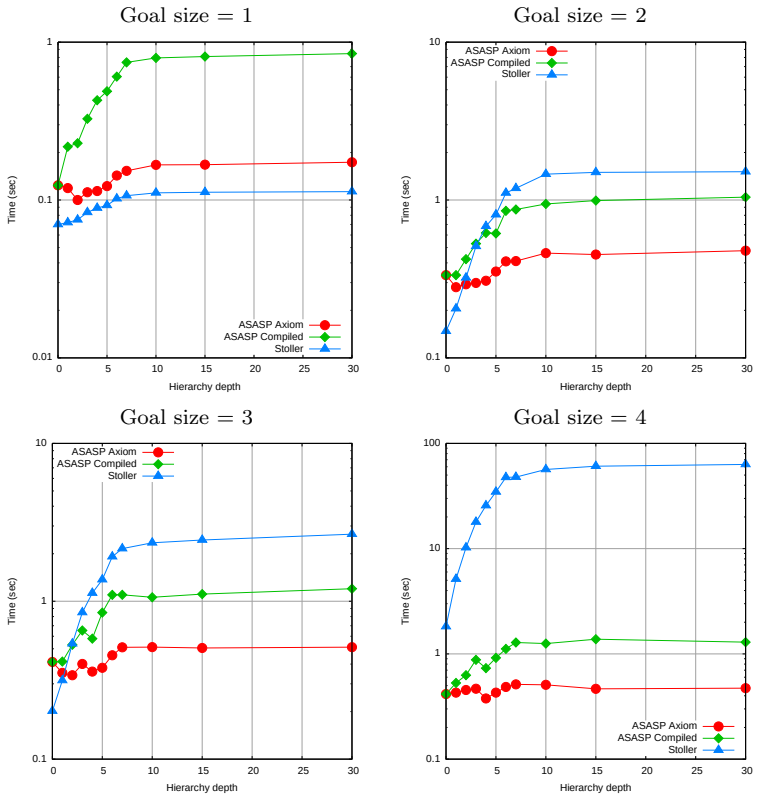
Model checking techniques have also been used by security analysts; see, e.g., [15, 22, 25, 14]. Except [15] (which uses push-down systems for verifying various properties of the SPKI/SDSI policy language), these techniques leverage symbolic finite state model checking and thus must adopt abstraction techniques to finitely approximate the infinite state space of RBAC policies. In contrast, our approach supports the symbolic specification of infinite state spaces by using fragments of first-order logic along the lines of [11]. Unfortunately, we cannot use the system [12] implementing the approah as this only supports at most two existentially quantified formulae to describe transitionsand the administrative actions considered in this paper require more than two existential variables. Another difference is that our tool uses automated theorem proving techniques for BSR formulae while [12] has *ad hoc* techniques to handle quantifiers in richer theories associated to the transition systems. A more extensive account of the techniques presented in this paper can be found in [6].

7. ACKNOWLEDGMENTS

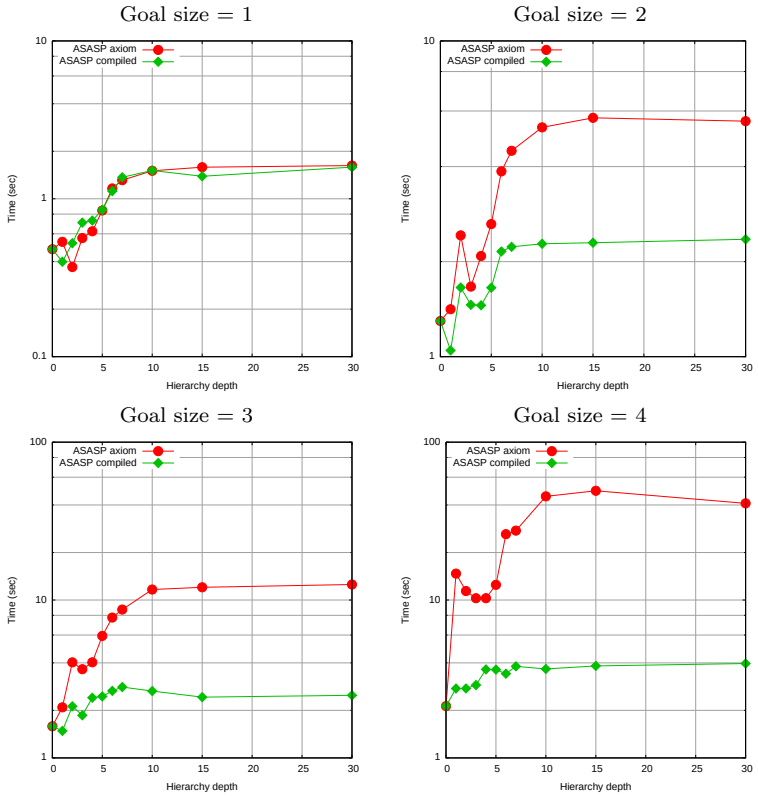
This work was partially supported by the "Automated Security Analysis of Identity and Access Management Systems (SIAM)" project funded by Provincia Autonoma di Trento in the context of the "team 2009 - Incoming" COFUND action of the European Commission (FP7) and the FP7-ICT-2007-1 Project no. 216471, "AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures."

8. REFERENCES

[1] `http://research.microsoft.com/en-us/um/redmond/projects/z3`.
[2] `http://www.cs.man.ac.uk/~korovink/iprover`.
[3] `http://www.cs.miami.edu/~tptp`.
[4] `http://www.cs.stonybrook.edu/~stoller/ccs2007`.
[5] `http://www.smt-lib.org`.
[6] A. Armando and S. Ranise. Automated Symbolic Analysis of ARBAC Policies. In *STM Workshop*, 2010.
[7] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE TCAD*, 35(8), 1986.
[8] J. Crampton. Understanding and developing role-based administrative models. In *Proc. 12th ACM CCS, pages 158–167, ACM Press, 2005.*
[9] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, Inc., 1972.
[10] H. Gallaire, J. Minker, and J.-M. Nicolas. Logic and Databases: A Deductive Approach. *Computing Surveys*, 16(2):153–185, 1984.
[11] S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Towards SMT Model-Checking of Array-based Systems. In *Proc. of IJCAR*, LNCS, 2008.
[12] S. Ghilardi and S. Ranise. MCMT: a Model Checker Modulo Theories. In *Proc. of IJCAR*, LNCS, 2010.
[13] T. Hillenbrand and C. Weidenbach. Superposition for Finite Domains. Res. Rep. RG1-002, MPI, 2007.
[14] S. Jha, N. Li, M. V. Tripunitara, Q. Wang, and H. Winsborough. Towards formal verification of role-based access control policies. *IEEE Trans. on Dependable and Secure Comp.*, 5(4):242–255, 2008.
[15] S. Jha and T. Reps. Model Checking SPKI/SDSI. *J. of Comp. Sec.*, 12:317–353, 2004.
[16] A. Kern, A. Schaad, and J. Moffett. An Administrative Concept for the Enterprise Role-Based Access Control Model. In *SACMAT*, pages 3–11, 2003.
[17] N. Li and Z. Mao. Administration in Role-Based Access Control. In *Proc. of ASIACCS*, 2007.
[18] N. Li and M. V. Tripunitara. Security analysis in role-based access control. *ACM TISSEC*, 9(4), 2006.
[19] R. Piskac, L. de Moura, and N. Bjoerner. Deciding Effectively Propositional Logic Using DPLL and Substitution Sets. *JAR*, 44(4):401–424, 2010.
[20] R. Sandhu, E. Coyne, H. Feinstein, and C. Youmann. Role-Based Access Control Models. *IEEE Computer*, 2(29):38–47, 1996.
[21] A. Sasturkar, P. Yang, S. D. Stoller, and C. Ramakrishnan. Policy analysis for administrative role based access control. In *Proc. of 19th CSF Workshop*. IEEE, July 2006.
[22] A. Schaad, V. Lotz, and K. Sohr. A model-checking approach to analysing organisational controls in a loan origination process. In *SACMAT*, pages 139–149, 2006.
[23] S. D. Stoller, P. Yang, M. I. Gofman, and C. R. Ramakrishnan. Symbolic Reachability Analysis for Parameterized Administrative Role Based Access Control. In *SACMAT'09*, pages 445–454, 2007.
[24] S. D. Stoller, P. Yang, C. Ramakrishnan, and M. I. Gofman. Efficient policy analysis for administrative role based access control. In *ACM CCS*, 2007.
[25] N. Zhang, M. Ryan, and D. P. Guelev. Evaluating access control policies through model checking. In *8th Info. Sec. Conf.*, number 3650 in LNCS, 2005.



(a) With role hierarchy



(b) With attributes

Figure 5: Performances for increasing depth of the role hierarchy and fixed goal sizes