ASASP: Automated Symbolic Analysis of Security Policies

Francesco Alberti[1], Alessandro Armando[2,3] and Silvio Ranise[3]

[1] Università della Svizzera Italiana, Lugano (Svizzera)
[2] Università degli Studi di Genova (Italia)
[3] FBK-Irst, Trento (Italia)

Abstract. We describe ASASP, a symbolic reachability procedure for the analysis of administrative access control policies. The tool represents access policies and their administrative actions as formulae of the Bernays-Shönfinkel-Ramsey class and then uses a symbolic reachability procedure to solve security analysis problems. Checks for fix-point—reduced to satisfiability problems—are mechanized by Satisfiability Modulo Theories solving and Automated Theorem Proving. ASASP has been successfully applied to the analysis of benchmark problems arising in (extensions of) the Role-Based Access Control model. Our tool shows better scalability than a state-of-the-art tool on a significant set of instances of these problems.

1 Introduction

Access control is one of the key ingredients to ensure the security of software systems where several users may perform actions on shared resources. To guarantee flexibility and scalability, access control is managed by several security officers that may delegate permissions to other users that, in turn, may delegate others. Indeed, such chains of delegation may give rise to unexpected situations where, e.g., untrusted users may get access to a sensitive resource. Thus, security analysis is critical for the design and maintenance of access control policies.

In this paper, we describe the *Automated Symbolic Analysis of Security Policies* (ASASP) tool, based on the model checking modulo theories approach of [2]. Security analysis is reduced to repeatedly checking the satisfiability of formulae in the Bernays-Schönfinkel-Ramsey class [6] by hierarchical combination of Satisfiability Modulo Theories (SMT) solving and Automated Theorem Proving (ATP). The use of an SMT solver allows us for quick and incremental—but incomplete—satisfiability checks while (refutation) complete and computationally more expensive checks are performed by the theorem prover only when needed. A *divide and conquer* heuristics for splitting complex access control queries into simpler ones is key to the scalability of ASASP on a set of benchmark problems arising in the security analysis of (extensions of) Administrative Role-Based Access Control (ARBAC) policies [1].

Theoretically, the techniques underlying ASASP are developed in the context of the model checking modulo theories approach [2]. In practice, it is difficult—or even impossible—to use the available implementation [3], called MCMT, for

the security analysis of access control policies. There are two reasons for this. First, the satisfiability problems for fix-point tests of access control policies seem easier to solve as they fall in the Bernays-Shönfinkel-Ramsey (BSR) class for which specialized tools exist. In MCMT instead, *ad hoc* instantiation techniques have been designed and implemented [3] to integrate the handling of universal quantifiers with quantifier-free reasoning in rich background theories. These are needed to model the data structures manipulated by systems [2]. Second, MCMT permits only mono-dimensional arrays [3] while access control policies routinely uses binary relations for which at least bi-dimensional arrays are needed to represent their characteristic functions. An encoding of multidimensional arrays by mono-dimensional arrays indexed by tuples is possible, although it makes it useless some heuristics of MCMT with an unacceptable degradations of its performances. Another limitation of MCMT concerns the number of existentially quantified variables in formulae representing transitions which is bounded to at most two (although this is sufficient to specify several different classes of systems as shown in [2]) while administrative actions usually require many more of such variables. An extensive discussion of the related work about the techniques underlying ASASP can be found in [1].

2 Background on administrative access control

Although ASASP can be used for the automated analysis of a more general class of administrative access control policies [1], here—for lack of space—we consider only a sub-class.

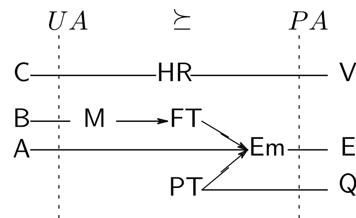
Role-Based Access Control (RBAC) regulates access through roles. Roles in a set R associate permissions in a set P to users in a set U by using the following two relations: $UA \subseteq U \times R$ and $PA \subseteq R \times P$. Roles are structured hierarchically so as to permit permission inheritance. Formally, a role hierarchy is a partial order $\succeq$ on R, where $r_1 \succeq r_2$ means that r_1 is *more senior than* r_2 for $r_1, r_2 \in R$. A user u is an *explicit* member of role r when $(u, r) \in UA$ while u is an *implicit* member of r if there exists $r' \in R$ such that $r' \succeq r$ and $(u, r') \in UA$. Given UA and PA, a user u *has permission* p if there exists a role $r \in R$ such that $(p, r) \in PA$ and u is a member of r, either explicit or implicit. A *RBAC policy* is a tuple $(U, R, P, UA, PA, \succeq)$.

Administrative RBAC (ARBAC). Usually (see, e.g., [10]), administrators may only update the relation UA while PA is assumed constant; so, a RBAC policy $(U, R, P, UA, PA, \succeq)$ will be abbreviated by UA. To be able to specify administrative actions, we need to preliminarily specify the pre-conditions of such actions. A *pre-condition* is a finite set of expressions of the forms r or $\overline{r}$ (for $r \in R$), called *role literals*. In a RBAC policy UA, a user $u \in U$ *satisfies* a pre-condition C if, for each $\ell \in C$, u is a member of r when ℓ is r or u is not a member of r when ℓ is $\overline{r}$ for $r \in R$. Permission to assign users to roles is specified by a ternary relation *can_assign* containing tuples of the form (C_a, C, r). Permission to revoke users from roles is specified by a binary relation *can_revoke* containing tuples of the form (C_a, r). We say that C_a is the *administrative pre-*

condition, a user u_a satisfying C_a is the *administrator*, and C is a *(simple) pre-condition*. The relation *can_revoke* is only binary because it has been observed that simple pre-conditions are useless when revoking roles (see, e.g., [10] for a discussion on this point). The semantics of the administrative actions in $\psi := (\mathit{can_assign}, \mathit{can_revoke})$ is given by a transition system whose states are the RBAC policies and a state change is specified by a binary relation $\rightarrow_\psi$ on pair of RBAC policies as follows: $UA \rightarrow_\psi UA'$ iff either (i) there exists $(C_a, C, r) \in \mathit{can_assign}$, u_a satisfying C_a, u satisfying C, and $UA' = UA \cup \{(u, r)\}$ or (ii) there exists $(C_a, r) \in \mathit{can_revoke}$, u_a satisfying C_a, and $UA' = UA \setminus \{(u, r)\}$.

A simple example. We consider the access control system of a small company as depicted in Figure 1, where a simple line between a user u—on the left—and a role r—in the middle—(resp. permission p—on the right) indicates that $(u, r) \in UA$ (resp. $(p, r) \in PA$) and an arrow from a role r_1 to a role r_2 that $r_1 \succeq r_2$. For example, B is an implicit member of role Em because M is more senior than Em and A is an explicit member of role M; thus, both A and B have permission E. Figure 1 also contains two administrative actions: the tuple in *can_assign* says that a member of role HR (the administrator) can add the role PT to a user who is a member of Em and is not member of FT while the pair in *can_revoke* says that a member of role M can revoke the role membership of a user of the role FT. For instance, it is easy to see that user A satisfies pre-condition $\{\mathsf{Em}, \overline{\mathsf{FT}}\}$ of the triple in *can_assign* and that user C can be the administrator, so that the application of the action is the RBAC policy $UA' := UA \cup \{(\mathsf{A}, \mathsf{PT})\}$. In UA', user A may get permission R besides E.

RBAC policy



Administrative actions
$(\{\mathsf{HR}\}, \{\mathsf{Em}, \overline{\mathsf{FT}}\}, \mathsf{PT}) \in \mathit{can_assign}$
$(\{\mathsf{M}\}, \mathsf{FT}) \in \mathit{can_revoke}$

Fig. 1. An ARBAC policy

User-role reachability problem. A pair (u_g, R_g) is called an *(RBAC) goal* for $u_g \in U$ and R_g a finite set of roles. The cardinality $|R_g|$ of R_g is the *size* of the goal. Given a set S_0 of (initial) RBAC policies, a goal (u_g, R_g), and administrative actions $\psi = (\mathit{can_assign}, \mathit{can_revoke})$, (an instance of) the *user-role reachability problem* [10] consists of establishing if there exist $UA_0 \in S_0$ and UA_f such that $UA_0 \rightarrow^*_\psi UA_f$ and u_g is member of each role of R_g in UA_f, where $\rightarrow^*_\psi$ denotes the reflexive and transitive closure of $\rightarrow_\psi$.

3 ASASP: architecture, implementation, and experiments

By using standard techniques [1], we can symbolically represent sets of policies and related administrative actions as BSR formulae and—along the lines of [2]—a backward reachability procedure may solve the user-role reachability problem.

The binary predicate ua and its primed version ua' denote an RBAC policy UA immediately before and after, respectively, of the execution of an administrative action.

An example of formalization. We briefly sketch how to represent the ARBAC policy in Figure 1 with BSR formulae. Let $User, Role$, and $Perm$ be sort symbols, $ua : User, Role$, $pa : Role, Perm$, and $\succeq : Role, Role$ be predicate symbols (constants symbols will be written in sanserif, as those in Figure 1 and implicitly assumed to be of appropriate sort). The fact that we have only three permissions E, V, and R can be axiomatized by the following sentences: $\mathsf{E} \neq \mathsf{V}$, $\mathsf{E} \neq \mathsf{R}$, $\mathsf{V} \neq \mathsf{R}$, and $\forall p.(p = \mathsf{E} \vee p = \mathsf{V} \vee p = \mathsf{R})$, for p variable of sort $Perm$. The fact that there are only five roles can be formalized similarly while the role hierarchy of Figure 1 is formalized by adding $\mathsf{M} \succeq \mathsf{FT}$, $\mathsf{FT} \succeq \mathsf{Em}$, and $\mathsf{PT} \succeq \mathsf{Em}$ to the BSR sentences for reflexivity, antisymmetry, and transitivity of $\succeq$ (constraining it to be a partial order). The (initial) relation UA of Figure 1 can be specified as the following formula in the BSR class: $\forall u, r.(ua(u, r) \Leftrightarrow (u = \mathsf{C} \wedge r = \mathsf{HR}) \vee (u = \mathsf{B} \wedge r = \mathsf{M}) \vee (u = \mathsf{A} \wedge r = \mathsf{Em}))$; and the relation PA as $\forall p, r.(pa(r, p) \Leftrightarrow ((r = \mathsf{HR} \wedge p = \mathsf{V}) \vee (r = \mathsf{Em} \wedge p = \mathsf{V})))$, where u, p, r are variables of sorts $User, Perm, Role$, respectively. The goal $(\mathsf{A}, \{\mathsf{PT}\})$ can be represented by the following formula in the BSR class: $\exists u, r.(r \succeq \mathsf{PT} \wedge ua(u, r))$. The administrative action $(\{\mathsf{HR}\}, \{\mathsf{Em}, \overline{\mathsf{FT}}\}, \mathsf{PT}) \in can_assign$ of Figure 1 can be represented as

$$\exists u_a, u, r_a, r_1. \begin{pmatrix} r_a \succeq \mathsf{HR} \wedge ua(u_a, r_a) \wedge \\ r_1 \succeq \mathsf{Em} \wedge ua(u, r_1) \wedge \neg \exists r_2.(r_2 \succeq \mathsf{FT} \wedge ua(u, r_2)) \wedge \\ \forall x, y.(ua'(x, y) \Leftrightarrow ((x = u \wedge y = \mathsf{PT}) \vee ua(x, y))) \end{pmatrix}. \quad (1)$$

The tuple in can_revoke can be represented similarly. Notice the presence of the (implicit) universal quantification in the sub-formula $\neg \exists r_2.(r_2 \succeq \mathsf{FT} \wedge ua(u, r_2))$ of (1). This may be problematic to guarantee closure under pre-image computation in the backward reachability procedure (see the discussion below). Fortunately, since there are only five roles (as shown in Figure 1), the universal quantifier can be replaced by the logically equivalent quantifier-free formula $\neg ua(u, \mathsf{FT}) \vee \neg ua(u, \mathsf{M})$, thereby avoiding problems for pre-image computation.

The core algorithm of ASASP[4] consists of iteratively computing the symbolic representation $\mathcal{R}(ua)$ of the set of backward reachable states as follows: $\mathcal{R}_0(ua) := G(ua)$ and $\mathcal{R}_{i+1}(ua) := \mathcal{R}_i(ua) \vee Pre(\mathcal{R}_i, T)$ for $i \geq 0$, where G is the symbolic representation of a RBAC goal, $T(ua, ua')$ is the symbolic representation of ψ, and $Pre(\mathcal{R}_i, T) := \exists ua'.(\mathcal{R}_i(ua') \wedge T(ua, ua'))$ is the *pre-image of* $\mathcal{R}_i$. To mechanize this, there are two issues to address. First, it should be possible to find a BSR formula which is logically equivalent to the second-order formula $Pre(\mathcal{R}_i, T)$, i.e. we have closure under pre-image computation. Second, there should exist decidable ways to stop computing formulae in the sequence $\mathcal{R}_0, \mathcal{R}_1, \mathcal{R}_2, ...$ This is done by performing either a *safety check*, i.e. test the satisfiability of $\mathcal{R}_i(ua) \wedge I(ua)$ where $I(ua)$ characterizes the set S_0 of initial

[4] The sources of the tool, the benchmark problems discussed below, and some related papers are available at `http://st.fbk.eu/ASASP`.

policies or a *fix-point check*, i.e. test the validity of $\mathcal{R}_{i+1}(ua) \Rightarrow \mathcal{R}_i(ua)$. Both logical problems should be decidable. In case the safety check is positive, ASASP concludes that the reachability problem has a solution, i.e. a sequence of administrative actions transforming one of the policies in I into one satisfying the goal G, and returns it. If the safety check is negative and the fix-point check is positive, ASASP returns that there is no solution. Following [2], restrictions on the shape of I, G, and T can be identified to guarantee the full automation of the procedure—i.e. the two requirements mentioned above and termination [1].

Architecture and refinements. The following three refinements of the core algorithm described above are crucial for efficiency. First, since $Pre(\mathcal{R}_i, \bigvee_{j=1}^{n} t_j)$ is equivalent to $\bigvee_{j=1}^{n} Pre(\mathcal{R}_i, t_j)$ when t_j is of a suitable form (e.g., that obtained when representing the administrative actions of ARBAC), it is possible to store the formula representing the set of backward reachable states by using a labelled tree, whose root node is labelled by the goal G, its children by $Pre(G, t_j)$, the edge connecting each child with the root by t_j, and so on (recursively) for $j = 1, ..., n$ (see [1] for details). It is easy to see that, by taking the disjunction of the formulae labelling all the nodes of a tree of depth k, we obtain a formula which is logically equivalent to $\mathcal{R}_k$ above. A key advantage of this data structure is to allow for an easy computation of the sequence of administrative actions leading from an initial policy to one satisfying the goal of a user-role reachability problem by simply collecting the labels of the edges from a leaf (whose label denotes a set of states with a non-empty intersection with the initial states) to the root node. This information is crucial for administrators to fix bugged policies. Another advantage consists in the possibility, without loss of precision, of deleting a node that would be labelled by an unsatisfiable preimage $Pre(\mathcal{R}_i, t_j)$, thereby reducing the size of the formula representing the set of backward reachable states and reducing the burden for SMT solvers and ATPs when checking for safety or fix-point. A similar approach for the elimination of redundancies in $\mathcal{R}_i$ has already been found extremely useful in MCMT [3,2]. To furtherly eliminate redundancy, after $Pre(\mathcal{R}_i, t_j)$ is found satisfiable, we check if $Pre(\mathcal{R}_i, t_j) \Rightarrow \mathcal{R}_{i-1}$ is valid and if so, $Pre(\mathcal{R}_i, t_j)$ is discarded. This can be seen as a simplified, and computationally cheap version, of the fix-point check, which is performed afterwards. Second, when checking for satisfiability, we first invoke an SMT solver and if this returns 'unknown,' we resort to an ATP (we call this a hierarchical combination). This is so because SMT solvers natively support (stack-wise) incrementality but, in many cases, approximate satisfiability checks for BSR, while ATPs are refutation complete and perform very well on formulae belonging to the BSR class but do not support incremental satisfiability checks. This is crucial to obtain a good trade-off between efficiency and precision. Third, the size of the goal critically affects the complexity of the security analysis problem for RBAC policies (see, e.g., [10]). In order to mitigate the problem, we incorporated a *divide et impera* heuristic: we generate a reachability problem for each role in the set R_g of roles in the goal. If (at least) one of the $k = |R_g|$ sub-goals is unreachable, ASASP concludes that the original goal is also so. Otherwise, if each sub-goal $j = 1, ..., k$ is reachable with a certain sequence

σ_j for administrative actions, then ASASP tries to solve an additional problem composed by the original goal with the transitions in $\bigcup_{j=1}^{k} \sigma_j$ (regarding σ_j as a set), that is hopefully smaller than the original set of administrative actions. If the last problem does not admit a solution (because some transitions may interfere), 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.

Implementation. ASASP is based on a client-server architecture, where the client (implemented in C) computes pre-images and generates the formulae encoding tests for safety and fix-point and the server consists of a hierarchic combination of the SMT solver Z3 (version 2.16) [12] and two ATPs, the latest versions of SPASS [9] and iProver [5]. To facilitate the integration of new satisfiability solving techniques, ASASP uses the new version (2.0) of the SMT-LIB format [8] for the on-line invocation of SMT solvers and the TPTP format [11] when using ATPs.

Experiments. We evaluated ASASP on a set of significant benchmarks and the results clearly demonstrate its scalability. We consider three classes of problems of increasing difficulty: (a) the synthetic benchmarks described in [10] for the ARBAC model without role 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 policies introduced in [1]. Each class consists of randomly generated user-role reachability problems which are classified w.r.t. the size of their goal, that was shown [7] to be the parameter characterizing the computational difficulty. 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. Figure 2 shows the plots of the median time (logarithmic scale) of ASASP and the tool—using backward search—described in [10],[5] called *Stoller* here, to solve the problems in the benchmark class (a) for increasing values of the goal size (for the results on the other two benchmark classes, see [1]). For small goal sizes, ASASP is slower than *Stoller* since it incurs in the overhead of invoking general purpose reasoning systems for safety and fix-point checks instead of the *ad hoc* algorithms used by *Stoller*. However, the time taken by *Stoller* grows quickly as the size of the goal increases and for goal size larger than 6, we do not report the median value

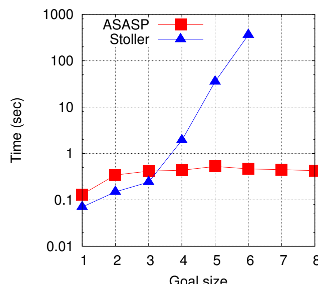


Fig. 2. ASASP vs. *Stoller*

[5] In [10], an algorithm based on forward reachability is also presented, which is evaluated on two classes of benchmarks whose goals are never reachable by construction. This corresponds to the worst-case choice for a forward search since it requires the exploration of the whole state space. For our tool—based on backward reachability—they are too easy and thus not considered here.

as *Stoller* solves less than 50% of the instance problems in the given time-out (set to 1,800 sec). There is a "cut-off effect" for goal sizes larger than 5 when problem instances become over-constrained (as it is unlikely that more and more goal roles are reachable). ASASP outperforms *Stoller* for larger values of the goal size; the key to obtain such a nice asymptotic behavior for ASASP is the third refinement described above. Similar observations also hold for the behavior of ASASP and *Stoller* on benchmark class (b). The results on benchmark class (c) show a similar asymptotic behavior for ASASP; a comparison with *Stoller* on these problems is impossible given its restrictions in the input format.

4 Conclusions and future work

We have presented ASASP, a tool for the symbolic analysis of security policies based on the model checking modulo theories approach of [3,2]. ASASP shows better scalability than the state-of-the-art tool in [10] on a significant set of instances of benchmarks. An interesting line of future work is to extend our approach to perform incremental analysis, i.e. incrementally updating the result of the analysis when the underlying policy changes as it is common in real-world applications [4].
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).

References

1. F. Alberti, A. Armando, and S. Ranise. Efficient Symbolic Automated Analysis of Administrative Role Based Access Control Policies. In *ASIACCS*, 2011.
2. S. Ghilardi and S. Ranise. Backward Reachability of Array-based Systems by SMT solving: Termination and Invariant Synthesis. *LMCS*, 6(4), 2010.
3. S. Ghilardi and S. Ranise. MCMT: a Model Checker Modulo Theories. In *Proc. of IJCAR*, LNCS, 2010.
4. M. I. Gofman, R. Luo, and P. Yang. User-Role Reachability Analysis of Evolving Administrative Role Based Access Control. In *ESORICS*, volume 6345 of *LNCS*, pages 455–471, 2010.
5. iProver. `http://www.cs.man.ac.uk/~korovink/iprover`.
6. F. P. Ramsey. On a problem in formal logic. In *Proc. of the London Mathematical Society*, pages 264–286, 1930.
7. A. Sasturkar, P. Yang, S. D. Stoller, and C.R. Ramakrishnan. Policy analysis for administrative role based access control. In *19th CSF Workshop*. IEEE, July 2006.
8. SMT-LIB. `http://www.smt-lib.org`.
9. SPASS. `http://www.spass-prover.org`.
10. S. D. Stoller, P. Yang, C.R. Ramakrishnan, and M. I. Gofman. Efficient policy analysis for administrative role based access control. In *ACM CCS*, 2007.
11. TPTP. `http://www.cs.miami.edu/~tptp`.
12. Z3. `http://research.microsoft.com/en-us/um/redmond/projects/z3`.