VAC - Verifier of Administrative Role-based Access Control Policies

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

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

Abstract. In this paper we present VAC, an automatic tool for verifying security properties of administrative Role-based Access Control (RBAC). RBAC has become an increasingly popular access control model, particularly suitable for large organizations, and it is implemented in several software. Automatic security analysis of administrative RBAC systems is recognized as an important problem, as an analysis tool can help designers check whether their policies meet expected security properties. VAC converts administrative RBAC policies to imperative programs that simulate the policies both precisely and abstractly and supports several automatic verification back-ends to analyze the resulting programs. In this paper, we describe the architecture of VAC and overview the analysis techniques that have been implemented in the tool. We also report on experiments with several benchmarks from the literature.

1 Introduction

Access control models allow to restrict access to shared resources by selectively assigning permissions to users. Role-based Access Control (RBAC) has become an increasingly popular access control model [5], it is standardized by NIST and is implemented in several software, such as Microsoft SQL Servers, Microsoft Active Directory, SELinux, and Oracle DBMS. RBAC reduces the complexity of user permissions administration by grouping users into roles and assigning permissions to each role. An *Administrative* RBAC User-Role Assignment (ARBAC-URA) policy defines a set of administrative roles and rules which specify how administrators *can assign* or *can revoke* roles to users [27].

Automatic security analysis of ARBAC systems is recognized as an important problem, as an analysis tool can help designers check whether their policies meet the expected security properties [25]. This is particularly desirable whenever policies need to be correct by design, for instance when accesses are not mediated by a monitor [6]. Most interesting security properties, such as privilege escalation and separation of duties, can be phrased as the *role-reachability problem* (i.e., is there a reachable configuration where some user can eventually be assigned to a target role?) [19, 7]. The role-reachability problem is known to be PSPACE-complete and hard to solve on real-world policies having hundreds of roles and rules and thousands of users [28].

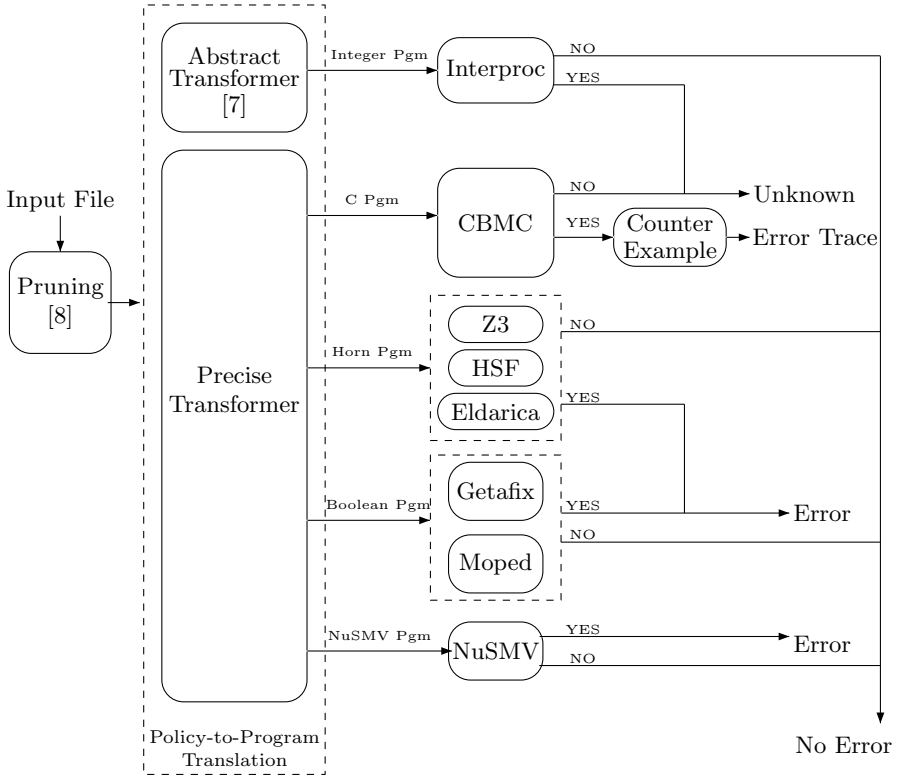

Fig. 1. VAC's Architecture.

In this paper, we present VAC (*Verifier of Access Control*), an automatic and scalable tool for solving the role-reachability problem of ARBAC-URA policies. The main components of VAC are a pruning module that aims at simplifying the state space by reducing a policy to a smaller one that preserves the reachability of the target role, and a policy-to-program translation module that converts a policy to an imperative program that simulates the policy both precisely and abstractly. VAC supports a plethora of automatic verification back-ends for the analysis of the resulting programs and has a built-in counterexample generator.

In the rest of the paper, we describe the architecture of VAC and overview the analysis techniques implemented in the tool. Finally, we present experimental results showing the effectiveness of VAC on analyzing realistic and complex benchmarks from the literature.

2 Software Architecture and Verification Approaches

The high-level architecture of VAC is shown in Fig. 1. We first describe the input format of VAC and then its components. **Input Format.** We refer to [8] for the syntax and the semantics of an ARBAC-URA system. To illustrate the input format of VAC[4] we use the toy example of an hospital policy shown in Fig. 2.

[4] VAC's input format is compatible with that of Mohawk [16].

```
ROLES EMPLOYEE DOCTOR MANAGER PATIENT;
USERS ANNA LUKE STEVE LUCY;
UA ⟨ANNA, DOCTOR⟩ ⟨LUCY, MANAGER⟩ ⟨LUKE, DOCTOR⟩ ⟨STEVE, PATIENT⟩;
CR ⟨DOCTOR, PATIENT⟩ ⟨DOCTOR, MANAGER⟩;
CA ⟨DOCTOR, EMPLOYEE & -DOCTOR, MANAGER⟩ ⟨DOCTOR, true, PATIENT⟩;
ADMIN ANNA LUKE;
SPEC DOCTOR;
```

Fig. 2. A VAC's input file.

ROLES and **USERS** are keywords used to list roles and users, respectively. **UA** defines the initial user-role assignment, whereas **CR** and **CA** specify the administrative rules *Can-Revoke* and *Can-Assign*, respectively. A **CR** rule is a pair of roles. For instance, the **CR** rule ⟨ DOCTOR, PATIENT ⟩ says that any administrator with role DOCTOR can revoke the role PATIENT from any user. A **CA** rule also contains a precondition, that is, a Boolean formula written as a conjunction of literals over roles. For instance, the **CA** rule ⟨ DOCTOR, EMPLOYEE & -DOCTOR, MANAGER ⟩ says that any administrator with role DOCTOR can assign any user u to the role MANAGER provided that u is member of EMPLOYEE and not a member of DOCTOR. The keyword **ADMIN** is used to list all users that are also administrators[5]. Finally, the keyword **SPEC** is used to specify the role-reachability query, namely the target role. VAC can also be used to check whether the target role is reachable by a specific user. For instance, the query **SPEC** LUCY DOCTOR allows to check whether LUCY can ever obtain role DOCTOR.

Pruning Module. This module takes as input a policy, which we refer as the original policy, and outputs a simplified one (in the same format as the input) that preserves the reachability of the target role. The module implements the pruning heuristic from [8] which is crucial for scalability. It eliminates roles, rules and users with the aim of reducing the state space to explore. This heuristic relies upon a fundamental theorem which states that the role-reachability problem can be solved by tracking only $k+1$ users, where k is the number of administrative roles [8]. Thus, the heuristic exploits sufficient conditions to eliminate administrative roles that are not relevant for the analysis. The effectiveness of the above method is amplified by a static pruning algorithm consisting of six pruning actions: the first three aim at discarding roles that are irrelevant to the reachability of the target role while the remaining ones identify administrative rules that can be combined or eliminated. Furthermore, whenever the target role is reachable within two steps in the intermediate policy, the pruning procedure terminates immediately returning the counterexample.

Policy-to-Program Translation. This module takes as input a policy and translates it into a program that simulates the evolution of the system. VAC provides the following two policy-to-program translations:
Abstract Transformer. This module implements the policy-to-program translation proposed in [7]. A policy P is translated into an imperative non-deterministic while-program P' with an error location. P' uses only integer variables to capture the number of users in a subset of role combinations and abstractly simulates the evolution of the system in such a way that if the error location is not reachable

[5] The list of administrators can be obtained from **CA** and **CR** rules. However, we include the keyword **ADMIN** to be consistent with Mohawk's input format [16].

then the role reachability problem has a negative answer on P. On the contrary, if the error location is reachable in P', this may correspond to a false positive as P' over-approximates the behaviour of P. P' is then analyzed by VAC using Interproc [18] with the box abstract domain.
Precise Transformer. This module translates a policy P into a Boolean program P' that precisely simulates the evolution of the system tracking at most $k+1$ users picked non-deterministically, where k is the number of administrative roles. The correctness of this approach relies on a fundamental theorem proven in [8]. The program uses $k+1$ blocks of n Boolean variables, where n is the number of roles in the policy. Each block tracks the role-membership of a selected user. The rest of the program consists of an infinite loop in which the administrative rules are non-deterministically simulated on a non-deterministically chosen user. The loop contains also an error location that is reachable whenever a tracked user reaches the target role. The role-reachability problem admits a positive answer on P if and only if the error location is reachable in P'. The reachability problem for Boolean programs is decidable and VAC supports several automated tools as back-ends for the analysis of P'. In particular, a complete analysis can be performed by using either (1) one of the following tools for Horn clauses: Z3 (μZ) [4, 13], HSF [11, 12], and Eldarica [15, 14], or (2) Moped [20, 29] and Getafix [23, 24] which are model checkers for Boolean Programs based on BDDs, or (3) NuSMV a model checker based on BDDs and SAT solvers [2, 3]. VAC uses the C bounded model checker CBMC [21, 22] for under-approximate analysis, particularly effective to find errors. If CBMC finds an error, it returns a counterexample showing how the error location can be reached in the program. Otherwise, VAC reports Unknown.

Counterexample Module. VAC implements an involved built-in counterexample generation module that takes as input the counterexample of the pruned policy returned by CBMC along with some information collected during the execution of the pruning, and outputs a counterexample (attack) of the original policy.

3 Implementation and Availability

Implementation. VAC is implemented in C and has dependencies with ANTLR (v3.2 for C), ROXML, and CCL libraries[6].
Availability. The source code, a set of benchmarks and static Linux binaries are available at: http://users.ecs.soton.ac.uk/gp4/VAC.
Usage. The shell command `./vac.sh <InputFile>` runs VAC with the default setting: (1) runs the abstract transformer and Interproc to prove correctness; (2) if a proof cannot be provided, VAC runs the precise transformer and CBMC (with `unwind` set to 2) to find a counterexample; (3) if CBMC does not find an error, VAC runs μZ for complete analysis. VAC has options to print the translated programs and the simplified policies, and select the back-end for the analysis.

[6] ANTLR, ROXML and CCL are respectively available at http://www.antlr.org/, http://www.libroxml.net, and https://code.google.com/p/ccl/.

	ARBAC Policy					Pruning					Reach	
	name	#roles	#rules	#admin	#users	#roles	#rules	#admin	#users	Time	Answer	Time
1	Hospital1	13	37	5	1092	4	5	3	6	0.009s	No	0.029s
	Hospital2	13	37	5	1092	4	5	3	6	0.009s	No	0.023s
	Hospital3	13	37	5	1092	3	2	1	4	0.009s	Yes	0.103s
	Hospital4	13	37	5	1092	4	4	1	4	0.009s	Yes	0.110s
2	University1	32	449	9	943	6	7	3	13	0.009s	No	0.034s
	University2	32	449	9	943	6	8	3	13	0.004s	Yes	0.192s
	University3	32	449	9	943	4	5	1	6	0.006s	No	0.021s
	University4	32	449	9	943	12	37	4	31	0.004s	Yes	1.571s
3	Bank1	343	2225	1	2	3	2	1	2	0.007s	Yes	0.112s
	Bank2	683	4445	1	2	3	2	1	2	0.019s	Yes	0.139s
	Bank3	1023	6665	1	2	3	2	1	2	0.024s	Yes	0.167s
	Bank4	1363	8885	1	2	3	2	1	2	0.030s	Yes	0.168s
	Bank5	343	2225	1	2	3	2	1	2	0.044s	Yes	0.138s
	Bank6	683	4445	1	2	3	2	1	2	0.155s	Yes	0.247s
	Bank7	1023	6665	1	2	3	2	1	2	0.300s	Yes	0.435s
	Bank8	1363	8885	1	2	3	2	1	2	0.522s	Yes	0.663s
	Bank9	531	5126	1	2000	2	0	1	2	0.244s	No	0.253s
	Bank10	531	5126	1	2000	2	0	1	2	0.248s	No	0.254s
	Bank11	531	5126	1	2000	3	2	1	2	0.245s	Yes	0.396s
	Bank12	531	5126	1	2000	6	5	1	2	0.066s	Yes	0.223s

Table 1. VAC's results on realistic case studies.

4 Experimental Results

We evaluate VAC, using the default setting, on several benchmarks from the literature. All experiments have been performed on a Linux 64-bit machine with Intel Core i7-3770 CPU and 16GB of RAM.

Table 1 shows the results on three sets of benchmarks based on realistic case studies. The first two case studies are carried out by Stoller at al. [30] and represent policies for a university and for an hospital, respectively. The third case study, conducted by Jayaraman et al. [16], models a bank with several branches[7]. While the first eight bank policies are from [16], we have built the last four from [16] by slightly modifying their policies to add more users and to make two of them correct. Table 1 reports the number of roles, rules, administrative roles and users of both the original policy and that after pruning. It also reports the tool's answer, the time taken by the pruning, and the overall analysis time.

Table 1 shows that the pruning module significantly reduces the size of these policies. Furthermore, VAC is extremely efficient in verifying these policies, regardless of whether the target role is reachable or not. More precisely, all benchmarks with a negative answer can be proved correct in less than a second. Similarly, on benchmarks with a reachable target the analysis takes less than 2 seconds including the time to generate the counterexample.

Table 2 shows the results on three sets of complex test suites, synthetically generated by Jayaraman et al. [16], with the aim of capturing the complexity of real systems. Each suite consists of ten policies where the number of roles and rules ranges respectively from 4 to 40k and 10 to 200k. The role-reachability problem has a positive answer on all these benchmarks. VAC is very effective on

[7] The number of roles and rules depends on the number of branches considered. For instance, 343 roles corresponds to 10 branches and 1363 to 40 branches.

Size Policy		VAC								
		First Suite			Second Suite			Third Suite		
#roles	#rules	Pruning		Verification Time	Pruning		Verification Time	Pruning		Verification Time
		#roles	#rules		#roles	#rules		#roles	#rules	
4	10	3	1	0.080s	3	1	0.084s	2	1	0.085s
5	25	4	2	0.087s	4	2	0.096s	2	1	0.092s
20	100	3	1	0.099s	3	1	0.089s	3	2	0.087s
40	200	4	2	0.099s	4	2	0.096s	2	1	0.091s
200	1000	2	1	0.101s	2	1	0.088s	2	1	0.096s
500	2500	3	1	0.100s	3	1	0.104s	3	2	0.128s
4000	20000	2	1	0.239s	2	1	0.198s	4	3	0.252s
20000	80000	2	1	0.844s	2	1	0.579s	3	2	0.922s
30000	120000	2	1	1.288s	2	1	0.849s	2	1	1.285s
40000	200000	2	1	1.586s	2	1	1.100s	4	3	1.646s

Table 2. VAC' s results on complex test suites.

these policies as well. The analysis takes less than 2 seconds on all policies and the pruning module reduces the policies to equivalent systems with a handful of roles and rules.

5 Conclusions

We have presented VAC an automatic and efficient tool for verifying security properties of administrative role-based access control policies. The main components of VAC are a pruning module which is essential for scalability, and a policy-to-program translation module that reduces the role-reachability problem to program verification problems. It supports several tools for the analysis, such as CBMC, Eldarica, Getafix, Interproc, Moped, NuSMV, HSF, and Z3 (μZ). Furthermore, it can provide counterexamples.

Related work. Among the state-of-the-art tools for the analysis of ARBAC-URA systems, VAC is the only tool that simultaneously has the following features: (1) complete analysis (2) counterexample generation, and (3) scalable analysis on large policies. Mohawk [16] performs only under-approximate analysis, though it now considers thresholds for completeness [17]; RBAC-PAT [10] is unable to handle large policies. They also can only analyze policies with *separate administration* where administrators cannot change their role-membership; this is not realistic, but simplifies analysis as only a single user needs to be tracked.

ASASPXL is the latest tool developed by Ranise et al. for the analysis of ARBAC policies [26]. A previous version (ASASP [1]) was not able to scale on large policies. ASASPXL is mainly designed to handle large policies and does so by encoding the instances to MCMT [9] which is a model checker for infinite state systems based on SMT solvers and backward reachability. In contrast, VAC does not target any specific kind of instances, and handles large policies by carrying out an effective pruning that is independent of the verification technique used for the analysis. VAC and ASASPXL can potentially handle the same kind of instances though they have different input formats.

All tools above do not generate counterexamples. Furthermore, VAC, on the policies of Section 4, has either the same performances or outperforms the tools mentioned above. VAC has also been used for the analysis of *temporal* RBAC [31].

Acknowledgements: Research was partially supported by NSF CCF #1018182.

References

[1] F. Alberti, A. Armando, and S. Ranise. ASASP: Automated Symbolic Analysis of Security Policies. In N. Bjørner and V. Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *Lecture Notes in Computer Science*, pages 26–33. Springer, 2011.

[2] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV: A New Symbolic Model Checker. http://nusmv.fbk.eu.

[3] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In E. Brinksma and K. G. Larsen, editors, *CAV*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer, 2002.

[4] L. de Moura, J. Berdine, and N. Bjorner. Z3 High-performance Theorem Prover. http://z3.codeplex.com.

[5] D. Ferraiolo and R. Kuhn. Role-Based Access Control. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563. Springer, 1992.

[6] A. L. Ferrara, G. Fuchsbauer, and B. Warinschi. Cryptographically Enforced RBAC. In *CSF*, pages 115–129. IEEE, 2013.

[7] A. L. Ferrara, P. Madhusudan, and G. Parlato. Security Analysis of Role-Based Access Control through Program Verification. In *CSF*, pages 113–125, 2012.

[8] A. L. Ferrara, P. Madhusudan, and G. Parlato. Policy Analysis for Self-administrated Role-Based Access Control. In *TACAS*, pages 432–447, 2013.

[9] S. Ghilardi and S. Ranise. MCMT: A Model Checker Modulo Theories. In J. Giesl and R. Hähnle, editors, *IJCAR*, volume 6173 of *Lecture Notes in Computer Science*, pages 22–29. Springer, 2010.

[10] M. I. Gofman, R. Luo, A. C. Solomon, Y. Zhang, P. Yang, and S. D. Stoller. RBAC-PAT: A Policy Analysis Tool for Role Based Access Control. In *TACAS*, pages 46–49, 2009.

[11] S. Grebenshchikov, A. Gupta, N. P. Lopes, C. Popeea, and A. Rybalchenko. HSF(C): A Software Verifier based on Horn Clauses. http://www7.in.tum.de/tools/hsf.

[12] S. Grebenshchikov, A. Gupta, N. P. Lopes, C. Popeea, and A. Rybalchenko. HSF(C): A Software Verifier Based on Horn Clauses - (Competition Contribution). In C. Flanagan and B. König, editors, *TACAS*, volume 7214 of *Lecture Notes in Computer Science*, pages 549–551. Springer, 2012.

[13] K. Hoder, N. Bjørner, and L. M. de Moura. μZ - An Efficient Engine for Fixed Points with Constraints. In G. Gopalakrishnan and S. Qadeer, editors, *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 457–462. Springer, 2011.

[14] H. Hojjat, F. Konecný, F. Garnier, R. Iosif, V. Kuncak, and P. Rümmer. A Verification Toolkit for Numerical Transition Systems - Tool Paper. In D. Giannakopoulou and D. Méry, editors, *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 247–251. Springer, 2012.

[15] H. Hojjat, P. Rümmer, and F. Konecny. A Predicate Abstraction Engine. http://lara.epfl.ch/w/eldarica.

[16] K. Jayaraman, V. Ganesh, M. V. Tripunitara, M. C. Rinard, and S. J. Chapin. Automatic Error Finding in Access-Control Policies. In *CCS*, pages 163–174, 2011.

[17] K. Jayaraman, M. V. Tripunitara, V. Ganesh, M. C. Rinard, and S. J. Chapin. Mohawk: Abstraction-Refinement and Bound-Estimation for Verifying Access Control Policies. *ACM Trans. Inf. Syst. Secur.*, 15(4):18, 2013.

[18] B. Jeannet, G. Lalire, and M. Argoud. The Interproc Analyzer. http://pop-art.inrialpes.fr/interproc/interprocweb.cgi.

[19] S. Jha, N. Li, M. Tripunitara, Q. Wang, and W. Winsborough. Towards Formal Verification of Role-Based Access Control Policies. *IEEE Transactions on Dependable and Secure Computing*, 5(4):242–255, 2008.

[20] S. Kiefer, S. Schwoon, and D. Suwimonteerabuth. A Model Checker for Pushdown Systems. http://www2.informatik.uni-stuttgart.de/fmi/szs/tools/moped.

[21] D. Kroening and E. Clarke. CBMC - Bounded Model Checking for ANSI-C. http://www.cprover.org/cbmc.

[22] D. Kroening and M. Tautschnig. CBMC - C Bounded Model Checker - (Competition Contribution). In E. Ábrahám and K. Havelund, editors, *TACAS*, volume 8413 of *Lecture Notes in Computer Science*, pages 389–391. Springer, 2014.

[23] S. La Torre, P. Madhusudan, and G. Parlato. Getafix: A Symbolic Model-checker for Recursive Programs. http://www.cs.uiuc.edu/∼madhu/getafix.

[24] S. La Torre, P. Madhusudan, and G. Parlato. Analyzing Recursive Programs using a Fixed-point Calculus. In M. Hind and A. Diwan, editors, *PLDI*, pages 211–222. ACM, 2009.

[25] N. Li and M. V. Tripunitara. Security Analysis in Role-Based Access Control. *ACM Trans. Inf. Syst. Secur.*, 9(4):391–420, Nov. 2006.

[26] S. Ranise, A. Truong, and A. Armando. Boosting Model Checking to Analyse Large ARBAC Policies. In A. Jøsang, P. Samarati, and M. Petrocchi, editors, *STM*, volume 7783 of *Lecture Notes in Computer Science*, pages 273–288. Springer, 2012.

[27] R. S. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 Model for Role-Based Administration of Roles. *ACM Trans. Inf. Syst. Secur.*, 2(1):105–135, 1999.

[28] A. Sasturkar, P. Yang, S. D. Stoller, and C. Ramakrishnan. Policy analysis for Administrative Role-Based Access Control. *Theoretical Computer Science*, 412(44):6208 – 6234, 2011.

[29] S. Schwoon. *Model-Checking Pushdown Systems*. Ph.D. Thesis, Technische Universität München, June 2002.

[30] S. D. Stoller, P. Yang, C. R. Ramakrishnan, and M. I. Gofman. Efficient Policy Analysis for Administrative Role Based Access Control. In *CCS*, pages 445–455, 2007.

[31] E. Uzun, V. Atluri, S. Sural, J. Vaidya, G. Parlato, A. L. Ferrara, and P. Madhusudan. Analyzing temporal role based access control models. In V. Atluri, J. Vaidya, A. Kern, and M. Kantarcioglu, editors, *SACMAT*, pages 177–186. ACM, 2012.