Role-based Trust Management Security Policy Analysis and Correction Environment (RT-SPACE)

Mark Reith*
University of Texas
at San Antonio
One UTSA Circle
San Antonio, Texas, USA
mreith@cs.utsa.edu

Jianwei Niu
University of Texas
at San Antonio
One UTSA Circle
San Antonio, Texas, USA
niu@cs.utsa.edu

William H. Winsborough
University of Texas
at San Antonio
One UTSA Circle
San Antonio, Texas, USA
wwinsborough@acm.org

ABSTRACT

This paper presents RT-SPACE, a tool suite for authoring, verifying, and correcting RT access control policies. RT is a role-based trust management framework well suited for use in systems that must protect the interests of multiple stakeholders in a decentralized environment.

Categories and Subject Descriptors: D.2 [Software Engineering]: Software/Program Verification

General Terms: Security, Verification.

Keywords: Trust Management Policy, Model Checking.

1. INTRODUCTION

When access control policies are subject to change, analyzing them for security properties such as safety (*e.g.* access to the database is limited to employees) and liveness (*e.g.* managers will always have access to the database) requires significant tool support. Such properties can be expressed in the form of role containment queries, such as the safety property in the form $Company.employee \sqsupseteq Database.access$ or the liveness property in the form $Database.access \sqsupseteq Company.manager$. We present RT-SPACE as a tool for authoring, verifying, and correcting RT policies, based on the notion of security analysis [5].

2. POLICY ANALYSIS FRAMEWORK

The architecture and process of RT-SPACE are illustrated as a data flow graph in Figure 1, while the technical details are fully described in [7]. The policy author crafts or updates a policy, and then submits it, along with some set of desirable properties, as input to the tool. The tool performs a conservative conversion of the input to one or more policy models. Each model is passed to a model checking component for automated verification. For a policy model that fails to satisfy the desired properties, the model checker generates a set of counterexamples. The policy correction component analyzes the counterexample set in order to generate a list of suggested corrections. The policy author may select an appropriate correction from this list or make other changes. The modified policy would then serve as input for the next iteration of the analysis. This subsequent iteration is necessary to ensure other properties have not been invalidated. The process of authoring, analyzing, and correcting policies is called *iterative correction*.

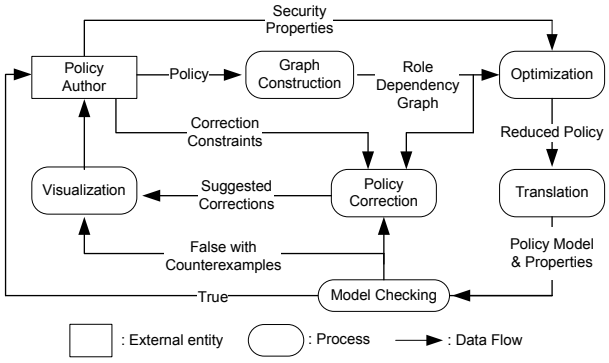

Figure 1: Policy Analysis Framework

Our policy analysis framework consists of tool support and methodology for automated verification and correction of security policies. With the exception of the model checking component, each of the following is implemented in Java.

Graph Construction. The role dependency graph is a key data structure for representing a policy as it describes relationships between roles. Optimization, translation, correction, and visualization depend on this data structure.

Optimization. Structural optimization may reduce the policy size and/or complexity by reducing the number of policy statements necessary for verifying desired properties. This includes calculating a smaller equivalent initial policy under a suitable notion of equivalence, decomposing a policy into sub-policies, and removing extraneous policy statements (cone of influence reduction).

Translation. Translation is the automated conversion of an RT policy into a finite state machine suitable for a model checker. It consists of unrolling any circular dependencies, calculating the maximum number of principals necessary to expose all policy flaws, constructing the SMV model from the policy, and constructing temporal logic property specifications from desired properties. It may optionally provide two optimizations [7]. The first optimization is called chain

*The views expressed in this article are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

Copyright is held by the author/owner(s).
ICSE'08, May 10–18, 2008, Leipzig, Germany.
ACM 978-1-60558-079-1/08/05.

reduction and may prune the state space and reduce analysis time. The second optimization is called role abstraction and may collapse certain sets of states into a smaller set of states, thus improving analysis time.

Model Checking. The model checking process determines whether a given policy satisfies desired security properties. In the case when the property is not satisfied, the model checker produces a counterexample, a reachable state where the properties fail to hold. We used the well established model checker SMV (Symbolic Model Verifier) [6] as an external model checking tool called by RT-SPACE.

Correction. The correction component takes a counterexample and the role dependency graph of the original policy as input and analyzes how the policy may be changed in order to make the counterexample unreachable. This process generates a list of such changes. Potential changes are filtered by a set of constraints provided by the policy author.

Visualization. The visualization component illustrates the initial, translated, and counterexample policy states. It primarily serves to focus attention on the relevant portion of the policy based on the counterexample so the policy author can better diagnose the policy flaw. For visualization, we used a graph layout package called *dot* and a Java rendering package called *Grappa*, both from the Graphviz [1] suite.

3. ADDRESSING INTRACTABILITY

The translation process maps an initial policy to a finite state system (policy model) that represents the reachable state space of the policy. In order to provide a conservative conversion in the case of role containment queries, it is necessary to introduce at most an exponential number of new n principals [5], in order to ensure that all possible policy flaws of the initial policy are exposed in the model. This may create intractable policy models and it may not always be feasible to verify that a policy satisfies a desired security property. One option is to underapproximate the policy model by introducing m number of new principals where $m < n$. Instead of generating the entire reachable state space of the policy using n principals, underapproximation generates a subset of this state space using m principals. This is conceptually similar to bounded model checking from the perspective that any counterexample generated is valid, but there is no guarantee that all counterexamples can be generated. This is significant because it provides some degree of assurance that the required security property holds, even when attaining full assurance is intractable.

4. RELATED WORK

Security analysts of access control systems and policies have proposed several means of determining if desired properties are satisfied. Jha *et al.* [3] and Sistla *et al.* [8] offer approaches for verifying security properties of SPKI/SDSI and RT policy languages respectively. Jha's work supports a subset of the RT language and is limited to analyzing static policies. Sistla's work does not address usability issues as in the case of intractably sized policies. Neither address policy correction as a means of assisting the user to fix the policy. Other related work includes Jha *et al.* [2] which empirically compares the efficiency of Datalog verses model checking techniques for evaluating security properties on RBAC policies.

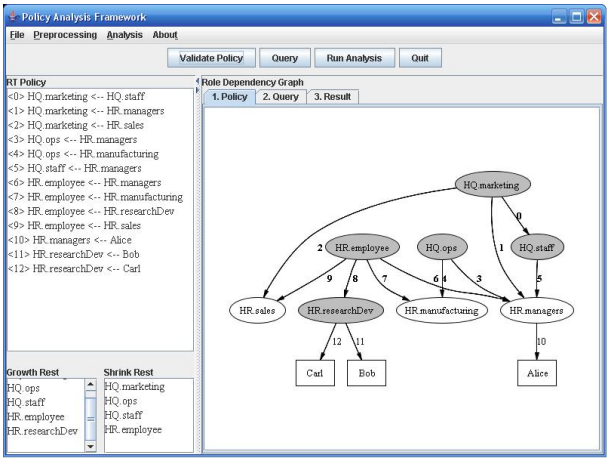

Figure 2: RT-SPACE Screenshot

5. CONCLUSION

We introduce a tool that provides an environment for evaluating security policies. It employs model checking, optimizations, and an underapproximation technique for addressing intractable cases. A significant benefit of using model checking to verify security properties is the counterexample that is produced upon verification failure. Such information can be used to generate suggested corrections, and through an iterative correction process the policy is shaped into a form that satisfies those properties. Although we have evaluated the translation component, further work remains to evaluate the degree of assurance when underapproximation is used.

6. REFERENCES

[1] E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Software — Practice and Experience*, 30(11):1203–1233, 2000.

[2] S. Jha, N. Li, M. Tripunitara, Q. Wang, and W. Winsborough. Towards formal verification of role-based access control policies. To appear in IEEE Trans. on Dependable & Secure Computing (TDSC).

[3] S. Jha and T. Reps. Model checking SPKI/SDSI. In *Journal of Computer Security*, volume 12, pages 317–353, 2004.

[4] J. Jurjens. Sound methods and effective tools for model-based security engineering with UML. In *27th ICSE*, pages 322–331, 2005.

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

[6] K. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. 1993.

[7] M. Reith, J. Niu, and W. H. Winsborough. Policy analysis framework for verification and correction. Technical Report CS-TR-2007-006, UTSA, 2007.

[8] A. P. Sistla and M. Zhou. Analysis of dynamic policies. In *Foundations of Comp. Sec. & Automated Reasoning for Security Protocol Analysis*, pages 233–262, 2006.