ACPT: A Tool for Modeling and Verifying Access Control Policies

JeeHyun Hwang[1] Tao Xie[1] Vincent Hu[2] Mine Altunay[3]
[1]Department of Computer Science, North Carolina State University, Raleigh
[2]Computer Security Division, National Institute of Standards and Technology, Gaithersburg
[3]Computing Division, Fermi National Laboratory, Batavia
jhwang4@ncsu.edu xie@csc.ncsu.edu vincent.hu@nist.gov maltunay@fnal.gov

Abstract

Access control mechanisms are a widely adopted technology for information security. Since access decisions (i.e., permit or deny) on requests are dependent on access control policies, ensuring the correct modeling and implementation of access control policies is crucial for adopting access control mechanisms. To address this issue, we develop a tool, called ACPT (Access Control Policy Testing), that helps to model and implement policies correctly during policy modeling, implementation, and verification.

1 Introduction

Access control is one of the most fundamental and widely used privacy and security mechanisms at both application and network levels. Access control mechanisms control which principals such as users or processes have access to which resources based on access control policies. Since access decisions on requests are based on policies, misconfigurations or faults in policies result in consequences such as disallowing an authorized user to access her resources. Moreover, such faulty policies may lead to security vulnerabilities (e.g., malicious users can access critical resources).

Correctly specifying and maintaining access control policies is an important and yet challenging task due to two main factors. First, a policy may consist of a large number of rules, especially when being used to control access on a large amount of distributed and sensitive information in large-scale computing environments. Second, a policy can become complex in order to meet various security and privacy requirements (such as legal issues) of policy authors. To ensure the correctness of policy specifications, policy authors must conduct rigorous policy verification and validation to ensure that the policy specifications truly encapsulate the requirements of the policy authors.

To address this issue, we develop a tool, called ACPT (Access Control Policy Testing), that assists policy authors in implementing policies correctly during policy modeling, implementation, and verification. Our previous work [7, 9, 10] focuses on verifying access control policies (in XACML [1]) with property verification (static verification) or test-input generation (dynamic verification). However, our previous work does not help create policies embedded in models or generate enforceable policies. ACPT bridges the gap between policy requirements and policy implementations by generating enforceable policies (in XACML) directly from policy models (reflected by policy requirements). In addition, ACPT supports static and dynamic verification of the generated policies to reduce faults in the policies.

ACPT provides three major functionalities. First, ACPT helps specify and combine policies based on well-known existing policy models. Second, ACPT analyzes and converts a policy (generated based on policy models) into an enforceable format such as XACML. Third, to ensure policy correctness, ACPT conducts both static and dynamic verification of a policy. Given a user-specified property set and a policy, our static verification checks whether the properties are satisfied. However, the confidence on policy correctness based on the static verification is dependent on the quality of the specified properties [8]. ACPT conducts dynamic verification (i.e., testing) to complement static verification by generating and executing test suites. Through dynamic verification, the policy authors can attain higher confidence on policy correctness.

2 ACPT Components

This section presents our tool for modeling and verifying access control policies. Our tool (shown in Figure 1) includes four components: policy modeling, static verification, dynamic verification, and policy implementation.

2.1 Policy Modeling

ACPT allows policy authors to create policies based on well-known access control models such as Role-Based Ac-

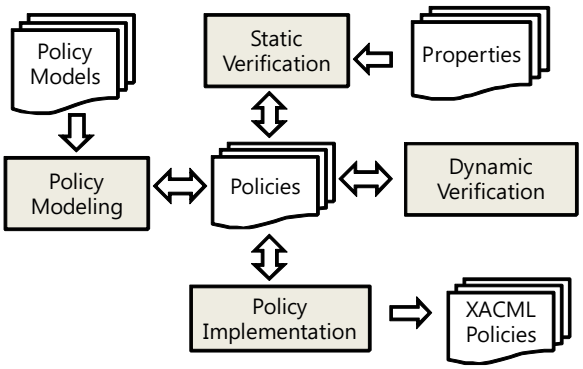

Figure 1. ACPT overview

cess Control (RBAC) [5], Attribute-Based Access Control (ABAC) [11], and Multi-Level security [3].

ACPT provides GUI to help policy authors specify policies and their properties interactively and effectively. Moreover, ACPT also supports additional features where the policy authors can edit, add, or delete policies and their attributes interactively. Figure 2 shows GUI in ACPT to model policies.

2.2 XACML Policy Implementation

XACML (eXtensible Access Control Markup Language) [1] is a standard language used to represent and evaluate access control policies. It was mainly designed as a standard for expressing both access requests and access control policies. ACPT takes a policy as an input and generates an XACML-represented policy, by mapping attributes in the policy to their corresponding XACML attributes. We use *condition*, i.e., a boolean function (in XACML) associated with a rule to specify its constraints; the rule matches with a request when each of attribute values in the rule matches against a request and the *condition* is evaluated to be true.

2.3 Static Verification

To ensure correct behaviors of a policy against its properties, we apply static verification on a policy to verify whether its properties are satisfied.

ACPT takes a policy p and its property r as inputs and verifies p against r using NuSMV [4]. Our previous work [6, 7] proposed an approach to represent a policy and its properties as a corresponding finite state machine (FSM) model and temporal logics (e.g., computation tree logic), respectively, using the SMV specification language [4]. SMV is a language that can represent various types of access control policy models and properties of access constraints. A property can be specified as a logical formula that represents whether a specific state can be reachable for the given constraints. NuSMV explores states to detect any states that violate the property. Properties can be either constraints or specific policy behaviors that should be preserved in the policy. The work focuses on modeling popular mandatory access control policies (RBAC [5] and Multi-Level security [3]). ACPT takes a policy p and converts p to an FSM model p'. Through state space search of p', a symbolic model checker called NuSMV [4] can search for the set of states where a property r is true or false; if the set of states (where r is false) is found, NuSMV reports that r is violated with counterexamples. The policy authors then inspect the counterexamples and fix the problems in p until no further violations are detected. However, ACPT does not provide any suggestions on how to fix the problems. Fixing problems depends on the policy authors' knowledge decision. The policy authors manually add/delete/modify rules in the policy to fix the problems.

2.4 Dynamic Verification

Given policy specifications, policy authors implement policies P_{imp} in a system. Dynamic verification is a testing process to assure the correctness of P_{imp}. By observing the evaluation of a policy implementation with a test input (i.e., request), policy authors may identify faults in the policy, and validate whether the corresponding output (i.e., decision) is intended. Confidence on policy correctness gained via static verification is dependent on the quality of the specified properties [8]. The policy authors have additional confidence on policy correctness by complementing static verification with dynamic verification (i.e., verifying a policy through executing test inputs on P_{imp}).

ACPT generates test inputs based on structural coverage [10] and combinatorial coverage [7]. Our previous work [7, 9, 10] proposed structural test generation (white-box testing) and combinatorial test generation (black-box testing) to reduce the size of generated requests for output inspection while providing a sufficient level of confidence on policy correctness. The objective of structural test generation is to generate test inputs to achieve 100 % structural coverage (e.g., rule coverage) in a policy. Achieving high structural coverage helps investigate a large portion (e.g., all or most of the rules being evaluated at least once) of policy entities (e.g., rules) for fault detection. We first use structural test generation before using combinatorial test generation to augment the generated test inputs for achieving the n-wise (e.g., pairwise) coverage criterion. The reason for using combinatorial test generation after structural test generation is that structural test generation (white-box testing) cannot detect some types of faults including omission faults (e.g., missing a rule). Specifically, combinatorial test generation is used to test all n-way combinations of input parameter values. In the policy context, for combinatorial testing, three attributes (subject, action, and object) and attribute values correspond to parameters and parameter inputs, respectively. For combinatorial test generation, Figure 3

	Subjects	Resources	Actions
1	Faculty	Grades	Write
2	Faculty	Records	View
3	Student	Grades	View
4	Student	Records	Write

Figure 3. 2-way combinatorial test inputs

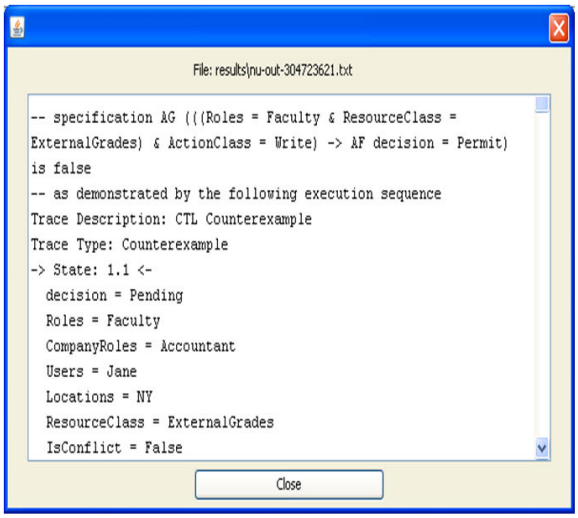

Figure 4. Verification results provided by ACPT

shows 2-way combinatorial test inputs, respectively, for the given subjects (e.g., Faculty and Student), resources (e.g., Grades and Records), and actions (e.g., Write and View). In Figure 3, each row corresponds to a single request.

3 ACPT Prototype

We implemented ACPT using Java. ACPT includes features of policy modeling, implementation, static and dynamic verification. ACPT is a Windows GUI-based tool to help users model and implement a policy and its properties interactively and effectively. We integrated NuSMV and ACTS[1] into ACPT for symbolic model checking and combinatorial test generation. We also integrated our previous structural test generation tool [10] for generating test inputs to achieve 100% structural coverage (e.g., rule coverage) of a policy under test. ACPT measures the structural coverage of the policy under test after test inputs (i.e., requests) are evaluated.

Figure 2 shows GUI that helps policy authors specify policies based on various access control models such as RBAC, ABAC, and Multi-Level security. The policy workspace in Figure 2 shows specified policy models as a tree structure. The policy editor provides a working area for the policy authors to edit a selected model. In Figure 2, the policy authors specify an RBAC policy with a set of roles (e.g., Faculty and Student), user-role relations (e.g., Jane is a faculty member and Jim is a student), and roles' permissions (e.g., a faculty member can write grades and a student cannot write grades).

After modeling policies using the GUI, the policy authors can conduct static verification. Static verification is useful to ensure the correctness of an individual policy or combined policies against a set of properties. ACPT allows that the policy authors specify multiple policies each of which includes a set of rules. For combining policies, the policy authors can specify which policy has to be given higher priority than other policies to resolve conflicts when multiple policies or rules are applied to the same request during policy evaluation. Figure 4 shows a screen snap shot, where ACPT provides property verification results when multiple policies are combined. Verifying properties against the combined policies is useful for the policy authors to understand whether properties (that are satisfied in an individual policy) are further satisfied after combination. With this feature, the policy authors prevent unintended policy behaviors (i.e., property violations) after combining policies.

After conducting static verification, ACPT helps the policy authors implement XACML policies. ACPT automatically converts a policy (specified in ACPT) to a corresponding XACML policy. We use XACML policy templates for RBAC, ABAC, and Multi-Level security models. ACPT creates an XACML policy (reflected by a policy specified in ACPT) by mapping attributes in the policy to their corresponding attributes in the XACML templates. We use the Sun's XACML Policy Decision Point (PDP) [2] to evaluate a request against the XACML policy to decide whether the request is permitted or denied.

To ensure the correctness of the policy under test, the policy authors often evaluate test requests and inspect whether the evaluated decisions (e.g., Permit or Deny) of the requests are correct. In practice, inspecting all possible test inputs is not trivial due to a large number of possible test requests. ACPT automatically generates test inputs by analyzing the policy under test based on a given criterion (e.g., achieving high rule coverage). In addition, ACPT keeps track of test evaluation results. For structural coverage, test evaluation results provide information such as the number of rules that are covered and the number of rules that are not covered.

4 Conclusions

Access control mechanisms play an important role to control access on contents for applications. For example, people increasingly share their files and access their email accounts in web applications, which control user's access privileges based on access control policies. ACPT bridges

[1] http://csrc.nist.gov/groups/SNS/acts/index.html

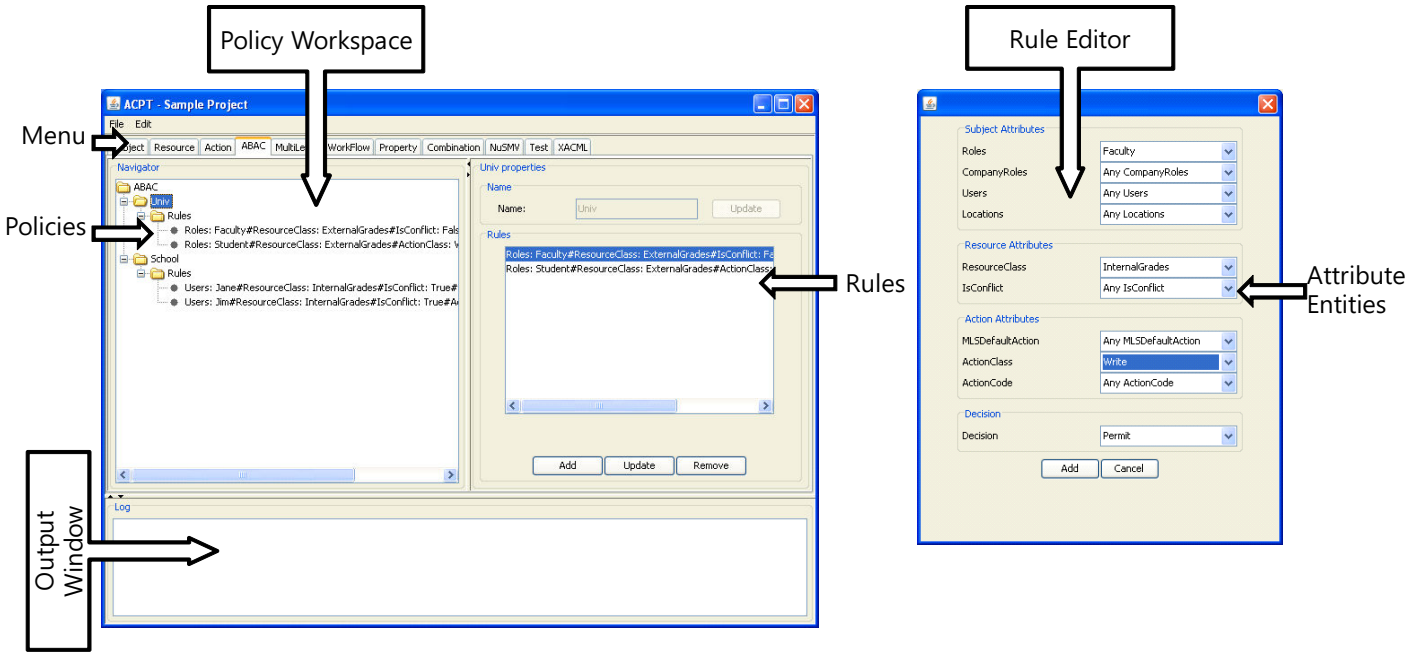

Figure 2. GUI-based ACPT tool to help policy authors specify and combine policies

the gap between policy requirements and policy implementations by generating enforceable policies (in XACML) directly from policy models (reflected by policy requirements). In addition, ACPT supports static and dynamic verification to reduce faults in policies. The current version of ACPT provides only the first-applicable combining algorithm (i.e., resolving conflicting decisions based on an order of policies). We plan to extend ACPT to combine policies using other popular combining algorithms such as deny-overrides, or permit-overrides combining algorithms.

Acknowledgment

This work is supported in part by NSF grant CNS-0716579 and a NIST contract.

References

[1] OASIS eXtensible Access Control Markup Language (XACML). `http://www.oasis-open.org/committees/xacml/`, 2005.

[2] Sun's XACML implementation. `http://sunxacml.sourceforge.net/`, 2005.

[3] D. E. Bell and L. J. Lapadula. Secure computer systems: Mathematical foundations. Technical Report ESD-TR-73-278, Mitre Corporation, 1973.

[4] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. 14th International Conference on Computer-Aided Verification (CAV)*, pages 359–364, 2002.

[5] D. F. Ferraiolo, R. S. Sandhu, S. I. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274, 2001.

[6] V. Hu, R. Kuhn, and T. Xie. Property verification for generic access control models. In *Proc. IEEE/IFIP International Symposium on Trust, Security and Privacy for Pervasive Applications (TSP)*, pages 243–250, 2008.

[7] V. Hu, R. Kuhn, T. Xie, and J. Hwang. Model checking for verification of mandatory access control models and properties. *To appear International Journal of Software Engineering and Knowledge Engineering*, 2010.

[8] E. Martin, J. Hwang, T. Xie, and V. Hu. Assessing quality of policy properties in verification of access control policies. In *Proc. Annual Computer Security Applications Conference (ACSAC)*, pages 163–172, 2008.

[9] E. Martin and T. Xie. Automated test generation for access control policies via change-impact analysis. In *Proc. 3rd International Workshop on Software Engineering for Secure Systems (SESS)*, pages 5–11, 2007.

[10] E. Martin, T. Xie, and T. Yu. Defining and measuring policy coverage in testing access control policies. In *Proc. 8th International Conference on Information and Communications Security (ICICS)*, pages 139–158, 2006.

[11] L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In *Proc. 2nd ACM Workshop on Formal Methods in Security Engineering (FMSE)*, pages 45–55, 2004.