

# Paradigm in Verification of Access Control

(Position Paper)

JeeHyun Hwang<sup>1</sup>, Vincent Hu<sup>2</sup>, Tao Xie<sup>1</sup>

<sup>1</sup>Department of Computer Science, North Carolina State University, Raleigh, USA

<sup>2</sup>Computer Security Division, National Institute of Standards and Technology, Gaithersburg, USA

jhwang4@ncsu.edu, vincent.hu@nist.gov, xie@csc.ncsu.edu

Access control (AC) is one of the most fundamental and widely used requirements for privacy and security. Given a subject's access request on a resource in a system, AC determines whether this request is permitted or denied based on AC policies (ACPs). In a system, an ACP is implemented at various places with different purposes. For example, operating systems adopt AC to regulate which users or groups are permitted to read/write/execute files or folders.

The main objective of AC is to protect resources against unauthorized user access. Faults in AC may result in critical consequences such as unauthorized user access on sensitive resources. However, it is a challenging task to implement and maintain AC correctly for two main reasons. First, AC can be complex, especially, when an ACP includes a large number of resources in a sophisticated structure for various groups and users. Second, policy authors may make mistakes when specifying or combining ACPs.

This position paper introduces our approach to ensure the correctness of AC using verification. More specifically, given a model of an ACP, our approach detects inconsistencies between models, specifications, and expected behaviors of AC. Such inconsistencies represent faults (in the ACP), which we target at detecting before ACP deployment. At a high level, ACPs are policy specifications, which encapsulate the expected AC behaviors from policy authors. An ACP model is a representation of ACP behaviors in a formal language.

An ACP consists of a set of rules, which regulate which subject can take a specific action on a specific resource under which condition. In the context of ACPs, input and output are a request (e.g., can user  $A$  access resource  $B$ ?) and a response (e.g., Permit), respectively. Policy authors may write properties, which can be verified against a given AC model. Properties are different from rules because users create properties based on business practice or user experience. For example, properties can be known security vulnerabilities or a user's security and privacy concerns of interest in AC. We use safety and liveness properties where safety and liveness are characteristics of a given property, denoted by  $p$ .

*Safety property.* Safety denotes that  $p$  is satisfied against an AC model. In other words, there exist no rules in the AC model to violate  $p$ . Therefore, verification of safety properties is to ensure that "something bad" (i.e., faults) does not happen. For example, a conference program committee member should not review her own submitted paper.

*Liveness property.* Liveness denotes that an AC model does "something good" (i.e., desired system behaviors). Therefore, verification of a liveness property is to ensure that a "good

thing" does happen eventually. One example is deadlock free. Deadlock denotes that a system does not make progress forever since a system waits for an action forever due to more than two competing actions, each of which waits for the other to finish.

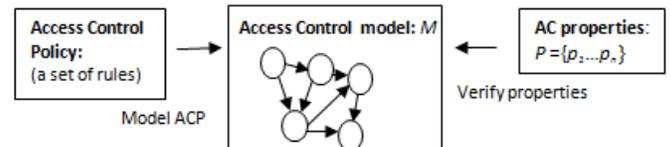


Figure 1. Overview of our approach

Figure 1 illustrates our approach. More specifically, we translate an ACP to its corresponding AC model, which is represented as a finite state machine. In this paper, our approach is applied to mandatory access control (MAC) policies, which regulate user and process access to resources.

Our verification uses black-box and white-box checking techniques. For black-box checking, policy authors specify either properties  $P$ . Given an AC model  $q$ , if there is no violation, we ensure that  $q$  is correct according to  $P$ . Otherwise,  $q$  is not correct and should be fixed to satisfy property  $p' \in P$  that causes violations. In such cases, we use white-box checking to modify  $q$  to satisfy  $P$ . For example, we create another  $p''$  (called a confined property [1]) modified from  $p'$  where  $p''$  is a subset of  $p'$  that is responsible for violations.  $p''$  can be converted to a rule. We add this rule in  $q$  where  $p'$  is satisfied after this addition based on the confined property.

We use NuSMV (<http://nusmv.irst.itc.it/>), a symbolic model checker to model an ACP. NuSMV supports both BDD-based and SAT-based model-checking approaches, and various analyses including Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) model checking for safety and liveness properties, and counterexample generation. Manually writing properties is tedious and error-prone. To address this issue, our approach generates test requests that can be used as properties for testing AC implementations. An AC implementation evaluates test requests and produces responses, which testers need to inspect to determine whether the responses are correct. We have implemented a prototype [1, 2] for the approach.

**Acknowledgment.** This work is supported in part by a NIST grant.

## REFERENCES

- [1] V. Hu, R. Kuhn, T. Xie, and J. Hwang. Model checking for verification of mandatory access control models and properties, in IJSEKE, Volume 21, Issue 1, Pages 103-127, 2011.
- [2] J. Hwang, T. Xie, V. Hu, and M. Altunay, ACPT: A tool for modeling and verifying access control policies. In Proc. POLICY, Demo, Pages 40-43, 2010.