Testing Noninterference, Quickly [*]

(short talk)

Cătălin Hriţcu[1] John Hughes[2] Benjamin C. Pierce[1] Antal Spector-Zabusky[1] Dimitrios Vytiniotis[3]
Arthur Azevedo de Amorim[1] Leonidas Lampropoulos[1]

[1]University of Pennsylvania [2]Chalmers University [3]Microsoft Research

Information-flow control mechanisms are difficult to design and labor intensive to prove correct. To reduce the time wasted on doomed proofs for broken definitions, we advocate modern random testing techniques for finding counterexamples during the design process. We show how to use QuickCheck, a property-based random-testing tool, to guide the design of a simple information-flow abstract machine. We find that both sophisticated strategies for generating well-distributed random programs and readily falsifiable formulations of noninterference properties are critically important. We propose several approaches and evaluate their effectiveness on a collection of injected bugs of varying subtlety. We also present an effective technique for shrinking large counterexamples to minimal, easily comprehensible ones. Taken together, our best methods enable us to quickly and automatically generate simple counterexamples for all these bugs.

Secure information-flow control (IFC) is nearly impossible to achieve by careful design alone. The mechanisms involved are intricate and easy to get wrong: static type systems must impose numerous constraints that interact with other typing rules in subtle ways, while dynamic mechanisms must appropriately propagate taints and raise security exceptions when necessary. This intricacy makes it hard to be confident in the correctness of such mechanisms without detailed proofs; however, carrying out these proofs while designing the mechanisms can be an exercise in frustration, with a great deal of time spent attempting to verify broken definitions! The question we address in this work is: Can we use modern *testing* techniques to discover bugs in IFC enforcement mechanisms quickly and effectively? If so, then we can use testing to catch most errors during the design phase, postponing proof attempts until we are reasonably confident that the design is correct.

To answer this question, we take as a case study the task of extending a simple abstract stack-and-pointer machine to track dynamic information flow and enforce termination-insensitive noninterference. Although our machine is simple, this exercise is both nontrivial and novel. While simpler notions of dynamic *taint tracking* are well studied for both high- and low-level languages, it has only recently been shown that purely dynamic mechanisms are capable of soundly enforcing strong security properties. Moreover, sound dynamic IFC has been studied only in the context of lambda-calculi and While programs; the unstructured control flow of a low-level machine poses additional challenges. (Testing of static IFC mechanisms is left as future work.)

We show how QuickCheck, a popular property-based testing tool, can be used to formulate and test noninterference properties of our abstract machine, quickly find a variety of missing-taint and missing-exception bugs, and incrementally guide the design of a correct version of the machine. One significant challenge is that both the strategy for generating random programs and the precise formulation of the noninterference property have a dramatic impact on the time required to discover bugs; we benchmark several variations of each to identify the most effective choices. In particular, we observe that checking the unwinding conditions of our noninterference property can be much more effective than directly testing the original property.

Our results may be of interest both to researchers in language-based security, who can now add random testing to their tools for debugging subtle enforcement mechanisms; and to the random-testing community, where our techniques for generating and shrinking random programs may be useful for checking other properties of abstract machines. Our primary contributions are: (1) a demonstration of the effectiveness of random testing for discovering counterexamples to noninterference in a low-level information-flow machine; (2) a range of program generation strategies for finding such counterexamples; (3) an empirical comparison of how effective combinations of these strategies and formulations of noninterference are in finding counterexamples; and (4) an effective methodology for shrinking large counterexamples to smaller, more readable ones. Our information-flow abstract machine, while simple, is also novel, and may be a useful artifact for further research.

Keywords random testing, security, design, dynamic information-flow control, noninterference, QuickCheck

[*] To appear in ICFP 2013: http://www.crash-safe.org/node/24