Research on Proof-Carrying Code for Untrusted-Code Security

George Necula Peter Lee
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
{petel,necula}@cs.cmu.edu

1 Introduction

A powerful method of interaction between two software systems is through mobile code. By allowing code to be installed dynamically and then executed, a host system can provide a flexible means of access to its internal resources and services. There are many problems to be solved before such uses of untrusted code can become practical. For this position paper, we will focus on the problem of how to establish guarantees about the intrinsic behavior of untrusted programs. Of particular interest are the following: (1) How can the host system ensure that the untrusted code will not damage it, for example, by corrupting internal data structures? (2) How can the host ensure that the untrusted code will not use too many resources (such as CPU, memory, and so forth) or use them for too long a time period?; and, (3) How can the host make these assurances without undue effort and deleterious effect on overall system performance?

Our position is that the theory of programming languages, including formal semantics, type theory, and applications of logic, are critical to solving the untrusted-code security problem. To illustrate the possibilities of programming language theory, we will briefly describe one rather extreme but promising example, which is *proof-carrying code* (PCC).

2 Proof-Carrying Code

Proof-Carrying Code is a technique by which the host establishes a set of safety rules that guarantee safe behavior of programs, and the code producer creates a formal *safety proof* that proves, for the untrusted code, adherence to the safety rules. Then, the host is able to use a simple and fast *proof validator* to check, with certainty, that the proof is valid and hence the foreign code is safe to execute.

In order to gain some preliminary experience with PCC, we have performed several experiments with user-level code downloaded in an operating system kernel. For example, in [2] we show that, using PCC, we can safely allow the user to download network packet filters written in hand-optimized assembly language, with obvious performance benefits over approaches using interpretation or safe languages. And all this with safety proofs smaller than 1K and one-time proof validation below 2ms. We measured similar costs of using PCC in other experiments [1].

There are many advantages in using PCC for mobile code. First, almost the entire burden of ensuring security is shifted to the code producer. The host, on the other hand, has only to perform a fast, simple, and easy-to-trust proof-checking process. The trustworthiness of the proof-checker is an important advantage over approaches that involve the use of complex compilers or interpreters in the host.

Second, PCC programs are "tamperproof," in the sense that any modification (either accidental or malicious) will either result in a proof that is no longer valid, or one that does not correspond to the enclosed program. In both these cases the program will be rejected.

Third, as opposed to cryptography, no trusted third parties are required because PCC is checking intrinsic properties of the code and not its origin. In this sense, PCC programs are "self-certifying." On the other hand, PCC is completely compatible with other approaches to untrusted-code security.

We have briefly described one approach, proof-carrying code, that illustrates the potential of programming-language theory in this arena, essentially through the exploitation of static checking. It is our position that the theory of programming languages, which we take to include formal semantics, type theory, and logic, provides methods and systems that will be critical to achieving a high level of security in mobile-code applications. While there are still many difficult research problems to be solved, we believe that the past and current results show enough potential to warrant a great deal of further work.

References

[1] NECULA, G. C., AND LEE, P. Proof-carrying code. Technical Report CMU-CS-96-165, Computer Science Department, Carnegie Mellon University, Dec. 1996. Also appeared as FOX memorandum CMU-CS-FOX-96-03.

[2] NECULA, G. C., AND LEE, P. Safe kernel extensions without run-time checking. In *Second Symposium on*

Operating Systems Design and Implementations (Oct. 1996), Usenix, pp. 229–243.