Lifting Reference Monitors from the Kernel

Fred B. Schneider

Computer Science Department
Cornell University
Ithaca, New York 14853
fbs@cs.cornell.edu

Abstract. Much about our computing systems has changed since reference monitors were first introduced, 30 years ago. Reference monitors haven't—at least, until recently—but new forms of execution monitoring are now possible, largely due to research done in the formal methods and programming languages communities. This talk will discuss these new approaches: why they are attractive, what can be done, what has been done, and what problems remain.

In contrast to 1972, operating systems today are too large to be considered trustworthy and security policies are needed not only to protect one user from another but also to protect programs from themselves, since so much of today's software is designed to be extensible. Thus, while the principle of complete-mediation remains sound, it no longer makes sense to locate mechanisms that do execution monitoring in the operating system kernel:

– The integrity of such a reference monitor is difficult to guarantee, by virtue of its location in a large, complex body of code.

– Access to only certain resources could observed, which restricts the vocabulary of abstractions that policies could then govern.

An alternative to deploying software—in the kernel or elsewhere —that intercepts run-time events is to automatically rewrite programs prior to execution, effectively in-lining the reference monitor. The approach, called an "in-line reference monitor" (IRM), has been prototyped for X'86 and JVM as well as for a variety of high-level languages. It has been a clear candidate for commercial deployment, though none has yet occurred. The added run-time checks do not seem to affect performance; and an extremely broad class of policies can be enforced.

With the expressive power of IRMs comes a burden: formulating the policies to enforce. For sure, the translation of informal requirements into formal specifications is not a new challenge, though whether for security we know what should be those informal requirements is certainly a valid question. Security policies also present fundamentally new technical difficulties. The writer of a security policy seemingly must not only understand the semantics of system's interfaces but also must understand what is hidden by those interfaces. Moreover, policy formalizations can have subtle ramifications, not only with regard to run-time efficiency but also with regard to the trusted computing base. The technical problems can be posed; promising directions for solutions can only be suggested.

Finally, the exploration of execution monitoring and program rewriting in policy enforcement is providing a new lens through which security policies can be viewed. Ad hoc arguments about the virtues of one protection mechanism over another is starting to be replaced by mathematical arguments about limits and by a rigorously defined hierarchy of enforcement mechanisms. Some very recent results (done with Kevin Hamlen Greg Morrisett) concerning the power of program-rewriting for policy enforcement will be discussed.

References

1. Ulfar Erlingsson and Fred B. Schneider. SASI enforcement of security policies: A retrospective. Proceedings of the *New Security Paradigms Workshop* (Caledon Hills, Ontario, Canada, September 1999), Association for Computing Machinery, 87–95. `http://www.cs.cornell.edu/fbs/publications/sasiNSPW.ps`

2. Ulfar Erlingsson and Fred B. Schneider. IRM enforcement of Java stack inspection. Proceedings 2000 *IEEE Symposium on Security and Privacy* (Oakland, California, May 2000), IEEE Computer Society, Los Alamitos, California, 246–255. `http://www.cs.cornell.edu/fbs/publications/sasiOakland.ps`

3. Fred B. Schneider Enforceable security policies. *ACM Transactions on Information and System Security* 3, 1 (February 2000), 30–50.