Formal, Informal, and Null Methods

Position statement on formal methods and security
for NRC workshop on trustworthiness

Martín Abadi
Digital Equipment Corporation
Systems Research Center
ma@pa.dec.com

January 23, 1997

There is a considerable body of work on specification and verification methods, and on their application to security. (See for example Gasser's book [Gas88] for a review of the state of the art in 1988.) Although it now seems plausible that one can build useful, formally-verified secure systems, this is seldom done, and we may guess that it will not be done to any large extent in either academia or industry in the near future.

It often seems that even informal methods are rarely applied for guaranteeing or evaluating the security of common systems and components. Even security gear often lacks specification, formal or informal. For example, one can find documents that describe the workings of the SSL protocol [FKK96] and others that describe some of its shortcomings and suggest improvements [WS96], but to my knowledge none that details the guarantees that SSL provides. Not surprisingly, then, the security properties of common applications are generally not clear.

Two obstacles to both formal and informal methods are the complexity of a system or component, and the ambiguity of its descriptions. These two obstacles render analysis difficult. They also make any trust assumptions seem rather implausible and ill-defined. More attention should be paid to keeping the trusted computing base (TCB) as small and simple as possible. (Do you know what is in your TCB?)

- For example, the Java bytecode verifier checks a program by constructing an invariant of a particular kind for the program [LY96]. It would be desirable to break the verifier into two parts: (1) a part that guesses or constructs an invariant candidate, by whatever means; (2) a part that checks that this candidate is in fact an invariant. Only the latter part would be in the TCB; it would be considerably smaller and simpler than the whole. (Raymie Stata and I have started to explore this idea.)

- Other examples are in the area of security protocols. These protocols often include many options and variants, which may be regarded as important features but obscure the definition of a TCB and create opportunities for error.

These remarks suggest the following applications for formal methods:

- Formal methods can be used to analyze particular system components. These might be only simplified or abstract versions of the components actually deployed. With this caveat, we should be able to develop rigorous approaches for specifying and verifying the security properties of network protocols and interpreters for mobile code, for example.

- Formal methods—and informal methods—can serve to organize the search for security flaws. This search can take place even with imperfect methods, and with inaccurate descriptions of systems. (The use of logics for finding mistakes in authentication protocols is the example with which I am most familiar.) Dijkstra once wrote [DDH72]:

  > Program testing can be used to show the presence of bugs,
  > but never to show their absence!

  In some cases, formal methods suffer from this limitation as well, but they are still useful.

- Even in situations when specification and verification techniques are not applied, for whatever reason, it is useful to ask whether they could be. A system that lends itself to rigorous treatment seems more likely to be secure than one that does not. Formal and informal methods can guide a system design in which they are not used explicitly, when the designer tries to make the system clean enough that those methods could in principle be used (see for example [AN96, Kra96]). Isolating a small and simple TCB is a good first step.

These applications are under way. They are more modest than providing formal end-to-end guarantees, but they seem worthwhile.

Acknowledgments

Leslie Lamport and Raymie Stata commented on drafts of this note.

References

[AN96] Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.

[DDH72] O.-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare. *Structured Programming.* Academic Press, 1972.

[FKK96] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL protocol: Version 3.0. Available at `http://home.netscape.com/eng/ssl3/ssl-toc.html`, March 1996.

[Gas88] Morrie Gasser. *Building a Secure Computer System.* Van Nostrand Reinhold Company Inc., New York, 1988.

[Kra96] Hugo Krawczyk. SKEME: A versatile secure key exchange mechanism for internet. In *Proceedings of the Internet Society Symposium on Network and Distributed Systems Security*, February 1996. Available at `http://bilbo.isu.edu/sndss/sndss96.html`.

[LY96] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification.* Addison-Wesley, 1996.

[WS96] David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings of the Second USENIX Workshop on Electronic Commerce Proceedings*, pages 29–40, November 1996. A revised version is available at `http://www.cs.berkeley.edu/~daw/me.html`.