

# Using a Declarative Language to Build an Experimental Analysis Tool

Catherine Meadows

Naval Research Laboratory, Code 5543, Washington, DC 20375, USA

**Abstract.** In this paper we give a brief summary of our experience in using a declarative language, Prolog, to develop an experimental formal analysis tool, the NRL Protocol Analyzer, which was updated and modified over the years to incorporate new theories and techniques. We discuss the benefits of using such an approach, and also some of the downsides...

The application of formal methods to cryptographic protocol analysis is now an established field. The types of assumptions that need to be made, and the techniques for automatically proving properties of cryptographic protocols, are well known, at least for a certain subclass of problems. However, when we began working on this problem in the late 80's, this was definitely not the case. Only a few tools, such as Millen's Interrogator [6], and a few algorithms, such as those devised by Dolev, Even, and Karp, [1], existed. Although these could be used as a basis for my research, it was unclear where we would ultimately wind up. Thus, we needed to ability to build a tool that could be rapidly reconfigured to incorporate new techniques and models, and that updated over (possibly) over a long period of time.

The earliest version of the Analyzer [2] consisted of a simply of a state generation tool. The user specified a state, and the Analyzer would use equational unification to generate all states that immediately preceded it. The search strategy was largely guided by the user, and was input by hand. This was very tedious, but allowed me to collect data that could be used to build the next version of the Analyzer.

The second version of the Analyzer allowed some automatic guidance of the search. In particular, it was possible to write and then use the Analyzer to prove inductive lemmas that put conditions on infinite classes of states. The search could then automatically avoid states that were unreachable according to the lemmas. However, it was up to the user to figure out what lemmas needed to be proved.

As we continued to use the Analyzer, it was found that many of the lemmas obeyed certain canonical forms. This made it easier to automate the generation as well as the proof of lemmas. Thus, the current version of the Analyzer, although it still requires some input from the user, generates most lemmas automatically [3]. It also proves a much greater variety of lemmas than it did before, and supports a higher-level and more flexible specification language than earlier versions. The most up-to-date description of the Analyzer is given in [4].

Throughout this process, we found the use of a declarative language such as Prolog a great boon. The ease of writing and reading such programs made it easier to update the Analyzer incrementally, over long periods of time, and even with long periods of inactivity. On the other hand, we found that many of the special tricks that can be used to improve Prolog's performance worked against this, and as a result we intended to avoid this after a while. Because of this, and because of other design decisions that we made in order to make this incremental modification easier (the use of generate-and-test as a theorem proving strategy, for example), there are a number of cryptographic protocol analysis tools designed with more specialized applications in mind that outperform the Analyzer. However, we believe that the Analyzer is still one of the most flexible tools around, and it has been used in the analysis of more complex protocols (see for example [4, 5]) than almost any other tool. Moreover, many of the newer tools make use of techniques that were pioneered by the NRL Protocol Analyzer.

In summary, we would definitely recommend declarative programming as a rapid prototyping tool, especially one which is expected to undergo major changes as a project progresses. On the downside, the very techniques that would improve such a program's performance appear to mitigate against its usefulness for rapid prototyping by making the program more opaque. However, this is a tradeoff that one might expect.

## References

1. D. Dolev, S. Even, and R. Karp. On the Security of Ping-Pong Protocols. *Information and Control*, pages 57–68, 1982.
2. C. Meadows. A system for the specification and verification of key management protocols. In *Proceedings of the 1991 IEEE Symposium in Research in Security and Privacy*. IEEE Computer Society Press, May 1991.
3. Catherine Meadows. Language generation and verification in the NRL protocol analyzer. In *Proceedings of the 9th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1996.
4. Catherine Meadows. Analysis of the Internet Key Exchange protocol using the NRL Protocol Analyzer. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1999.
5. Catherine Meadows. Experiences in the formal analyzer of the GDOI protocol. In *Proceedings of Verlässliche IT-Systeme 2001 - Sicherheit in komplexen IT-Infrastrukturen*, 2001.
6. J. K. Millen. The interrogator: A tool for cryptographic protocol security. In *Proc. 1984 Symp. Security and Privacy*, pages 134–141. IEEE Computer Society Press, 1984.