A Symbiotic Relationship Between Formal Methods and Security

Jeannette M. Wing

December 1998

CMU-CS-98-188

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This paper will appear in an edited volume resulting from the series of two workshops held in 1998, sponsored by the ONR and NSF, entitled *Workshops on Computer Security, Fault Tolerance, and Software Assurance: From Needs to Solution.*

Abstract

Security played a significant role in the development of formal methods in the 70s and early 80s. Have the tables turned? Are formal methods now ready to play a significant role in the development of more secure systems? While not a panacea, the answer is yes, formal methods can and should play such a role. In this paper I first review the limits of formal methods. Then after a brief historical excursion, I summarize some recent results on how model checking and theorem proving tools revealed new and known flaws in authentication protocols. Looking to the future I discuss the challenges and opportunities for formal methods in analyzing the security of systems, above and beyond the protocol level.

This research is sponsored in part by the Defense Advanced Research Projects Agency and the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, F33615-93-1-1330, and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-97-2-0031 and in part by the National Science Foundation under Grant No. CCR-9523972. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency Rome Laboratory or the U.S. Government.

Keywords: formal methods, specification, verification, model checking, theorem proving, security, authentication, electronic commerce.

1 Introduction

Formal methods owes much to the security community. In the United States, the National Security Agency was a major source of funding in the 70s and early 80s for formal methods research and development. Results included the development of formal security models, tools for reasoning about security, and applications of these tools to proving systems secure. Security provided a challenging research application for the formal methods community.

The burgeoning use of the Internet brings security now to the attention of the masses. The success of Amazon.com Inc. and the like suggests that people trust sending their credit card numbers over the wire. People are, however, justifiably hesitant to send their mother's maiden name to an electronic banking system when asked for it on-line. Simultaneous with the increasing desire to perform transactions over the Internet is the maturation of key (pun intended) technology. For example, public-key encryption is no longer an academic exercise. We have fortuitously come to a balancing point between a *need* and a *solution*: People are more willing today to pay the price for increased security.

Are formal methods part of this maturing technology? With respect to security, what problems can formal methods help to solve? What problems will formal methods never help to solve?

In an attempt to answer these questions, the remainder of this paper first delimits the bounds of formal methods, and then reviews briefly the historical role of the security community in the early development of formal methods[1]; summarizes recent results in the use of formal methods tools for reasoning about security protocols; and discusses directions for the future role of formal methods applied to security.

2 The Limits of Formal Methods

2.1 What Formal Methods Cannot Do

It is axiomatic that systems will never be made 100% secure. Formal methods will not break that axiom. Moreover, substitute the word "proven" for "made" in the previous sentence, and we have a corollary. It would be foolish for anyone in formal methods to stand up and say "I can prove your system is 100% secure." Why?

Systems do not run in isolation; they operate in some *environment*. The formal specification of a system must always include the assumptions one makes about the system's environment. A proof of correctness is valid only when these assumptions hold. So, if any assumption is violated, all bets are off. Indeed, to break into a system, clever intruders find out how to violate these assumptions.

Moreover, even if one were careful to state these assumptions explicitly, which is often impractical, there would inevitably be conditions missed. And, even if one were comprehensive about stating these assumptions, which is always impracticable, a system could very well be deployed in an environment for which it was not originally designed, perhaps for convenience or lack of an alternative.

These remarks are not particular to security; they apply in general to the process of verifying a system meets its specification.

It is also axiomatic that the more secure one wants a system to be the more one must be willing to pay. This truth implies that security itself is not an either/or property. If you add more locks to your door, you make it harder for someone to break into your house. More locks means more secure.

Requiring passwords, using biometrics, encrypting data, signing messages, setting file access control bits, using firewalls, sandboxing, managing a private network, running secure coprocessors, or hiring a guard are just examples of the collection of "locks" one can use to make a system more secure. In practice, security is a combination of many properties, including data integrity, privacy, entity identification, message authentication, and accountability. Some of these properties are also not either/or, but measured in terms of degree. Also, depending on the application or the user's end goal, some properties will be more important than others. One property may even conflict with another, or achieving one may make it harder to achieve another. For example, both anonymity and accountability are desired properties of electronic payment systems.

[1] This paper does not do justice to either the history of security research or the history of formal methods; rather it focuses on the intersection of the two and even so, only sketchily.

Achieving some security properties may conflict with achieving other system goals. Requiring passwords for access conflicts with convenience; using a public-key encryption scheme conflicts with performance.

2.2 What Formal Methods Can Do

Formal methods can help us

- Delimit the system's boundary: the interface between the system and its environment.
- Characterize a system's behavior more precisely. Most current methods focus on functional behavior only (What is the correct answer?) but some can handle real-time behavior too (Is the correct answer delivered on time?).
- Define the system's desired properties precisely.
- Prove a system meets its specification. Some methods, by providing counterexamples like intruder scenarios, can also tell us under what circumstances a system does not meet its specification.

These capabilities of formal methods help the practitioner in two ways:

- Through *specification*, focusing the system designer's attention. What is the interface? What are one's assumptions about the system's environment? What is the system supposed to do under this condition or that condition? What happens if that condition is not met? What are the system's invariant properties?
- Through *verification*, providing additional assurance. Relying on a proof that a system meets its security goals is better than relying on a gut feeling.

It should be emphasized that any proof of correctness is relative to both the formal specification of the system and the formal specification of the desired properties. A system "proven correct" with respect to an "incorrect" specification leaves us with no assurance about the system at all. Finally, there will always be a gap between what is in a person's head and the first codification of the system or desired property. No amount of formalization will eliminate this gap.

3 Past

The early formal methods research funded directly by the National Security Agency, or indirectly through the National Computer Security Center (NCSC), centered on *proving systems secure*.

Addressing the question of what *secure* means, researchers defined models and policies to express Lampson-style [26] access rights of subjects to objects. This work led to the Bell-LaPadula No-Read-Up and No-Write-Down secrecy model [3], the Biba No-Read-Down and No-Write-Up integrity model [5], and the less formal Clark-Wilson integrity model based on a set of nine rules of practice [11]. The Bell-LaPadula model in particular gained notoriety when McLean introduced System Z, which satisfies Bell-LaPadula properties but is clearly insecure [29].

The *systems* of interest to prove secure were operating systems. More specifically, kernels. Given that one cannot prove an entire system secure, better to try to prove a small piece of it. Trust the kernel and nothing else. This approach emphasized the importance of the reference monitor concept: the functionality of the operating system that mediates access by subjects to objects. For example, a user-level process should not have access to the kernel-level stack.

The formal methods community played a fundamental role in fleshing out what *proving* means. The process of proving entails three parts (not necessarily done in this order): First, one must state the property of the system to prove, as expressed explicitly in a *formal specification*. In the security context, this specification might simply be a list of properties such as the so-called *-property (No-Write-Down) of the Bell-LaPadula model. Second, one must model the system so that one can formally prove the property. This mathematical model might be a semantic structure like a state machine or a syntactic structure like a logical expression. Third, the proof. Typically, the proof might rely on induction over traces of the state machine model or it might rely on deduction to show that an implication holds ($SystemModel \Rightarrow SystemProperty$). The proof might be discovered automatically by the machine or require interactive guidance from the human user. *Formal verification* is the process of proving, by hand or machine, that the model of the system satisfies the formal specification. In practice, most theorem proving tools are more like proof checkers; they differ in the amount of human intervention needed to check the proof.

One of the most influential documents of the time, produced by the NCSC, is the U.S. Trusted Computer System Evaluation Criteria, better known as "The Orange Book" [10]. Amoroso [2] summarizes its goals:

- To provide a standard metric for the NCSC to compare the security of different computer systems.
- To guide computer system vendors in the design and development of secure systems.
- To provide a means for specifying security requirements in Government contracts.

In particular for a system to be certified A.1 according to the Orange Book means that one formally specify the system's security requirements, formally model the system, and formally prove that the model meets its specification.

In this context, the major results of the early 80s by the formal methods community in the United States, in particular those funded heavily by the security community, were in the development of theorem proving tools. To get one's system certified A.1, one would use one of these tools to produce the proof.

In fact, these tools were general-purpose theorem provers; they were applied to examples from the security arena, but were applicable in general to all kinds of systems. By 1986, in Kemmerer's landmark Verification Assessment Study [23], four tools were the most mature, known, or used:

- Affirm, developed at the University of Southern California's Information Sciences Institute, best known for its support for reasoning about equational specifications, in particular through its implementation of the Knuth-Bendix completion procedure.
- The Formal Development Methodology (FDM) System, developed at System Development Corporation's Santa Monica Research Center, best known for its support for a non-deterministic state machine model and the Ina Jo specification language.
- Gypsy, developed at the Institute for Computing Science at the University of Texas at Austin, best known for its support for program verification of a subset of Pascal, including a verification condition generator.
- (Enhanced) Hierarchical Development Methodology (HDM), developed at Stanford Research International's Computer Science Laboratory, best known for its SPECIAL specification language and its collection of decision procedures for propositional logic as the heart of its theorem prover.

The general-purpose Boyer-Moore theorem prover [8], also developed during the same time period, was representative of the state of the art in automated theorem proving tools and was applied to many examples, notably the "CLInc Stack" [4].

A few researchers developed tools specific to reasoning about security. The two best known examples are the Interrogator [33] and the NRL Protocol Analyzer [31]. With both of these tools, one specifies an insecure state and the tool searches backwards to determine whether that state is reachable. The Interrogator is based on Prolog, does an exhaustive search (and hence is fully automatic), and has a built-in notion of encryption. The NRL Protocol Analyzer's search is less automatic. It is based on Dolev and Yao's pioneering work on an algebraic term rewriting model for two-party cryptographic protocols [14]. Meadows used the NRL Protocol Analyzer to discover previously unknown flaws in the Simmons Selective Broadcast Protocol and the Burns-Mitchell Resource Sharing Protocol. Kemmerer, Meadows, and Millen's [22] paper summarizes the strengths and weaknesses of FDM, the NRL Protocol Analyzer, and the Interrogator, using the Tatebayeshi-Matsuzaki-Newman (TMN) protocol as the common running example.

In 1990, Burrows, Abadi, and Needham published their work on a Logic of Authentication (aka the BAN Logic) [9], a formal logic designed specifically to reason about authentication protocols. The logic's main construct allows one to reason in terms of *belief*, and in particular the beliefs a principal accumulates during the run of a protocol. One kind of belief a principal might acquire is about the freshness of messages, e.g., through the freshness of message components such as nonces. The lack of proof that a message is fresh suggests a possible replay attack—a well-known vulnerability of the original Needham-Schroeder symmetric-key protocol. The BAN work attracted both praise and criticism. It inspired some to define their own belief logics, e.g., the GNY (Gong, Needham, and Yahalom) logic [16], the SVO (Syverson and van Oorschot) logic [39], and AUTLOG [24]. Unlike the aforementioned general-purpose tools, BAN and its derivatives focused on only authentication protocols, and except for AUTLOG [24], lack tool support. Despite these limitations, the BAN Logic's influence was positive overall: it demonstrates that formal logics have a role in revealing flaws in an important class of security protocols.

One of the criticisms against the original BAN paper was the absence of a semantic model. Solving this problem led to the definition of various state-machine semantic models for authentication protocols, including Abadi and Tuttle's [1], Woo and Lam's [40], and Heintze and Tygar's [18]. Woo and Lam also introduced the need to check not just for secrecy, but also *correspondence*, a property that assures that the authenticating principal is indeed "talking" to the intended authenticated principal.

In their comprehensive 1993 survey report, Rubin and Honeyman [38] use Meadows's four-type classification scheme [30] to categorize twenty-seven different formal approaches to the analysis of authentication protocols. The four types are (1) using general-purpose specification languages and tools, e.g., Ina Jo; (2) using special-purpose rule-based tools, e.g., the Interrogator, to help the protocol designer; (3) using belief logics, e.g., BAN; and (4) using special-purpose algebraic-based tools, e.g., the NRL Protocol Analyzer.

Two international meetings caused the formal methods and security communities to cross paths: the FM'89 [13] and FM'91 workshops, sponsored by the governments of the United States, Canada, and the United Kingdom (in particular by the National Security Agency and its Canadian and UK counterparts). The focus in FM'89 was on the role of formal methods for trustworthy computer systems. Here, trustworthy meant not just security but also safety-critical. The main outcome was the recognition of two different styles of formal methods:

- The UK and European style: The focus was on specification, on the system's high-level design, and on paper-and-pencil analysis.
- The US and Canadian style: The focus was on verification, from the system's high-level design through its code-level implementation down to its bit-level representation in hardware (the "CLInc Stack" approach), and on machine-assisted analysis.

Debate over which style was better subsided by FM'91 where instead there was consensus to embrace all methods, to acknowledge that tools are necessary, and to direct the community's effort to producing more convincing case studies. Another outcome of FM'91 (for the US at least) was the move of mainstream formal methods research out from under the shadow of the security agencies, witnessed by the absence of subsequent workshops sponsored by those three agencies.

4 Present

Since the early 90s the formal methods community has experienced an explosion of new developments: new methods, new tools, and countless large-scaled projects and non-trivial case studies.[2] Clarke and Wing capture the state of the art in their 1996 *ACM Computing Surveys* paper detailing the progress of three threads in the development of formal methods: model checking, theorem proving, and software specification. Model checking, in particular, is a proven success for hardware verification; companies such as Intel are establishing their own hardware verification groups, building their own verification systems, and hiring people trained in formal methods.

In 1996 another convergence of the two communities occurred. Lowe [27] used Roscoe's model checker, FDR, to exhibit a flaw in the Needham-Schroeder public-key authentication protocol, first published eighteen years earlier. Lowe actually discovered the flaw on his own, but used the tool to check both the flawed and

[2] As of 22 December 1998, the Oxford Formal Methods Web page `http://www.comlab.ox.ac.uk/archive/formal-methods/` lists 76 different formal methods notations and tools, and 648 "formal methodists".

the amended protocols. This paper started a flurry of activity in (1) the use of other model checkers to show the same thing, (2) the use of other tools and techniques to show the same thing, and (3) the application of all these tools to other authentication protocols and to simplified electronic commerce protocols. Here is a sampling:

- Model checking approaches
  - Mitchell, Mitchell and Stern [34] use Dill's Murϕ model checker (originally designed for hardware verification) on the Needham-Schroeder public-key, TMN, and Kerberos protocols. Current efforts at Stanford are aimed at specifying and verifying SSL 3.0.
  - Marrero, Clarke, and Jha [28] describe a special-purpose model checker, Brutus, which has a built-in model of an intruder. It has direct support for checking correspondence properties. Marrero used it to verify fifteen classic authentication protocols and is currently applying it to examine electronic commerce protocols, including 1KP, 2KP, and Netbill.
  - Heintze, Tygar, Wing, and Wong [19] used FDR to check atomicity properties of Netbill and a simple digital cash protocol.
- Theorem proving approaches
  - Paulson used a general-purpose theorem prover, Isabelle, to show how to use induction to reason about five classic authentication protocols and their variations [37].
  - Dutertre and Schneider embed CSP in the general-purpose theorem prover PVS and used the embedding to verify authentication protocols [15].
  - Bolignano used the general-purpose theorem prover, Coq, to analyze the Needham-Schroeder public-key protocol [6] and is investigating its use for analyzing electronic commerce standards like the Secure Electronic Transaction (SET) protocol [7].
- Hybrid approaches
  - Meadows has recently made improvements to the NRL Protocol Analyzer so that it should best be viewed as special-purpose tool that embodies both model checking (e.g., brute force search) and theorem proving (e.g., lemma generation) functionality. She is currently applying it to analyze the Internet Key Exchange protocol [32] and the SET protocol.
  - Kindred and Wing [25] invented a new technique, called *theory generation*, which automatically generates a finite representation of a protocol's theory, as represented in terms of BAN-like formulae. Kindred has applied this approach to the classic set of authentication protocols and variants of the NetBill electronic payment protocol.

The common theme in almost all of the above recent work is the demonstration of how formal methods can be applied to authentication protocols, particularly Needham-Schroeder's public-key protocol. Indeed at the September 1997 DIMACS Workshop of Cryptographic Protocol Design and Verification many of the speakers presented how their method reveals the flaw discovered by Lowe.

In June 1998, Heintze and Wing [20] ran the well-attended Workshop on Formal Methods and Security Protocols and there are numerous similar workshops scheduled for 1999 worldwide. The interest in the intersection of these two communities remains unabated.

The motivation from the formal methods community is clear: security still remains a challenge. The motivation from the security community is strong too. More and more people place their trust in computing systems today for doing everything from casual shopping to medical recordkeeping; and more and more systems are built out of commercial-off-the-shelf components. It is no longer just the government, the military, or the universities who are the purchasers, users, or conveyors of large, complex computing systems. Thus, system designers and implementers are more willing to pay the price for increasing the assurance that their systems are secure. Formal methods can provide such increased assurance.

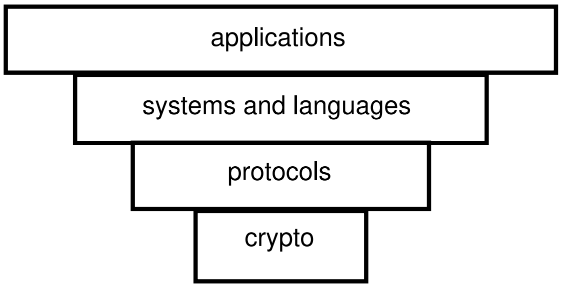

Figure 1: System Layers

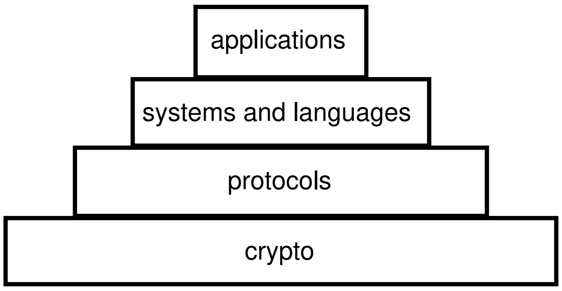

Figure 2: Security Guarantees

5 Future

5.1 The Practice of Building Secure Systems

Figure 1 depicts how we build secure systems. We first and foremost rely on a solid cryptographic base. Out of these primitives for encryption, decryption, signatures, hashing, etc., we define protocols such as for authentication and key-exchange, and we rely on standard reliable network protocols like TCP/IP. We rely on these protocols to build security services, some of which we use on a daily basis; for example, I invoke Kerberos's `kinit` every morning to access my files stored remotely. All of these protocols and system services are implemented in general-purpose programming languages such as C or Java. Finally, above the systems and languages level, we have applications which are what the public sees and uses. These applications include on-line shopping, banking, bill payment, and tax forms submission, all of which should provide some guarantees of privacy and protection to the user.

Ironically, the "strength" of what we can guarantee is inversely proportional to the "size" of the layer (Figure 2). There are fundamental and deep results in cryptography that tell us precisely what we can guarantee, what we cannot, and what is still an open question (e.g., the equivalence of the RSA problem and factoring). At the protocol level we have a handful of formal methods, even mechanized ones, that let us provide some guarantees about authentication protocols. At the system/protocol layer, we have protocols like SSL and SHTTP, which provide minimal encryption and authentication functionality for setting up secure channels. At the systems and languages layer, commercial technology such as Authenticode, Active X, Java, and JavaScript provide varying degrees of security, but are subject to widely publicized attacks such as denial of service and spoofing. At the application layer, in terms of security guarantees, we don't have very much at all. What then are the challenges for the future?

5.2 Challenges and Opportunities for Formal Methods Researchers

Below I use N to indicate near-term research; L, long-term.

First, I focus on the *protocol level (N)*. If a protocol has a design flaw, it does not matter if the implementation is correct. The protocol is vulnerable to attack. We know that protocols are notoriously difficult to get right and the more complex a protocol, the harder it is to understand. Good progress has been made in proving or disproving that individual protocols meet certain properties. Progress has also been made in using different mechanized methods like model checking and theorem proving to help with the proof process. This work should continue, as should the more general work of building and integrating formal methods tools and applying them to larger and larger systems.

With respect to security, however, I would like to move the formal methods community to look beyond the protocol level.

- Multiple protocols
  - *Protocol composition (L).* We expect in practice that more and more people will be designing and deploying their own protocols by using existing protocols as building blocks. We should design new and integrated protocols with compositionality in mind. By composition, I mean running protocols interleaved (concurrently), sequentially (back-to-back), and layered (as subprotocols of each other).

    In general we need all composition mechanisms to work together. For example, authentication requires using encryption as a subprotocol; digital cash requires a blind signature scheme; and the SET standard relies on a public-key infrastructure.

    Some newly discovered attacks arise because of multiple, interleaved runs of the same or different protocols. For example, the correctness of the separate runs of two protocols does not imply the correctness of a system where an intruder can participate in both runs at the same time. We need to look at multiple, simultaneous, and possibly interleaved runs of the same *and different* protocols.
- Systems and language level
  - *Program analysis tools (N).* Even if the protocol design is correct, the implementation could be flawed. Many of the Computer Emergency Response Team (CERT) advisories can be traced to a buffer overflow problem, e.g., resulting from using the unsafe `strcpy` C library string routine. From the CERT report the following are given as examples of "weaknesses in how protocols and software are implemented" [12]:
    * race conditions in file access
    * non-existent checking of data content and size
    * non-existent checking for success or failure
    * inability to adapt to resource exhaustion
    * incomplete checking of operating environment
    * inappropriate use of system calls
    * re-use of software modules for purposes other than their intended ones

    We should develop program analysis tools that will help us detect these kinds of weaknesses in the software implementations of protocols and systems. Work such as applying program slicing to do a vulnerability analysis of TCP/IP [17] is a step in this direction.
  - *Certified library components (N).* As more and more systems are built out of off-the-shelf components, it pays to have additional assurance that the components have been certified to meet some degree of security. Since these building blocks will be used multiple times and in different contexts, the cost of certifying them could be amortized over the overall cost of developing the systems in which they are embedded.

    These library components might be in hardware too. A milestone was recently achieved when the IBM 4758 PCI Cryptographic Coprocessor, which provides a tamper-sensing and tamper-responding environment to enable secure electronic business transactions, earned the highest certification for commercial security awarded by the US Government. It is the first product to ever meet the Federal Information Processing Standard 140-1 Level 4 [21].

  - *Benchmark suite (N).* It would be a great service to the community to have a benchmark suite of intruder scenarios that can be used as a testbed for designing, analyzing, and debugging existing and future protocols and systems. These benchmarks can be used as test cases against source code as well as test cases for formal methods analysis tools.
  - *Programming language design (L).* Language designers should investigate incorporating into the programming language more "type" information that would permit some security guarantees to be statically enforced or dynamically checked. Myers's work on JFlow for statically analyzing flow control [35] is one approach based on additional annotations to Java programs. Another approach by Necula and Lee is to use proof-carrying code, allowing clients to execute untrusted remote code safely [36].

- Applications level
  - *Case studies (N).* We should do large-scale examples to show the applicability and scalability of our formal methods. Current studies of the SSL, IKE, and SET standards are good examples. Moreover, formalizing a standard has a higher payoff than formalizing a single system. This work takes time, effort, and people power; it also can be extremely tedious. Thus, the case studies need to be chosen wisely; when completed, the targets of these studies must still be relevant.

- Horizontal and vertical slices *(N, L)*
  - *Global properties.* At each level and above all levels, we need to reconsider global security properties of systems. We need to understand which properties can be decomposed such that local proofs imply they hold globally, and which do not. We may need new proof techniques to handle global properties that cannot be decomposed.
  - *Intruder models.* We need to build a suite of intruder models, each class representing different intruder capabilities (passive, active, etc.). Some intruder models may be protocol-specific; others, more generic. As the taxonomy of protocols of interest expands, e.g., to include electronic payment protocols and secure auction protocols, so must our models of intruders.
  - *Crossing abstraction boundaries.* We need to look at slices that cut across the four levels depicted in Figure 1, tracing a property or function from an application at the top all the way down to how it is implemented at the cryptographic level. We should pay particular attention to when we are crossing boundaries between the levels since interfaces often do not match at the boundaries.

Even beyond these layers and slices, we need to take a more *holistic* view of a system. Again, from the CERT report:

> Vulnerabilities in the category of system and network configurations are not caused by problems inherent in protocols or software programs. Rather, the vulnerabilities are a result of the way these components are set up and used. Products may be delivered with default settings that intruders can exploit. System administrators and users may neglect to change the default settings, or they may simply set up their system to operate in a way that leaves the network vulnerable.
>
> An example of a faulty configuration that has been exploited is anonymous File Transfer Protocol (FTP) service. Secure configuration guidelines for this service stress the need to ensure that the password file, archive tree, and ancillary software are separate from the rest of the operating system, and that the operating system cannot be reached from this staging area. When sites misconfigure their anonymous FTP archives, unauthorized users can get authentication information and use it to compromise the system.

Thus, we see that it is not enough to look at just the system or even the system and its intended operating environment. Formal methods need to be integrated with other methods that can address issues—some of which are beyond the scope of formalization—raised by examples like the one above. These analyses include risk analysis, hazard analysis, fault analysis, and intrusion detection analysis. Formal methods also need to be better integrated into the entire software development lifecycle such as during requirements analysis, testing, and simulation.

Finally, we must introduce the human factor, which in principle is part of the system's environment. Human factors cannot be neglected. Research in modeling human behavior, human-computer interaction, and management of processes and organizations can all complement the more formal nature of research of formal methdos.

Acknowledgments

Opinions expressed in this paper are my own, not of any of my sponsors or even necessarily of any of my formal methods or security colleagues.

References

[1] M. Abadi and M. Tuttle. A semantics for a logic of authentication. In *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, pages 201–216, August 1991.

[2] E. Amoroso. *Fundamentals of Computer Security Technology.* AT&T Bell Laboratories, 1994.

[3] D. Bell and L. LaPadula. Secure computer systems: Mathematical foundations. Technical Report ESD-TR-73-278, The MITRE Corporation, Bedford, MA, 1973.

[4] W.R. Bevier, W.A. Hunt, Jr., J S. Moore, and W.D. Young. An approach to systems verification. *Journal of Automated Reasoning*, 5:411–428, 1989. See also three other articles in the same issue by Young, Moore, and Hunt.

[5] K. Biba. Integrity considerations for secure computer systems. Technical Report MTR-3153, The MITRE Corporation, Bedford, MA, 1975.

[6] D. Bolignano. An approach to the formal verification of cryptographic protocols. In *Proceedings of the Third ACM Conference on Computer and Communications Security*, pages 106–118. ACM Press, 1996.

[7] D. Bolignano. Towards the formal verification of electronic commerce protocols. In *Proceedings of the Tenth IEEE Computer Security Foundations Workshop*, June 1997.

[8] R. Boyer and J. Moore. *A Computational Logic*. ACM monograph series. Academic Press, New York, 1979.

[9] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[10] National Computer Security Center. Department of Defense Trusted Computer Security Evaluation Criteria. Technical Report DoD 5200.28-STD, NCSC, 1985.

[11] D. Clark and D. Wilson. A comparison of commercial and military computer security policies. In *IEEE Symposium on Security and Privacy*, 1987.

[12] Computer Emergency Response Team Coordination Center Staff. Security of the internet. In *Encyclopedia of Telecommunications*, 1997.

[13] D. Craigen and K. Summerskill. *Formal Methods for Trustworthy Computer Systems (FM89)*. Springer-Verlag, 1990. Workshops in Computing Series.

[14] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, March 1989.

[15] B. Dutertre and S. Schneider. Using a PVS embedding of CSP to verify authentication protocols. In *Theorem Proving in Higher Order Logics*, pages 121–136, August 1997. LNCS 1275.

[16] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 234–248, May 1990.

[17] B. Guha and B. Mukherjee. Network security via reverse engineering of TCP code: Vulnerability analysis and proposed solutions. In *Proc. IEEE Infocom'96*, pages 603–610, San Francisco, CA, March 1996.

[18] N. Heintze and J. Tygar. A model for secure protocols and their compositions. *IEEE Transactions on Software Engineering*, 22(1):16–30, January 1996.

[19] N. Heintze, J. Tygar, J. Wing, and H. Wong. Model checking electronic commerce protocols. In *Proceedings of the Second USENIX Workshop in Electronic Commerce*, pages 147–164, November 1996.

[20] N. Heintze and J.M. Wing. Proceedings of the workshop on formal methods and security protocols. URL: http://cm.bell-labs.com/cm/cs/who/nch/fmsp/index.html, June 1998.

[21] IBM. IBM Coprocessor First to Earn Highest Security Validation. http://www.ibm.com/security/cryptocards/html/pr_fips.html.

[22] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

[23] R.A. Kemmerer. Verification assessment study final report. Technical Report C3-CR01-86, National Computer Security Center, Ft. George G. Meade, MD, March 1986. Five volumes.

[24] V. Kessler and G. Wedel. AUTLOG—an advanced logic of authentication. In *Proceedings of the Computer Security Foundations Workshop VII*, pages 90–99. IEEE Comput. Soc., June 1994.

[25] D. Kindred and J. Wing. Fast, automatic checking of security protocols. In *USENIX 2nd Workshop on Electronic Commerce*, 1996.

[26] B. Lampson. Protection. In *Proceedings of the Fifth Princeton Symposium on Information Sciences and Systems*, 1971. Reprinted in *ACMU Operating Systems Review*, Vol. 8, 1974.

[27] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[28] Will Marrero, Edmund Clarke, and Somesh Jha. A model checker for authentication protocols. In *Proc. of the DIMACS Workshop on Design and Formal Verification of Security Protocols*. DIMACS Rutgers University, September 1997.

[29] J. McLean. A Comment on the Basic Security Theorem of Bell and LaPadula. *Information Processing Letters*, 20, 1985.

[30] C. Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1:5–53, 1992.

[31] C. Meadows. The NRL Protocol Analyzer: An Overview. *Journal of Logic Programming*, pages 113–131, 1996.

[32] C. Meadows. Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer. submitted to 1999 Securitye and Privacy, 1998.

[33] J.K. Millen, S.C. Clark, and S.B. Freedman. The Interrogator: Protocol Security Analysis. *IEEE Trans. on Soft. Eng.*, 13(2), February 1987.

[34] J. Mitchell, M. Mitchell, and U. Stern. Automated Analysis of Cryptographic Protocols Using Murphi. In *Proceedings of the IEEE Conference on Secuirty and Privacy*, pages 141–151, 1997.

[35] A. Myers. JFlow: Practical Static Information Flow Control. In *Proceedings of the 26th ACM Symposium on Principles of Programming Languages*, January 1999.

[36] G. Necula and P. Lee. Safe Kernel Extensions Without Run-Time Checking. In *Proc. of Second Symp. on Operations Systems Design and Implementation*, October 1996.

[37] Lawrence C. Paulson. Proving properties of security protocols by induction. Technical report, University of Cambridge, December 1996.

[38] A Rubin and P Honeyman. Formal methods for the analysis of authentication protocols. Technical Report 93–97, CITI, November 1993.

[39] P. Syverson and P. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*. IEEE Computer Society Press, May 1994.

[40] T. Woo and S. Lam. A semantic model for authentication protocols. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 1993.