How Far Can You Trust A Computer?[1]

Carl E. Landwehr
Center for High Assurance Computing Systems,
Naval Research Laboratory
Washington, D.C., U.S.A

Abstract

The history of attempts to secure computer systems against threats to confidentiality, integrity, and availability of data is briefly surveyed, and the danger of repeating a portion of that history is noted. Areas needing research attention are highlighted, and a new approach to developing certified systems is described.

1 Introduction

Concerns about the security of data processed by or stored in computers are probably as old as computing, at least in the sense that some of the earliest modern computing machines were built and used in sensitive applications -- for example, the Polish "Bombe" and its British descendants that were used to attack German ciphers during World War II [1]. But it was only with the development of large-scale, shared multiprocessing systems that computer security, in the sense that term is used today, began to be an issue of general concern. The advent of time-sharing systems in the late 1960's and early 1970's brought the difficulties of protecting users from each other within a single computing environment into sharper focus, because people expected to store data for long periods in such systems, as well as to receive a fair share of interactive computing services. This paper will review briefly some of the history of computer security work starting from that time, summarize some of the lessons we have learned, sketch a recently developed approach to developing and certifying computer systems with security requirements, and suggest some research directions.

Because words like "security" and "trust" are commonly, but imprecisely, used, we introduce a few definitions before proceeding. We say a computer system is *secure* if it can preserve the *confidentiality*, *integrity*, and *availability* of the data it processes and stores against some anticipated set of threats. Preserving confidentiality has typically denoted protecting the data against unauthorized disclosure; preserving integrity has denoted preventing its unauthorized modification; and preserving availability has denoted preventing its unauthorized withholding. These definitions are perhaps narrower than the casual reader might expect, and indeed they have provoked some debate even within the computer security community. Integrity, in particular, continues to be a much-debated term [2]. A computer system or component is *trusted* if we rely on it to perform some critical function or preserve some critical property (such as security); it is only *trust-*

1. Invited Paper, SAFECOMP '93, Proceedings 12th International conference on Com-puter Safety, Reliability and Security, Oct., 1993. Proceedings published by Springer-Verlag, ISBN 0-387-19838-5. Revised following presentation.

worthy if we have evidence to justify the trust we place in it. A computer system is called multilevel secure (MLS) if it is trusted to separate users with different clearances from data with different classifications.

2 Penetrate and Patch

When, in the late 1960's and early 1970's, operating system developers (and their customers) began to discover that their operating systems were somewhat less secure than they had thought, they treated security flaws like any other bugs, and installed fixes. Customers who were particularly interested in the security of their systems sometimes hired "tiger teams" to try to penetrate them, so that all holes might be found and patched. One product of such efforts was the flaw hypothesis methodology, which suggested an informal but systematic approach to this activity [3].This approach to computer security was a victim of its own success -- not its success in achieving secure computer systems, but its success in penetrating insecure ones. Every time a new person or group attempted to penetrate a system, even one that had previously been penetrated and patched, new holes were found [4]. Although records were sometimes collected concerning the holes found, they were not widely circulated, and many of them have since been lost [5]. A forthcoming research report collects fifty surviving examples and proposes a taxonomy for organizing this kind of data [6]. Figure 1 reproduces a chart from that report characterizing the genesis, location, and time in the system life cycle where these flaws were introduced. The taxonomy and charts in that report are intended to provide a helpful method for abstracting current flaw data so that future attempts to improve system security can build on a stronger empirical base.

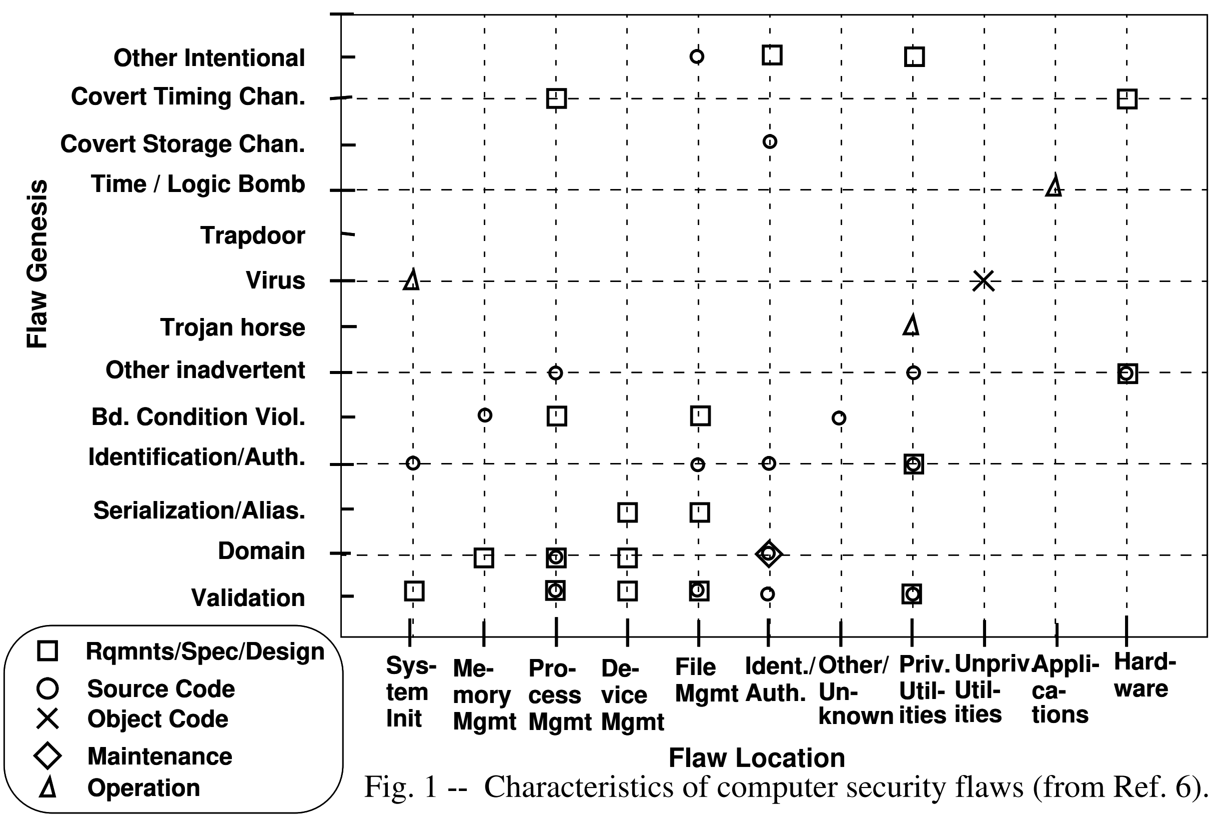

Fig. 1 -- Characteristics of computer security flaws (from Ref. 6).

3 Technology for Trustworthy Operating Systems

The discouraging results of penetrate and patch activities led to the realization that a more systematic approach to building secure systems was needed. From the operating system notion of *monitors* as software components that controlled access to critical system resources, computer security researchers developed the concept of a *reference monitor* that would validate that all references issued by subjects (executing programs) to objects (memory and files) were consistent with an access control policy [7]. If the hardware and software needed to perform the reference monitor functions could be isolated and encapsulated in a small and simple enough part of a system so that high confidence in its correctness could be established, that component would be called a *security kernel* [8]. For some time, researchers labored to demonstrate prototype security kernel implementations. The difficulty of actually isolating all of the code on which security-enforcement depended proved greater than had been supposed, but it nevertheless seemed clear that much more trustworthy systems could be built using this approach than by the penetrate-and-patch method.

To persuade vendors to build more trustworthy systems and to make it easier for users to purchase such systems, the U.S. government established a National Computer Security Evaluation Center in 1981. It was to produce a set of computer security evaluation criteria and then evaluate products submitted voluntarily by vendors. Results would be maintained on an Evaluated Products List that could be used to qualify products for government purchase; this qualification provided the incentive for vendors to submit their products for evaluation.

The *Trusted Computer System Evaluation Criteria,* better known as "the Orange Book," which defined seven different evaluation levels, appeared officially in 1983. It defined a *trusted computing base* as the "totality of protection mechanisms within a computer system ... that is responsible for enforcing a security policy." Thus a system that had its security enforcement mechanisms distributed throughout its software and hardware could be said to have a trusted computing base, but not a security kernel. Only the highest two levels defined by the TCSEC (B3 and A1) require that the TCB be structured to exclude code not essential to security policy enforcement -- that is, to have a security kernel.

The Orange Book was tacitly based on an abstraction of the dominant computing model of the 1970's: a shared, central-server timesharing system. As the computing world shifted toward workstations and networks, it became clear that, at the very least, some additional thought was required to see how to apply the Orange Book in this context. One result of this re-thinking was the *Trusted Network Interpretation of the TCSEC* (the "Red Book"), published in 1987.

It also became clear that having an operating system that could separate differently classified information from differently cleared users was not the same thing as having an application that provided a useful MLS interface. The operating system might support a variety of applications, each operating at single security levels or it might support several instances of the same application, with each instance operating at a different (single) level. But to support a single application that operated across a range of security levels required that some of the security enforcement be provided by the application -- the product TCB would have to be modified or extended to incorporate parts of the application in this case. To address this issue, particularly in the context of database management systems, the *Trusted Database Interpretation* was published in 1991.

The rest of the world did not sit still during this period. Toward the end of the 1980's, France, Germany, the Netherlands, and the United Kingdom produced the *Information Technology Security Evaluation Criteria* (the ITSEC) -- a "harmonised" version of evaluation criteria created jointly by representatives from all four countries. The ITSEC permit greater flexibility than the TCSEC, in that vendors and customers can separately specify functional requirements and assurance requirements, which are joined in the Orange Book classes. The ITSEC also attempt to define a structure that supports both product and system evaluation, although their utility for the latter role is not universally accepted. The *Canadian Trusted Product Evaluation Criteria* (CTCPEC) also permit separating function and assurance, but (as their title indicates) is restricted to product evaluations. However, the CTCPEC have gone farthest in explicitly addressing integrity and denial-of-service (availability) issues.

4 What We Can Do Today

Many products have been built and submitted for evaluation in the decade since the Orange Book first appeared. As of June 1992, the U.S. Evaluated Products List included over a dozen products rated C2, four rated B1, two rated B2, and one at B3; in addition there are two systems that have been evaluated as network components under the TNI; one received a B2 rating and the other an A1. Products have also been evaluated successfully against the ITSEC and the CTCPEC.

On this basis, it seems fair to conclude that today we can specify, design, and build operating system products to meet the requirements reflected by Orange Book levels D through B3. We also know how to evaluate products against those criteria, and there is some evidence that products satisfying the higher levels of the criteria are indeed more difficult to penetrate than those that don't.

These conclusions should be tempered with the understanding that the evaluation process is still typically long, arduous, and, particularly at the higher assurance levels, expensive. This fact is partly attributable to the way that the evaluation process has evolved, as Steve Lipner noted in his insightful paper presented two years ago [9], but it is also attributable to the fact that developing high assurance systems based on security kernels requires greater control and documentation of the system engineering process, and particularly the software engineering process, than most developers customarily provide.

Something else that we can (and do) do today is plug together systems from products, including workstations, local area networks, routers, gateways, and software from a wide variety of sources. Often these systems perform their functions quite effectively, but the security they provide is usually hard to determine and hard to control, even if the security properties of the individual components are known.

5 Problems We Face

There are many problems that need to be solved before we will see the widespread and effective application of trustworthy computing technology. A few of these problems are highlighted in this section; the following section reports some recent advances in addressing the first of them.

We need less costly product evaluation/system certification techniques that can be applied more quickly, but with effectiveness at least equal to current approaches. Problems with the current product evaluation process in the U.S., as described by Lipner [9], have stimulated attempts to improve it. Unquestionably, better methods are needed, but there seems to be an increasing tendency in some quarters to accept commercial off-the-shelf products, together with some form of testing, as sufficient to assure the security of a product. We must be sure that this tendency does not simply lead us back to the discredited "penetrate-and-patch" paradigm. Research areas relevant to this problem include techniques for assessing software development methods, techniques for documenting and assessing software specifications and designs, tools and methods for product testing, and reverse engineering methods.

We need better ways to understand the security provided by composite and distributed systems. The only reason that we can build systems today by plugging them together is that there is a degree of standardization at the level of physical connectors and device protocols. These standards permit the possibility of component-based system engineering. We are not likely to be able to deduce much about the security provided by systems built this way, however, until at least a comparable level of standardization of the security functions and assurances provided by components is achieved. Further, we must take into account the potentially world-wide distribution of modern systems, which, particularly in the commercial sphere, often means that reliable authentication of the originator and recipient of a message, rather than the confidentiality of its contents, is the paramount security concern. Research areas that address this issue encompass abstract, formal work on security modeling techniques that support composition and decomposition, methods and tools for reasoning about and finding flaws in cryptographic protocols, and concrete approaches for standardizing security function and assurance requirements.

We need better ways to control the security functions that current (and future) systems provide. Many actual security problems occur not because security controls are lacking but because the existing controls have not been set up correctly. Steve Kent of BBN Communications has observed that the US is a country of people who are unable to program their videotape recorders, yet we are building interfaces to our security controls that are much more complex than the average VCR control panel. It is difficult to find much research aimed at this problem presently, but work to identify common requirements for application-based security controls and to develop user and administrator interfaces to them that are based on work in the area of human-computer interaction could lead to significantly improved security in practice.

We need to develop practical methods for building high assurance systems. There are definite needs for systems that can provide very high confidence that they will not have security failures. The leading technology for developing high assurance software is to apply formal techniques to its specification and development. Although a recent study shows increased industrial application of formal methods [10], their use is still seen as a significant cost factor, and there is uncertainty as to whether they can be successfully applied in large projects. Further, it is difficult to assess the cost-effectiveness of their application because it is hard to quantify the security provided by the resulting system. Imaginative approaches are needed to organize systems so that requirements for high assurance software are kept to a minimum. We need practical methodologies for exploiting formal methods on those portions of systems that unavoidably require high assurance, and we need methods to estimate or measure the security actually provided.*We need to broaden the scope of "security" and to develop methods for addressing security properties in conjunction with other critical system properties.* Few systems are purchased strictly to provide security. Typically, a customer requires a system to perform some function -- communication, record keeping, real-time control, etc. -- and may acknowledge that to perform the function properly, some security requirements must be met as well. In commercial applications, confidentiality may frequently take a back seat to integrity and authenticity, and availability may be the strongest security concern. In control systems, timely delivery of results may be paramount. If we are to build systems that incorporate security as well as the other properties users require, we need techniques for developing designs that can meet a variety of critical requirements and that permit a system designer to make rational trade-offs among them. Research that permits quantification of covert channel bandwidths, for example, is a step in this direction to the extent that it permits us to quantify the rate at which a particular system design permits information to leak [11]. Work to model denial of service protection is similarly relevant [12].

6 Developing and Certifying Trustworthy Systems: A New Approach

As noted above, we cannot at present develop and certify the security properties of integrated systems nearly as well as monolithic products. In this section, we briefly describe an informal, but structured, approach to system development and certification developed recently at the Naval Research Laboratory. This approach has yet to be applied in sufficient detail to a large example to permit us to make strong claims about its effectiveness, but it is based on concepts proven in our earlier work [13,14]. It has strong intuitive appeal, both as a way to address security requirements during system development and as a way to explain to the *accreditor* (the person responsible for deciding whether to permit the system to be operated) what security the system provides and what risks its operation would pose. *Certification* denotes a technical assessment of the ability of a system to meet specified technical standards (e.g. for enforcing security requirements). A more comprehensive description of this approach has recently appeared [15].

The approach is based on recording *assertions* and *assumptions* that capture the system security requirements within the framework of a documented *assurance strategy*. At the beginning of the project this strategy records both an initial, high-level, abstract version of the *assurance argument* for the system as well as the plan for creating the final, more detailed and concrete assurance argument that will form the primary technical basis for the certification decision. As the project progresses, the assurance strategy is elaborated to reveal the increasingly detailed outline of the assurance argument, which demonstrates that the system as designed and built actually satisfies its security requirements. This argument will not exist as a separate document; the final assurance strategy will in effect be an index to other parts of system documentation (software specification and design documents, test plans and results, etc.) that provide the "nuts and bolts" of the assurance argument. With this approach, certification of a trusted system can largely be accomplished as an audit of the development process.

For a given system, assertions are predicates that are enforced by the system, and assumptions are predicates that must be enforced in the system's environment. The system itself is unable to enforce its assumptions, but must rely upon them. Together, assumptions and assertions represent what must be true of the system and its environment to satisfy the security policy. If an assumption or an assertion is false, a security violation may occur.

For example, consider a medical information system used by physicians, nurses, and pharmacists within a single hospital to record the current symptoms, diagnosis, treatment plan, and billing information for each patient. Suppose that the system's security policy requires that (1) only an administrator can create a new patient record or enter authorizations for doctors, nurses, and pharmacists to use the system, (2) only physicians may update the recorded diagnosis and treatment plan, (3) nurses may update the record of symptoms and medication administered, and (4) pharmacists may read the treatment plan but can only update the billing information. Finally, (5) patients are prohibited from any access to the system.

The architect of such a system has a number of security disciplines available to help satisfy system security requirements, including personnel security, physical security, procedural security, communications security, computer security, and others. The system architect typically seeks the most cost-effective combination of methods drawn from these disciplines that will satisfy the overall system security policy in the face of anticipated threats.

If the system architect decided to rely primarily on the discipline of computer security to enforce the security policy, most of the predicates would be enforced by the medical information system software, and they would be assertions about that software system. If the architect chose to rely on personnel and procedural security measures (e.g., by training the users in their roles and relying on them to invoke only the system functions appropriate to those roles) then, from the standpoint of the information system, all of the predicates would be assumptions about the environment in which it operates.

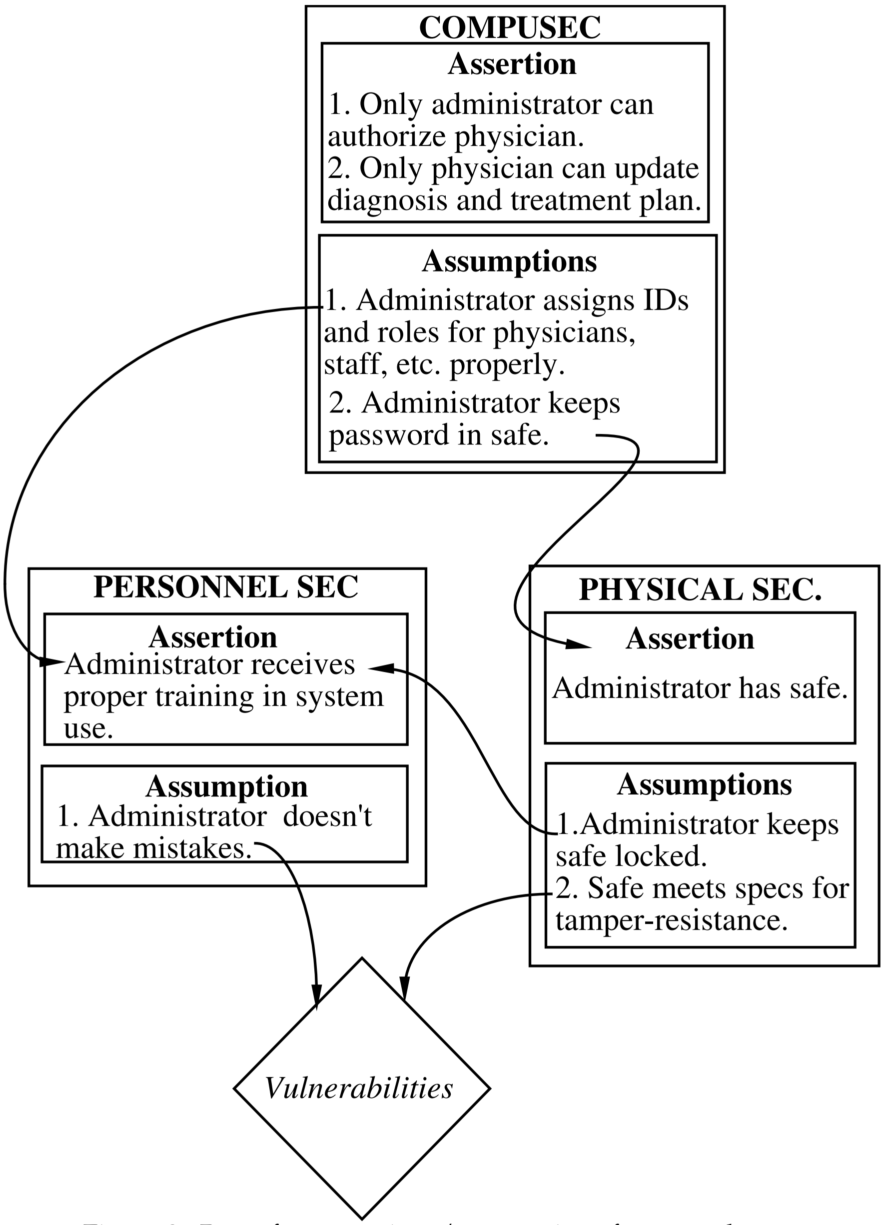

Figure 2. Part of an assertions/assumptions framework.

The assurance strategy provides a framework for recording the assertions and assumptions according to a chosen system security architecture. Suppose a design is developed that requires the administrator to use, and to write down on paper, a password for authentication purposes. Figure 2 illustrates part of an initial assertions/assumptions framework for the medical information system example. Notice that each assumption is supported by an assertion from another discipline or else maps to the "vulnerabilities" symbol. In this representation, assertions do not map to other parts of the framework; they form a set of requirements to be satisfied by the discipline in which they occur. The full assurance strategy for COMPUSEC, for example, would have to explain what kind of assurance would be provided that the COMPUSEC assertions are enforced (e.g., through use of a trusted computing base supporting access controls).

A certifier can review the assurance strategy at the beginning of the development and decide at that time whether this strategy (if followed) is likely to produce an acceptable assurance argument at delivery. The assurance strategy may be modified during the development, since as design trade-offs are made it may be appropriate to modify the kind or degree of assurance (e.g., code reviews, formal verification, testing) required to support particular parts of the assurance argument that is being created. It allows certifiers to assess the role various design decisions play in the overall assurance argument and to determine whether the proposed assurance techniques are effective for demonstrating the validity of the decision.

This approach is not a panacea; correctly defining the security policy and creating a complete assurance argument will continue to be a challenging task. But it does promote the integration of system and security engineering, it can reduce the risk that unforeseen certification issues will impede system development and delivery, and it can make the risks of operating the system more clearly visible to the accreditor.

7 Summary and Conclusion

We have reviewed briefly the development of trustworthy computing technology. Readers should particularly note the lessons of the "penetrate and patch" approach, so that we do not repeat the experiences of that era. We have noted needs for improvements in system certification methods, in understanding security implications of composite and distributed systems, in the control interfaces for security functions, in methods for developing high assurance systems, and in integrating security and other critical properties. Finally, we have glimpsed a new approach to system certification.How far can we trust computers? We already trust them to fly our airplanes and rockets, and it is certainly easier to purchase a computer system and know its security properties now than it was ten years ago. But we still have far to go to before we can make rigorous statements about the *trustworthiness* of the software in our trusted systems.

References

1. Hodges A. *Alan Turing: the enigma*. Simon and Schuster, New York, 1983.

2. Sandhu R. On four definitions of data integrity. In: Keefe, T (ed) Proc. IFIP WG11.3 seventh working conf. on database security, Sept., 1993 (to appear as *Database Security VII: Status and Prospects*, Elsevier, 1994).

3. Linde R. Operating system penetration. In: *Proc. National Computer Conference*, 1975. AFIPS Press, Montvale, N.J., 1975, pp 361-368.

4. Neumann P G. Computer security evaluation. In: *Proc. National Computer Conference*, 1978. AFIPS Press, Montvale, N.J., 1978, pp 1087-1095.

5. Bisbey R. Personal communication. 26 July 1990.

6. Landwehr C E, Bull A R, McDermott J P, Choi W S. A taxonomy of computer program security flaws with examples. NRL Report (forthcoming), Naval Research Laboratory, Washington DC, 1993.

7. Anderson J P. Computer security technology planning study (vols I and II). ESD-TR-73-51, Hanscom Field, Bedford MA; NTIS AD 758 206, 1972.

8. Gasser M. *Building a secure computer system*. Van Nostrand Reinhold, New York, 1988.

9. Lipner S B. Criteria, evaluation, and the international environment: where have we been, where are we going? In: Lindsay and Price (ed), *Proc. IFIP-SEC 91*, Brighton, England. Elsevier - North Holland, 1991.

10. Craigen D, Gerhart S, Ralston T. An international survey of industrial applications of formal methods. NRL Report 9554, Naval Research Laboratory, Washington DC, 1993.

11. Gray J W. On introducing noise into the bus-contention channel. In: *Proc. 1993 IEEE CS Symp. on Research in Security and Privacy*. IEEE Computer Society Press, 1993, pp 90-99.

12. Millen J K. A resource allocation model for denial of service. In: *Proc. 1992 IEEE CS Symp. on Research in Security and Privacy*. IEEE Computer Society Press, 1992, pp 137-147.

13. Landwehr C E, Heitmeyer C L, McLean J. A security model for military message systems. *ACM Trans. on Computer Systems* 1984; 2(3):198-222.

14. Froscher J N, Carroll J M. Security requirements of Navy embedded computers. NRL Memorandum Report 5425, Naval Research Laboratory, Washington DC, 1984.

15. Payne C N, Froscher J N, Landwehr C E. Toward a comprehensive INFOSEC certification methodology. In: *Proc. 16th National Computer Security Conference*. National Institutes of Standards and Technology / National Computer Security Center, Baltimore, MD, Sept. 1993.