From *Proceedings of the Department of Defense Computer Security Center Invitational Workshop on Network Security*, New Orleans, LA, March 1985. Reprinted in [1, pp. 300–316].

Networks are Systems
A Discussion Paper

John Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

1 Preamble

The DoD Computer Security Center's Invitational Workshop on Computer Network Security is concerned with extension of the Criteria in the Center's "Orange Book" [5] to cover the case of computer networks. My assignment was to prepare a position paper on the role of (Formal) Verification in ensuring network security—but before I could address the issue of verification, I found I had first to consider the notion of "network": just what *is* a network, and what does it mean for a network to be "secure"?

2 Networks

2.1 What Sort of Networks should be considered?

First, we must decide how general a notion of "network" we wish to consider. The letter of invitation indicates that the goal is the production of "Trusted Network Evaluation Criteria for long-haul, packet switched networks similar to the Arpanet or DDN". This restricted focus is, in my view, shortsighted, undesirable, and unnecessary.

The most significant development in modern systems design is the trend towards *distributed* systems; that is, systems composed of smaller systems which communicate over some medium in order to cooperate towards the achievement of a common goal. This trend is especially true in the case of military systems, where C^3I and message systems—which are nothing if not distributed systems—play such a significant, and growing, role. But quite aside from systems whose very function involves communications and networking, modern hardware developments make it inevitable that many general and special purpose systems—even systems that fit in a single box—will be structured internally as multi-processor systems communicating over a private bus or LAN.

To ignore these developments in our consideration of network security is to risk the production of Network Evaluation Criteria whose application is restricted to a limited, and declining, section of the market. More importantly, by failing to comprehend these new developments in systems design, we will fail to influence their course and will have to struggle, belatedly, for a proper consideration of their security implications.

Orthogonal to the use of *physical* networks to build distributed systems has been the development of system *concepts* based on the idea of communicating processes. This approach, which regards systems as composed of self-contained, autonomous processes connected by ideal communications channels, seems likely to dominate future language and system design—its influence can already be seen in languages such as CSP, Gypsy, and NIL [9] (and even, to a limited extent, in Ada) and in operating systems such as Thoth [4] and Accent [11]. Using this approach, the decision whether the processes constituting a design should be supported on separate machines connected by physical communications lines, or should be time-shared on a single processor, is relegated to an implementation issue.

Thus, a second reason for considering a broad interpretation of the notion of "network" is the need for Security Evaluation Criteria not only to remain abreast of the best modern approaches to system design, but to exploit the insights which they confer. One of the great merits of the approach based on communicating processes, and a reason for its popularity, lies in the opportunity it provides for integrating networking, inter-process communication, and operating system services. Thus, issues which had formerly seemed quite dissimilar have now been unified. I believe we could expect a similar benefit in the case of security: a unification of the issues of system and network security. Conversely, if we follow the narrow approach, there is a danger that systems based on communicating processes and implemented on a network will fall outside the scope of the Network Evaluation Criteria, while those implemented on a single processor will likewise fall outside the rather narrow scope of the Orange Book!

2.2 What *is* a Network?

If we decide to take a broad view of the sort of networks that should be considered for security evaluation, then we need to agree on what constitutes a network. The definition proposed in letter of invitation states:

> "A network is an entity composed of a communications medium and those parts of all devices attached to that medium whose responsibility is facilitating, controlling, monitoring, or otherwise participating

> in the transfer of information across the medium. Such devices may include, but are not limited to, packet switches, host front ends, access controllers, monitoring centers, and hosts."

This definition concentrates on the physical characteristics of networks and views them primarily as agents for data transmission. This view is entirely appropriate for networks such as the Arpanet or the X25 networks maintained by PTTs, where the services provided by the network amount to little more than the ability to connect to a remote host or to transfer files from one host to another. In these cases, the user is left in no doubt that the network is simply a data transmission medium and that all services are provided by the host machines attached to the network.

More recently, however, networks themselves have come to provide services or resources—file servers, name servers etc.—while some of the services provided by host machines have become so integrated with those of the network that they give the appearance of a single network-system-wide service—electronic mail, for example. The most recent stage of development is exemplified by systems such as Unix United [3] and Locus [10], and by the products of Apollo, Sun and certain other manufacturers which, to a greater or lesser extent, present their users with a single coherent system interface that hides the fact that their implementation involves separate machines and network communications. In the best of these systems, copying a file from one machine to another is no different than copying it from one directory to another.

If we consider the special issue of access control, a similar evolution may be perceived. In the beginning, access control was exclusively the responsibility of the hosts attached to the network; later, networks themselves came to provide some form of access control; most recently, there have emerged systems in which access control is fully distributed throughout the host systems and the network. In the limit, this last approach can be used to synthesize a multilevel secure system out of largely untrusted components [15].

What we are witnessing in all these cases is the eclipse of the idea that networks are "special" in some way, and its replacement by an integrated, systems-level view that regards networks as simply an implementation mechanism used in certain types of system. There can be few today who would dispute that this is, indeed, the correct view. If I send mail to someone, it is a matter of indifference to me whether his mailbox is on the same machine as mine, or is on a machine of an entirely different sort at the other side of the world: the mechanisms by which our mail is exchanged are an implementation detail, it is the *service*—the ability to exchange mail—that is of interest. Similarly, in the case of security, the properties (and even the existence!) of networks are an implementation issue; the significant matter is the security of the services provided by the total *system*.

I submit, therefore, that security is a property of systems; networks are an implementation mechanism used in certain systems and the security of a network cannot be considered in isolation from that of the overall system of which it is a component. It is therefore unnecessary, and undesirable, to generate Security Evaluation Criteria specifically for networks. Instead, revisions to the Orange Book are needed in order to establish Security Evaluation Criteria for *systems* that will admit implementations using networks. This is not to deny that the security characteristics of network implementations are important; it is simply to recognize that they should not be treated in isolation.

The significant property of networking technology is that it makes it possible to put systems together to produce bigger systems, or, looked at from the other direction, it makes it possible to build systems out of smaller systems. Thus, the novel problems raised by networks are those concerning systems built from systems. Thus, I claim that *networks are systems*, and the issue we should be concerned with is one of **putting systems together**.

2.3 What is different about Systems built from Systems?

As far as security is concerned, systems made up of other systems differ from monolithic systems in two important respects. The first of these concerns the fact that the component systems are *active* entities; the second is due to the fact that they are *autonomous* entities. These two properties undermine the application of certain cherished principles of secure systems design and modeling.

The Simple-Security Condition doesn't apply The principle known as the "simple security condition" asserts that security is preserved if an untrusted subject at one security level reads from an object at a lower level.[1] This is sound because, by definition, objects are *passive* entities: they can make no clandestine use of "read requests" received from higher level subjects—a memory doesn't "know" when it has been read by a high level subject.[2]

The situation is different when the participants are active systems. In order for one system to read information from another, the first must send a "read request" message to the second—which must decode the request, obtain the information requested, and return it in a message to the first system. Thus, the reading of

[1]For ease of expression, I am using inexact terminology here. I should, of course, speak of security classifications drawn from a poset with "dominates" as the ordering relation.

[2]In modern paged-memory systems this may not be so, since the "use" bit indicates whether a memory page has been read. If this bit can be read by user processes, then it provides a covert channel; even if it cannot be read, it may still provide a timing channel, since a used page is less likely to be removed from main memory and is therefore more likely to provide rapid access.

information can only be accomplished with the knowledge and active participation of the system being read from, and therefore it is *not* secure for a high level system[3] to read information from a lower level one—since the untrusted high level system may choose to encode high level information in the read requests which it sends to the low level system and the latter can decode this information and record it at a lower level of classification. Thus, an untrusted system may not read information (and, in general, may not request a service) from another such system operating at a lower level.

The *-property doesn't apply The dual situation arises when information is written from one system to another. The "*-property" asserts that security is preserved when an untrusted subject writes into an object at a higher level. But this rule, too, is inadequate when independent systems are concerned. The problem this time is due to the *autonomy* of the participants. When information is sent from one system to another, it is necessary to receive an acknowledgment that the recipient has received *and acted upon* the information that was sent. It is impossible to build a usefully reliable distributed system in the absence of such acknowledgments. And it is important to recognize that the need for acknowledgments is not merely due to unreliability in the underlying communication mechanism: the problem remains even when the communications channels are perfect—as when the parties to the transaction are not remote systems, but are independent processes sharing the same hardware.

The root cause of the problem is that if one system sends a file to another, then it needs to know that the receiving system has been able to write the file satisfactorily—which the receiver may be unable to do for any number of reasons quite independent of transmission errors. It may, for example, be out of disk space, or temporarily busy on some other, more urgent task. In fact, since the receiver is an autonomous system, it may choose to reject the "write request" for any reason of its own choosing. Thus *end-to-end* acknowledgments are required and it is these that render it unsecure for a low level system to write information to (or, in general, to request a service from) an untrusted system at a higher level—since the latter can choose modulate its acknowledgments to the former in a manner that encodes high level information.

I should stress that I am not claiming that the Bell and La Padula model of security [2] is wrong; that model is very careful to distinguish (active) subjects from (passive) objects. Rather, it is the case that the assumptions of that model (developed for monolithic systems) do not accord with the different reality of distributed systems.

[3]By which I mean a system with access to highly classified information.

Reference Monitors must be Application-Specific The requirement for end-to-end acknowledgments also undermines another tenet of secure systems design: the idea that it is possible to localize most, if not all, security issues, in a small and application independent "security kernel". Because the acknowledgments must come from the remote *application process* (since it alone knows whether it has performed the requested service correctly), a trusted, application-specific reference monitor is required if the application is to service requests from lower level systems.[4] It is difficult to overstate the importance of this last point; it is also the one least appreciated by those unfamiliar with distributed systems.

In fact, this problem is not unique to distributed systems; it arises also with "conventional" multilevel secure systems. Consider, for example, an MLS system that maintains a SECRET file containing structured information (such as a database or message file). There must be some "guardian" process that ensures the structural integrity of that information. Since this process must be able to both read and write the file containing the information, it must operate at the SECRET level. If an UNCLASSIFIED user wishes to update the information in the file, he must send his proposed update to the guardian, which will perform the update on his behalf only if it accords with its integrity criterion. But what if the update does not meet this criterion? It is unsecure for the guardian (a SECRET level process) to communicate this (or any other) fact back to its UNCLASSIFIED client—and the MLS system will ensure that it does not do so, since its reference monitor will prevent the "write-down" that would be required were it to attempt it. Now in a conventional MLS system in which the guardian's clients are human users, this lack of feedback may be acceptable, or it may be feasible to interpose some (human mediated) guard function that lowers the risk of clandestine use of the feedback channel to some acceptable level.[5] In a distributed system, on the other hand, where the client may be another process, and where communications may be expected to operate at high bandwidth, these solutions are likely to be less acceptable. The moral seems to be not so much that distributed systems are intrinsically different than conventional ones, but rather that expectations are greater and that more ambitious functionality may be desired. The single reference monitor of a conventional MLS system (providing secure access only to unstructured files) may not be sufficient to support future applications.

[4]It is because each multilevel service requires its own reference monitor that our DSS system provides only one such service—namely, file storage and retrieval [15]. Notice that an application-independent reference monitor will suffice if applications are only allowed to service requests from clients at their own level.

[5]Alternatively, the guardian may be made a "trusted process"—which is simply a euphemism for an application-specific reference monitor.

2.4 A Simple Security Model

Fortunately, it seems possible to deduce some fairly straightforward principles for secure operation in distributed systems. I will describe these principles in the context of a simple security model for distributed systems.

A **system** is composed of interconnected **domains**. A **domain** is a self-contained computational entity provided with **sockets**. No information may enter or leave a domain except through one of its sockets. A domain may either be **atomic**, meaning that it has no internal structure (visible at this level of abstraction), or it may be composed, recursively, of a set of interconnected smaller domains (i.e., a system). Domains are interconnected by linking their sockets. Within this model, a conventional monolithic system is an example of an atomic domain; a (simple Arpanet-type) network can be regarded as a domain that provides lots of sockets but rather limited computational power.

In order to talk about security, we associate a **security label** with each socket. For simplicity, I will ignore compartments and speak as if the levels TS, S, C, and U were the only labels—the extension to compartments is obvious. [6] A domain is **secure** if the information *leaving* the domain through a socket tagged with level *l* is entirely unaffected by any information that may have entered the domain through sockets at levels *higher* than *l*.[7] Notice that any domain can be rendered secure by causing all its sockets to have the same level. Consequently, I will talk about **single-level** domains (those whose sockets all have the same level) and **multilevel** domains (those whose sockets may have different levels). An atomic multilevel domain is simply a conventional MLS system—one that could be certified using the criteria in the Orange Book. The question of interest here is: what are the rules for interconnecting domains securely? Because of the problems discussed earlier, the rules are Draconian and, therefore, quite simple.

The basic rule is: sockets may be connected if they have exactly the same security label. Thus, if we have a network that provides no security at all,[8] then it can be used as a single-level domain at, say, the S level, and may interconnect other single-level S domains, and the S level sockets of multilevel domains. Notice that the resulting system is also a domain and may partake in further interconnections.

The simplicity of the scheme described above must be compromised somewhat in order to accommodate domains that are "not quite" multilevel. Such domains

[6]At this level of abstraction, sockets are single-level. In practice, sockets of different levels may be multiplexed onto a single physical socket—but this is an implementation-level issue which may be ignored here.

[7]This is an informal re-statement of the "standard" SRI security model [6–8, 13, 14].

[8]By which I mean that it cannot reliably associate security labels with its sockets. Remember, it is an axiom that information cannot enter or leave a domain except through its sockets—thus, even an unsecure network is required to be protected against both active and passive wiretapping.

are typified by (monolithic) systems cleared to operate in “system high” mode. In this mode of operation, a system may be allowed to operate as multilevel (say, S and TS), but all its users are required to be cleared to the level of the most sensitive information held by the system (TS in this case) and no information may be released at less than the highest level without manual review. This mode of operation is intended for those systems that are believed to do a “good job” of correctly labeling information and of enforcing policy, but which are not trusted absolutely. We can accommodate such systems within our model by labeling their sockets with the entire set of security labels associated with the information they process (e.g., $\{$S, TS$\}$).

It should be clear that it is unsecure to connect a $\{$S, TS$\}$ socket with one labeled $\{$C, S$\}$, or one labeled simply S. On the other hand, it is secure to connect it to sockets labeled $\{$S, TS$\}$ or TS. It is not obvious to me whether it would be secure to allow a connection to a socket labeled $\{$C, S, TS$\}$, but I will assume that it is. With this assumption, the rule seems to be: sockets may be connected if their associated label sets share the same least upper bound. It seems clear that similar rules can be developed to deal with the “controlled” and “compartmented” modes of operation.

Another extension to the simple model is desirable in order to admit certain types of systems (typically, networks) that do provide a fairly strong form of security, but which do not enforce DoD policy. The motivation is as follows: if it is secure to interconnect two sockets, then the length and nature of the interconnecting “wire” is unimportant (provided it is secure against active and passive wiretapping). Thus we should also allow sockets to be interconnected by a “network” domain that provides “pure channels” that are functionally identical to a dedicated wire, even if the network is ignorant of DoD policy.

Simple extensions to the model accommodate this situation. Instead of (or as well as) a security label, a socket may be marked as a **pure channel** connection and labeled with an identifier. At most two sockets may share the same identifier.[9] The requirement is that any information *leaving* the domain through a pure channel socket with identifier i must be entirely unaffected by any information that may have *entered* the domain though sockets bearing any other label or identifier. Notice that this is entirely in accord with the policy on conventionally labeled sockets: the label [pure channel, identifier i] can be regarded as a new security level that neither dominates, nor is dominated by, any other. This requirement ensures that two pure channel sockets bearing the same identifier can be regarded as “collapsed” to a single point. Thus one pure channel socket with identifier i may be connected to a

[9]Actually, I think this requirement can be relaxed to allow broadcast systems.

socket of label X and another to one of label Y exactly if sockets X and Y may be directly interconnected.

2.5 Certification Criteria

Very little of the security model developed in the previous section is specific to networks, yet it seems to address many of the relevant issues. I believe that with a little rewording, many of the criteria in the Orange Book can be similarly generalized to cover network, as well as monolithic, implementations. To take a single example, the system architecture requirement for B2 systems states, in part: "the TCB shall maintain process isolation through the provision of distinct address spaces." Now the real requirement here is to ensure that no information enters or leaves a domain except through its sockets. The use of distinct address spaces for each domain is an implementation mechanism suitable for achieving this requirement on a single processor. In a distributed system, the appropriate implementation mechanisms include the use of separate processors for each domain and end-to-end encryption to protect their intercommunications. If the Orange book were to say more about underlying requirements and less about specific implementation techniques, networks and distributed systems could be accommodated along with conventional systems.

3 Verification

Just as it is necessary to understand what we mean by "networks", so we need to be clear what "verification" is about before we can consider its role in the evaluation of trusted systems.

3.1 What does Verification Accomplish?

Verification is the demonstration of consistency between two (or more) descriptions the same system. The exact meaning of "consistency" depends on the type of verification being performed. **Formal Verification** operates on system descriptions called **specifications** which are **theories** expressed in some **logic**; in this case, "consistency" means that one specification (the more "concrete" one) is a valid **interpretation** of the other (the more "abstract" one)—i.e., any **model** of the more concrete theory will also be a model of the more abstract one. This form of consistency is demonstrated by showing that the **axioms** of the more abstract theory are **theorems** of the more concrete one.

If two *independent* specifications are verified to be consistent, then one's confidence that both are "correct" is surely increased. If the two specifications are not independent, however (e.g., if the more concrete one is consciously derived from the other), then all that can be said is that one's confidence in the correctness of the derived specification is raised near to that of one's confidence in the abstract specification. The ultimate goal—confidence in the correctness of an actual running system—can be achieved through a chain of consistency demonstrations running all the way from a highly abstract specification, through possibly many levels of intermediate specifications, to the running code. The starting point for such a verification chain is a highly abstract specification whose correctness is accepted on the basis of extensive scrutiny and peer review. It is not necessary that this most abstract of specifications should describe *all* the properties desired of the system: some properties may be deemed more important than others and it will be these that are included in the most abstract specification and subject to the rigor of verification. In the case of security verification, the properties present in the most abstract specification are concerned exclusively with security issues and the specification is usually called a **security model**.[10]

In practice, it is generally too tedious and too expensive to perform a chain of formal verification all the way from the most abstract specification right down to the running code. Instead, the formal verification is generally broken off at some level and informal techniques are used to carry assurance from there down to the code level. The reason that *formal* verification is used at all is that (especially when machine-checked) it is more complete and reliable than informal verification. This being the case, it is important to choose the right point in the verification chain at which to turn from formal to informal verification: in particular, formal verification should not be broken off too early—the major design decisions affecting security should certainly be subject to the rigor of formal verification, as should any particularly subtle issues (such as covert channels). On the other hand, formal verification should not be pursued too far; in my view (which is by no means universal) the vast cost of code proof yields a poor return in terms of increased confidence in the security of the overall system.

Just as the decision where to *break off* the formal verification chain is an important one, so is the choice of the point at which the chain should *start*—i.e., in the case of security verification, the choice of security model. If the security model is so highly abstract that it can apply to almost any system design, then a number of intermediate specifications and verification steps will be needed to reach a reasonably detailed description of the actual system of interest. Far fewer steps will be needed if one starts with a more concrete and detailed security model, but that model will already embody system details and design decisions that restrict the class of systems to which it can be applied, and there is also the non-trivial problem of establishing the correctness of such a detailed model in the first place.

[10]Confusingly, this sense of the word "model" is different from that used in logic.

The choice of security model is therefore a delicate and important matter. A security model comprises two components (termed the **system** and the **security** component, respectively):

- Assumptions about the sort of systems to be verified, and
- An interpretation of "security" that is appropriate to that class of systems.

Since most existing security models were developed with monolithic systems in mind, their system components may not accord with the different reality of distributed systems (recall Sections 2.3 and 2.3). It therefore seems necessary to consider how the modeling of distributed systems should differ from that of conventional ones.

3.2 What is different about Systems built from Systems? (Revisited)

Most modeling and verification techniques for monolithic systems depend on the notion of "state": at each instant the system is "in" some state, and progress is represented by the transition from one state to the next. This state-based approach is viable because the monolithic systems can be regarded as *sequential* processes. This is not the case with distributed systems: the components of such a system proceed in parallel and, in general, asynchronously. Although the instantaneous state of a distributed system can be defined as the product of the states of its sequential components, the evolution of that composite state will not be deterministic—since its components may proceed at different rates. For this reason, attempts to describe the behavior of distributed systems in terms of their composite state have not been altogether satisfactory. Most current work attempts to describe distributed systems in terms of the communications in which their components participate (i.e., their input/output behavior—sometimes known as "trace semantics"). The benefit of this approach is that it permits the specification of system behavior without excessive placing of constraints on system implementation: unlike the case with state semantics, the designer is free to choose either a distributed or a sequential implementation.

The arguments that have led those concerned with general system properties to prefer trace to state semantics apply no less forcefully in the case of security properties. I believe that a formal security model for "putting systems together" is more likely to develop from a security model expressed in terms of input/output behavior (e.g., the SRI model [6–8, 13, 14]) than one expressed in terms of states. The informal model described in Section 2.4 can easily be formalized using the techniques of the SRI model and seems to provide a step in the right direction.

3.3 What should be Verified?

The model of Section 2.4 is built on two key assumptions, which become *requirements* on the next lower level of modeling:

- No information can enter or leave a domain, except through its sockets, and
- Information leaving a domain through a socket labeled at level l must be independent of any information that entered the domain through sockets labeled at levels not dominated by l.

The first of these requirements is sometimes referred to as "domain isolation" or "separation". It is clearly fundamental to everything else and, in my view, is definitely a candidate for formal verification. For the case of multiple domains supported on a single processor, I have proposed a technique for verifying this property which I call "Proof of Separability" [12]. In the case of networks, the requirement may be achieved in many different ways (e.g., link, node, or end-to-end encryption, or physically protected cableways) and the appropriate verification techniques will be different also. It is likely that verification of domain isolation in networks will have to consider a "distributed reference monitor" (e.g., crypto boxes and KDC in the case of end-to-end encryption) and this will present an interesting challenge.

The second requirement is to ensure that the services provided at the interface to each domain are secure. This also seems a prime candidate for verification and is one for which several techniques are available. Those based on information flow analysis have the special merit that they identify covert channels as well as the more gross forms of insecurity. The question of covert channels, and whether they (or rather, their absence) should be a target for formal verification, was one that we were specifically asked to consider. It seems to me that if an *overt* channel is a conduit for information that is provided deliberately, then a *covert* channel is simply one that is provided accidentally! It is significant that there is no distinction between overt and covert (storage) channels in the mathematical security model of information flows [6–8, 13, 14].

Since distributed systems are only feasible in the presence of high-bandwidth communications, it is very dangerous to assume that any covert channels will be of sufficiently low bandwidth that they will not pose a serious threat. Also, because distributed systems are composed of active entities and generally require explicit acknowledgments to messages, the opportunities for covert channels are much greater than in monolithic systems (recall the channels described in Sections 2.3 and 2.3). Furthermore, since covert channels are provided "accidentally" it is also dangerous to assume that they can be discovered by casual examination.

Rather, I believe that the systematic detection of covert channels is one of the main contributions of formal verification to the evaluation of secure distributed systems.

4 Suggested Verification Criteria

The following is a sketch of Verification Criteria that are suggested for Class A1. The bulk of the existing A1 criteria can be retained, with slight rewording. For Class B3, replace the term "formal" by "descriptive" and the phrase "mathematical proof" by "careful argument".

> A formal model shall be maintained of the security policy supported by *the complete system* of which the evaluated product is to form a component, and by the evaluated product itself. Formal statements shall be provided of the assumptions made about all system components and a mathematical proof shall be provided to demonstrate that these assumptions, in conjunction with the policy supported by the evaluated product, are sufficient to support the policy of the overall system. A FTLS of the evaluated product TCB shall be maintained and verified with respect to the security model established for the product. The formal models employed shall address the issue of covert channels and the techniques used to substantiate the security policy of the evaluated product and to verify its FTLS shall be ones that identify covert channels.

References

[1] Marshall D. Abrams and Harold J. Podell, editors. *Tutorial: Computer and Network Security*. IEEE Computer Society Press, 1986.

[2] D. E. Bell and L. J. La Padula. Secure computer system: Unified exposition and Multics interpretation. Technical Report ESD-TR-75-306, Mitre Corporation, Bedford, MA, March 1976.

[3] D. R. Brownbridge, L. F. Marshall, and B. Randell. The Newcastle Connection, or UNIXes of the world unite! *Software—Practice and Experience*, 12(12):1147–1162, December 1982.

[4] D. R. Cheriton. *The Thoth System: Multi-process Structuring and Portability*. Operating and Programming Systems Series. North-Holland, 1982.

[5] Department of Defense. *Department of Defense Trusted Computer System Evaluation Criteria*, December 1985. DOD 5200.28-STD (supersedes CSC-STD-001-83).

[6] R. J. Feiertag. A technique for proving specifications are multilevel secure. Technical Report CSL-109, Computer Science Laboratory, SRI International, Menlo Park, CA, January 1980.

[7] R. J. Feiertag, K. N. Levitt, and L. Robinson. Proving multilevel security of a system design. In *Sixth ACM Symposium on Operating System Principles*, pages 57–65, November 1977.

[8] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings of the Symposium on Security and Privacy*, pages 11–20, Oakland, CA, April 1982. IEEE Computer Society.

[9] F. N. Parr and R. E. Strom. NIL: A high-level language for distributed systems programming. *IBM Systems Journal*, 22(1/2):111–127, 1983.

[10] G. Popek et al. Locus: A network transparent, high reliability, distributed system. In *Eighth ACM Symposium on Operating System Principles*, pages 169–177, Asilomar, CA, December 1981. (ACM Operating Systems Review, Vol. 15, No. 5).

[11] R. Rashid and G. Robertson. Accent: A communications oriented network operating system kernel. In *Eighth ACM Symposium on Operating System Principles*, pages 64–75, Asilomar, CA, December 1981. (ACM Operating Systems Review, Vol. 15, No. 5).

[12] John Rushby. Proof of Separability—A verification technique for a class of security kernels. In *Proc. 5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 352–367, Turin, Italy, April 1982. Springer-Verlag.

[13] John Rushby. The security model of Enhanced HDM. In *Proceedings 7th DoD/NBS Computer Security Initiative Conference*, pages 120–136, Gaithersburg, MD, September 1984.

[14] John Rushby. The SRI security model. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, Forthcoming.

[15] John Rushby and Brian Randell. A distributed secure system. *IEEE Computer*, 16(7):55–67, July 1983.