A Hookup Theorem for Multilevel Security

DARYL MC CULLOUGH

Abstract—**In this paper, the author describes a security property for trusted multilevel systems, *restrictiveness*, which restricts the inferences a user can make about sensitive information. This property is a *hookup property*, or *composable*, meaning that a collection of secure restrictive systems when hooked together form a secure restrictive composite system. The author argues that the inference control and composability of restrictiveness make it an attractive choice for a security policy on trusted systems and processes.**

Index Terms—**Composable, information, process, security.**

I. Introduction

Multilevel security requires that sensitive information be disclosed only to authorized personnel. In the "paper world," this is enforced by assigning to each document and each employee a *security level* indicating sensitivity and authority. Commonly used levels in the government are *unclassified*, *confidential*, *secret*, and *top_secret*. The levels form a partially ordered set, so that an employee can be said to be authorized to read a document only if his level is greater than or equal to that of the document.

For information processing systems multilevel security becomes more complicated, because not all information is in the form of documents and not all consumers of information are employees. Generally for such systems the problem of multilevel security consists of two aspects:

1) *Access control*—determining who can see information of a given sensitivity leaving the system.

2) *Correct labeling*—determining the sensitivity of information entering and leaving the system.

The second issue, correct labeling, becomes much more important in computer systems because of the possible presence of *Trojan Horse* programs. A Trojan Horse program is a malicious program that when run by a high level user will try surreptitiously to obtain classified information from that user and convey it to an accomplice, usually the unauthorized user who programmed the Trojan Horse. These programs often masquerade as useful programs such as word processors or compilers.

Rather than inspect each program on a system to see if it is a Trojan Horse program (it may not be possible to decide by inspection whether a program is a Trojan Horse or not), people instead try to build computer systems which are secure even in the presence of malicious programs.

In this paper we will describe a security property that addresses these issues, called *restrictiveness*, which is *composable*. This means that for a collection of trusted processes "hooked up" to make a system, or for a collection of system "hooked up" to make a network, the system or network is secure if each component is secure.

Using restrictiveness as a definition of security for trusted systems provides confidence in building large, complex systems from smaller, easier to verify trusted components. Security is modularized and so becomes more manageable. We first consider two earlier models for security, and point out some of their shortcomings.

A. The Bell–LaPadula Model: Trusted and Untrusted Processes

The issues of access control and correct labeling are addressed by the Bell–LaPadula model [1]. In this model, all entities involved with a computer system—users, files, programs, etc.—are divided into two classifications: *subjects* and *objects*. Subjects are the active entities, such as users and processes, which are capable of reading and modifying system state information, while objects are passive containers for information, such as files. Subjects which are processes are further divided into *trusted processes* and *untrusted processes*.

1) Rules for Untrusted Subjects: To enforce security for untrusted processes, Bell–LaPadula requires that every object and every untrusted process be assigned a security level. The model then demands that

a) An untrusted process may only read from objects of lower or equal security level. (Motto: Read down.)

b) An untrusted process may only write to (or modify) objects of greater or equal security level. (Motto: Write up.)

The *read down* rule insures that a process is only allowed to read information it is entitled (by its security classification) to read. The *write up* rule ensures that all objects are correctly labeled; it is impossible to put information into an object which comes from a source whose level is higher than the security label on the object.

In the Bell–LaPadula model, users are interpreted as untrusted subjects, except that allowance is made for users to act at any security level less than their maximum. For example, a *secret* user may login as a *secret* or *unclassified* subject, but not as a *top_secret* subject.

Manuscript received August 1, 1989; revised January 29, 1990. Recommended by T. A. Berson and S. B. Lipner. This work was supported by the Rome Air Development Center, Griffiss Air Force Base, NY, under Contract F30602-85-C-0098.

The author is with Odyssey Research Associates, 301A Harris B. Dates Drive, Ithaca, NY 14850.

IEEE Log Number 9034811.

2) Trusted Processes: The answers the model provides are incomplete, however, because it does not provide guidance for determining the sensitivity of information coming from the *trusted processes* of a system. A trusted process is any process which is exempted from the stringent requirements enforced on untrusted processes. They are called "trusted" because, since they are not bound by the normal rules, it is necessary to have reason to trust that they do not behave maliciously.

Trusted processes are needed whenever an activity potentially involves information at several different security levels. For example, the file server must be able to read and write files at all different levels, and so cannot possibly be bound by the access control rules given above. In the Bell–LaPadula model, the need for such multilevel processes was recognized, but no precise security rules for the behavior of such processes were given.

To fill this gap, it is desirable to have a security property for a trusted process that guarantees that information leaving the process is correctly labeled: that high-level information does not "leak" into low-level outputs. We next consider a candidate for such a property.

B. Deducibility Security

A *multilevel process* is a system which takes in information of different security levels, processes it and outputs information of different security levels. Note that this description can equally well describe a trusted process, or an entire computer system, or a network of computer systems. A general framework for defining security for such systems, called *deducibility security*, is found in Sutherland [9]. We give an informal description of a special case of Sutherland's model.

We assume that all the sensitive information contained in a process enters the process in the form of discrete *inputs* labeled with a security level indicating their sensitivity, and that information leaves the process in the form of labeled *outputs*.[1] The word *event* will be used to refer to an input or an output.

For each security level l we will call the events of level less than or equal to l the *view* for that level, and all other events we will say are *hidden* from that level. By the assumption that access control is enforced on the untrusted parts of the system, we can know that users of level l are unable to see any events outside their own view. The hidden events for a level l are the inputs made by users of higher (or incomparable) level. Under these circumstances, we will say that a multilevel process is *deducibility secure* if for each security level l and for every sequence of events possible in some history of the process:

The information contained in the events in the view for level l does not reveal anything about information in inputs hidden from level l.

[1]We are ignoring the possibility that new sensitive information might be created *inside* the process. To consider this possibility would go beyond the present paper.

To formalize this statement, it is necessary to say what it means for one set of observations (the sequence of events in the view of some level) not to reveal anything about some other source of information (the sequence of inputs hidden from the level). In Goguen–Meseguer's noninterference policy [3], this was formalized by saying (in our terminology) that if one removes all the hidden inputs the observations in the view remain unchanged.

This is not as general as one would like, since it is only meaningful for deterministic systems (ones where what is observed is completely determined by the inputs). A more general definition is to say that any possible set of observations is *consistent* with any possible sequence of hidden inputs. (That is, it is impossible for a user of level l to "rule out" any sequence of hidden inputs.) We choose this more general definition, since we intend to apply it in cases, such as concurrency, where there is nondeterminism.

Deducibility security prevents leaks due to Trojan Horses. If an entire system is deducibility secure including any Trojan Horses that may be lurking around, and it initially has no classified information in it, then no unauthorized user of that system will ever learn any classified through the system. In other words, any system which allows illegal information flow through the system is not deducibility secure. The informal argument goes as follows:

> Suppose that a low level user learns through the system some piece of high level information on the system. This means that the user's view of the system behavior, call it b, has revealed that some condition C holds for the system, and the fact that this condition holds is classified information. Since we are only considering systems which do not create any new classified information internally, if C holds, then there must have been an earlier moment at which high level information was put into the system by high level users; some action must have been taken by the high level user which caused condition C to hold.
>
> Therefore, behavior b allows the low level user to infer C, which allows him to infer that high level users performed some action. So the high level behavior *do nothing* can be ruled out. Since some high level behavior can be ruled out, the system is not deducibility secure.

If deducibility security prevents all leaks due to Trojan Horses, then what more could one want in a definition for multilevel security? Well, in practical terms, one does not verify everything, one only verifies certain "security relevant" portions of the system. Is it possible that a user can combine information obtained from two different components of a system in order to learn information that he could not receive from either component in isolation? The answer turns out to be yes, as the next section shows.

1) Deducibility Security Is Not Preserved by Hookup: A multilevel system may have many interacting trusted processes, so it is not sufficient to guarantee that each process is secure in isolation; it is necessary to also show that several processes working together cannot conspire to violate security. In other words, for trusted processes, it is necessary to have a definition of security that is *composable*. While deducibility security may be a good overall definition of security for an entire system, it unfortunately is not composable. In this section we will demonstrate this result by showing two processes which alone are deducibility secure, but which when "hooked up" form a composite system which is not secure.

In the following example, we will consider systems with only two levels: *high* and *low*.

Conventions and Notations for Systems: To illustrate the possible behavior of systems, let us introduce a pictorial notation for the *traces*, or possible histories, of systems. We depict a trace of a system by giving a timeline running vertically, with the future of the system toward the top and with the past of the system toward the bottom. Horizontal vectors directed toward or away from the time line of a system represent inputs to and outputs from that system, respectively. We will use unbroken lines to represent *low* inputs and outputs, and broken lines to represent *high* inputs and outputs. To represent two systems operating in parallel, we show their timelines together. A message sent from one machine to another will be shown as a horizontal arrow pointing away from the time line of the sending system and toward the time line of the receiving system.

A Counterexample: Consider a system, called 𝒜, which has the following set of traces: each trace starts with some number of high-level inputs or outputs followed by the low-level output *stop* followed by the low-level output *odd* (if there has been an odd number of high-level events prior to *stop*) or *even* (if there has been an even number of high-level events prior to *stop*). The high-level outputs and the output of *stop* leave via the right channel of the process, and the events *odd* and *even* leave via the left channel. The high-level outputs and the output of *stop* can happen at any time.

Two possible event sequences of system 𝒜 are portrayed in the top left corner of Fig. 1.

System 𝒜 actually is deducibility secure; regardless of high-level inputs, the possible low-level sequences are 1) *stop* followed by *odd* or 2) *stop* followed by *even*. A high-level input does not affect these possibilities, because it is always possible for such an input to be followed by a high-level output, and the pair would leave the low-level outputs unaffected.

System ℬ behaves exactly like system 𝒜, except that:

- its high-level outputs are out its left channel.
- its *even* and *odd* outputs are out its right channel.
- *stop* is an input to its left channel, rather than an output.

System ℬ, system 𝒜, is deducibility secure. A typical event sequence of system ℬ is shown in the upper right corner of Fig. 1.

If systems 𝒜 and ℬ are connected, so that the left channel of ℬ is connected to the right channel of 𝒜 then we have the situation pictured in the bottom of Fig. 1.

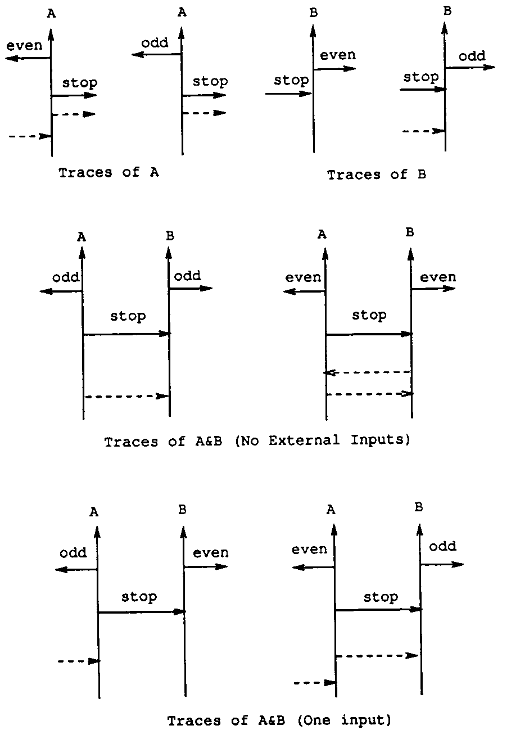

Fig. 1. Deducibility security is not composable.

Now we see that the combined system is no longer deducibility secure. Since the number of shared high-level signals is the same for 𝒜 and ℬ, the fact that 𝒜 says *odd* while ℬ says *even* (or vice versa) means that there has been at least one high-level input from outside. If all high-level inputs are deleted, then systems 𝒜 and ℬ will necessarily both say *even* or both say *odd*. By looking at the low-level events, a user can deduce something about the high-level inputs.

Composition allows one to turn small information leaks into large leaks. Although the example above leaks only a small amount of information; it is not difficult to devise schemes for combining several slightly leaky systems to get a very leaky system. For some examples of how insecurities can multiply see [7].

II. Introducing Restrictiveness

An important thing to notice about the failure of composability for deducibility security is that, although a system obeying the property ensures that no *single* high-level input will affect the future low-level behavior, it does not guarantee that a *pair*, consisting of a high-level input followed immediately by a low-level input, will have the same effect on the low-level behavior as the low-level input alone. From Fig. 1, it is clear that system ℬ does not ensure this latter, stronger form of noninterference. For example, the pair consisting of a high-level input followed by *stop* does *not* have the same effect as *stop* alone.

The additional requirement can be intuitively understood as follows. Only some facts about the past of a system are relevant for the future low-level behavior of the system. These relevant facts can be thought of as defining the "low-level state" of the system. The requirement of noninterference is that a high-level input may not change the low-level state of the system. Therefore, the system should respond the same to a low-level input whether or not a high-level input was made immediately before.

Restrictiveness is a property of systems which formalizes this requirement.[2] In the next sections we formalize restrictiveness as a property of state machines[3] and prove that it is composable.

A. State Machines

A *state machine* is a way of describing a computer system in terms of its internal structure, and its input–output behavior. To describe a system as a state machine, one must give the following.

1) The set of possible *states*.

2) The set of possible *events*, the inputs, outputs, and internal signals of the system.

3) The set of possible *transitions*.

4) The *initial state*.

A transition is denoted by

$$\sigma_0 \xrightarrow{e} \sigma_1$$

where σ_0 is the state of the machine before the transition, e is the accompanying event for the transition, and σ_1 is the state of the machine after the transition. A sequence of transitions starting in σ_0 and ending in σ_n, involving events $[e_1, \cdots, e_n]$ is denoted by $\sigma_0 \rightarrow^{[e_1, \cdots, e_n]} \sigma_n$. We say that σ_0 *can accept* event e if for some state σ_1, $\sigma_0 \rightarrow^{e} \sigma_1$.

The Traces: Traces of a state machine are all sequences of events γ such that for some state σ_1 *start* $\rightarrow^{\gamma} \sigma_1$, where *start* is the initial state.

1) Input Total State Machines: A state machine is said to be *input total* if in any state it can accept an input. The significance of this property for our purposes is that for an input total machine, one can only learn about its state by watching its outputs; no information is conveyed to the user by accepting inputs. In contrast, in a system which is not input total, one learns something about the state of the system whenever an input is accepted; namely that it is in an accepting state for that input.

By restricting our attention to input total processes, we achieve a technical simplification for our theory of security: information enters a system through its inputs, and leaves a system through its outputs. We will make input totality a condition for a state machine to be restrictive, but this is not intended to imply that only such machines are secure. Rather, restrictiveness as a definition of security only applies to input total machines.

B. Security for State Machines

Once again, we will only consider the case of two security levels, *low* and *high*. To prevent a low-level user from discovering information he is not allowed to know, we first need to specify the high-level *state information*, and *event information*. We can summarize this information through the use of two equivalence relations, one on states and one on event sequences.

If σ_1 and σ_2 are two states, then we say $\sigma_1 \approx \sigma_2$ if the states differ only in their high-level information. In other words, if to each state variable we assign either the security level *high* or *low*, then $\sigma_1 \approx \sigma_2$ if the values of all low-level variables are the same in the two states.

If γ_1 and γ_2 are two sequences of events, then we say that $\gamma_1 \approx \gamma_2$ if the two sequences agree for low-level events. For example, if the event a is low, and the event b is high, then the following three events are all considered equivalent:

$$[a, b, b, a] \approx [b, a, b, a] \approx [b, b].$$

A low-level user's view of the system, the state information he may know, and the events he may see, is completely determined by the equivalence relations on states and event sequences (both of which we will denote by $\approx$).

III. Restrictive State Machines

A state machine is defined to be *restrictive* for the view determined by $\approx$ if:

1) It is input total.

2) For any states σ_1, σ_1', and σ_2, and for any two input sequences β_1 and β_2, if

a) $\sigma_1 \rightarrow^{\beta_1} \sigma_1'$
b) $\sigma_2 \approx \sigma_1$
c) $\beta_1 \approx \beta_2$

then for some state σ_2'

a) $\sigma_2 \rightarrow^{\beta_2} \sigma_2'$
b) $\sigma_2' \approx \sigma_1'$.

3) For any states σ_1, σ_1', and σ_2, and for any output sequences γ_1, if

a) $\sigma_1 \rightarrow^{\gamma_1} \sigma_1'$
b) $\sigma_2 \approx \sigma_1$

then for some state σ_2' and some output sequence γ_2

a) $\sigma_2 \rightarrow^{\gamma_2} \sigma_2'$
b) $\sigma_2' \approx \sigma_1'$
c) $\gamma_2 \approx \gamma_1$.

Rules 2 and 3 say, roughly, that "equivalent states are affected equivalently by equivalent inputs, produce equivalent outputs and then remain equivalent." This is a non-

[2]Goguen and Meseguer's notion of noninterference formalized this notion for deterministic state machines. For the class of machines they considered, restrictiveness and noninterference in their sense agree.

[3]In [6] a similar property was defined solely in terms of the traces of a system; states were not involved.

interference assertion; that high-level inputs and high-level information cannot affect the behavior of the system, as viewed by a low-level user. Restrictiveness thus generalizes the Goguen–Meseguer definition of noninterference to nondeterministic systems. (For the particular kind of deterministic machines that Goguen and Meseguer considered, restrictiveness and noninterference reduce to the same property.)

One should note that an immediate consequence of 2 is that if β_1 is a high-level input sequence, then it must not affect the state at all. Also, in rule 3, it is easy to show by induction that it is enough to consider cases in which γ_1 (but not necessarily γ_2) consists of a single event.

IV. Hooking Up Machines

If $\mathcal{A}$ and $\mathcal{B}$ are two machines, then hook them up by sending some of the outputs of $\mathcal{A}$ to be inputs to $\mathcal{B}$, and vice versa. These common events will then be communication events, which will be treated like output events for the composite machine. The inputs of the composite machine are the inputs of either component machine which are not supplied by the other. The states of the combined machines are pairs $< \sigma, \nu >$, where σ is a state of $\mathcal{A}$, and ν is a state of $\mathcal{B}$.

The Events: An event of the composite machine is any event from either component machine. For any sequence of events γ from the composite machine, we will let $\gamma \uparrow E_{\mathcal{A}}$ be the sequence of events in γ engaged in by machine $\mathcal{A}$, and $\gamma \uparrow E_{\mathcal{B}}$ be the sequence of events engaged in by $\mathcal{B}$. Because of the shared communication events, some events from gamma will occur in $\gamma \uparrow E_{\mathcal{A}}$ and in $\gamma \uparrow E_{\mathcal{B}}$.

The Transitions: $< \sigma, \nu > \rightarrow^{\gamma} < \sigma', \nu' >$ is a valid transition for the composite machine if and only if $\sigma \rightarrow^{\gamma \uparrow E_{\mathcal{A}}} \sigma'$ and $\nu \rightarrow^{\gamma \uparrow E_{\mathcal{B}}} \nu'$ are valid transitions of the component machine. This notion of hookup, or parallel composition, is taken from CSP [4]. It assumes that the only correlation between state transitions of the two components is through the shared communication events.

Security: The equivalence relations for the composite machine are inherited from those of the component machines:

- $< \sigma, \nu > \approx < \sigma', \nu' >$ if and only if $\nu \approx \nu'$ and $\sigma \approx \sigma'$.
- $\gamma \approx \gamma'$ if and only if $\gamma \uparrow E_{\mathcal{A}} \approx \gamma' \uparrow E_{\mathcal{A}}$ and $\gamma \uparrow E_{\mathcal{B}} \approx \gamma' \uparrow E_{\mathcal{B}}$.

A. Restrictiveness is Composable

If state machines $\mathcal{A}$ and $\mathcal{B}$ are restrictive, then a composite machine formed from hooking them up is restrictive.

The composite machine is input total. If β is any sequence of inputs for the composite machine, and $< \sigma, \nu >$ is any starting state, then $\beta \uparrow E_{\mathcal{A}}$ is a sequence of inputs for $\mathcal{A}$, and $\beta \uparrow E_{\mathcal{B}}$ is a sequence of inputs for $\mathcal{B}$. Since $\mathcal{A}$ and $\mathcal{B}$ are input total, there are states σ' and ν' such that $\sigma \rightarrow^{\beta \uparrow E_{\mathcal{A}}} \sigma'$ and $\nu \rightarrow^{\beta \uparrow E_{\mathcal{B}}} \nu'$. Therefore, $< \sigma, \nu > \rightarrow^{\beta} < \sigma', \nu' >$.

Inputs affect equivalent states equivalently. If $< \sigma_1, \nu_1 >$, $< \sigma_1', \nu_1' >$, and $< \sigma_1, \nu_1 >$ are states and β_1 and β_2 are input sequences, then since $\mathcal{A}$ and $\mathcal{B}$ are restrictive, there must be states σ_2' and ν_2' such that

1) a) $\sigma_2 \rightarrow^{\beta_2 \uparrow E_{\mathcal{A}}} \sigma_2'$
   b) $\nu_2 \rightarrow^{\beta_2 \uparrow E_{\mathcal{B}}} \nu_2'$.
2) a) $\sigma_2' \approx \sigma_1'$
   b) $\nu_2' \approx \nu_1'$.

Therefore,

1) $< \sigma_2, \nu_2 > \rightarrow^{\beta_2} < \sigma_2', \nu_2' >$.
2) $< \sigma_2', \nu_2' > \approx < \sigma_2', \nu_2' >$.

Equivalent states produce equivalent outputs, which lead again to equivalent states. As remarked earlier, it is sufficient to consider outputs of single events. Suppose then that e is an output, and that

1) $< \sigma_1, \nu_1 > \rightarrow^{e} e < \sigma_1', \nu_1' >$
2) $< \sigma_1, \nu_1 > \approx < \sigma_2, \nu_2 >$.

An output for the composite machine must be an output for one of the component machines. We assume then that e is an output from $\mathcal{A}$; the other case is handled similarly.

Since $\mathcal{A}$ is restrictive, and $\sigma_1 \rightarrow^{e} \sigma_1'$, and $\sigma_2 \approx \sigma_1$, then for some state σ_2' and some output sequence γ: $\sigma_2 \rightarrow^{\gamma} \sigma_2'$, and $\sigma_2' \approx \sigma_1'$, and $\gamma \approx [e]$.

Now, since the sequence γ is an output sequence, any events shared by both $\mathcal{A}$ and $\mathcal{B}$ must be inputs to $\mathcal{B}$. Since $\gamma \approx [e]$, it follows that $\gamma \uparrow E_{\mathcal{B}} \approx [e] \uparrow E_{\mathcal{B}}$. Therefore, we have:

1) $\nu_1 \approx \nu_2$
2) $\nu_1 \rightarrow^{[e] \uparrow E_{\mathcal{B}}} \nu_1'$
3) $\gamma \uparrow E_{\mathcal{B}} \approx [e] \uparrow E_{\mathcal{B}}$.

From the fact that $\mathcal{B}$ is restrictive, we get that for some state ν_2'

1) $\nu_2 \rightarrow^{\gamma \uparrow E_{\mathcal{B}}} \nu_2'$
2) $\nu_2' \approx \nu_1'$.

Therefore, we have that

1) $< \sigma_2, \nu_2 > \rightarrow^{\gamma} < \sigma_2', \nu_2' >$
2) $< \sigma_2', \nu_2' > \approx < \sigma_1', \nu_1' >$.

V. Conclusions

In analogy with the Bell–LaPadula model, we can require that every untrusted process be assigned a security level, and also require that every output be greater than or equal to this level (motto: send up), and that every input be less than or equal to this level (motto: receive down). It is easy to see that every such untrusted process is *manifestly secure*; it is necessarily restrictive, if we make all state information the level of the process. Therefore, an immediate consequence of the hookup theorem for restrictive machines is as follows.

If every component of a system is proved restrictive, or is untrusted and manifestly secure, then the entire system is restrictive. Therefore, any Trojan

Horse in a restrictive system is harmless, as long as it is only allowed to send up and receive down.

In this paper, we have argued for the need for a formal definition of security that is applicable for a wide range of processes, systems, and networks. The Bell–LaPadula model is not sufficient for this purpose, because while it defines security in terms of access controls, it does not provide formal guidance for the security of trusted processes. Sutherland's deducibility security provides a general definition of security for total systems, but unfortunately is not composable, and so cannot be applied to components of a large system. The final property described, restrictiveness, is generally composable and so can be used as a definition of security for small processes and entire systems.

References

[1] D. E. Bell and L. J. LaPadula, "Secure computer system: Unified exposition and multics interpretation," Electron. Syst. Division, AFSC, Hanscom AF Base, Bedford, MA, Tech. Rep. ESD-TR-75-306, 1976.

[2] J. A. Goguen and J. Meseguer, "Security policies and security models," in *Proc. 1982 IEEE Symp. Security and Privacy.*

[3] ——, "Unwinding and inference control," in *Proc. 1984 IEEE Symp. Security and Privacy.*

[4] C. A. R. Hoare, *Communicating Sequential Processes.* Englewood Cliffs, NJ: Prentice-Hall, 1985.

[5] D. McCullough, "Foundations of Ulysses: The theory of security," Odyssey Research Associates, Ithaca, NY, Tech. Rep., 1988.

[6] ——, "Specifications for multilevel security and a hookup property," in *Proc. 1987 IEEE Symp. Security and Privacy.*

[7] ——, "Covert channels and degrees of insecurity," in *Proc. 1988 Franconia Computer Security Foundations Workshop: The Mitre Corporation.*

[8] R. A. Milner, *A Calculus of Communicating Systems* (*Lecture Notes Comput. Sci.*, Vol. 92). New York: Springer, 1980.

[9] D. Sutherland, "A model of information," in *Proc. 9th Nat. Comput. Security Conf.*, 1986.

Daryl McCullough was born in Clarksville, TN, and grew up in Rome, GA. He received the Bachelor's degree in physics from Northwestern University, Evanston, IL, in 1980, and the Master's degree in physics from Iowa State University, Ames, in 1986.

Since 1984 he has worked at Odyssey Research Associates in Ithaca, NY. There he has worked in the fields of computer security and program verification.