Security Models and Information Flow

John McLean

Center for High Assurance Computer Systems
Naval Research Laboratory
Washington, D.C. 20375

We develop a theory of information flow that differs from Nondeducibility's, which we see is really a theory of information sharing. We use our theory to develop a flow-based security model, FM, and show that the proper treatment of security-relevant causal factors in such a framework is very tricky. Using FM as a standard for comparison, we examine Noninterference, Generalized Noninterference, and extensions to Noninterference designed to protect high-level output, and see that the proper treatment of causal factors in such models requires us to consider programs as explicit input to systems. This gives us a new perspective on security levels. The Bell and LaPadula Model, on the other hand, more successfully models security-relevant causal information although this success is bought at the expense of the model being vague about its primitives. This vagueness is examined with respect to the claim that the Bell and LaPadula Model and Noninterference are equivalent.

1. Introduction

There is a general belief in the security community that the correct explication of security should be formulated in terms of Shannon-style information flow.[1] This belief has been explicitly expressed at foundational workshops on computer security and is implicit when we consider the history of general security models, a history whose sequence of products includes the Bell and LaPadula Model, Noninterference, and Nondeducibility.

Despite this general trend to more information-theoretic models of security, a security model that is based purely on information flow is still forthcoming. The general security model that comes closest, Nondeducibility, is really based on a theory of information sharing, which is more appropriate to compartmentalization than to security. Noninterference and its derivatives, including Restrictiveness, incorporate a sense of information flow that is more intuitive, but there has been little work done in developing a method for evaluating such models.

This paper formulates a security model based on information flow and shows that when developing such a model, one must be very careful that the model properly treats all security-relevant causal factors. This gives us a new perspective from which we can evaluate other general security models. Using our new model as a standard for comparison, we examine Nondeducibility, Noninterference, and the Bell and LaPadula model. We also examine the relationship between Noninterference and the Bell and

1. The modifier "Shannon-style" is to distinguish our use of information flow, which is based on information theory, from the type of information flow that is detected by flow tools. These two senses of the term, though related, are not identical.

LaPadula Model.

2. A Security Model Based on Information Flow

The general security model that is most self-consciously based on information theory is Sutherland's Nondeducibility Model [16]. This model states that information flows in a system from high-level objects to low-level objects if and only if some possible assignment of values to the low-level objects in the state is incompatible with a possible assignment of values to the state's high-level objects.[2] More formally, for any assignments H and L to a system's high-level objects and low-level objects, respectively, there is no information flow from the high-level objects to the low-level objects if and only if $p(H)>0$ & $p(L)>0 \rightarrow p(H|L)>0$.[3] Although intuitively appealing, this definition does not capture information flow but rather information sharing. As such it is more relevant to compartmentalization than to security. Security allows for information flows from low-level objects to high-level objects; Nondeducibility does not.[4] From the fact that $p(H|L)p(L) = p(L|H)p(H)$, it follows that $p(H)>0$ & $p(L)>0 \rightarrow p(H|L)>0$ if and only if $p(H)>0$ & $p(L)>0 \rightarrow p(L|H)>0$.

Nondeducibility accommodates its bidirectional concept of information flow by limiting the objects we are to consider when evaluating a system's security. It compensates for the loss of information about system state by extending the notion of an assignment to a sequence of the different values assumed by these objects, although the sequence does not contain information about how long a particular value was assumed. According to Nondeducibility, a system is secure if and only if $p(H)>0$ & $p(L)>0 \rightarrow p(H|L)>0$ where H is an assignment sequence to the system's high-level input port and L is an assignment sequence to the system's low-level input and output ports. As we shall see in the next section, protecting only high-level input is insufficient for ensuring security since in many systems high-level output is generated solely from low-level input. More pertinent to our discussion here is that Nondeducibility's view of information flow seems to render it incapable of even being extended to model a system where high-level output can be generated from low-level input. It cannot distinguish between permissible flows from low-level output to high-level output, as, e. g., in auditing, and nonpermissible flows that run in the other direction.

The assumption that high-level output cannot be generated solely from low-level input is not the only assumption forced upon Nondeducibility by its theory of information flow. For example, consider a system in which a low-level user is required to give as input an arbitrary number from some set S. This number is echoed as high-level output to a second user who must then give as high-level input a different member of S. We can think of these inputs as constituting unique identifiers, but the system's purpose is not important. What is important is that for any number $n \in S$, if *low* is the system's low-

---

2. Here and throughout this paper we use "object" in its broadest sense. System objects include files, programs, and I/O devices.

3. Sutherland's formulation is in terms of model theory, but ours is equivalent to his.

4. One may argue on implementation grounds that unless we are content with broadcasting we must accept two-way information flow whenever there is one-way flow. However, it is, at the very least, premature to conclude that our theory of security should rule out unidirectional flow as impossible.

level input port and *high* is the system's high-level output port then $p(low=n)>0$ and $p(high=n)>0$, yet $p(high=n \mid low=n) = p(low=n \mid high=n) = 0$. Our system is intuitively secure since the real information flow is from the low-level user to the high-level user and not vice-versa. However, this is a distinction Nondeducibility cannot make. It can avoid the problem only by being restricted to systems that are input total. Whether or not such a restriction has other justification, it should not be forced upon a model of security by the model's concept of information flow.

It may seem that any information theoretic approach must accept bidirectional information flow as inevitable. After all, $p(L \mid H) = p(L)$, i. e. low-level information is independent of high-level information, if and only if $p(H \mid L) = p(H)$, i. e. high-level information is independent of low-level information. However, we can avoid this conclusion by taking time into account in our security model and viewing information as flowing from H to L only if H assigns values to objects in a state that precedes the state in which L makes its assignment. Let us refer to the assignment of values of all the low-level objects and high-level objects in state t as L_t and H_t, respectively. Security does not require that $p(H_t \mid L_{t-1}) = p(H_t)$, i. e. that knowledge of the values of low-level objects in a state gives us no additional information about the values of high-level objects in the successor state. Such a model would rule out high-level auditing of low-level users where whenever a low-level file, l, assumes a new value in some state, t, the high-level audit file is changed so that its value in state $t+1$ reflects l's value in state t. Instead, security requires that the values of the low-level objects in a state be independent of the values high-level objects have in the previous state, i. e., previous values for high-level objects do not affect what a low-level user can see now.[5] Formally, this requirement is that $p(L_t \mid H_{t-1}) = p(L_t)$ or equivalently, that $p(H_{t-1} \mid L_t) = p(H_{t-1})$. Although such a model requires that knowing L_t does not give users any information about H_{t-1} that they do not know initially, it does not rule out the possibility that if users consider, e. g., L_{t-1} and L_t together, they can learn something new about H_{t-1}. To address this concern, we require that $p(L_t \mid H_{t-1} \;\&\; L_{t-1}) = p(L_t \mid L_{t-1})$.

Such a model has several advantages over Nondeducibility. First, it prohibits information flow from H to L without ruling out possible flows of information from L to H. Although $p(L_t \mid H_{t-1} \;\&\; L_{t-1}) = p(L_t \mid L_{t-1})$ if and only if $p(H_{t-1} \mid L_t \;\&\; L_{t-1}) = p(H_{t-1} \mid L_{t-1})$, the latter equality prohibits information only from flowing backwards from L_t to H_{t-1}, something presumably prohibited by causal considerations anyway. It does not prohibit information from flowing from L_t to, e. g., H_{t+1}. Second, it rules out systems in which low-level users can gain probabilistic knowledge about high-level events even if they cannot rule out any particular high-level event as being impossible. Finally, if we interpret H and L to include information, not only about object values, but also about user-detectable system resources consumed in generating those values, our model prohibits timing channels.

It may seem that by focusing on H_{t-1} our model fails to protect L_t from H_s where $s < t-1$ or $s \geq t$. We do not consider, e. g., H_{t-2} since we assume that for any high-level

---

5. For simplicity, we do not here consider the passing of information by using the existence of high-level objects as a channel. To prohibit such flows we can treat the name space of high-level objects as the value of an additional high-level object.

object, h, h_{t-2} can affect L_t only if h_{t-2} affects L_{t-1} or if H_{t-1} affects L_t. This amounts to the assumption contained in any model based on objects that a state can affect its successor only *via* information stored in its objects. Similarly, protecting L_t from future and current values of high-level objects is not necessary. If $p(L_t \mid H_s) \neq p(L_t)$ where $s > t$, then causal considerations force us to conclude that the flow is from L_t to H_s and not vice-versa. Future states of a state machine cannot causally affect previous states. If $p(L_t \mid H_t) \neq p(L_t)$, then causal considerations force us to conclude that the flow is not between H_t and L_t, but rather to both H_t and L_t from objects in a previous state. A state cannot not causally influence itself.

The problem with our model is not that it is too weak, but rather that it is too strong. It fails to adequately address the fact that for a security violation to take place, not only must the value of L_t be statistically correlated with the value of H_{t-1}, but the value of H_{t-1} must have exerted some causal influence on the value of L_t. That is, L_t must have its value *because* H_{t-1} has a particular value. It cannot be the case that the the statistical correlation arises simply because the values of both are caused by some other low-level event.

As an example, consider a system with a very strict auditing requirement. To prevent low-level users from crashing the system before some particular piece of data they have have written to a low-level file has been recorded in a high-level audit file, the system requires that the audit file be written first, i. e., a program can write a value X to a low-level object only if it has previously written X to the high-level object that constitutes the audit file. This "audit first" requirement that a low-level object can assume a value X in system state t only if the high-level audit file has the value X in the system's predecessor state $t-1$ gives rise to a statistical dependency between the value of the low-level file in state t and the high-level file in state $t-1$. Yet, the requirement does not represent a security violation since the value of the low-level file is not causally influenced by the high-level file. The moral is that for a true security breach to occur, we need both a statistical dependency and a causal dependency. A statistical dependency, by itself, is not enough.

This fact does not imply that an adequate information-theoretic model of security is impossible. It may be the case that although our model, which considers only information flow between a state and its successor state, fails to sufficiently model all security-relevant causal connections, there is a more inclusive information flow model that succeeds. For example, consider the model, which we shall call the *Flow Model* (FM), that states that a system is secure only if $p(L_t \mid H_s \ \& \ L_s) = p(L_t \mid L_s)$ where H_s and L_s are the sequences of values assumed by high-level objects and low-level objects, respectively, in every state that precedes t.[6] One could argue that if we know the value of L_{t-2}, we can deduce the values of the audit file in state $t-1$ and then deduce the value of L_t from this information and, possibly, L_{t-1}. Hence, the value of our audit file in state $t-1$ adds no new information to that contained in the values of the low-level objects in states

6. It should be noted that this notion of sequence, which we shall use throughout the rest of this paper, differs from the sequences used above when describing Nondeducibility in that it records an object's value for each state, not simply the sequence of value changes that an object has undergone.

$t-1$ and $t-2$.

Even if FM can handle this particular example of a statistical correlation between high-level objects and low-level objects that is not a security violation, we have not shown that it can handle more elaborate examples. FM's correctness depends on the assumption that if we know the entire set of objects that comprise a system and the history of all of these objects, we have the machinery necessary to separate those statistical correlations between high-level objects and low-level objects that are security violations from those that are not. Rather than pursuing this issue here, we simply note that when formulating any security model based on information flow, we must be careful that the model adequately addresses all security-relevant causal considerations. For the rest of this paper we shall use FM as a standard with which we can evaluate other security models based on information flow.

3. Noninterference and Information Flow

Like Nondeducibility, Goguen and Messeguer's Noninterference Model is also heavily influenced by information theory [4]. In its original form, a system was said to be noninterfering if its low-level output was independent of its high-level input in the sense that for any system with output function $out(u,I)$, whose value is the output generated by input history I to user u, $out(u,I) = out(u,I^*)$, where I^* is I purged of all inputs from users with security levels that are greater than u's.

When compared with FM, Noninterference does very well, although its domain is limited. Noninterference views a system as comprising four objects: $high-in$, $low-in$, $high-out$, and $low-out$, which are high-level and low-level input ports and output ports, respectively. Since Noninterference assumes that system output is completely determined by system input, probabilistic considerations are irrelevant. Nevertheless, realizing that the relevant probabilities are limited to the domain $\{0,1\}$, we can formulate Noninterference as the requirement that $p(low-out_t \mid high-in_S \ \& \ low-in_S) = p(low-out_t \mid low-in_S)$ where x_S denotes the sequence of values x has assumed for all times previous to t.[7] Although on the surface this requirement differs from FM by not considering the object $low-in$ on the left-hand side of the first probability or the objects $high-out$ and $low-out$ on the right-hand side, this omission follows from Noninterference's assumptions that $low-in$ cannot be influenced by the system, that output is determined by input, and that high-level output must be generated from high-level input and therefore, cannot be compromised without also compromising high-level input.

For future reference let us list Noninterference's assumptions that will concern us:

(1) Systems initially contain programs. System input is data to these programs and system output is the result of these programs operating on this data.

---

7. This formulation only approximates Noninterference. The formulation, like FM, requires that $high-in_S$ not add any new information to $low-out_t$ not already contained in $low-in_S$. Noninterference is more properly viewed as the requirement that $high-in_S$ not share any information with $low-out_t$. This does not seem to be a problem for Noninterference since it is hard to imagine $high-in_S$ and $low-out_t$ sharing information without $high-in_S$ causally affecting $low-out_t$. However, we shall see that it does cause problems for some of Noninterference's extensions.

(2) System programs are deterministic.

(3) Systems cannot generate high-level output solely from low-level input.

(4) Given a system, its output to a user can be determined solely by its input history.

Assumption (1) is a fundamental assumption of the trace-based specification methodology on which Noninterference is based [2, 11]. This methodology assumes that a system is a set of programs that takes input data and returns output data. A system's identity is the set of programs that constitute the system and determine its input/output behavior. Assumption (2) follows from the fact that *out* is a function. Assumption (4), which follows from the fact that *out* takes only u and I as arguments, rules out any system whose output cannot be determined without knowledge of previous output. As such, Assumption (4) follows from Assumption (2). Assumption (3) is necessary if Noninterference is to be an adequate security specification since, like Nondeducibility, Noninterference protects only inputs.

Noninterference's adequacy stands or falls with these assumptions. Unfortunately, Assumption (3) is not generally true [14]. Systems that generate cryptographic keys convert low-level input seeds into high-level output in their normal course of operation.[8] In less esoteric realms, many systems that perform resource-intensive analysis, such as the analysis required by space-based sensing systems or by marketing analysis systems, also convert low-level input into high-level output. The security value gained by data in such systems is due to the time involved in system processing or the nature of the processing, itself, not the input. Since Noninterference protects only high-level input, it does not protect the information such systems generate. For example, given any key generating system, we could append a low-level output channel that repeats all high-level output without violating Noninterference since our system does not have any high-level input to protect.[9]

The problem of protecting high-level output is even more pressing in the setting of McCullough's generalization of Noninterference designed to include nondeterministic systems [10]. In this model, a system is noninterfering if for every legal trace of the system and every alteration we can make to that trace by deleting or inserting high-level inputs, there is a legal trace that is equivalent to the first trace except perhaps with respect to high-level outputs.[10] Generalized Noninterference does not simply allow for nondeterminism, i. e. for a sequence of inputs to be compatible with different outputs; it also allows for output to affect a trace's set of legal futures. For example, there is nothing in Generalized Noninterference to rule out a system where *low-in(a).low-out(x)*, *low-in(a).low-out(y)*, and *low-in(a).low-out(x).low-in(b).low-out(c)* are all legal, but *low-in(a).low-out(y).low-in(b).low-out(c)* is not. In such a system we cannot determine the

8. This example was suggested to me by Robert Morris.

9. The problem of protecting high-level data that is not generated from high-level input is not a mere theoretical contrivance. The LOCK developers found it pressing enough to supplement their Noninterference model with a separate access control model [3].

10. McCullough actually rejected this definition of security, now called *Generalized Noninterference*, in favor of a stronger property called *Restrictiveness* since Generalized Noninterference is not composable. However, the details of Restrictiveness do not concern us here. Everything we say in this paper about Generalized Noninterference applies to Restrictiveness as well.

output generated by the input sequence *low-in(a).low-in(b)* without knowing the output generated by *low-in(a)*. Hence, Generalized Noninterference rejects not simply Assumption (2), but Assumption (4) as well. However, as is the case with Noninterference, for Generalized Noninterference to be an adequate specification of security, we must make the assumption, which is not generally true, that a program cannot generate high-level output if the program is given solely low-level input.

Extensions to Generalized Noninterference that address this problem by protecting high-level output have been developed by Guttman and Nadel [5] and by McLean and Meadows [14]. Both extensions require that a trace's high-level outputs cannot interfere with its low-level outputs any more than its high-level inputs can. Since both extensions are formulated in the setting of Generalized Noninterference, they are not all that helpful with the problem of protecting high-level outputs in the setting of simple Noninterference. A more serious problem appears when we consider the extensions *vis-a-vis* FM. It may seem that both extensions are equivalent to FM insofar as they can both be approximately characterized by the requirement that $p(low-out_t \mid high-in_s \ \& \ high-out_s \ \& \ low-in_s) = p(low-out_t \mid low-in_s)$ if we realize that this characterization is stronger than either extension since it includes probabilistic considerations that Generalized Noninterference, the Guttman-Nadel extension, and the McLean-Meadows extension all ignore.[11] However, this is not the case. FM prohibits $high-out_s$ from adding knowledge to $low-out_t$ not previously contained in $low-in_s$; the Guttman-Nadel extension and the McLean-Meadows extension more closely approximate the stronger requirement that $high-out_s$ not share any information with $low-out_t$. In other words, neither extension takes into account the fact that sometimes there is nothing wrong for there to be an information-theoretic link between high-level output and low-level output. Modifying our "audit first" example slightly, consider a system that in order to prevent users from crashing the system before some particular piece of low-level output they have just generated is sent as high-level output to an audit file, requires the audit file to be written first. In such a system, low-level output is legal only if it has already appeared as high-level output.

Although such systems violate both extensions to Generalized Noninterference, we have seen above in our development of FM that they do not violate security. For a true security breach to occur we need both a statistical dependency and a causal dependency between the two outputs. Hence, Generalized Noninterference faces two problems. First, it faces the problem faced by Nondeducibility and any form of Noninterference that protecting high-level input is not always sufficient to protect high-level output. Second, since Generalized Noninterference rejects Assumption (4) of Noninterference, any extension to it that protects high-level outputs faces the problem faced by other information-theoretic models of security that causal dependencies must be adequately modeled to distinguish between statistical correlations among outputs that are security violations and those that are not.

---

11. The price of ignoring probabilistic considerations should not be underestimated. For example, a system with low-level commands to nondeterministically print on a low-level terminal either previous high-level input or a randomly generated character string satisfies Generalized Noninterference but is clearly not secure. On the other hand, encryption systems fail to satisfy Noninterference but can satisfy FM if we treat encryption as introducing random noise.

To correctly model these causal dependencies, we need to consider the programs that connect trace events. This can be done within an input/output-based specification methodology only by considering programs as input. In other words, any formulation of Noninterference that rejects Assumption (4) must also reject Assumption (1). The result is that a system is no longer regarded as a set of programs that operate on input data, but rather as a primitive interpreter that operates on input programs and data. Fortunately, by rejecting Assumption (1) we can solve our first problem as well.

It should be noted that rejecting Assumption (1) and regarding all programs that are executed as system input does not necessitate changing the syntax of Noninterference or any of its derivatives. For example, traditionally a Noninterference specification of a system that receives low-level input L and returns to user u high-level output H, which is the result of applying some program P to L, would be specified by saying that $out(u, low-in(L))=H$. Now, we also have to regard P, the program the performs the analysis on L, as being input as well. Hence, our specification would say that $out(u, low-in(L).high-in(P))=H$. For Generalized Noninterference, our specification would say that if $low-in(L).high-in(P)$ is legal, then $low-in(L).high-in(P).high-out(H)$ is legal. Note that we regard P as being high-level. In general, on this approach high-level output requires high-level input either in the form of data or in the form of a program. In other words, we extend Assumption (3) to require that our system initially contains neither high-level data nor high-level programs. Note also that we regard a program as being high-level, not necessarily because the program, itself, is classified, e. g. a key-generating program, but also if the application of the program can generate high-level output from low-level input, e. g., programs that are not classified but are extremely expensive to run. This necessitates our re-thinking what a high-level security level means in such contexts.

Given this simple addition to Noninterference, we can solve our problem of protecting high-level output. We assume that high-level output that must be protected cannot be generated from low-level data unless that data has been operated on by a high-level program. Hence, regarding the program as further necessary input, high-level output that must be protected can be generated only from high-level input. Hence, protecting high-level input is sufficient to assure the necessary protection of high-level output.[12] The fact that we are once again faced solely with the task of protecting input also solves our second problem. The high-level output that can interfere with low-level output is exactly the high-level output that was not generated from any high-level input (i. e. output generated solely from low-level input but is, nevertheless, sent to high-level users) and is, hence, not intrinsically "high". If we like, we could explicitly distinguish between output that is high-level, in itself, e. g. output that is generated from high-level input or from a high-level program, and output that is high-level simply because it is sent to a high-level user or file, e. g., low-level data that is sent to a high-level audit file. However, there is no need to make this distinction explicit in our Noninterference specification.

---

12. Regarding programs as explicit input also permits us to solve another problem faced by Noninterference: the fact that some systems, e. g. encryption systems, generate low-level output from high-level input. By modeling the programs of such systems explicitly, we can treat some programs as being trusted and explicitly exclude them from our requirement of Noninterference.

4. The Bell and LaPadula Model and Information Flow

By comparing FM to Nondeducibility, Noninterference, and several of the latter's derivatives, we have unearthed several inadequacies. In the case of Noninterference and its derivatives, we have also seen how to correct these inadequacies. In this section we consider to what extent the problems faced by these models are also faced by the access control model of Bell and LaPadula (BLP) [1] and the relation between BLP and Noninterference.[13] Unfortunately, the question of whether BLP adequately models all security-relevant causal information does not permit a conclusive answer since the primitives of BLP do not have a rigorous general semantics.[14] If we interpret the statement "program P *read*s file F" as implying that the value of F can causally influence the behavior of P and the statement "program P *write*s file F" as implying that P can causally influence the value of F, then BLP comes very close to capturing the causal dependencies necessary for security. On this view *read* and *write* are basically causal notions and not based solely on statistical correlations.[15] Further, insofar as there is any consensus in the security community as to what *read* and *write* mean, the consensus seems to be consistent with such an interpretation. This is the source of BLP's intuitiveness as a security model. It is also not surprising that BLP avoids any problem that is raised by a purely information-theoretic explication of security when we consider that of all the general security models, BLP is that model that is least indebted to information theory. However, we must take BLP's correctness in this sense with a grain of salt since there is nothing in the model, itself, that rules out more bizarre interpretations of *read* and *write*.

The looseness of BLP's primitives is the source of the model's great flexibility. Although Noninterference is also a model without a formal semantics, there seems to be little, if any, disagreement as to what the primitives of Noninterference mean, i. e., what constitutes input and output. Unfortunately, Noninterference, in general, is too strong: in most systems we are willing to tolerate high-level input having potentially security-compromising effects on low-level output, e. g., with respect to operating system messages concerning available storage, device status, etc. Since disallowing all such information flow may lead to performance degradation we are unwilling to accept, we may permit the flow but require that it be monitored. BLP, on the other hand, permits us to formulate a policy that allows all flows (if, e. g., we interpret the symbol *read* as a relation that holds between no program-object pair) or a policy that allows no flows (if, e. g., we interpret the symbols *read* and *write* as denoting relations that hold between every program-object pair): hence, the need for covert channel analysis to determine the effectiveness of a particular interpretation of the models primitives for a particular system

13. We ignore problems with BLP raised in [12] and assume that we have transition restrictions as formulated in [13]

14. In what follows we do not treat the rules as part of the *formal* part of BLP. One could claim that the rules give a partial semantics to the primitive terms of BLP, but it is not clear how helpful such a semantics is. Without going too deeply into a well-discussed set of issues, we note that the BLP rules are allowed to change from model application to model application, and we are given no guidelines about how far the rule set is allowed to change [8]. As such, the set of rules that are contained in [1] are vacuous as far as providing a general semantics for the model's primitive terms.

15. As such, this view differs from the interpretations of *read* and *write* found in [17] and [9].

[15].

The claim that BLP captures security-relevant causal information, however loosely, that extensions to Generalized Noninterference do not may seem odd to many since it is widely believed that BLP and Noninterference are generally equivalent [6, 18]. The sense of "equivalence" used here seems to be that the set of systems which one model condones is the set of systems the other condones. It is worthwhile examining this thesis here since it is related to our claim that BLP takes security-relevant causal information into account that Noninterference-style explications do not. Just as the lack of a general semantics for BLP forces us to take the latter claim with a grain of salt, it also forces us to take the former claim with a grain of salt.

The equivalence thesis is supported by a proof published by Tom Haigh [6]. However, despite Haigh's proof, adherence to the equivalence thesis is puzzling when we consider it together with a second wide-spread belief, which we can call the *GM adequacy thesis*, that when one uses Noninterference one does not need to do a covert channel analysis [7]. What is puzzling is that nobody subscribes to the view which follows from these two theses, a view which could be called the *BLP adequacy thesis*: when one uses BLP, one does not need to do a covert channel analysis. In fact, everybody seems to regard the BLP adequacy thesis as being false.

One problem with the equivalence thesis, is that Haigh establishes it by making use of the assumption that we cannot use a file $f1$ to modify another file $f2$ without observing $f1$. The purpose of the assumption is to side-step the fact that BLP prohibits certains types of access which are allowed by Noninterference. In particular, Noninterference allows an unclassified user to modify a top secret file $f1$ on the basis of a secret file $f2$ as long as the user does not view either file. As an example, Noninterference permits the command **cp(f1,f2)** where **cp(x,y)** copies x to y. On the standard view, BLP would prohibit such a command since the operation would have to grant to the user *read* access to $f1$. The assumption contained in Haigh's proof forces Noninterference to prohibit the access as well.

There are two points to note here. First, the assumption that one cannot use one file to modify another file without observing the former goes against the heart of Noninterference. Noninterference is an input/output specification of security that meticulously avoids any mention of how security is to be implemented. Eliminating the ability to access without viewing a file is clearly an implementation decision. Nevertheless, we can incorporate the assumption in the theorem, itself, and say that any system which does not contain the ability to use a file $f1$ to modify another file $f2$ without observing $f1$ satisfies Noninterference if and only if it satisfies BLP.

However, this brings us to a more serious problem with the equivalence thesis. When we are talking about equivalence of security models, we are really talking about equivalence of the classes of systems models condone as secure. Hence, we are talking, not just about security models, but also about mappings of models to systems, for we cannot prove that a system satisfies a model without a mapping between model primitives and system primitives.

Returning to our copy command, it may be true that **cp(f1,f2)** is ruled out by the standard interpretation of BLP. However, there is nothing in the simple security condition and the *-property that forces us to adopt the standard interpretation. We could simply not view the command as having *read* access to $f1$ or *write* access to $f2$. As stated above, BLP is a *formal* model of computer security that contains no interpretations of its primitive terms. We could just as well state the simple security condition as the assertion "If x is a subject and y is an object and x has ϕ access to y, then the security level of x dominates that of y." The point is strengthened if we remember that *subject*, *object*, *security level* and *dominates* are formal terms as well, with the only restriction being that *security level* must be a finite lattice under *dominates*. A more accurate way of expressing the information contained in the simple security condition is "if Sx and Oy and Rxy, then $Lx \geq Ly$" under the *dominates* relation. The equivalence of such a formulation of BLP and Noninterference obviously depends on our interpretation of S, O, R, and L just as much as the incorporation of security-relevant causal considerations into BLP does.

5. Conclusion

We have seen that Nondeducibility's bidirectional concept of information flow is more appropriate to a theory of compartmentalization than to a theory of security. Nevertheless, we have been able to formulate a security model, FM, solely in terms of information flow that allows upward only flows of information, but we have seen that the proper treatment of security-relevant causal information is tricky in an information theoretic framework. Although this fact does not show that a purely information-theoretic model of security is impossible, it should make us careful when evaluating such a model. Using FM as a tool for evaluating other security models based on information flow, we have seen that Nondeducibility and extensions to Generalized Noninterference designed to protect high-level output generated from low-level input fail to take necessary causal information into account. The problems with Noninterference-style specifications can be fixed by considering programs as input. Insofar as the customary interpretation of BLP's primitives captures the necessary causal information, however loosely, BLP is not subject to the same criticisms. Finally, we have shown the relationship between this fact and the thesis that Noninterference and BLP are equivalent.

Acknowledgements

I wish to thank James Gray, Ira Moskowitz, and Todd Wittbold for their suggestions regarding this paper.

References

1. D. E. Bell and L. J. LaPadula, ''Secure Computer System: Unified Exposition and Multics Interpretation,'' MTR-2997, MITRE Corp., Bedford, MA, March, 1976. Available as NTIS AD A023 588.
2. S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe, ''A Theory of Communicating Sequential Processes,'' *J. ACM*, vol. 31, no. 3, pp. 560-599, July 1984.
3. T. Fine, T. Haigh, R. O'Brien, and D. Toups, ''Noninterference and Unwinding for LOCK,'' in *Proceedings of the Computer Security Foundations Workshop*, Franconia, NH, 1989.

4. J. A. Goguen and J. Meseguer, ‘‘Security Policies and Security Models,’’ in *Proc. 1982 IEEE Symposium on Security and Privacy*, pp. 11-20, IEEE Computer Society Press, April, 1982.

5. J. Guttman and M. Nadel, ‘‘What Needs Securing?,’’ in *Proc. Computer Security Foundations Workshop I*, pp. 34-57, Franconia, New Hampshire, June 1988.

6. J. T. Haigh, ‘‘A Comparison of Formal Security Models,’’ in *Proc. 7th National Computer Security Conference*, pp. 88-119, Gaithersburg, MD., Sept. 1984.

7. J. T. Haigh, R. A. Kemmerer, J. McHugh, and W. D. Young, ‘‘An Experience Using Two Covert Channel Analysis Techniques on a Real System Design,’’ *Proc. 1986 IEEE Symposium on Security and Privacy*, pp. 14-24, IEEE Computer Society Press, Oakland, CA., 1986.

8. L. J. LaPadula, ‘‘The ‘Basic Security Theorem’ of Bell and LaPadula Revisited,’’ *Cipher*, January 1989.

9. L. Marcus and T. Redmond, ‘‘A Semantics of Read,’’ in *Proc. Ninth National Computer Security Conference*, 1986.

10. D. McCullough, ‘‘Specifications for Multi-Level Security and a Hook-up Property,’’ in *Proc. 1987 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, April 1987.

11. J. McLean, ‘‘A Formal Method for the Abstract Specification of Software,’’ *J. ACM*, vol. 31, no. 3, pp. 600-627, July 1984.

12. J. McLean, ‘‘Reasoning about Security Models,’’ in *Proc. 1987 IEEE Symposium on Security and Privacy*, pp. 123-131, IEEE Computer Society Press, April 1987. Also in *Advances in Computer System Security*, vol. III, ed. R. Turn, Artech House, Dedham, MA, 1988.

13. J. McLean, ‘‘The Algebra of Security,’’ in *Proc. 1988 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, April 1988.

14. J. McLean and C. Meadows, ‘‘Composable Security Properties,’’ *Cipher*, Fall 1989.

15. J. McLean, ‘‘Specifying and Modeling Computer Security,’’ *IEEE Computer*, vol. 23, no. 1, pp. 9-16, January 1990.

16. D. Sutherland, ‘‘A Model of Information,’’ in *Proc. of the 9th National Computer Security Conference*, Gaithersburg, MD., September, 1986.

17. I. Sutherland, ‘‘Relating Bell-LaPadula-Style Security Models to Information Models,’’ in *Proceedings of the Computer Security Foundations Workshop*, Franconia, NH, 1988.

18. T. Taylor, ‘‘Comparison Paper between the Bell and LaPadula Model and the SRI Model,’’ *Proc. 1984 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Oakland, CA., 1984.