Security Models[*]

John McLean

1 Introduction

The term *security model* has been used to describe any formal statement of a system's confidentiality, availability, or integrity requirements. In this article we focus on the primary use of security models, which has been to describe general confidentiality requirements. We then give pointers to security model work in other areas.

2 Models of Confidentiality

Even if we limit ourselves to models of confidentiality, there are two related, but distinct, senses of the term *security model* in the computer security literature [McL90b]. In the more limited use of the term, a security model specifies a particular mechanism for enforcing confidentiality, called *access control*, which was brought over into computer security from the world of documents and safes. In the more general usage of the term, security models are specifications of a system's confidentiality requirements and are not "models" at all in that they specify security requirements without describing any particular mechanism for implementing these requirements. These models specify restrictions on a system's interface (usually its input/output relation) that are sufficient to ensure that any implementation that satisfies these restrictions will enforce confidentiality. In this section, we consider access control models and interface models in turn.

2.1 Access Control Models for Confidentiality

The access control model for confidentiality was first formulated by Lampson (see, e.g., [Lam71]) and later refined by Graham and Denning [GD72]. The structure of the model is that of a state machine where each state is a triple (S, O, M), where S is a set of subjects, O is a set of objects (which has S has a subset), and M is an access matrix which has one row for each subject, one column for each object, and is such that cell $M[s, o]$ contains the access rights subject s has for object o. These access rights are taken from a finite set of access rights, A. States are changed by requests for altering the access matrix, M. An individual machine in the model is called a *system*. Despite its simplicity, the model has had a long, useful life in computer security, based primarily on the work of Harrison, Ruzzo, and Ullman and the work of Bell and LaPadula.

[*]From *Encyclopedia of Software Engineering* (ed. John Marciniak), Wiley Press, 1994.

2.1.1 The HRU Model and Its Derivatives

Harrison, Ruzzo, and Ullman [HRU76] used Lampson's concept of an access control model to analyze the complexity of determining the effects of a particular access control policy. To make the problem precise they considered a particular access control model, called the *HRU Model*, which is similar to Lampson's but contains requests only of the following form:

```
if
    $a_1$ in $M[s_1, o_1]$ and
    $a_2$ in $M[s_2, o_2]$ and
        ...
    $a_m$ in $M[s_m, o_m]$ and
then
    $op_1$
    ...
    $op_n$
```

where each a_i is in A, and each op_i is one of the following primitive operations:

enter a into (s,o),
delete a from (s,o),
create subject s,
create object o,
destroy subject s,
destroy object o.
The semantics of the primitive operations are exactly what one would expect.

Given a system, an initial configuration Q_0, and a right a, we say that Q_0 is *safe* for a if there is no sequence of system requests that, when executed starting in state Q_0, will write a into a cell of the access matrix that did not already contain it. Harrison, Ruzzo, and Ullman proved two fundamental theorems about the complexity of the safety problem. The first is that the safety problem is decidable for *mono-operational* systems, i.e., systems in which every request has only one operation, and the second is that the safety problem is undecidable in general.

Theorem: There is an algorithm for determining whether or not a given mono-operational system and initial configuration of that system is safe for a given right a [HRU76].

Proof: The proof depends on the fact that for every sequence of requests $r_1, r_2, ..., r_n$ that leaks a from an initial configuration (S_0, O_0, M_0), there is a sequence of requests from that configuration that also leaks a, but which contains only **enter** requests except for an initial **create subject** request. To form this new sequence we first insert an initial **create subject** request to the beginning of $r_1, r_2, ..., r_n$ that creates a new subject, call it s_{init}, and then drop all **delete** and **destroy** requests from $r_1, r_2, ..., r_n$. This new sequence will still leak a since the conditional part of a request can test only for the presence of rights, and not for the absence of either rights or objects. Next, consider the rightmost **create subject** request. We can remove this request from the sequence and simply replace all references to the new subject in future requests by references to s_{init}. We continue with this procedure until we reach our initial **create subject** request, which we leave intact. Finally, we can remove the **create object** requests, rightmost first, again replacing all references to the new object in future requests by references to s_{init}. Finally, we can drop any **enter** requests that enter a right a_i into a cell that already contains it.

The resulting sequence will still leak a, but can be at most $l = (|A| \cdot (|S_0|+1) \cdot (|O_0|+1)) + 1$ requests long since each request, except the initial **create subject** request, must enter a new symbol from A into an access matrix cell and the number of cells in the matrix cannot be greater than $(|S_0|+1) \cdot (|O_0|+1)$. Hence, we can decide whether or not a system is safe by looking at all possible sequences of **enter** requests (preceded by an initial **create subject** request) of length less than or equal to l, and we are done. $\Box$

Theorem: The general problem of determining whether or not a given configuration of a given system is safe for a given right a is undecidable [HRU76].

Proof: The proof depends on the fact that the access control model is expressive enough to model any given Turing Machine. As an example, consider a Turing Machine that is in state q and reading position 3 of its tape. Further, assume that so far, the machine has written the symbols c_1, c_2, c_3, and c_4 on tape positions 1, 2, 3, and 4, respectively and that the machine has proceeded no further than the fourth position of the tape. Such a machine can be represented by a state machine whose access rights are the machine's states and tape symbols, plus the two additional rights **own** and **end**. The particular state of the machine would correspond to a state that had four subjects s_1, s_2, s_3, and s_4 corresponding to tape positions 1, 2, 3, and 4, respectively. $M[s_1, s_2]$, $M[s_2, s_3]$, and $M[s_3, s_4]$ would each contain the symbol **own** representing the fact that for $1 \leq i \leq 3$, position i is to the immediate left of position $i+1$ on the tape. For $1 \leq i \leq 4$, $M[s_i, s_i]$ would contain c_i, representing the fact that position i contains the symbol c_i. Finally, $M[s_3, s_3]$ would also contain the symbol q and $M[s_4, s_4]$ would also contain the symbol **end**, representing the facts that the machine is currently in state q reading tape position 3 and that it has not proceeded beyond tape position 4. Moves of the machine correspond to the requests that keep the system's state in correspondence with the machine's. The problem of determining whether a right is leaked now corresponds to the problem of determining whether a machine enters a certain state. If we equate that right with the machine's final state, we have reduced the safety problem to the halting problem, and we are done. $\Box$

The two theorems force upon modelers the following dilemma: on one hand, the general HRU model can express a wide variety of policies, but there is no general, computationally feasible way to determine the effects of these policies; on the other hand, there is a general, computationally feasible way to determine the effects of policies modeled via mono-operational HRU, but the model is too weak to express many policies of interest. For example, mono-operational systems cannot express policies that give subjects special rights to objects they create since there is no single operation that both creates an object and flags it as belonging to the creating subject.

Harrison, Ruzzo, and Ullman proved that the safety problem is decidable (although PSPACE complete) for systems that have no **create** requests [HRU76] and that it is decidable for systems that are both *monotonic*, i.e., contain no **destroy** or **delete** requests, and *monoconditional*, i. e., have only requests whose condition parts have, at most, one clause [HR78]. However, such systems are still very limited as far as expressing useful properties, and safety for even monotonic systems becomes undecidable if we allow biconditional requests

(requests whose condition parts have two clauses) [HR78]. Lipton and Snyder have shown that the safety problem for systems with a finite set of subjects is decidable, but it is computationally intractable [LS78].

Taking another tack, the *take-grant model*, introduced by Jones, Lipton, and Snyder [JLS76] and extended by Snyder [Sny81] and others [BS79, Bis84], has a linear time algorithm for safety, yet falls outside the known decidable cases of HRU. Closer to the HRU model is the *Schematic Protection Model* (*SPM*) developed by Sandhu [San88]. SPM, which contains security types, has a decidable subset that is more expressive than the take-grant model. An extension by Ammann and Sandhu [AS90] yields a model that is formally equivalent to monotonic HRU, but maintains positive safety results. More recently, Sandhu has had success with the Typed Access Matrix model(*TAM*), which introduces strong typing into HRU [San91]. Like HRU, monotonic TAM is undecidable. However, if we limit all commands to three parameters and avoid cyclic creates, the resulting model is decidable in polynomial time yet expressive enough to capture a wide variety of policies. This model is the current state of the art with respect to models for generalized access control policies.

2.1.2 The Bell and LaPadula Model and Its Derivatives

The HRU results show that it is often very hard to predict how access rights can propagate in a given access control model, even if we have complete knowledge of the programs that propagate those rights. A related, but distinct, problem arises from the fact that users are often unaware of everything a program operating on their behalf is doing. As an example, consider a user who accepts the right to execute another user's program. The first user may be unaware that executing the program will pass to the second user some entirely unrelated set of rights possessed by the first user. Programs such as these, which on the surface perform one function, e.g. provide editing capability for a file, but clandestinely perform another, e.g., distribute read rights for the same file, are known as *Trojan Horses*.

The Trojan Horse problem has led to a distinction between two types of access control policies: *Discretionary Access Control*, or *DAC*, and *Mandatory Access Control*, or *MAC*. Whereas DAC allows users to pass rights they possess to other users without constraint, MAC restricts how users can pass rights to other users. The existence of Trojan Horses that pass a user's rights without the user's knowledge is generally viewed as making DAC an insufficient method of access control in high-assurance environments.

The best known example of MAC policies are in the military with its well-known lattice of security levels that range from *top secret*, perhaps with various compartments, down through *secret* and *confidential* to *unclassified*. Rights to read a top secret file, for example, cannot be passed by any mechanism to an unclassified user. However, other examples of MAC policies pervade our everyday lives. For example, although an employee may grant an employer rights to view his or her salary, the employee would not want the employer to be able to pass these rights on to another employee. More generally, people are often willing to grant to a second party (perhaps a doctor or loan officer) the right to gather information about themselves, but only on the condition that the right to gather this information is not passed on to arbitrary third parties.

The best known security model for MAC is that of Bell and LaPadula [BL75]. Like the HRU model, the *Bell and LaPadula Model*, or *BLP*, employs subject, objects, rights, and an access control matrix. However, BLP differs from HRU in that the sets S and O do not change from state to state and the set A contains only two rights *read* and *write*.[1] BLP also introduces an unchanging lattice of security levels L and a function $F : S \cup O \rightarrow L$ which when applied to a subject or object in a state, yields the security level of its argument in that state. The set of states, V, in the model is a set of ordered pairs (F, M), where, as in HRU, M is the access matrix. A system consists of an intial state v_0, a particular set of requests R, and a transition function $T : (V \times R) \rightarrow V$ that transforms the system from one state to another when a request is executed. However, the most important difference between BLP and HRU is the introduction of a series of definitions which culminate in necessary and sufficient criteria for a system to be secure.

Definition: A state (F, M) is *read secure* (called *simple security* in [BL75]) if and only if for every $s \in S$ and every $o \in O$, $read \in M[s, o] \rightarrow F(s) \geq F(o)$.

Definition: A state (F, M) is *write secure* (called the **-property* in [BL75]) if and only if for every $s \in S$, $o \in O$, $write \in M[s, o] \rightarrow F(o) \geq F(s)$.

Definition: A state is *state secure* if and only if it is read secure and write secure.

Definition: A system (v_0, R, T) is *secure* if and only if v_0 is state secure and every state reachable from v_0 by executing a finite sequence of one or more requests from R is state secure.

Read security prohibits low-level users from gaining read access to high-level files. Write security prevents high-level Trojan Horses from copying the contents of high-level files to files to which low-level users can gain read access. Bell and LaPadula go on to prove the following theorem about secure systems, known as the *Basic Security Theorem*, or *BST*:

Theorem: A system (v_0, R, T) is secure if and only if (1) v_0 is a secure state and (2) T is such that for every state v reachable from v_0 by executing a finite sequence of one or more requests from R, if $T(v, c) = v^*$, where $v = (F, M)$ and $v^* = (F^*, M^*)$, then for each $s \in S$ and $o \in O$:

- if $read \in M^*[s, o]$ and $read \notin M[s, o]$ then $F^*(s) \geq F^*(o)$;
- if $read \in M[s, o]$ and $F^*(s) \not\geq F^*(o)$, then $read \notin M^*[s, o]$;
- if $write \in M^*[s, o]$ and $write \notin M[s, o]$ then $F^*(o) \geq F^*(s)$; and
- if $write \in M[s, o]$ and $F^*(o) \not\geq F^*(s)$, then $write \notin M^*[s, o]$.

Proof: Going from left to right, if the system is secure, then v_0 must be a secure state by definition. If there were some state v reachable from v_0 by executing a finite sequence of one or more requests from R such that $T(v, c) = v^*$ yet v^* does not satisfy one of the first two restrictions on T, then v^* would be a reachable state that failed to be read secure. If v^* failed to satisfy one of the second two restrictions on T, then v^* would be a reachable state that failed to be write secure. In either case, the system would not be secure.

Going the other direction, assume that the system is not secure. In that case, either v_0 must be a nonsecure state or there must be a nonsecure state reachable from v_0 by executing

[1]In Bell and LaPadula's formulation A also contains the rights *append* and *execute*, but their existence makes no difference in what follows. We have taken other similar liberties in simplifying the model where the simplifications do not affect the discussion. Readers interested in the original formulation are referred to [BL75].

a finite sequence of one or more requests from R. If v_0 is not a secure state, we are done. If v_0 is a secure state, let v^* be the first state in the request sequence that is not secure. This means there is a reachable, secure state v such that $T(v, c) = v^*$ where v^* is not secure. However, this is ruled out by the four restrictions on T, and we are done. □

Many have taken the BST as a justification for the definition of security offered by BLP. Although the argument is seldom made explicit, the belief is probably based on the fact that the BST seems to show that there is a natural notion of a *secure transition* (i.e., a transition that satisfies the four restrictions placed on T by the BST) that yields the same class of systems as the state restrictions of BLP yield. The fact that there are two natural definitions of security that yield the same class of systems gives credence to the belief that they are correct.

The trouble with this interpretation of the BST is that it is transparent with respect to the definition of secure state. An analogous theorem would hold no matter how we defined a secure state [McL85]. A truly secure transition must assure not only that every state reachable from a secure state is secure, but that the new state must be reachable in an intuitively secure manner. To see that BST fails to do this, consider the system Z whose initial state is state secure and which has only one type of transition: when a subject s requests any type of access to an object o, every subject and object in the system are downgraded to the lowest security level and access is granted. System Z satisfies BLP's notion of security, but it is obviously not secure in any meaningful sense [McL90b].

To address the problem raised by System Z, McLean defines a framework of security models that contain transition restrictions [McL90b]. A framework is a quadruple (S, O, L, A), where each element of the quadruple keeps the same meaning it has in BLP. As in BLP, a model within the framework is a set of state machines whose states are of the form (F, M) where F and M are as before. However, the framework contains a new function, $C : S \cup O \rightarrow P(S)$, which returns the set of subjects that are allowed to change the security level of its argument. As before, a system consists of an intial state v_0, a particular set of requests R, and a transition function T, but T is now the function $T : (S \times V \times R) \rightarrow V$ which gives the new state that results from a subject executing a request in a current state. Given this framework, we can define a secure system as follows;

Definition: A transition function T is *transition secure* if and only if $T(s, v, r) = v^*$, where $v = (f, m)$ and $v^* = (f^*, m^*)$, implies that for all $x \in S \cup O$ if $f(x) \neq f^*(x)$ then $s \in C(x)$.

Definition: A system (v_0, R, T) is *secure* only if (1) v_0 and all states reachable from v_0 by a finite sequence of one or more requests from R are (BLP) state secure, and (2) T is transition secure.

This framework forms a Boolean Algebra of models whose bottom (most restrictive) element is BLP with *tranquility*, the transition restriction that no security level can change, and whose top element (least restrictive element) is BLP with no restrictions on security level changes whatsoever [McL90b]. Since it gives only necessary conditions for a system to be secure, the policies in the framework do not contradict each other, and hence, sense can be made of the Boolean meet, join, and complement of policies. McLean also considers the framework that results when S is replaced by $P(S)$ and shows that the resulting framework forms a lattice of models that can be used to model multi-person rules such as the restriction that it takes two people to launch a missile or that payment for a shipment requires approval from both a receiving official, who vouches that the shipment was received, and an accounting official, who vouches that the charge is correct and has not previously been paid. These two frameworks constitute the current state of the art with respect to models for mandatory access control.

2.1.3 Problems with Access Control Models

An advantage of access control models is that they are intuitive and and can be implemented with high assurance. One provides a tamper-proof, non-bypassable *reference monitor* that controls all subjects' accesses to objects and is small enough to be susceptible to rigorous verification methods. However, it should be emphasized that determining a system's subjects, objects, read accesses, and write accesses is not as trivial as it first may seem. For example, consider a program that opens a high-level file for reading, reads a bit from the file, and then branches to one of two internal subroutines, write-one or write-zero, depending on what the high-level bit is. If write-one (write-zero) closes the high-level file, downgrades the program of which it is a subroutine, opens a low-level file for writing, and then writes 1 (0) to the low-level file, the result is a nonsecure information flow that violates BLP only if one regards the program counter, itself, an object [McL90b].

However, it is not the case that all such channels are so easy to detect and eliminate. Consider, for example, a reference monitor's response to a subject that attempts to write to a nonexistent file. If the reference monitor informs the subject of the mistake, it will allow a channel where a low-level subject attempts to write to a high-level file that a high-level Trojan Horse systematically creates and removes. If the subject is not informed of the mistake or if a file is automatically created when a subject attempts to write to a nonexistent file, then the subject will not be made aware of legitimate attempts to write which were thwarted through a minor typing error.

Channels such as these, first noticed by Lampson [Lam73] and now called *covert channels*, are caused by the difficulty of mapping an access control model's primitives to a computer system. The problem is exacerbated in distributed systems where a program must first write to a subsystem in order to read a file located on that subsystem. It is not surprising that such a problem is inherent in access control models if we consider the origin of the model. The concept of access control did not originate with computers, but rather, was brought over to computers from the paper and safe world. In that world, papers are kept in safes and access to safes by individuals are moderated by a security official. Covert channels are possible in such a scheme. For example, a high-level user can pass information to a low-level user by either approaching or not approaching a safe. However, such channels are not troublesome for three reasons. First, high-level users are trusted not to exercise such channels. This is partly because they have been subject to background checks, but primarily because if a high-level user wanted to communicate with a low-level user, he or she could do so in a much more efficient manner after hours. Second, any such channel would be extremely slow. Third, the monitoring security official could detect something funny if such a channel were often exploited.

When we turn to computers, covert channels become a real problem. First, we may trust users not to divulge information they are cleared to see, but given the existence of Trojan Horses, we can't trust all programs. Second, the speed of a computer raises the capacity of covert channels to a unacceptable level. Third, seldom is there a human who can determine that such a channel is being exploited in real time.

For this reason covert channel analysis goes hand-in-hand with the implementation of access control models. It assures us that a system's interpretation of the model's primitives is not too weak. Such analysis is usually based on tracing the information-flow paths of programs [Den76, Den82], checking programs for shared resources that can be used to transfer information [Kem83], or checking systems for clocks that can be used for timing channels [Wra91]. However, although such channels can often be detected, their detection comes at the end of the system development process when system changes are much more expensive to correct [Boe76]. It would be cheaper to rule out such channels from the beginning and make sure that they were never introduced into the system in the first place.

2.2 Interface Models of Confidentiality

Rather than specifying a particular method for enforcing security, interface models specify restrictions on a system's input/output relation that are sufficient for ruling out nonsecure implementations. It is up to the implementor to determine a method for satisfying the specification. Such an approach allows implementors more flexibility in designing and building systems, is more natural for dealing with networks, and, in general, does better with respect to covert storage channels. However, as we shall see, although the interface approach is relatively straight-forward with respect to deterministic systems, it becomes rather subtle when extended to nondeterministic systems

2.2.1 The Noninterference Model for Deterministic Systems

Most interface models for confidentiality are based on Noninterference, the restriction that high-level user input cannot interfere with low-level user output. The original formulation of Noninterference, due to Goguen and Meseguer [GM82], is based directly on the work of Feiertag [Fei80] and indirectly on earlier work by Cohen [Coh77] and by Popek and Farber [PF78]. Goguen and Meseguer consider a deterministic system whose output to user u is given by the function $out(u, hist.read(u))$ where $hist.read(u)$ is an input history (trace) of the system whose last input is $read(u)$, a read command executed by user u.[2] Security is defined in terms of purges of input histories, where a purge removes commands executed by a user whose security level is not dominated by u.

Definition: Let cl be a function from *users* to *security levels* such that $cl(u)$ is the clearance of u. Further, let *purge* be a function from $users \times traces$ to *traces* such that

- $purge(u, <>) = <>$, where $<>$ is the empty trace
- $purge(u, hist.command(w)) = purge(u, hist).command(w)$ if $command(w)$ is an input executed by user w and $cl(u) \geq cl(w)$, and
- $purge(u, hist.command(w)) = purge(u, hist)$ if $command(w)$ is an input executed by user w and $cl(u) \not\geq cl(w)$.

[2]In [GM82], *out* actually takes three arguments, u, *read*, and the state s reached by executing the commands in *hist*. Our formulation is identical for purposes of exposition and more in keeping with other interface models we will be discussing.

A system satisfies $Noninterference$ if and only if for all users u, all histories T, and all output commands c, $out(u, T.c(u)) = out(u, purge(u, T).c(u))$.

To help verify that a system satisfies Noninterference, Goguen and Meseguer developed a set of "unwinding conditions" that are sufficient for establishing Noninterference in state machines [GM84]. Although these conditions are relatively straight-forward to verify, their application depends on the development of a state machine model of the system under consideration. More recently, McLean has shown how to side-step the development of such a state machine and verify Noninterference directly [McL92]. These verification techniques help make Noninterference as useful, in practice, as BLP. Although verifying Noninterference, in general, may be harder than verifying BLP, there is no covert storage channel analysis remaining to do after the verification.

Since the primitives of BLP lack a precise semantics, one cannot precisely compare the two models [McL90a]. However, it can be noted that (1) in general BLP is weaker than Noninterference in that the latter prohibits many of the covert channels that the former would allow under the standard interpretation of its primitives, and (2) Noninterference is weaker than BLP in that it allows low-level users to copy one high-level file to another high-level file, which BLP would normally disallow as a high-level *read* by the low-level user. In both cases Noninterference seems to be closer to our intuitive notion of security than BLP.

In fact, Millen has shown in [Mil87] that for deterministic systems, Noninterference is practically perfect in that if input sequence X is noninterfering with output sequence Y and X is independent of the input from other users, then $I(X, Y) = 0$, where $I(X, Y)$ is the *mutual information* between X and Y and represents the information flow over the system from X to Y (see, e.g., [Jon79]). Of course, for Noninterference to rule out timing channels, time must be considered as part of the input and output alphabet.

The reason why Noninterference is only "practically perfect" is that, as shown originally by Sutherland [Sut86], it can be too strong. Consider, for example, a system, where user X and user V are each independently given an opportunity to give an input from the alphabet $\{0, 1\}$ and that Y receives as output $x \oplus v$, where x (v) is the input from X (V) if there is one, else 0. Clearly, X interferes with Y. For example, if $x = 1$ and $v = 1$, then $y = 1 \oplus 0 = 0$, but if we eliminate X's input, $y = 0 \oplus 0 = 1$. In general, we should not allow such a system if X and V were high-level users and Y were a low-level user since there are input sequences from V that would allow X to communicate with Y. (Remember that V could be a Trojan Horse using the system or a user whose interface to the system was under the control of a Trojan Horse.) However, if V's inputs were randomly distributed over $0, 1$, then V would, in effect, be providing perfect encryption for X's input, and $I(X, Y)$ would be 0. In such a case, X would be interfering with Y, but no information could flow since Y cannot detect X's interference. In this sense, Noninterference is possibly too strong in that it makes a worse-case assumption about the behavior of other users on the system. As such, it rules out cryptographic systems as being nonsecure.

Despite this limitation, Noninterference constitutes the current state of the art with respect to interface models for deterministic systems. However, it would be nice if we could apply Noninterference to nondeterministic systems as well. Although, ultimately, all systems may be (ontologically) deterministic, it is unreasonable to require that all system specifications (the system descriptions that will, in fact, be analyzed for security flaws) be (epistemically) deterministic. Further, the limitation to deterministic systems rules out probabilistic algorithms. In the next two sections, we examine ways of generalizing Noninterference to nondeterministic systems.

2.2.2 Possibilistic Models for Nondeterministic Systems

Before giving a nondeterministic version of Noninterference, we need a framework for describing nondeterministic systems. We could simply generalize the language in which we presented Noninterference and consider *out* to be a relation instead of a function, i.e., allow the same input to generate different output. However, to catch channels where information is passed by the order in which output is transmitted by the system, we will, instead, include outputs in the history itself. The resulting traces represent acceptable input/output behaviors, and a system is a set of acceptable traces. For example, a system in which a user can give as input either 0 or 1 and immediately receives that input as output is specified by the following set of traces: $\{<>, in(0), in(1), in(0).out(0), in(1).out(1), in(0).out(0).in(1), ...\}$. For simplicity, we assume that any prefix of an acceptable trace must also be an acceptable trace and that a user can give input at any time (although the system may choose to ignore it).

The obvious way to generalize Noninterference is to require that the purge of an acceptable trace be an acceptable trace, where the purge of a trace is formed by removing all high-level inputs from the trace. The problem with this definition is that the purged trace may not be unacceptable due to any security violations, but due to other system requirements. For example, consider the system described in the previous paragraph and assume that all input and output is high-level. Since the system generates no low-level output, it is trivially secure. Now, we have seen that $in(0).out(0)$ is an acceptable trace of the system. However, the purge of this trace, viz. $out(0)$, is not an acceptable trace since it contains an unsolicited output, which the system is not supposed to give.

At this point, the obvious approach is to keep the requirement that the purge of an acceptable trace be an acceptable trace, but redefine the purge operator so that it removes, not simply all high-level input, but all high-level output as well. This approach, however, also has problems. First, it is too strong in that it rules out any system where low-level input must generate high-level output. As a result, we would have to regard as nonsecure a system that secretly monitors low-level usage and sends its audit records as high-level output to some other system for analysis. A more severe problem is that it allows nonsecure systems. Consider a system in which the following traces are acceptable: $\{<>, highin(0), highin(1), lowout(0), lowout(1), highin(0).lowout(0), highin(1).lowout(1)\}$. This system satisfies our security property since if we remove all the high-level events from an acceptable trace, the result is an acceptable trace. However, it is not hard to come up with a scenario where a Trojan Horse acting on "behalf" of a high-level user can pass information to a low-level user using such a system. For example, if we assume that the high-level user always has the option if giving input before the next low-level output is generated, then information can be passed noiselessly. If the Trojan Horse wants to send a 0 or 1 to the low-level user, it simply gives the appropriate bit as input before the next low-level output is generated.

The problem with the approaches so far is that when we allow the same input to issue a variety of outputs to the low-level user, the requirement that the purge of an acceptable trace be an acceptable trace is too weak. We also need the requirement that high-level events can be introduced into an acceptable trace without rendering the resulting trace unacceptable to the system. For our last system to meet this stronger requirement, it would also have to regard the traces $highin(0).lowout(1)$ and $highin(1).lowout(0)$ as being acceptable, which would close the nonsecure channel. Such a requirement was not necessary when we considered only deterministic systems since if the insertion of a high-level input altered the low-level output of a trace, then we could not satisfy the requirement that the trace that would be the purge of both traces must have the same output as the original traces.

Of course, it would be too strong to require that any arbitrary insertion of high-level events into an acceptable trace must be acceptable. The high-level events, themselves, must be acceptable to the system, and we must take into account the fact that these new events (possibly in conjunction with existing low-level events) can alter the values of high-level outputs. These considerations lead us to the requirement that for any two acceptable traces, T and S, there is an acceptable trace R consisting of T's low-level events (in their respective order), S's high-level inputs (in their respective order), and possibly some other events that are neither low-level events from T nor high-level inputs from S. This property, known as *Nondeducibility*, was first put forward by Sutherland [Sut86] to capture the requirement that whatever the low-level user sees is compatible with any acceptable high-level input.

Although Nondeducibility is more general than Noninterference in that it does not assume determinism, it is equivalent to Noninterference if we limit ourselves to deterministic systems with only two users. However, Nondeducibility is strictly weaker than Noninterference if we consider deterministic systems with more than two users. For example, consider the system described in the previous section of this article where user Y receives as output the exclusive or of the input from user X and user V. We saw that in such a system user X interferes with Y's output, although Y may not be able to detect this. As such, this system fails to satisfy Noninterference with respect to X and Y. It does satisfy Nondeducibility, however, since any acceptable input sequence from X and any acceptable output sequence to Y can be combined into an acceptable trace by inserting suitable inputs from V.[3]

The trouble with Nondeducibility is that it is too weak. For example, consider a system where a high-level user H gives arbitrary high-level input (presumably a secret messages of some sort) and some low-level user L gives the low-level input, *look*. When L issues *look*, he or she receives as low-level output the encryption of H's input up to that time, if there is any, or else a randomly generated string. Such a system models an encryption system where low-level users can observe encrypted messages leaving the system, but to prevent traffic flow analysis, random strings are generated when there is no encrypted output. This system satisfies Nondeducibility since low-level users can learn nothing about high-level input. The problem arises when we realize it would still satisfy Nondeducibility even if we removed the encryption requirement. For example, given the high level input $highin(Attack\ at\ dawn)$, and the low-level trace $lowin(look).lowout(xxx)$, we can construct the legal system trace $lowin(look).lowout(xxx).highin(Attack\ at\ dawn)$. Similarly, given the acceptable traces $<>$ and $highin(Attack\ at\ dawn).lowin(look).lowout(Attack\ at\ dawn)$, we can construct a legal trace from the high-events of the former and the low-events from the latter, viz.

[3]In fact, there is a problem with our formulation since Y's output trace must be long enough to account for all of X's and V's inputs. This is an artifact of our formalism, however, and is not a problem for the original statement of the model. We can get around the problem within our formalism, but as we shall soon see, Nondeducibility has other problems that render any solution to this problem otiose.

lowin(*look*).*lowout*(*Attack at dawn*) since it is possible that the string "Attack at dawn" was randomly generated. This problem was first noticed by McCullough [McC87].

A second problem with Nondeducibility is that it is not composable. Composability is the second-order property that holds of a first-order property if and only if any composite system formed by connecting two subsystems that satisfy the first-order property in an appropriate way satisfies the first-order property as well. McCullough showed that Nondeducibility is not preserved by secure composition (i.e., composition in which outputs from one subsystem are connected to inputs of another subsystem only if the outputs and inputs have the same security level) [McC90].

Referring back to the definition of "Nondeducibility", we see that the cause of these problems is that it allows us too much freedom in constructing an acceptable trace R from the high-level inputs of an acceptable trace T and the low-level events from an acceptable trace S. We should require, not only that event order be maintained within T and S, but that we also be limited in how we can intersperse the events from the two different traces. In effect, we want to require not simply that there is some place in an acceptable trace where we can insert a high-level event and still obtain an acceptable trace, but that we can obtain an acceptable trace no matter where we insert a high-level event. This observation is the motivation for the following security property, known as *Generalized Noninterference*: given any acceptable system trace T and an alteration T_1 formed by inserting or deleting a high-level input to or from T, there is an acceptable trace T_2 formed by inserting or deleting high-level outputs to or from T_1 after the occurrence of the alteration in T made to form T_1 [McC87]. For example, consider the system described above where a low-level user monitors high-level output. As we noted, a possible trace for that system is *lowin*(*look*).*lowout*(*xxx*). If we alter this trace to obtain *highin*(*Attack at dawn*).*lowin*(*look*).*lowout*(*xxx*), we are left with an unacceptable trace that cannot be made acceptable by inserting or deleting high-level outputs after the occurrence of the inserted high-level input. Hence, the system fails to satisfy Generalized Noninterference.

Unfortunately, although Generalized Noninterference solves the first problem with Nondeducibility, it does not solve the second. Generalized Noninterference is not composable either [McC87]. To create a composable security property, we must be even more restrictive about how T_2 is formed. This leads us to the following definition, known as *Restrictiveness*: given any acceptable trace T and alteration T_1, formed by inserting or deleting a high-level input to or from T, there is an acceptable trace T_2 formed by inserting high-level outputs to or from T_1 after the occurrence of the alteration in T made to form T_1 and any sequence of low-level inputs that immediately follow the alteration to T [McC87]. For example, consider the acceptable trace $wxyz$ where z is a high-level output and both x and y are low-level inputs. Now consider the alteration we obtain by inserting a high-level input h after w to form $whxyz$. Generalized Noninterference would allow us to form an acceptable trace from this alteration either by removing z or by inserting a high-level output anywhere in the trace after h. Restrictiveness forces us to form an acceptable trace only by removing z or by inserting a high-level output somewhere after y. McCullough showed that this restriction to Generalized Noninterference, yields a composable security property [McC90].

Although Restrictiveness goes a long way toward providing a nondeterministic version of Noninterference, it is not problem free. One problem is that it is not preserved by many standard views of refinement [Jac89]. For example, consider a system where a high-level user can either input 0 or 1 and a low-level user can receive as output either 0 or 1. In other words, assume that the following set of traces are all acceptable $\{highin(0), highin(1), lowout(0), lowout(1), highin(0).lowout(0), highin(0).lowout(1), highin(1).lowout(0), highin(1).lowout(1)\}$. Since low-level output is compatible with any high-level input, the system is obviously Restrictive. If we consider the notion of refinement used in a number of software engineering paradigms, however, a perfectly correct implementation of this program could eliminate the nondeterminism contained in the specification and produce a program that accepted, e.g., only the following traces: $\{highin(0).lowout(0), highin(1).lowout(1)\}$. Such a program is not Restrictive. Since we can have functionally correct implementations of Restrictive specifications that are not, themselves, Restrictive, we must check for Restrictiveness at each level of the software development process. This leads to a major increase in the cost of software engineering.

This problem is caused by the fact that most of these methodologies view properties as sets of traces and view a program as satisfying a property if its acceptable traces are a subset of the property. The rub is that security properties such as Noninterference are not sets of traces, but rather properties of sets of traces – i.e., meta-properties. Properties such as this, which includes average response time as well, are not preserved by subsetting. Although there are specification/refinement methodologies that do a better job of preserving security under refinement, they have, so far, been applied only to relatively simple generalizations of Noninterference [McL92, Mea92].

A second problem with Restrictiveness is that it addresses only noise-free channels. For example, consider the system described above, but assume that the traces $lowout(0)$, $lowout(1)$, $highin(0).lowout(1)$, and $highin(1).lowout(0)$, although possible, occur with only a very small probability, say .0001. Assume further, that whenever a high-level input occurs, a low-level output immediately occurs with a very high probability. Although this system is Restrictive, it passes high-level information to low-level users at an extremely high rate. If a low-level user sees an output, he or she can be almost certain that that the output was given as a high-level input, and whenever a high-level user gives an input, it is almost certain that the low-level user will soon receive it as a low-level output. Such noisy or probabilistic channels are beyond the scope of the possibilistic approach to modeling we have so far considered.

2.2.3 Probabilistic Models for Nondeterministic Systems

The first models formulated explicitly to deal with probabilistic channels were put forward in 1990. They were the Flow Model (FM), formulated by McLean [McL90a], and P-Restrictiveness, formulated by Gray [Gra90]. FM is an extremely general model which Gray applied to a specific system description the next year, calling the system-specific interpretation AFM [Gra91]. In that same paper, Gray also introduced a new security model, Probabilistic Noninterference (PNI), which he compared with AFM. The year after that, Gray and Syverson produced a verification logic which supports the formal verification of systems implementing probabilistic models [GS92].

Both AFM and PNI regard a system as a 4-tuple $< S, I, C, O >$. S is a set of *information sources*, i.e., entities that introduce probabilistic behavior into the system. S normally consists of all system input channels and any internal random number generators. C is the set of system output channels. I and O are the alphabets of S and C, respectively. For simplicity, we will assume that S consists of two input channels, *highin* (input from the high-level user) and *lowin* (input from the low-level user), and that O consists of two output channels, *highout* (output to the high-level user) and *lowout* (output to the low-level user). At any time, the low-level user knows the history of the two low channels and the high-level user knows the history of all the system channels.

At any given time, t, the input to the system consists of the ordered pair $< highin_t, lowin_t >$, which we denote by in_t, and the output from the system consists of the ordered pair $< highout_t, lowout_t >$, which we denote by out_t. We denote the history of inputs and outputs up to and including t by $< in_1, in_2, ..., in_t >$ and $< out_1, out_2, ..., out_t >$, respectively.

We assume that there is a function $\hat{O}$ which intuitively gives the probability of a certain output occurring at time $t+1$ given the input and output histories up to time t, i.e., $\hat{O}(out_{t+1}| < in_1, in_2, ..., in_t >, < out_1, out_2, ..., out_t >)$ is the probability that the system will produce out_{t+1} as output at time $t+1$ given input history $< in_1, in_2, ..., in_t >$ and output history $< out_1, out_2, ..., out_t >$. We also assume that there is an analogous function $\hat{I}$ which intuitively gives the probability of a certain input occurring at time $t+1$ given the input and output histories up to time t. Although it is unrealistic, in general, to assume that a particular $\hat{I}$ correctly models a user's input, in the security properties that follow quantification is made over all such functions so we never need to assume that a particular $\hat{I}$ models any individual user. The reason for introducing $\hat{I}$ is to prevent high-level Trojan Horses from communicating information to low-level users via game-theoretic strategies (see, e.g., [WJ90]) and, more importantly, to define a probability measure on system events. Given $< S, I, C, O >$, $\hat{O}$, and $\hat{I}$, a probability measure, P, can be constructed that gives the probability of any event in which the system can engage.

To limit ourselves to systems in which high-level users do not pass information to low-level users outside the system (since if they do, computer security becomes moot), we require that low-level input at time t can depend only on previous low-level events and that, conditioned on previous history, the low-level input must be statistically independent of high-level input at time t. This requirement can be formalized by requiring that there are two probability measures H and L, called the *high environment behavior* and *low environment behavior*, respectively, such that (1) $H(highin_{t+1}| < in_1, ..., in_t >, < out_1, ..., out_t >)$ gives the probability of $highin_{t+1}$ being the high-level input at time $t+1$, (2) $L(lowin_{t+1}| < lowin_1, ..., lowin_t >, < lowout_1, ..., lowout_t >)$ gives the probability of $lowin_{t+1}$ being the low-level input at time $t+1$, and (3) $\hat{I}(in_{t+1}| < in_1, ..., in_t >, < out_1, ..., out_t >) = H(highin_{t+1}| < in_1, ..., in_t >, < out_1, ..., out_t >) \cdot L(lowin_{t+1}| < lowin_1, ..., lowin_t >, < lowout_1, ..., lowout_t >)$.

AFM, Gray's formalization of FM within this framework, is the requirement that given $< S, I, C, O >$ and $\hat{O}$, for any $\hat{I}$ that satisfies the secure environment criteria, $P(lowout_{t+1}| < in_1, ..., in_t > \& < out_1, ..., out_t >) = P(lowout_{t+1}| < lowin_1, ..., lowin_t > \& < lowout_1, ..., lowout_t >)$. Hence, AFM intuitively says that conditioned on low-level history low-level output at time t is statistically independent of previous high-level events.

To formalize PNI, we note that since P is determined by $< S, I, C, O >$, $\hat{O}$, and $\hat{I}$, while $\hat{I}$ is determined by H and L, it follows that P is determined by $< S, I, C, O >$, $\hat{O}$, H, and L. Given $< S, I, C, O >$ and $\hat{O}$, Gray defines PNI as the requirement that for any two high environment behaviors $H1$ and $H2$, any low environment behavior L, and any low-level event e, $P_{H1,L}(e) = P_{H2,L}(e)$, i.e., the probability measure constructed from $< S, I, C, O >$, $\hat{O}$, $H1$, and L assigns the same probability measure to e as the one constructed from $< S, I, C, O >$, $\hat{O}$, $H2$, and L. Hence, PNI intuitively says that the probability of a low-level event occurring is independent of any high-level user behavior.

Gray has shown that PNI is sufficient for guaranteeing that there is no information flow from high-level users to low-level users and that (A)FM is strictly stronger than PNI. In other words both models are sufficient to guarantee confidentiality, but there are systems that satisfy PNI, but fail to satisfy (A)FM. For example, consider a system in which low-level data is randomly generated. If such data appears as high-level output and then as low-level output at a later time, the system will satisfy PNI, but not (A)FM. Whether such systems show that (A)FM is too strong is debatable since it is unclear why low-level data should be regarded as high-level output. The real strength in (A)FM, however, is that it is easier to verify than PNI and can be used as a verification condition for PNI [GS92]. Whether we can build systems that conform to either model, and, if not, whether we can weaken the models to be more generally applicable are open research issues. If the answer to both questions is negative, we may be forced to abandon general interface models for computer security and retreat to access control, relying on covert channel analysis to detect probabilistic channels [MM92]. For the time being, (A)FM and PNI constitute the current state of the art with respect to such models.

3 Other Types of Models

As noted in the introduction to this article, not all security models address general confidentiality concerns. Although space does not permit a thorough discussion of other types of security models, we shall give pointers for where to look. Whereas confidentiality prohibits the unauthorized reading of information, availability prohibits the unauthorized withholding of information. It's concern is not that low-level users can read high-level files, but that they can prevent high-level users from accessing these files. There has been a fair amount of formal work in this area, first by Gligor[Gli83], and then by Yu and Gligor [YG90] and by Millen [Mil92]. The latter two models present resource allocators. The model of Yu and Gligor uses temporal logic to specify constraints on such an allocator, and the one by Millen uses a finite state machine framework.

When we turn to integrity, which prohibits the unauthorized modification of information, less progress has been made. The most famous integrity model to date was formulated by Clark and Wilson [CW87]. The basis of the Clark-Wilson Model is that controlled data items can be altered only by certain transactions and that such transactions may require collaboration from other people. The disadvantage of the model is that it is far from formal, and it is unclear how to formalize it in a general setting (although the framework described [McL90b] can be used to formalize the multi-person rule part of the model). Part of the problem is that although we seem to have a clear formal concept of confidentiality (via information theory), we have yet to develop a general one for integrity.

A different approach to dealing with these problems is to move from general models to application-specific models. The first model to try this approach was the Secure Military Message System Model [LHM84]. The constraints expressed in this model are not general security constraints on subjects and objects, but specific security constraints that a message system must meet in its handling of messages. Since then, the application-specific approach has been applied in a number of areas, most notably in the area of database security [DLS$^+$88].

References

[AS90] P. Ammann and R. Sandhu. Extending the creation operation in the schematic protection model. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1990.

[Bis84] J. Biskup. Some variants of the take-grant protection model. *Information Processing Letters*, 19(3), 1984.

[BL75] D. Bell and L. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Technical Report MTR-2997, MITRE, Bedford, MA, 1975.

[Boe76] B. Boehm. Software engineering. *IEEE Transactions on Computers*, C-25(12):1226–41, December 1976.

[BS79] M. Bishop and L. Snyder. The transfer of information and authority in a protection system. In *Proc. 7th ACM Symposium on Operating Systems Principles*, 1979.

[Coh77] E. Cohen. Information transmission in computational systems. *ACM SIGOPS Operating System Review*, 11(5):133–139, November 1977.

[CW87] D. Clark and D. Wilson. A comparison of commercial and military computer security policies. In *Proceedings of the 1987 IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1987.

[Den76] D. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, May 1976.

[Den82] D. Denning. *Cryptography and Data Security.* Addison-Wesley, Reading, Massachusetts, 1982.

[DLS$^+$88] D. Denning, T. Lunt, R. Schell, W. Shockley, and M. Heckman. The sea view security model. In *Proceedings of the 1988 IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1988.

[Fei80] R. Feiertag. A technique for proving specifications are multilevel secure. Technical Report CSL-109, SRI, Menlo Park, CA, 1980.

[GD72] G. Graham and P. Denning. Protection – principles and practice. In *Proc. Spring Joint Computer Conference.* AFIPS Press, 1972.

[Gli83] V. Gligor. A note on the denial-of-service problem. In *Proceedings of the 1983 IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1983.

[GM82] J. Goguen and J. Meseguer. Security policies and security models. In *Proceedings of the 1982 IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1982.

[GM84] J. Goguen and J. Meseguer. Unwinding and inference control. In *Proceedings of the 1984 IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1984.

[Gra90] J. Gray. Probabilistic interference. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1990.

[Gra91] J. Gray. Toward a mathematical foundation for information flow security. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1991.

[GS92] J. Gray and P. Syverson. A logical approach to multilevel security of probabilistic systems. In *Proceedings of the 1992 IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1992.

[HR78] M. Harrison and W. Ruzzo. Monotonic protection systems. In R. DeMillo, D. Dobkin, A. Jones, and R. Lipton, editors, *Foundations of Secure Computation*, pages 337–365. Academic Press, New York, 1978.

[HRU76] M. Harrison, W. Ruzzo, and J. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, August 1976.

[Jac89] J. Jacob. On the derivation of secure components. In *Proceedings of the 1989 IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1989.

[JLS76] A. Jones, R. Lipton, and L. Snyder. A linear time algorithm for deciding security. In *Proc. 17th Annual Symp. on Found. of Comp. Sci.*, 1976.

[Jon79] D. S. Jones. *Elementary Information Theory.* Oxford University Press, Oxford, 1979.

[Kem83] R. Kemmerer. Share resource matrix methodology: An approach to identifying storage and timing channels. *ACM Transactions on Computer Systems*, 1(3):256–277, August 1983.

[Lam71] B. Lampson. Protection. In *5th Princeton Symposium on Information Sciences and Systems*, March 1971. Reprinted in *ACM Operating Systems Review*, 8(1) (1974).

[Lam73] B. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, October 1973.

[LHM84] C. Landwehr, C. Heitmeyer, and J. McLean. A security model for military message systems. *ACM Transactions of Computer Systems*, 2(3):198 – 222, August 1984.

[LS78] R. Lipton and L. Snyder. On synchronization and security. In R. DeMillo, D. Dobkin, A. Jones, and R. Lipton, editors, *Foundations of Secure Computation*, pages 367–385. Academic Press, 1978.

[McC87] D. McCullough. Specifications for multi-level security and a hook-up property. In *Proceedings of the 1987 IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1987.

[McC90] D. McCullough. A hookup theorem for multilevel security. *IEEE Transactions on Software Engineering*, 16(6):563 – 568, June 1990.

[McL85] J. McLean. A comment on the 'basic security theorem' of Bell and LaPadula. *Information Processing Letters*, 20(2):67 – 70, February 1985.

[McL90a] J. McLean. Security models and information flow. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1990.

[McL90b] J. McLean. The specification and modeling of computer security. *Computer*, 23(1):9 – 16, January 1990.

[McL92] J. McLean. Proving noninterference and functional correctness using traces. *Journal of Computer Security*, 1(1):37 – 57, January 1992.

[Mea92] C. Meadows. Using traces based on procedure calls to reason about composability. In *Proceedings of the 1992 IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1992.

[Mil87] J. Millen. Covert channel capacity. In *Proceedings of the 1987 IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1987.

[Mil92] J. Millen. A resource allocation model for denial of service. In *Proceedings of the 1992 IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1992.

[MM92] I. Moskowitz and A. Miller. The channel capacity of a certain noisy timing channel. *IEEE Transactions on Information Theory*, 38(4):1339 – 1344, July 1992.

[PF78] G Popek and D. Farber. A model for verification of data security in operating systems. *Communications of the ACM*, 21(9):237–249, September 1978.

[San88] R. Sandhu. The schematic protection model: Its definition and analysis for acyclic attenuating schemes. *Journal of the ACM*, 35(2):404–432, 1988.

[San91] R. Sandhu. The typed access matrix model. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1991.

[Sny81] L. Snyder. Theft and conspiracy in the take-grant model. *Journal of Computer and Systems Sciences*, 23(3):333–347, 1981.

[Sut86] D. Sutherland. A model of information. In *Ninth National Computer Security Conference*. National Bureau of Standards/National Computer Security Center, 1986.

[WJ90] T. Wittbold and D. Johnson. Information flow in nondeterministic systems. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society Press, 1990.

[Wra91] J. Wray. An analysis of covert timing channels. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society Press, 1991.

[YG90] C-F Yu and V. Gligor. A specification and verification method for preventing denial of service. *IEEE Transactions on Software Engineering*, 16(6):581 – 592, June 1990.