Defining Noninterference in the Temporal Logic of Actions

Todd Fine
Secure Computing Corporation
2675 Long Lake Road,
Roseville, Minnesota 55113-2536

Email: *fine@sctc.com*

Abstract

Covert channels are a critical concern for multilevel secure (MLS) systems. Due to their subtlety, it is desirable to use formal methods to analyze MLS systems for the presence of covert channels. This paper describes an approach for using Lamport's TLA to specify noninterference properties. In addition to providing a more intuitive definition of noninterference than previous attempts, this approach also supports analysis of systems that do contain covert channels to demonstrate limitations on their exploitations. In relating the definition of noninterference given here to prior definitions of noninterference, this paper discusses ways in which other definitions of noninterference can be formalized in TLA, too. Finally, this paper discusses how prior work on specification refinement and composition might be applied to the noninterference problem within the framework provided by TLA.

1. Introduction

A multilevel secure (MLS) system is one in which processes are assigned levels indicating the sensitivity of data they are trusted to access and files are assigned levels indicating the sensitivity of data they contain. For example, a process might have a level such as `SECRET` while a file might have a level such as `UNCLASSIFIED`. The levels are ordered from least sensitive to most sensitive, and the goal of the system is to ensure that processes cannot obtain information for which they are not trusted. For example, a process with level `UNCLASSIFIED` should not be able to access information in a file with level `SECRET`. An obvious requirement is the *Simple Security Property* which prohibits a process from reading files at higher levels. A slightly less obvious requirement is the $*$-*Property* which prohibits a process from writing files at lower levels; without this requirement, a misbehaving high-level process might compromise data in a file at its level by copying it into a file at a lower level that is accessible to low-level processes. Typically, the concern is that the high-level process is a Trojan horse that downgrades information in the background without the user's knowledge. For example, suppose a user with clearance to `SECRET` is running a word processor at level `SECRET` that has been modified without the user's knowledge to copy `SECRET` information entered by the user into a file with level `UNCLASSIFIED`. The $*$-*Property* addresses this concern by preventing the process running at level `SECRET` from writing files with level `UNCLASSIFIED` [6].

Unfortunately, the satisfaction of the Simple Security and $*$ properties does not ensure that low-level processes cannot obtain data from high-level files. The term *covert channel* is used to denote a mechanism by which a high-level process can communicate data to a low-level process [6]. The Simple Security and $*$ properties can be circumvented by a high-level process reading data from a file at its level and communicating it to a low-level process via a covert channel. Once again, the concern is typically that the high-level process is a Trojan horse that uses the covert channel to downgrade information without the user's knowledge.

There has been much prior work investigating techniques to formalize the definition of a covert channel and use formal methods to analyze a system model to identify covert channels [2, 9, 3, 4, 7]. This paper proposes a definition of noninterference that while similar to prior definitions is more intuitive. While much of the prior work has viewed noninterference as an absolute criteria indicating whether a system is "secure", the statement of noninterference proposed here defines what it means for one entity to not be able to interfere with another entity through the system. This leaves open the possibility that other entities in the system might be able to interfere with each other. In particular, the statement allows for the possibility that the system might contain a covert channel and the goal is to demonstrate that certain well-behaved entities do not exploit the channel.

Another advantage of the definition provided here is that it is couched in Lamport's Temporal Logic of Actions (TLA) [1]. Lamport's TLA work is widely known and provides a well-accepted model of computing. Stating noninterference in terms of this model makes it accessible to the relatively large number of people who are TLA-literate, allows TLA proof rules to be used in the analysis of the system, and opens the possibility of applying TLA theories such as composition and refinement to the noninterference problem. Earlier work has suggested that TLA is an inappropriate formalism for studying the noninterference problem [5]. We discuss here why TLA is an appropriate formalism for studying noninterference even though the specific issues identified in reference [5] is correct. In addition, we discuss how prior definitions of noninterference can be incorporated into the framework of TLA.

2. TLA and Composition

This section provides a brief overview of TLA and composition theory [1]. The presentation here is a slight refinement of the standard presentation of TLA with the differences noted here. The TLA specification language is based on state transitions. A state is a representation of a snap shot of the system at some point in time. A state transition is a triple consisting of a starting state, a new state, and an agent. The meaning of a triple is that whenever the system is in the starting state, the specified agent can cause a transition to the new state. The agent of a transition can be either a system entity or an environment entity. A behavior is an infinite sequence of state transitions representing an execution history. A system is specified as a set of behaviors indicating the execution histories possible in the system.

Note that a "system" is often a component of a larger system. Then, the component system's environment consists of other system components with which it interacts as well as entities external to the system. Also note that a behavior can also be viewed as an infinite sequence of states sts and an infinite sequence of agents ags. The correspondence between this and a sequence of state transitions is that the i^{th} state transition in the sequence is $(sts(i), (sts(i+1), ags(i))$. For convenience, we often use the representation as a pair of sequences rather than the representation as a sequence of transitions in the following.

Lamport has demonstrated that any component can be specified as a triple (I, N, L), where:

- I specifies the set of initial states for the system.
- N specifies the allowed state transitions by both the system and its environment.
- L specifies liveness conditions. For example, L might require that each component periodically be given the opportunity to perform processing.

I represents the set of states appearing as the first state in some behavior while N represents the set of state transitions appearing in some behavior.

For simplicity, we ignore L in the following and assume every component can be specified as a pair (I, N). Consequently, the definition of noninterference provided in Section 4 does not address liveness. Some prior definitions of noninterference such as those in references [10] and [7] do address liveness, so the definition provided here is currently deficient in this area by comparison. We expect that once we extend our representation of TLA to address liveness, the extension of our definition of noninterference to address liveness will be straightforward.

The description of N as specifying the transitions of both the system and the environment is slightly misleading. Clearly, the specification of a component will not completely describe the behavior of the environment. If it did so, then it would be a specification of the component *and* the environment rather than simply a specification of the component. The transitions in N that correspond to environment agents describe assumptions that a component makes about the environment. For example, the assumption that the environment never modifies a variable x would be captured by not including in N any transitions by the environment that modified x.

In TLA, a state actually corresponds to the state of the "entire universe" rather than simply the system state. Consequently, each component has a "view" of the state. We represent views as equivalence relations on the state. Two states are in the equivalence relation associated with a given component only if the two states "look" the same to the component. Components are required to always allow "stuttering" with respect to their view. In other words, given two states that appear the same to a component, any agent must be allowed to cause a transition from one state to the other. From the perspective of the component, the system is stuttering in that a null transition has been made. In reality, a stuttering step might actually perform modifications to many variables that are not visible to the component. This allows a system behavior to record operations performed by other components in the system even though the variables altered by those operations are unknown to the component.

For convenience, we represent each component through the following attributes:

- c_ags — the set of agents within the component
- $view$ — the equivalence relation defining which states "look" the same to the component
- $init$ — the set of allowable states for the component when the system is first started
- $guar$ — the set of transitions that can be caused by agents in the component

- $rely$ — the set of transitions by agents in the environment of the component that correspond to assumptions on the behavior of the environment

The attributes c_ags and $view$ are implicit in TLA rather than explicit as they are here. Typically, the c_ags sets for different components will be disjoint. Then, the agent associated with a transition indicates which component performed the transition. The attribute $init$ corresponds to TLA's I. The attributes $guar$ and $rely$ provide a partitioning of TLA's N based on whether the agent is within the component or an environment agent. This partitioning and the terminology "guarantee" and "rely" are based on earlier work by Shankar [11].

We refer to the above as the *component form* of a system and use terms such as cmp, cmp_1, and cmp_2 to denote specific instances. Given the component form representation cmp of a system, we use $behs(cmp)$ to denote the set of behaviors associated with cmp. This is simply the set of behaviors having an element of $init(cmp)$ as their starting state and having each transition being an element of either $guar(cmp)$ or $rely(cmp)$.

We now describe an operation for composing components. The composition operator described here is essentially that described in reference [1] where composition of systems involves simply intersecting their sets of behaviors. In other words, a behavior is accepted for the system if it is accepted by each of the components in isolation. We define our composition on component forms rather than sets of behaviors, but the definition is really the same. In defining the operator, we use subscripted component attributes to refer to attributes of specific components. For example, c_ags_1 is c_ags for cmp_1 and c_ags_2 is c_ags for cmp_2. Given two components cmp_1 and cmp_2, we define $cmp_1 \mid\mid cmp_2$ to be the following component form:

- c_ags is the union of c_ags_1 and c_ags_2.
- $view$ is the intersection of $view_1$ and $view_2$.
- $init$ is the intersection of $init_1$ and $init_2$.
- $guar$ is the union of $guar_1$ and $guar_2$.
- $rely$ is the intersection of $rely_1$ and $rely_2$.

The intuition behind these definitions is:

- Any agent of either component is an agent of the overall system.
- Two states "look" the same to the composite system exactly when they "look" the same to each of the component systems.
- The initial state must be appropriate for each component.
- Each system transition must be either a transition by the first component or a transition by the second component.
- Each transition of the composite system's environment must satisfy the assumptions each component makes of its environment.

To ensure that a composite specification is consistent, we must ensure that the transitions specified for each component ($guar_i$) satisfy the assumptions made by the other component (represented by $rely_{3-i}$). For example, if $rely_1$ prohibits the environment of the first component from modifying variable x, then that component cannot meaningfully be composed with a second component having $guar_2$ containing a transition that modifies x. Note that this is a little different than the Abadi-Lamport work. They allow any two components to be composed. However, their proof rules for composite systems are only applicable when the composition is consistent. In other words, they allow components to be composed even when the result is inconsistent but they avoid doing so in practice.

Although one approach would be to define components as being composable as long as $guar_1 \subseteq rely_2$ and $guar_2 \subseteq rely_1$, this has the unfortunate side-effect of often terming a component as not being composable with itself. For example, a component that modifies variable x itself but assumes other components do not modify x would be termed as not being composable with itself. In particular:

- $guar_1$ and $guar_2$ would be the same and both contain a transition that modifies x
- $rely_1$ and $rely_2$ would be the same and both not contain any transitions that modify x
- So, $guar_1 \not\subseteq rely_2$.

In Section 7 it is important to be able to compose a component with itself. Thus, we slightly relax the conditions on composability to:

- $guar_1 \subseteq guar_2 \cup rely_2$
- $guar_2 \subseteq guar_1 \cup rely_1$

This condition is clearly satisfied when $guar_1$ and $guar_2$ are the same, so any component can be composed with itself. Intuitively, the condition on composition is that any transition performed by a component can be jointly performed by the other component or is allowed by the other component's environment assumption. In cases when c_ags_1 and c_ags_2 are disjoint, the constraints on $guar$ for each component ensure there are no transitions that can be performed jointly by both components. Then, the conditions on composability reduce to $guar_1 \subseteq rely_2$ and $guar_2 \subseteq rely_1$ as originally proposed.

3. Covert Channels and Noninterference

A well-known example of a possible covert channel in MLS variants of Unix is the *access time channel*. Each file has an access time associated with it that indicates the time at which the file was last read. When a high-level process reads a low-level file, the access time is updated. A low-level process can receive a bit of information by using the `stat` command to determine whether the access time of the low-level file has changed. Although the Simple Security Property prohibits the high-level process from writing information into the low-level file, it does not prohibit the file's meta-data from being updated.[1]

Informally, a noninterference policy requires that the "view" a low-level process has of the system is the same regardless of whether a high-level process is executing. Note that "view" here is not necessarily the same as the notion of a component's view defined in the previous section. One attempt at a more rigorous definition of noninterference is:

- Let seq be an arbitrary sequence of instructions executed by processes (low-level as well as high-level) in the system.
- Let seq' be seq with the instructions executed by high-level processes removed.
- Then, the execution of seq on the system should "look" the same to low-level processes as the execution of seq'.

This definition detects the access time channel by choosing seq to be a high-level read of a file followed by a low-level `stat`. Then, seq' is simply the low-level `stat`. The value returned by `stat` is different in each sequence since the high-level read changes the access time of the file.

Approaches that have previously been used to formalize "look the same" include:

- The outputs generated by the system must be the same for each sequence.

  This approach detects the access time channel by detecting a different value output from `stat` in seq and seq'.

- The states resulting from the execution of each sequence "look" the same.

  This approach detects the access time channel by defining two states to "look" the same to low-level processes if the access times associated with all low-level files are the same. Since seq and seq' result in different access times being associated with the low-level file, the resulting states do not "look" the same.

Descriptions of this class of approach can be found in references [9] and [2].

Another class of approach is characterized by Thayer and Johnson's notion of *correctability* [3]. Essentially, this definition of noninterference is:

- Let seq be an arbitrary sequence of system events (including both inputs and outputs).
- Let seq' be a "perturbed" sequence that is identical to seq except for the addition or deletion of high-level inputs.
- Then, there must be some "correction" of seq' that is:
  - identical to seq' except for the addition or deletion of high-level outputs, and
  - identical to seq in low-level inputs and outputs.

This approach detects the access time channel using as seq the sequence consisting of a high-level read, a low-level `stat`, and the output of the new access time. One choice for seq' is seq with the high-level read removed.[2] Then, seq' is not a valid behavior of the system since the low-level `stat` must return the file's original access time rather than the new access time. There is no way to construct a "correction" by adding or removing high-level outputs, so the system does not satisfy the requirements of correctability.

The main difference between this approach and that described above is that it allows for nondeterministic system models. In other words, it is not required that there be a unique new state for each operation and starting state.

There are several other approaches based on considering sequences of events. For example, reference [10] defines noninterference in terms of sequences of events accepted by the system. In the case of deterministic systems, the policy requires that given any sequence seq with seq_l containing only the low-level events in seq, the set of low-level events the system accepts after seq is the same as the set of low-level events the system accepts after seq_l. This policy detects the access time channel with $seq = < read_{f,v} >$ and $seq_l = <>$ (the empty sequence). Then, the low-level event returning the current time in response to a `stat` call is accepted after seq but is not accepted after seq_l when the `stat` call would have to return the previous access time for the file.

A similar policy was presented in reference [7]. The approach there was to define noninterference in terms of determinism. The system under consideration is "merged" with arbitrary high-level behavior. Then, all of the high-level

[1] One way to close this channel in an MLS system is to change the semantics of the read operation so the access time is not updated unless the file is at the same level as the reading process.

[2] In this approach, requests processes make of the system are viewed as system inputs.

behavior is "hidden". The policy requires that the resulting behavior is deterministic. The intuition is that the next low-level event accepted is always a function of only the previous low-level behavior. Covert channels show up as instances in which the system can nondeterministically choose the next low-level event based on the high-level behavior that was hidden. To clarify this, consider the access time example. The merging with arbitrary high-level behavior means that a file's access time can be updated to the current time at any point. When this behavior is hidden, the resulting system can nondeterministically choose to update a file's access time at any point. Then, the output seen at the low-level is not a function of previous low-level behavior.

The final class of definitions of noninterference that we discuss here is those using an equivalence relation on states to explicitly specify restrictions on state transitions. The equivalence relation defines what it means for two states to "look" the same to low-level processes and the restrictions are essentially:

- The starting and new states for transitions by high-level processes "look" the same.
- Given two starting states that "look" the same, an operation executed by a low-level process leads to two new states that "look" the same.

This approach would detect the access time channel by defining two states to "look" the same if the access times for all low-level files are the same. Then, a read operation performed by a high-level process on a low-level file violates the first of the above restrictions by changing the access time of the low-level file. Reference [4] describes an example of this class of approach.

Of the various definitions, the ones stated in terms of sequences of events or operations are the most intuitive. However, there are many subtleties involved. For example, although correctability is somewhat intuitive, it actually is a flawed statement of noninterference that Thayer and Johnson fixed in a variant called forward correctability [3]. The final class of statements provides more design guidance through the explicit restrictions on transitions, but does not provide an abstract statement of the resulting noninterference property. Consequently, it is difficult to determinine what the security requirement really means in this approach. Ideally, an unwinding theorem [9] can be proved demonstrating that the conditions on individual transitions are sufficient conditions to establish a more intuitive definition of noninterference in terms of system behaviors.

Another disadvantage of most of the prior definitions is that they are absolute statements; a system satisfies them or it does not. In practice, covert channels are usually unavoidable. For example, a Unix-like MLS system that is required to satisfy Unix semantics would not be able to avoid the access time channel. Then, the real goal of the analysis is to determine how each channel can be exploited. Once this is done, countermeasures can be inserted to address intolerable exploitations. In these cases it is desirable to use a *conditional* noninterference policy which states that the system contains no covert channels except for certain exceptional cases. Of the policies described previously, only reference [7] provides a statement of such a noninterference policy. This policy requires there be no signaling from a high-level process to low-level processes as long as the high-level process is constrained to a specified behavior [7]. However, even this statement of noninterference is not quite general enough since it is sometimes necessary to restrict the behavior of the low-level processes, too. For example, it might be acceptable for arbitrary high-level processes to signal to certain trusted low-level processes.

4. Proposed New Statement

This section proposes a "new" statement of noninterference in terms of TLA. Although there is a great deal of similarity between the definition proposed here and the statements overviewed in the preceding section, there are some significant differences:

- The proposed statement corresponds more closely with the informal definition of noninterference.
- The proposed statement is a property of process-process-system triples rather than a property of systems. In other words, the statement defines what it means for a system to prohibit a process from interfering with a second process rather than what it means for the system to prohibit interference from high-level to low-level.
- The proposed statement is defined in terms of composition rather than in terms of purging or perturbing high-level inputs. Note, however, that the definition of noninterference provided in reference [7] is similar in this regard.

Before stating the proposed definition of noninterference, it is first necessary to define a couple more concepts. First, we need to represent a process cmp executing on a system sys. We view each as peers cooperating to perform a task. Thus, we use $cmp \parallel sys$ to denote the process represented by cmp executing on sys.

Second, we need to define a process' "view" of a system (as opposed to its view of the system state). We define $cmp_1 \approx_{cmp_3} cmp_2$ to denote that cmp_1 and cmp_2 "look" the same to cmp_3. This relation is defined as follows:

1. The set of states that "look" the same (with respect to $view_3$) as some state in $init_1$ is equal to the set of states that "look" the same as some state in $init_2$.

2. Each agent in c_ags_3 is in both c_ags_1 and c_ags_2.

3. If (st_1, st_1', ag) is a transition allowed by one component and st_2 "looks" the same as st_1(with respect to $view_3$), then there exists a st_2' such that (st_2, st_2', ag) is a transition allowed by the other component and st_2' "looks" the same as st_1'.

An alternative (and perhaps more intuitive) way to define $cmp_1 \approx_{cmp_3} cmp_2$ is through behaviors. Essentially, the requirement is that for each behavior beh_1 of one component, there exists a behavior beh_2 of the other component such that the corresponding states of beh_1 and beh_2 "look" the same with respect to $view_3$. By the two behaviors looking the same, we simply mean that the i^{th} state of beh_1 is equivalent to the i^{th} state of beh_2 with respect to $view_3$.
Given cmp_1 and cmp_2 we define:

$$cmp_1 \not\rightsquigarrow_{sys} cmp_2 = \\ ((cmp_1 \parallel cmp_2) \parallel sys) \approx_{cmp_2} (cmp_2 \parallel sys)$$

Then, the intuitive statement of "cmp_2 executing by itself on the system 'looks' the same to cmp_2 as cmp_2 executing concurrently with cmp_1" is formalized as:

$$cmp_1 \not\rightsquigarrow_{sys} cmp_2$$

At this level, the statement appears identical to some existing statements of noninterference. For example, this definition appears similar to that used by Rushby in reference [9]. However, there is a significant difference. In prior definitions, cmp_1 and cmp_2 would be either the identity of a process or a security attribute of a process. For example, Rushby defines MLS noninterference in a form similar to that used here but with $\rightsquigarrow$ defined as a relation on sensitivity levels rather than processes. By defining $\rightsquigarrow$ to be a relation on processes, the definition of noninterference can take into account the behavior of the processes.
To clarify this point, consider the access time channel. By defining cmp_1 to be a high-level process that reads low-level files and defining cmp_2 to be a low-level process that performs `stat` operations on the same files, it is clear that $cmp_1 \rightsquigarrow_{sys} cmp_2$. Now, suppose cmp_1 is defined instead to be a high-level process that only accesses high-level files. Then, $cmp_1 \not\rightsquigarrow_{sys} cmp_2$. Since sys has not been changed, it still contains a covert channel through access times. But, there is no interference from cmp_1 to cmp_2 through sys since no high-level process reads low-level files. Whereas many prior statements of noninterference have defined "security" as an absolute (either there are channels or there are not channels), the statement proposed here acknowledges the possibility that there are channels in the system and allows for a precise statement as to how the channels can be exploited. Once a set of exploitations has been identified and demonstrated to be complete, it suffices to develop countermeasures for each of the exploitations. While the countermeasures might involve closing the channel, they also might involve leaving the channel in the system and auditing its use or introducing noise or delays. Many prior statements of noninterference cannot deal with the latter case since the channel is still in the system. The proposed statement of noninterference addresses the issue by allowing for a precise definition to be made of the processes in the system that can signal through any residual channels.
Note that other *conditional noninterference* policies have also allowed for exceptions to pure noninterference to be identified. Typically, these policies have only allowed for restrictions to be placed on the sending subject. For example, the conditional noninterference policy in reference [7] provides a means for stating that there is no flow of information to the receiver as long as the sender behaves in a specified fashion. The noninterference policy described here is a further generalization that allows restrictions to be placed on the receiving process, too. For example, we could just as well change cmp_2 to not check the access time on files accessed by cmp_1. Then, there would be no flow from cmp_1 to cmp_2 through sys even though the system contains the access time channel and cmp_1 accesses low-level files. It is expected that this generalization will be useful in precisely defining the information flows in systems that (for example) collect global system information in data structures that are only accessible to trusted processes. Then, the policy could be used to identify flow from higher level processes to the trusted processes as being alright.
Although we have been using the term "process" to refer to cmp_1 and cmp_2, it is important to note that they can actually be any component. For example, cmp_1 might actually be a collection of processes executing on the system. A pure MLS policy could be stated by grouping high-level processes into cmp_1 and low-level processes into cmp_2 and demonstrating the system prevents interference. It is also possible that the components do not correspond to processes at all. For example, cmp_1 might denote a particular profile of traffic on one network while cmp_2 denotes a particular profile of traffic on a second network. If the system is a network guard, this would provide a policy statement concerning the ways in which the first network can interfere with the second network through the guard. In summary, although we use the term "process" for motivational reasons, the proposed statement of noninterference addresses much more than simply interference between processes in computing systems.

5. Discussion

The most common objection to using the Abadi-Lamport work for noninterference is that noninterference is not preserved by arbitrary refinements. A refinement of a system

sys is any sys_2 whose allowed behaviors are a subset of those allowed by sys. In other words, a refinement can further restrict the behavior of the system but cannot introduce new behavior. For example, an abstract specification of a sort procedure might simply indicate that at the completion of the sort, the elements in the input list have been rearranged into the proper order. This abstract specification might allow for the elements to be magically reordered in a single step, gradually reordered through bubble sort, gradually reordered through quicksort, …. This specification could be refined into a specification of a particular sorting algorithm such as quicksort.

Allowing for specification refinement is one of the reasons for requiring that components allow stuttering. The stuttering steps can be refined into transitions that alter lower-level variables in the refinement. For example, what appears as a stuttering step in the abstract specification of sorting might be seen to be the setting of an index variable in the specification of quicksort.

In the Abadi-Lamport work the term "property" is used to denote a set of behaviors that is closed under stuttering. Thus, properties are represented in the same manner as systems. A system is said to satisfy a property if every behavior of the system is contained in the set of behaviors denoting the property. A benefit of this approach is that any property that a system satisfies is also satisfied by any refinement of the system. This allows a top-down development approach to be used. Requirements are stated that ensure the desired system properties hold. Then, these requirements are refined into high-level design, low-level design, code, … with each successive refinement still ensuring the desired system properties.

Unfortunately, noninterference is not preserved by refinement when nondeterministic behavior is allowed. For example, suppose that the abstract specification of a system allows for either a 0 or a 1 to be output to a low-level process from a given system state. Although this behavior, might not allow covert signaling in the abstract specification, it is trivial to construct refinements in which covert signaling is allowed. For example, suppose the specification is refined so that a bit of data from a high-level file is read and used to determine whether to output a 0 or a 1. Then, the transition outputs data from a high-level file to a low-level process. Since noninterference is not preserved by refinement, it cannot be a property in the Abadi-Lamport sense. This has led others to reject the use of the Abadi-Lamport theory for exploring noninterference properties. For example, reference [5] mentions that noninterference is not a property in the Abadi-Lamport theory and then proceeds to study noninterference within a different formalism.

However, it is still possible to state noninterference in the temporal logic of actions as we have done in the previous sections. As pointed out in reference [5], the definition is technically a "property of Abadi- Lamport properties" rather than an "Abadi-Lamport property" itself. This means the TLA proof rules do not necessarily apply to noninterference. However, they do apply to safety properties such as the Simple Security Property and the $*$-Property. By writing specifications in TLA, the existing theories for TLA can be used in the analysis of such properties. By using the statement of noninterference proposed in the previous section, the same system model can be used for the noninterference analysis, too.

Furthermore, if the definition of noninterference can be reduced to a collection of *unwinding conditions*, the unwinding conditions can often be addressed using TLA proof rules. For example, the definitions in references [2, 9] can also be formalized in TLA as properties of properties. These definitions of noninterference have associated "unwinding theorems" that allow verification of the noninterference policy to be reduced to verification of conditions on individual instructions. Essentially, the conditions are:

- Operations by the transmitter do not change data visible to the receiver.
- Operations by the receiver modify data visible to the receiver in a manner determined entirely by data visible to the receiver.

The former condition is a restriction on the set of transitions that the system may allow. Consequently, it is a safety condition and is a property in the Abadi-Lamport sense. The second condition is generally a property of Abadi-Lamport properties rather than an Abadi-Lamport property itself. A more precise statement of the condition is:

> Given any transition (st_1, st'_1, ag) by the receiver and st_2 having the same data visible to the receiver as is visible in st_1, there exists a transition (st_2, st'_2, ag) with st'_1 and st'_2 having the same data visible to the receiver.

Rather than describing a set of allowed transitions, this condition describes a class of sets of allowed transitions and consequently is a property of properties rather than a property itself. Thus, the TLA proof rules do not directly apply to the second condition but do apply to the first condition.

In summary, the TLA proof rules are not directly applicable to noninterference, but the inference that noninterference cannot be studied within the framework provided by TLA is incorrect. Noninterference can be stated in TLA and portions of the noninterference analysis as well as analysis of any other safety properties of the system can be done using the TLA proof rules.

There is a strong connection between the refinement issue and determinism. Generally, the concern with refining systems satisfying a noninterference policy is that the removal of nondeterminism introduces covert channels through the removal of noise. For example, consider the access time channel. If the system specification says that it can choose to update the access time for any file at any time, then actions taken by the sending process are hidden by noise present in the system. When the receiving process detects that the access time of a file has been changed, it has no way to determine whether the time was updated by the sending process or randomly by the system. Thus, the sending process cannot interfere with the receiving process. The system that is obtained by removing all of the random system transitions updating the access times of files is a refinement of the initial system since it contains no new transitions. However, this system does allow the sending process to signal to the receiving process. Roscoe discusses the difficulties of analyzing noninterference in nondeterminism systems in reference [7].

A similar concern with the definition of noninterference given in the previous section is that even though a system might prevent a sending process from interfering with a receiving process, it might not prevent a refinement of the sending process from interfering with a refinement of a receiving process. For example, if the sending process is defined to allow arbitrary transitions by the sender, then composing the sending process with the system does not constrain the system in any way. Consequently, the composition of the receiver with the system is essentially the same as the composition of the receiver with the system and the sender, and it appears there is no information flow. A refinement of the sender can constrain the behavior of the system and possibly introduce information flows.

One way to address these concerns is to strengthen the definition of noninterference so that it requires that any refinement of the system prohibits any refinement of the sender from interfering with any refinement of the receiver.

A similar concern is that the sender and the receiver might not specify the behavior of all agents in the system. Then, the definition of noninterference allows noise to be introduced by other system agents. For example, the sending process might appear to not be able to use the access time channel to signal to the receiving process because any update to a file's access time caused by the sender could just as well have been caused by the unspecified behavior of other agents in the system. One way to address this concern is to ensure that the sending and receiving processes describe the behavior of all of the nonsystem agents.

In summary, in stating the definition of noninterference in the previous section, we chose to state it in such a way that the only interferences that are detected are those that the sender can force to occur. In practice, noise cannot be relied upon to obscure channels and it is desirable to strengthen the noninterference requirements.

6. Relation between TLA and CSP

We now discuss the relationship between definitions of noninterference in TLA and those in event based languages such as CSP. To do so, we suppose that each agent in a TLA specification is viewed as having the following structure:

- $ag.id$ — denotes the identity of the agent
- $ag.lvl$ — denotes the security level of the agent
- $ag.op$ — denotes the operation carried out by the agent

For example, some of the agents in an MLS operating system might be:

- $ag_1 = (ps, topsecret, req\_read_f)$ denoting a process ps at level `TOP SECRET` issuing a `read` request on file f
- $ag_2 = (kernel, topsecret, read_{ps,f})$ denoting the operating system kernel processing a request by process ps to read file f
- $ag_3 = (pr, unclassified, req\_stat_f)$ denoting a process pr at level `UNCLASSIFIED` issuing a `stat` request on file f
- $ag_{4,t} = (kernel, unclassisfied, stat_{pr,f,t})$ denoting the operating system kernel processing a request by process pr to get the access time of file f and returning t as the access time

The system would be specified so that a transition (st, st', ag) would be allowed if the state changes specified by (st, st') are consistent with the event specified by ag. For example, the system would allow its environment to make a transition (st_1, st_2, ag_1) in which st_2 is obtained from st_1 by recording that ps has requested f be read. As another example, the system would be specified to allow a transition (st_2, st_3, ag_2) when st_2 records that ps has requested f be read and st_3 is obtained by performing the read file operation starting in st_2. Examples of behaviors allowed by the resulting system are:

- Nothing happens

  $st_a \xrightarrow{ag} st_a \cdots$
- pr checks the access time

  $st_a \xrightarrow{ag_3} st_b \xrightarrow{ag_4,0} st_c \xrightarrow{ag} st_c \cdots$
- ps reads the file followed by pr checking the access time

  $st_a \xrightarrow{ag_1} st_d \xrightarrow{ag_2} st_e \xrightarrow{ag_3} st_f \xrightarrow{ag_4,1} st_g \xrightarrow{ag} st_g \cdots$

We now can view “agents” as being observable system events. Given a state view v and an allowed transition (st_1, st_2, ag) with st_1 and st_2 appearing different with respect to v, we can view ag as a visible event. If ag is in $cags$ for the system, then ag is a system event. Otherwise, it is an environment event. By considering the agent sequences associated with behaviors, we can obtain a representation of the system as a sequence of events. For example, the behaviors above correspond to the sequences:

- $<>$
- $< (pr, unclassified, req\_stat_f),$ $(kernel, unclassified, stat_{pr,f,0}) >$
- $< (ps, topsecret, req\_read_f),$ $(kernel, topsecret, read_{pr,f}),$ $(pr, unclassified, req\_stat_f),$ $(kernel, unclassified, stat_{pr,f,1}) >$

If we assume that pr can only see its input and output buffers, then events ag_1 and ag_2 are not visible with respect to pr’s view. We can then restrict each of the event sequences to the events visible to pr:

- $<>$
- $< (pr, unclassified, req\_stat_f),$ $(kernel, unclassified, stat_{pr,f,0}) >$
- $(pr, unclassified, req\_stat_f),$ $(kernel, unclassified, stat_{pr,f,1}) >$

Now, it is relatively straightforward to translate between definitions of noninterference in terms of TLA and CSP. As an example of applying a CSP version of noninterference to a system specified in TLA, consider the definition in reference [7] which requires that the system obtained by hiding sender events is deterministic when viewed by the receiver. The above sequences of events show that the system specified in TLA can perform two different outputs to the receiver as the result of the same input by the receiver. Consequently, the system appears nondeterministic to the receiver and does not satisfy the definition of noninterference in reference [7].
By translating existing theories of noninterference into TLA, it is possible to apply those theories to systems specified in TLA. This allows prior definitions to be used in TLA just as easily as the definition proposed here. In summary, the definition of noninterference in TLA given here is not the only way to define noninterference in TLA. To use another variant of noninterference, it is not necessary to abandon TLA; it is simply necessary to translate into TLA.

7. Composition and Refinement

A recent area of research regarding noninterference statements is their composability. Early work in this area includes McCullough’s statement of *restrictiveness* [4]. More recent work includes that described in references [5] and [8]. Essentially, the question is whether the composition of two “secure” components results in a “secure” composite system (where “secure” means “satisfies noninterference”). The motivator for this research is the desire to use a divide-and-conquer approach to analyzing systems. For example, it is desirable to divide the analysis of an MLS network into analysis of the following components:

- Analysis of each of the nodes in isolation.
- Analysis of the network protocol used to connect the nodes.

In theory, it is possible to continue to apply this divide-and-conquer approach to more detailed models of the system. For example, the model of each node might be as the composition of various components (file system, network protocol stack, ...) representing the modules comprising the node. In practice, a point is eventually reached at which point it is not practical to consider components in isolation.
To consider the composability of our proposed statement of noninterference, we now suppose that we have two systems, sys_1 and sys_2, and processes, cmp_1 and cmp_2, such that:

- $cmp_1 \not\rightsquigarrow_{sys_1} cmp_2$, and
- $cmp_1 \not\rightsquigarrow_{sys_2} cmp_2$

Ideally, it would be possible to demonstrate that $cmp_1 \not\rightsquigarrow_{sys_1 \| sys_2} cmp_2$. This type of composability result holds for policies such as those in references [4], [5], [10], and [7]. We have not spent much time considering the composability of the definition of noninterference given here, so it is clearly an area for future research. We simply make some initial observations here.
Suppose cmp_1 and cmp_2 are processes of the composite system that are desired to be shown to be noninterfering. In general, each process can refer to processing specific to each of the systems being composed. This means that when analyzing the security of sys_1, it might be necessary to consider sys_2, too. This would be undesirable since it would make it difficult to analyze systems individually. There might be a trade-off here between being able to state conditional noninterference policies and having the noninterference policies be composable.
The reason why composing two secure (in the noninterference sense) systems does not necessarily result in a secure system is that the composition is generally more deterministic than the original system. As noted earlier, removing

nondeterminism can introduce covert channels. For example, our definition of noninterference requires that for each behavior beh of the sender and receiver operating together on the system, there exists an equivalent behavior beh' of the receiver operating by itself on the system. The reason that this policy is not generally composable is that the behavior beh' might not be allowed by the composite system even though it is allowed by the first system individually. The problem arises because some of the environment steps of the first system are steps it allows to be taken by the second system. If any of these steps appear in beh' and are not actually selected by the second system, then beh' is not a valid behavior of the composite. Although more investigation is required, we suspect that by requiring noninterference to hold for all refinements of cmp_1 and cmp_2, this problem can be addressed.

8. Conclusion

In the approach described here, noninterference is a property (in the generic rather than Abadi-Lamport sense) of sys, cmp_1, and cmp_2. This has led to noninterference being referred to as a "property of properties" (in the Abadi-Lamport sense) since sys, cmp_1, and cmp_2 are Abadi-Lamport properties. While it is true that noninterference is not an Abadi-Lamport property, the preceding sections have demonstrated that noninterference can be specified in TLA and that examinations of the composability of noninterference can be carried out within TLA. We believe a similar approach can be used to examine refinement of noninterference in the context of TLA specifications. Since the TLA specification language and the composition and refinement theories developed for it are well-accepted tools for specifying and verifying systems, it is desirable to be able to use them for secure system development. By developing techniques to perform covert channel analysis on TLA specifications, it is possible to use TLA for all of the formal analysis performed on a system.

In addition to demonstrating how to state noninterference in TLA, the proposed statement is of interest by itself. It has a very close tie to the intuitive notion of noninterference. In addition, it allows precise statements of which processes are prevented from signaling even when a covert channel is present in the system. The given definition of noninterference is merely one example of how noninterference can be defined in TLA. We discussed how other definitions of noninterference in terms of CSP might be translated into TLA. This allows users of TLA to build upon prior definitions of noninterference while also providing a common framework for comparing various noninterference policies.

Much future work remains to be done. First, it would be desirable to determine whether the strong version of our noninterference policy is composable. In doing so, we will need to determine the feasibility of retaining the conditional nature of the policy while ensuring it is composable. Second, the examination into refinement of systems satisfying noninterference should be done. This could provide guidance to developers as to how abstract specifications can be refined into implementations without introducing security flaws. Finally, the range of policies that can be supported by the proposed statement of noninterference should be considered. In addition to allowing for the statement of policies prohibiting components from interfering even though a covert channel is present, the generality might be useful in areas such as intransitive noninterference [9]. For example, using cmp_1 to denote processing other than that specified for an assured pipeline and cmp_2 to denote a component intended to receive information through the assured pipeline, it should be possible to demonstrate that the information transfer is prevented if the system implements the assured pipeline correctly.

References

[1] M. Abadi and L. Lamport. Conjoining specifications. Technical Report 118, Digital Equipment Corporation, Systems Research Center, Dec. 1993.

[2] T. Fine. Constructively Using Noninterference to Analyze Systems. In *IEEE Symposium on Security and Privacy*, pages 162–169, Oakland, CA, May 1990.

[3] D. Johnson and F. Thayer. Security and the composition of machines. In *Proceedings of the Workshop on the Foundations of Computer Security*. IEEE, Oct. 1988.

[4] D. McCullough. Noninterference and the composability of security properties. In *Proceedings of the 1988 Symposium on Security and Privacy*, pages 177–186. IEEE, Apr. 1988.

[5] J. McLean. A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 1994.

[6] NCSC. Trusted Computer Systems Evaluation Criteria. Standard, DOD 5200.28-STD, US National Computer Security Center, Fort George G. Meade, Maryland 20755-6000, Dec. 1985.

[7] A. W. Roscoe. CSP and determinism in security modelling. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 114–127, Oakland, CA, May 1995.

[8] A. W. Roscoe and L. Wulf. Composing and Decomposing Systems under Security Properties. In *Proceedings of the IEEE Computer Security Foundations Workshop*, 1995.

[9] J. Rushby. Noninterference, Transitivity, and Channel-Control Security Policies. Technical report, SRI International, Dec. 1992.

[10] P. Ryan. A CSP Approach to Noninterference and Unwinding. *IEEE Cipher*, 1990.

[11] N. Shankar. A lazy approach to compositional verification. Technical Report TSL-93-08, SRI International, Dec. 1993.