

A General Theory of Security Properties

Aris Zakinthinos
Computer Laboratory
University of Cambridge
Aris.Zakinthinos@cl.cam.ac.uk

E.S. Lee
Centre for Communications Research
University of Cambridge
Stewart.Lee@cl.cam.ac.uk

Abstract

This paper presents a general theory of possibilistic security properties. We show that we can express a security property as a predicate that is true of every set containing all the traces with the same low level event sequence. Given this security predicate, we show how to construct a partial ordering of security properties. We also discuss information flow and present the weakest property such that no information can flow from high level users to low level users. Finally, we present a comparison of our framework and McLean's Selective Interleaving Functions framework [14].

1. Introduction

Each researcher has proposed a new security property has constructed his or her own notation and formalism. With different notations and assumptions about the model of components, comparing the strengths and weaknesses of the various security properties has been difficult. In this paper, we examine what constitutes a security property and how they can be expressed. We then present a framework for the specification and analysis of security properties.

One of the first attempts to provide a general theory of security properties was through the use of Selective Interleaving Functions [14]. McLean's framework is only applicable to a subset of security properties. Possibly its greatest weakness, however, is that it does not allow for an obvious specification of security. Our framework captures the intuitive notion of security properties and can be used to determine the composability of components that satisfy security properties.

In this paper we are only interested in confidentiality properties. The goal of confidentiality is to prevent low level users (LLU) from determining anything about high level activity. The security policy defines exactly what low level users are forbidden to discover. For example, it may be considered desirable to ensure that low level users

cannot determine which high level inputs have occurred. Or we may say, information about high level inputs may not flow to LLU. The security policy dictates which flows are permissible and which are not. A security property is an instantiation of a policy. There may be more than one property that satisfies a given policy. In this work, we do not advocate any specific security policy. We consider security properties in general¹.

2. Event Systems and Notation

The framework for our investigation into composability will be event systems as given by McCullough [12] and Johnson and Thayer [11]. McCullough's definition derives from the work on modeling concurrence of Hoare [9]. An event system interacts with its environment through events. These events correspond to the primitive actions done to or by the system. A sequence of events corresponding to a possible execution sequence of the system is called a *trace*. We will define an event system in terms of its possible traces.

A trace is denoted by a sequence of events, separated by commas and enclosed in angle brackets. Since traces play such a central role in our work we require some operations on traces.

Definition 1: Trace Concatenation

The notation $s^{\wedge}t$ will refer to the trace formed by putting together traces s and t in that order. We will use st to denote concatenation if s and t are obvious from the context. Formally, if $\langle X \rangle$ and $\langle Y \rangle$ are traces then $\langle X \rangle^{\wedge} \langle Y \rangle = \langle X, Y \rangle$

Definition 2: Trace Prefix

If s is a trace prefix of t , then it is possible to find some extension u of s such that $s^{\wedge}u = t$. Formally,

$$s \leq t \equiv \exists u. s^{\wedge}u = t$$

¹ In this work when referring to security we mean confidentiality.

Definition 3: Restriction Operator

The expression $t|A$ denotes the trace formed by removing from t all events not in A .

Example 1: Let $t = \langle a_1, a_2, a_1, a_3, a_2 \rangle$. Then $t|\{a_1, a_3\} = \langle a_1, a_1, a_3 \rangle$

An event system is defined as follows:

Definition 4: Event Systems.

An event trace system is a 4-tuple:

$$S = \langle E, I, O, T \rangle$$

where

E is the set of events

I , the input events, $I \subseteq E$

O , the output events, $O \subseteq E$ and $I \cap O = \emptyset$

$T \subseteq E^*$ is the set of traces

Throughout the remainder of this work we will need the set of traces of a given system. The following definition gives the set of traces of a system S .

Definition 5: The Set of Traces of a System

For a system S the function $traces(S)$ returns the set of traces T of S .

The set of traces of an event system must satisfy the following property. It must always be possible for the system to accept an input event. This condition is called *input totality* [12] [11]. Formally,

Definition 6: Input Totality

A system S is said to be input total if and only if

$$\forall \tau: traces(S) \cdot \forall e: I \cdot \tau \hat{\cdot} e \in traces(S)$$

This modeling abstraction simplifies the presentation of the results. The need for input totality is examined in section 6.

The standard set operators of union, \cup , and intersection, \cap , will be used to combine the various sets of the event trace system. The set difference operator, \setminus , will also be used. The set $A \setminus B$, for example, contains all elements in the set A that are not in the set B .

The specification of security properties usually requires a distinction between high level (trusted) and low-level (untrusted²) users. We will refer to these categories as HLU and LLU respectively. This division is accomplished by dividing E into the disjoint subsets L and H , such that every event is in exactly one of L or H . These are, respectively, the sets of low- and high-level events. Assuming two comparable levels simplifies the presentation of the results without altering the results;

the generalization to an arbitrary lattice of levels is straightforward but notationally cumbersome.

The following definition gives some notation for commonly used classes of events.

Definition 7: Event Classes.

The following notation will be useful in specifying security properties:

$HI = H \cap I$ high level input events,

$LI = L \cap I$ low level input events,

$HO = H \cap O$ high level output events and

$LO = L \cap O$ low level output events.

3. Security Properties

McLean [14] was one of the first to propose a general theory of secure system composition. McLean's desire to provide such a theory came from the realization that security properties do not fit into the Alpern-Schneider [2] safety liveness framework. This means the Abadi and Lamport's composition principle [1] cannot be used to reason about composition. We present a proof that security properties cannot be expressed in the Alpern-Schneider framework in Appendix A.

McLean's theory is based on selective interleaving functions (SIFs). Selective Interleaving Functions can be used to express properties that are an interleaving of two traces of the system. The justification for using SIFs is McLean's observation that certain security properties are "closure properties with respect to some function that takes two traces and interleaves them to form a third trace" [14]. In this section we propose a framework for specifying security properties that is more general than selective interleaving functions. This allows a system designer the ability to reason about a larger class of security properties. We defer a comparison between our framework and selective interleaving functions to section 6.

To understand what a security property is, we first must understand what a security property does. Consider a low level user using a system and observing a trace τ_{low} . He can construct the set of traces that are consistent with his observation. We will call this the low level equivalent set.

Definition 8: Low Level Equivalent Set.

Given a trace τ and a System S , $LLES(\tau, S)$ is the set of traces that have the same low level events as τ in the same order. Formally,

$$LLES(\tau, S) = \{ s \mid \tau|_L = s|_L \wedge s \in traces(S) \}$$

² Or less trusted.

The low level user knows that a member of $LLES(\tau_{low}, S)$ must have occurred³. The examination of the elements of $LLES(\tau_{low}, S)$ will indicate which high level sequences could have occurred and which could not have occurred.

The purpose of a security property is to prevent low level users from being able to make deductions about the events of the high level users. The exact deductions that the low level user should not be able to make is dependent on what information flow policy the security property is attempting to enforce. Security properties try to remove the ability of a low level user from deducing anything about high level events by ensuring that τ_{low} can be attributed to more than one high level event sequence. That is, a security property ensures that certain traces are elements of $LLES(\tau_{low}, S)$. A system satisfies a security property if all of the required traces are present in all low level equivalent sets of the system. This leads us to the following definition of a security property.

Definition 9: Security Properties

A system property P is a security property if and only if there is a predicate Q such that for any system S , P satisfies S , written $P(S)$, if and only if every low level equivalent set of S satisfies Q . Formally,

P is a security Property \Leftrightarrow

$$\exists Q \cdot \forall S \cdot P(S) \Leftrightarrow (\forall \tau: \text{traces}(S) \cdot Q(LLES(\tau, S)))$$

Which traces are required to be present (i.e., consistent with τ_{low}) is dependent on what information flows are to be prevented. It might seem strange that the relevant traces are not those that prevent all information flows. There are many reasons why other properties are required. For example:

1. The risk analysis of the system indicates little threat of Trojan horses. In this case a security property with the possibility of some unauthorized flows might be acceptable.
2. A desired component does not satisfy this property and a weaker property must be used.

We will discuss information flow further in section 4. The following sections will demonstrate how some of the properties presented in the literature can be expressed using definition 8.

3.1 Noninference

Noninference was introduced by O'Halloran [17]. It attempts to separate the low level activity from the high

level activity. Informally, Noninference requires that for any trace of the system removing all high level events results in a trace that is still valid. This can be expressed as follows:

$\forall \tau: \text{traces}(S) \cdot \text{NONINFERENCE}(LLES(\tau, S))$, where

$$\text{NONINFERENCE}(A) \equiv \exists t: A \cdot t|H = \langle \rangle$$

Notice that the predicate NONINFERENCE ensures that for a given low level observations τ_{low} , τ_{low} is a possible trace.

Noninference is too strong for systems that have a high level output without a high level input. As an example consider a system the only function of which is to keep a journal of all low level events on a high level device. This system is secure. The low level user does not know anything about what high level users are doing. This system, however, does not satisfy the Noninference property.

McLean [14] extends Noninference as follows. For any trace τ , it must be possible to find another trace σ such that the low level events of τ are equal to σ and σ has no high level inputs. McLean calls this weaker property Generalized Noninference. This can be expressed as a predicate over the low level equivalent set as follows:

$\forall \tau: \text{traces}(S) \cdot \text{GN}(LLES(\tau, S))$, where

$$\text{GN}(A) \equiv \exists t: A \cdot t|HI = \langle \rangle$$

This satisfies the definition of a security property. The GN predicate ensures that the trace without any high level inputs is always possible for any low level observation. Therefore, for all possible low level observations, a trace can be found with the same low level events but with no high level inputs.

3.2 Noninterference

Noninterference is a security property introduced by Goguen and Meseguer [6] [7]. It captures the attractive notion that system security is preserved whenever high level users are prevented from influencing the behavior of low level users. Goguen and Meseguer's original definition of Noninterference was only applicable to deterministic systems. McCullough [12] [13] extended the definition to encompass non-deterministic systems.

McCullough's definition of Generalized Noninterference (GNI) can be informally defined as follows: Given a trace τ , modifying it by inserting or deleting high level inputs results in a sequence σ , which is not necessarily a valid trace. It must be possible to construct a valid trace τ' from σ by inserting or deleting high level outputs.

³ Since we are only interested in possibilistic properties we assume that each member of $LLES(\tau, S)$ is equiprobable.

We can formally define Generalized Noninterference⁴ as:

$$\forall \tau: \text{traces}(S) \cdot \text{GNI}(\text{LLES}(\tau, S)), \text{ where } \text{GNI}(A)$$

$$\equiv \forall \tau: A \cdot \forall t: \text{interleave}(\text{HI}^*, \tau|L) \cdot \exists s: A \cdot t = s|(L \cup \text{HI})$$

The function *interleave* will return all possible interleavings of its arguments.

3.3 Non-Deducible Output Security

The previous two examples were of security properties founded on the notion of preventing a LLU from deducing anything about high level inputs. Our definition of a security property is not limited to this type of security. To illustrate a different form of security we present Guttman and Nadel's Non-Deducible Output Security [8]. Non-Deducible Output Security can be expressed as:

$$\forall \tau: \text{traces}(S) \cdot \text{NDO}(\text{LLES}(\tau, S)), \text{ where}$$

$$\text{NDO}(A) \equiv \forall \tau: A \cdot \forall t: \text{traces}(S) \cdot t|L = \tau|L \Rightarrow$$

$$\exists s: A \cdot s|(H \cup L) = t|(H \cup L)$$

If the LLU sees a trace τ_{low} , he can determine the set $\text{LLES}(\tau_{\text{low}}, S)$. All of these traces are indistinguishable to a low level user from a trace s that has the same low level events as τ , but the high level events come from another trace that has the same low level input events. Since the low level user cannot determine which high level events were chosen, the observation of τ_{low} gives the user no new information about high inputs or outputs. Furthermore, since the merging was performed arbitrarily, the low observation is also compatible with all interleavings and so give no information about which interleaving occurred.

3.4 Separability

Separability is an example of perfect security [14]. Separability is perfect security because no interaction is allowed between high level and low level events⁵. It is like having two separate systems, one running the high level processes and one running the low level processes.

Separability can be defined as follows. For every pair of traces τ_1 and τ_2 the trace τ such that $\tau|L = \tau_1|L$ and $\tau|H = \tau_2|H$ is a valid trace.

No matter what the low level user observes, every possible sequence of high level events is possible. Therefore, the low level user cannot gain any new information.

This property can be formalized as:

$$\forall \tau: \text{traces}(S) \cdot \text{SEPARABILITY}(\text{LLES}(\tau, S)), \text{ where}$$

$$\text{SEPARABILITY}(A) \equiv \forall \tau: A \cdot \forall t: \text{traces}(S) \cdot \text{interleave}(t|H, \tau|L) \subseteq A$$

4. Information Flow and the Perfect Security Property

The previous section defined security properties. We indicated that the traces that must appear in the low level equivalent set are dependent on what information flows are to be prevented. In this section, we present the weakest security property such that no information can flow from high level users to low level users. We will use this result to determine the strength of the various properties presented in the literature.

Information flows from high level users to low level users when the low level users observe something they believe is connected with high level activity.

“Information is transmitted along an object when variety in the events engaged by a [high level] user can be conveyed to a [low level] user as a result of [the high level users] interaction with the object.” [5]

Separability is an example of a system with no possibilistic information flow. This is ensured because all high level traces must be consistent with any low level observation τ_{low} . However, Separability is too strong. Consider a low level user observing a sequence τ_{low} from a system S . The absence of a high level sequence from $\text{LLES}(\tau_{\text{low}}, S)$ might indicate an information flow from high level to low level users, but this is not sufficient. For example, consider a system where the only high level behavior is to echo all low level output events to a high level device for archiving. A low level user observing τ_{low} can construct $\text{LLES}(\tau_{\text{low}}, S)$ and will notice that many high level sequences are not possible. However, there is no information flow from high level users to low level users. The problem with Separability is that it does not allow low level users to influence high level activity.

The ability for low level users to influence high level events does not reduce security since the low level users will not know how they have influenced the high level outputs. Even if the low level user knows exactly what

⁴ This version of Generalized Noninterference is weaker than McCullough's because it does not require high level outputs to only be altered after the point which the high level inputs have been altered. However, this does not change any of the results and simplifies the presentation.

⁵ We stress that this is in the possibilistic sense. A Separability secure system can be constructed that has a probabilistic covert channel.

the effect of his actions are, it does not help him in determining anything new about high level activity.

To see how to weaken Separability to allow low level events to influence high level outputs consider the following formulation of Separability:

$$\forall \tau: \text{traces}(S) \cdot \tau | L \in \text{LLES}(\tau, S) \wedge \forall p, s: p^\wedge s \in \text{LLES}(\tau, S) \wedge s | H = \langle \rangle \cdot \forall \alpha: H \cdot \exists t: \text{traces}(S) \cdot p^\wedge \langle \alpha \rangle | H = t | H \Rightarrow p^\wedge \langle \alpha \rangle^\wedge s \in \text{LLES}(\tau, S)$$

This formulation differs from the one given in section 3.4 by incrementally constructing all the interleavings. This can be seen by examining the expression:

$$\forall \alpha: H \cdot \exists t: \text{traces}(S) \cdot p^\wedge \langle \alpha \rangle | H = t | H \Rightarrow p^\wedge \langle \alpha \rangle^\wedge s \in \text{LLES}(\tau, S) (1)$$

The trace $p^\wedge s$ is such that s has no high level events while p might have some high level events. Since $\tau_{\text{low}} \in \text{LLES}(\tau, S)$ we can always find such a p and s . The antecedent of (1) is true if $p | H^\wedge \langle \alpha \rangle$ is a valid high level trace. If $p | H^\wedge \langle \alpha \rangle$ is a valid high level trace then it must be possible for the event α to occur between p and s .

The possibility of α occurring between p and s is only dependent on the preceding high level events in p . Therefore, a system where low level events influence high level outputs would not satisfy Separability because Separability would require that this high level output be possible even before the event that caused it has occurred.

The following property allows α to be dependent on all of p not just the high level events.

$$\forall \tau: \text{traces}(S) \cdot \tau | L \in \text{LLES}(\tau, S) \wedge \forall p, s: p^\wedge s \in \text{LLES}(\tau, S) \wedge s | H = \langle \rangle \cdot \forall \alpha: H \cdot p^\wedge \langle \alpha \rangle \in \text{traces}(S) \Rightarrow p^\wedge \langle \alpha \rangle^\wedge s \in \text{LLES}(\tau, S)$$

PSP stands for the perfect security property. We will demonstrate why we call this property perfect below. PSP allows high level outputs to be influenced by low level events. PSP might appear complex but fortunately there exists a simple procedure to determine if a component satisfies PSP [19]. The composability of PSP is proven below. We will now prove that this property is the weakest security property to ensure no information flow.

Theorem 1: PSP does not allow any information to flow from high level users to low level users.

Proof:

Assume that there is a system S that satisfies PSP and S allows high level information to flow to low level users. Since S has an information flow it does not satisfy Separability.

Therefore, there exists a τ_{low} such that the Separability predicate is false. For a τ_{low} that makes Separability false, construct A such that:

$$\text{SEPARABILITY}(\text{LLES}(\tau_{\text{low}}, S) \cup A) \text{ is true.}$$

Let τ be a trace of A . Let p and s be traces such that they satisfy the following:

$$\exists \alpha: H \cdot p^\wedge s | L = \tau_{\text{low}} \wedge p^\wedge \langle \alpha \rangle | H \leq \tau | H \wedge p^\wedge s \notin A \wedge p^\wedge \langle \alpha \rangle^\wedge s \in A$$

Such a p and s exist A is not empty and τ_{low} is a trace of system since it satisfies PSP.

Consider $p^\wedge \langle \alpha \rangle^\wedge s$:

Case 1: $\alpha \in HI$ or $\alpha \in HO$ and α is only dependent on high level events. In this case the definition of PSP and Separability are the same. Therefore, α cannot be such an element.

Case 2: $\alpha \in HO$ and is dependent on low level events. If α cannot occur at this point in the trace it is because the conditions for it to occur have not been satisfied. Does the low level user gain any knowledge from realizing α did not occur? Since he is controlling the existence of the event, the answer is no. The existence of such a trace does imply that the low level user can covertly communicate with the high level user, but such an information flow is already allowed. \square

Theorem 2: PSP is the weakest security property that does not allow information flow from high level users to low level users.

Proof:

Theorem 1 proved that PSP does not allow information to flow from high level users to low level users. We must therefore prove that any weaker property must allow flows from high level users to low level users.

Let P be a property that is weaker than PSP that does not have any unauthorized information flows. Let S be a system that satisfies P but not PSP. Since S does not satisfy PSP and is weaker than PSP:

$$\exists p, s: p^\wedge s \in A \wedge s | H = \langle \rangle \cdot \exists \alpha: H \cdot p^\wedge \langle \alpha \rangle \in \text{traces}(S) \wedge p^\wedge \langle \alpha \rangle^\wedge s \notin A, \quad \text{where } A = \text{LLES}(\tau_{\text{low}}, S)$$

If the subtrace p occurs and if α occurs then τ_{low} could not have occurred. The low level user has deduced something about high level events. Therefore no such P exists. \square

We will now prove that PSP is a composable property. The following defines the composition of two components.

Definition 10: Composition of Components

Given $S_1 = \langle E_1, I_1, O_1, T_1 \rangle$ and $S_2 = \langle E_2, I_2, O_2, T_2 \rangle$ that satisfy

$$\begin{aligned} I_1 \cap I_2 &= \emptyset \\ O_1 \cap O_2 &= \emptyset \\ (E_1 \setminus (I_1 \cup O_1)) \cap E_2 &= \emptyset \\ (E_2 \setminus (I_2 \cup O_2)) \cap E_1 &= \emptyset \end{aligned}$$

then the composition of S_1 and S_2 produces a new component $S = \langle E, I, O, T \rangle$ such that:

$$E = E_1 \cup E_2$$

$$I = (I_1 \setminus O_2) \cup (I_2 \setminus O_1)$$

$$O = (O_1 \setminus I_2) \cup (O_2 \setminus I_1)$$

and $T = \{ a \in E^* \text{ such that } a|E_1 \in T_1 \wedge a|E_2 \in T_2 \}$

Theorem 3: The composition of two components S_1 and S_2 that satisfy PSP will yield a system S to satisfy PSP.

Proof:

Assume that the composition of two components that each satisfy PSP does not satisfy PSP. This then implies that for some τ of S :

$$\tau_{low} \notin A \vee \exists p, s: p \wedge s \in A \wedge s|H = \langle \rangle \cdot \exists \alpha: H \cdot p \wedge \langle e \rangle \in \text{traces}(S) \wedge p \wedge \langle \alpha \rangle \wedge s \notin A, \text{ where } A = \text{LLES}(\tau, S)$$

Case 1: $\tau_{low} \notin A$

Since τ is a trace of S , $\tau|E_1 \in \text{traces}(S_1)$ and $\tau|E_2 \in \text{traces}(S_2)$. Since S_1 and S_2 satisfy PSP $\tau|L_1 \in \text{traces}(S_1)$ and $\tau|L_2 \in \text{traces}(S_2)$ therefore by the definition of composition $\tau|L \in \text{traces}(S)$.

Case 2: $\exists p, s: p \wedge s \in A \wedge s|H = \langle \rangle \cdot \exists \alpha: H \cdot$

$$p \wedge \langle \alpha \rangle \in \text{traces}(S) \wedge p \wedge \langle \alpha \rangle \wedge s \notin A$$

From the definition of composability:

$$\forall \tau: E^* \cdot \tau \in \text{traces}(S) \Leftrightarrow \tau|E_1 \in \text{traces}(S_1) \wedge \tau|E_2 \in \text{traces}(S_2)$$

Since $p \wedge s \in A \Rightarrow p \wedge s \in \text{traces}(S)$, there exists

$p_1 \wedge s_1 \in \text{traces}(S_1)$ and $p_2 \wedge s_2 \in \text{traces}(S_2)$ such that

$$p|E_1 = p_1, s|E_1 = s_1, p|E_2 = p_2 \text{ and } s|E_2 = s_2.$$

$$p \wedge \langle \alpha \rangle \in \text{traces}(S) \quad (\text{Composition})$$

$$= p \wedge \langle \alpha \rangle |E_1 \in \text{traces}(S_1) \wedge p \wedge \langle \alpha \rangle |E_2 \in \text{traces}(S_2) \quad (\text{PSP})$$

$$\Rightarrow p \wedge \langle \alpha \rangle \wedge s |E_1 \in \text{traces}(S_1) \wedge p \wedge \langle \alpha \rangle \wedge s |E_2 \in \text{traces}(S_2) \quad (\text{Composition})$$

$$= p \wedge \langle \alpha \rangle \wedge s \in \text{traces}(S)$$

Both Cases yield a contradiction therefore PSP is a composable property. \square

5. Comparing Security Properties

Our formalism provides an easy method of evaluating the relative strengths of security properties. Since we have a logical expression for our properties the comparison is simple. To compare properties P and Q , evaluate $P \Rightarrow Q$ and $Q \Rightarrow P$. If the first statement is true then P is stronger than Q . If the second statement is true then Q is stronger than P . If both are true, the properties are equal. If neither are true, they are not comparable.

Example 2: We will compare Generalized Noninference to Generalized Noninterference:

First we will show that GNI implies Generalized Noninference:

$$\begin{aligned} & \forall \tau: \text{traces}(S) | L \cdot \text{GNI}(\text{LLES}(\tau, S)) \quad \text{Definition of GNI} \\ = & \forall \tau: \text{traces}(S) | L \cdot \forall t: \text{interleave}(HI^*, \tau_{low}) \cdot \exists s: \text{LLES}(\tau, S) \cdot t = s|(L \cup HI) \quad \text{Specialization with } t = \tau \\ \Rightarrow & \forall \tau: \text{traces}(S) | L \cdot \exists s: \text{LLES}(\tau, S) \cdot t = s|(L \cup HI) \quad \text{Distributive} \\ \Rightarrow & \forall \tau: \text{traces}(S) | L \cdot \exists s: \text{LLES}(\tau, S) \cdot t|L = s|L \wedge t|HI = s|HI \quad \text{Definition of LLES}(t, S) \\ = & \forall \tau: \text{traces}(S) | L \cdot \exists s: \text{LLES}(\tau, S) \cdot t|HI = s|HI \\ & \text{It has no high level events} \\ = & \forall \tau: \text{traces}(S) | L \cdot \exists s: \text{LLES}(\tau, S) \cdot \langle \rangle = s|HI \quad \text{Definition of GN} \end{aligned}$$

$$= \forall \tau: \text{traces}(S) | L \cdot \text{GN}(\text{LLES}(\tau, S))$$

Now we show that Generalized Noninference does not imply GNI:

$$\forall \tau: \text{traces}(S) | L \cdot \text{GN}(\text{LLES}(\tau, S)) \Rightarrow$$

$$\forall \tau: \text{traces}(S) | L \cdot \text{GNI}(\text{LLES}(\tau, S))$$

Definition of GNI & GN

$$= \forall \tau: \text{traces}(S) | L \cdot \exists s: \text{LLES}(\tau, S) \cdot s|HI = \langle \rangle \Rightarrow$$

$$\forall \tau: \text{traces}(S) | L \cdot \forall t: \text{interleave}(HI^*, t_{low}) \cdot \exists s: \text{LLES}(\tau, S) \cdot t = s|(L \cup HI) \quad \text{Specialization such that } t|HI \neq \langle \rangle$$

$$\Rightarrow \forall \tau: \text{traces}(S) | L \cdot \exists s: \text{LLES}(\tau, S) \cdot s|HI = \langle \rangle \Rightarrow$$

$$\forall \tau: \text{traces}(S) | L \cdot \exists s: \text{LLES}(\tau, S) \cdot t = s|(L \cup HI)$$

Distributive, Definition of LLES(t, S) and the Specialization condition

$$\Rightarrow \forall \tau: \text{traces}(S) | L \cdot \exists s: \text{LLES}(\tau, S) \cdot s|HI = \langle \rangle \Rightarrow$$

$$\forall \tau: \text{traces}(S) | L \cdot \exists s: \text{LLES}(\tau, S) \cdot \neg s|HI = \langle \rangle$$

$$= \perp$$

Therefore, GN does not imply GNI and GN is a weaker property than GNI. \square

By applying the above technique to the security properties presented above, the following lattice can be constructed.

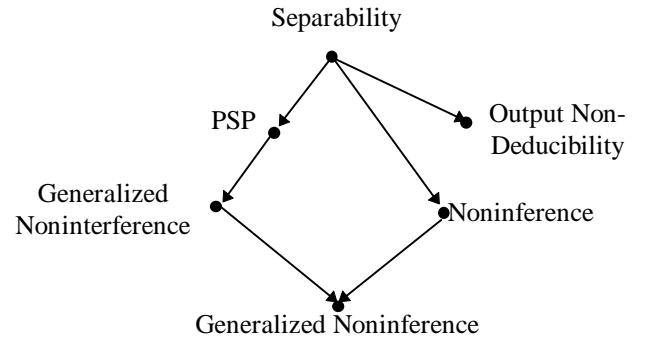


Figure 1: A Partial Ordering of Security Properties

The arrows in the lattice indicate which property implies which other. For example PSP implies Generalized Noninterference and by transitivity Generalized Noninference. An instructive way to represent part of the above lattice is to only consider the elements that can be totally ordered (see Figure 2).

Notice that PSP partitions the figure into two. This can be used to determine the strength of the properties. We can see that Separability is a stronger property than PSP. Therefore, systems with no information flow are being unnecessarily rejected. Most security properties defined in the literature are weaker than PSP. This might be surprising, but can be explained because high level interleavings are not considered by any of the weaker properties.

6. Comparison to Other Approaches

6.1 Selective Interleaving Functions

The only other general framework for the specification and analysis of security properties is McLean's Selective Interleaving Functions. In this section we compare our framework to McLean's. Specifically, we will compare the expressability of the two frameworks and the results one can obtain from each.

All of the security properties that can be expressed using SIFs must be a closure with respect to some interleaving function. If a security property cannot be so expressed, then the results of McLean's work are not applicable. It may be argued that if the security properties that cannot be handled by SIFs are "uninteresting", then SIFs are all that is required. This argument is naïve. What is considered interesting today, might not be interesting in the future. We will show that SIFs cannot represent all security properties. A framework should not place limits on what types of properties can be expressed.

PSP is an example of a property that can be expressed in our formalism but not using SIFs. Recall that PSP is similar to Separability but allows high level outputs to be dependent on low level events. A SIF for PSP would be

required to produce the interleaving of all input events and output events that are not dependent on low level events while ensuring that the high level outputs that did depend on low level events were not arbitrarily interleaved. This type of dependency is not expressible using SIFs.

In the SIF framework a component satisfies a property if and only if it is closed under some SIF. SIFs take two traces and perform an interleaving of the traces to produce a new trace. Consider the class SIFs that require the *lowin* and *lowout* from one trace and the *highin* and *highout* from the other. All of the security properties examined by McLean have this property. Furthermore, it is not clear how the events can come from different traces and still have a useful system. These SIFs can be expressed as a predicate over the low level equivalent set by ensuring that the interleaving specified by SIF is present in the low level equivalent set.

One of the main differences between our work and McLean's is our assumption of input totality. McLean requires the *a priori* knowledge of compatible traces for the composition. Our assumption of input totality removes this requirement.

Input totality makes the presentation of the results easier. If input totality were not required then it would be possible to find two components such that their cascade composition would not be allowed. Consider the communication events between two composed components S_1 and S_2 . If the outputs of S_1 were unacceptable as inputs at S_2 or an input event that must

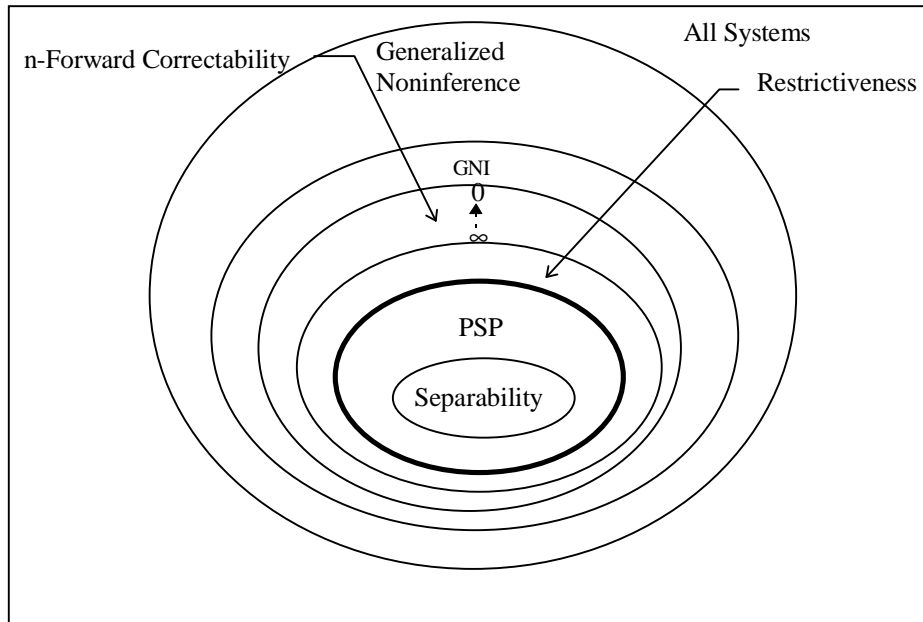


Figure 2: A Total Ordering of Most Possibilistic Properties

occur at S_2 cannot be generated by S_1 , then the composition would not succeed. Input totality removes this problem.

McLean does not require input totality in his theory of Selective Interleaving Functions. He uses an interface requirement, when it is needed, that ensures the composition will succeed. The input totality requirement can be replaced with an interface requirement. This would not change any of our results but would complicate their presentation.

McLean's composition results take the form of theorems indicating what SIF a system will satisfy if the condition of the theorem are true. If the conditions of the theorem are not satisfied, then nothing can be said about the resulting system. In our framework the property that the resulting system satisfies is evident even if the property is not composable. Furthermore, it allows one to determine why a property is not composable.

In specifying a property using SIFs, it might be required to restrict the domain of the interleavings. Unfortunately, there are domain assumptions in the composition proofs. These domain assumptions are what led McLean to incorrectly conclude Generalized Noninference was cascade composable when it is not⁶ [19]. Furthermore, even if the domain assumption had been made explicit they would not give any clue as to why Generalized Noninference failed to compose.

6.2 Security Specifications

Jacob [10] developed a model of systems that he uses to assess their security. He models components using the trace semantics of CSP [9]. Jacob's components differ from ours in one important way: there is no distinction between input events and output events.

Jacob defines a function *infer S B l* that is read as "the inferences about S that B can make having observed l " [10]. If we assume two comparable levels then Jacob's definition of *infer S B l* is equivalent to our definition of $LLES(l, S)$. Notice that by inference Jacob means what possible observations can a user make not what extra information his observation has given him.

With the definition of *infer S* Jacob defines what it means for a system R to be at least as secure as a system S . In our notation:

$$\forall l: \text{traces}(R) \cdot \exists l': \text{traces}(S) \cdot l \sqsubseteq l' \wedge \text{infer } S \text{ } l \sqsubseteq \text{infer } R \text{ } l$$

R is at least as secure as S if all possible low level traces of R are low level traces of S and the $LLES(l, S) \subseteq LLES(l, R)$. Since the low level user does not know

which element of $LLES(l, S)$ has occurred, enlarging the set enhances security.

Jacob then defines restricting information flow as (in our notation and assuming two levels):

$$\forall t: \text{traces}(S) \cdot \exists l: \text{traces}(S) \cdot t \sqsubseteq l \wedge l \sqsubseteq H = \langle \rangle$$

This definition can be seen to be equivalent to the definition of Noninference (section 3.1).

Jacob proceeds to generalize the *infer* function to allow a security specification that allows one to specify systems in which a restricted amount of information is allowed to flow from one user to another. Since our definition of a security property is a predicate over the low level equivalent set this type of specification can also be done in our framework. Furthermore, more complex security properties can be expressed in our framework.

6.3 Security Properties for CCS

Focardi and Gorrieri [3] [4] have also attempted to provide a basis for a formal comparison of security properties. They have chosen Milner's CCS [16] as a basis for modeling the components. Focardi and Gorrieri also formalized various information flow security properties and compared their strengths. Their work differs from ours because they do not provide a definition of a security property. It is this formalization of security properties that is a primary goal of our work.

7. Conclusions

In this paper we presented a framework for specifying and analyzing security properties. The definition of a security property is general and intuitively appealing. We have also presented a property that is the weakest property that allows no possibilistic information flow. We have demonstrated that this property cannot be expressed in the Selective Interleaving Functions Framework.

Acknowledgments

We would like to thank John McLean and the anonymous referees for their helpful comments and suggestions.

8. Bibliography

- [1] Martin Abadi and Leslie Lamport. "Composing Specifications," *Technical Report 66*, Digital Equipment Corporation Systems Research Center, Palo Alto, CA, 1990.
- [2] Bowen Alpern and Fred Schneider. "Defining Liveness," *Information Processing Letters*, 21(4), pages 181-185. October 1985.

⁶ This was corrected in [15]

- [3] Riccardo Focardi and Roberto Gorrieri. "A Taxonomy of Trace-based Security Properties for CCS," *Proceedings of the 7th Computer Security Foundations Workshop*, IEEE Computer Society, June 1994.
- [4] Riccardo Focardi. "Comparing Two Information Flow Security Properties," *Proceedings of the 9th Computer Security Foundations Workshop*, IEEE Computer Society, June 1996.
- [5] Simon N. Foley. "A Universal Theory of Information Flow," *Proceedings of the 1987 IEEE Symposium on Research in Security and Privacy*, pages 116-121. IEEE Press. 1987.
- [6] Joseph A. Goguen and José Meseguer. "Security Policies and Security Models," *Proceedings of the 1982 IEEE Symposium on Research in Security and Privacy*, pages 11-20. IEEE Press. April 1982.
- [7] Joseph A. Goguen and José Meseguer. "Unwinding and Inference Control," *Proceedings of the Symposium on Security and Privacy*, pages 75-86. IEEE Computer Society, May 1984.
- [8] J. D. Guttman and M. E. Nadal. "What Needs Securing?," *Proceedings of the Computer Security Foundations Workshop*, IEEE Computer Society, June 1988, pages 34-57. June 12-15 1988.
- [9] C. A. R. Hoare. "Communicating Sequential Process." London: Prentice-Hall International, UK, LTD., 1985.
- [10] Jeremy Jacob. "Security Specifications," *Proceedings of the 1988 IEEE Symposium on Research in Security and Privacy*. IEEE Press, May 1988.
- [11] Dale M. Johnson and F. Javier Thayer. "Security and the Composition of Machines," *Proceedings of the Security Foundations Workshop*, Franconia, NH, pages 72-89. June 1988.
- [12] Daryl McCullough. "Specifications for Multi-Level Security and a Hook-Up Property," *Proceedings of the 1987 IEEE Symposium on Research in Security and Privacy*. IEEE Press, May 1987.
- [13] Daryl McCullough. "Noninterference and the Composability of Security Properties," *Proceedings of the 1988 IEEE Symposium on Research in Security and Privacy*, pages 177-186. IEEE Press, May 1988.
- [14] John McLean. "A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions," *Proceedings of the 1994 IEEE Symposium on Security and Privacy*, pages 79-93. IEEE Press. May 1994.
- [15] John McLean. "A General Theory of Composition for a class of "Possibilistic" Properties," *IEEE Transactions on Software Engineering*, January 1996 Volume 22 Number 1, pages 53-67.
- [16] R. Milner. "Communication and Concurrency." Prentice-Hall.
- [17] Colin O'Halloran. "A Calculus of Information Flow," *Proceedings of the European Symposium on Research in Computer Security*. Toulouse, France. 1990.
- [18] A. Zakinthinos and E. S. Lee. "How and Why Feedback Composition Fails," *Proceedings of the Computer Security Foundations Workshop IX*. IEEE Press. June 1996.
- [19] A. Zakinthinos. "On The Composition Of Security Properties," Ph.D. dissertation, University of Toronto, Toronto, Ontario. 1996.

Appendix A: Security Properties vs. Safety/Liveness Properties

The Alpern and Schneider safety/liveness model of properties is currently the dominant model in the specification of analysis of programs [14]. Properties are regarded as sets of traces and a component satisfies a property if its set of traces is a subset of the property's set. This appendix proves that security properties cannot be expressed in the Alpern and Schneider framework.

Definition 11: Event System Space

An event system space is a 4-tuple $\langle E, I, O, T \rangle$ where E, I, O, T are defined as in the definition of an event system but with $T = E^*$. We will write \check{S} for the event system space.

Definition 12: An Element of a System Space.

A system $S = \langle E_1, I_1, O_1, T_1 \rangle$ is a subset of the system space $\check{S} = \langle E, I, O, T \rangle$ if and only if $E_1 \subseteq E, I_1 \subseteq I, O_1 \subseteq O, T_1 \subseteq T$.

Theorem 4: Security properties are not expressible as sets of traces.

Proof:

Assume that a security property P can be expressed as the intersection of a safety and liveness property. Let $T \subseteq \text{traces}(\check{S})$ be the set of traces that satisfy a security property P . Since P is the intersection of a safety and liveness property, any system that had traces $T' \subseteq T$ would also satisfy P . A security property ensures that a system has certain behaviors. For a system to satisfy a security property all low level equivalent sets must satisfy the security predicate. Construct a system $S \subseteq \check{S}$ and $\text{traces}(S) \subseteq T$ such that the security property predicate is false for some low level observation τ_{low} . Such a system exists because removing one of the required traces will make the predicate false, but will still result in the set of

traces being a subset of T . The set of traces of S does not satisfy the security property P but is a subset of T . This yields a contradiction. Therefore, P cannot be expressed as a set of traces. \square