Composing Secure Systems that have Emergent Properties

Aris Zakinthinos
Aristotle Computer Consulting Services Ltd.
Toronto, Canada
Aris@kcsl.ca

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

Abstract
A property is said to emerge if not all of its constituent components satisfy the property. This paper proves the existence of emergent properties and investigates how a property might emerge. We introduce a condition called stability on properties that allows one to predict under what circumstances a property might emerge

.

1. Introduction

Current work on the composition of components has focused on determining the effects of interconnecting components with known properties [1,2]. This is fine if what is required is the determination of the security property satisfied by the composition of the components. What these frameworks do not consider is what property do components have to satisfy so that the resulting system satisfies a given property. This question has been partially addressed by the search for composable security properties. If a set of components satisfies a composable security property then the interconnection of these components also satisfies that property.

If a system designer wants to construct a system that satisfies a particular property he can proceed in one of two ways:

1. If the property is a composable property then he can ensure that the components satisfy the property.
2. If the property is not composable or one of the components does not satisfy the property he must construct the system and verify if the resulting system satisfies the desired property.

System designers are interested in devising systems that satisfy a specified property. In this paper we examine what property or properties the system's components must satisfy so that the resulting system satisfies the desired property. For example, if a system designer wants to connect two components in cascade such that the resulting system satisfies Generalized Noninterference (GNI) then what property must each component satisfy? Since GNI is cascade composable [1,2], then if each component satisfies GNI the desired result is achieved. Is it possible that one, or both, components do not satisfy GNI yet the resulting system does satisfy GNI? As we shall demonstrate, the answer to this question is yes.

Our research leads us to an investigation of the existence of emergent properties. An emergent property is one that is not satisfied by all the constituent components of a system, but is satisfied by the overall system. If we can determine the condition for a property to emerge then it might be possible to build a desired system with that property from components that do not satisfy any security property.

2. Systems and Security Properties

The framework for our investigation into composability will be event systems [3,4]. McCullough's definition derives from the work on modeling concurrence of Hoare [5]. 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. We adopt the notation of separating events by commas and enclosing a trace 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 concatenating the 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 <X> and <Y> 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,

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

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 $t|\{a_1,a_3\} = \langle a_1,a_1,a_3 \rangle$

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 $I \cap O = \varnothing$

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

Throughout the remainder of this work we will need to refer to the set of traces of the system *S* of interest.

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* [3,4]. Formally,

Definition 6: Input Totality

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

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

This modeling abstraction simplifies the presentation of the results. There are some practical cases in which input totality does not correspond to reality. These can be modeled, but we adopt the input totality restriction for reasons that are purely notational and that concern the proofs we will present.

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, \, will also be used. The set A\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) users and low-level (untrusted or less trusted) 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 involves some cumbersome notation.

The following definition gives some designations for commonly used classes of events. We will also use the notation τ_{low} to represent a trace of only low-level events.

Definition 7: Event Classes.

The following notation will be useful in specifying security properties:

HI = H ∩ I high level input events,

LI = L ∩ I low level input events,

HO = H ∩ O high level output events and

LO = L ∩ O low level output events.

Definition 8: Low Level Equivalent Set.

Given a trace τ and a System *S*, LLES(τ,S) is the set of traces that have the same low level events as does τ, and they occur in the same order. Formally,

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

The purpose of a security property is to prevent low level users from being able to make deductions about the events that should be the concern of only the high level users. A security property is a restriction on the flow of information from any HLU to any LLU. The exact deductions that the LLU 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 with some known degree of certainty by ensuring that a low level observation can be attributed to more than one high level event sequence. That is, a security property ensures that certain traces are elements of $LLES(\tau_{low},S)$. Therefore, a system satisfies a security property if all required traces are present in all $LLES(\tau_{low},S)$. If this were not the case then the fact that some $LLES(\tau_{low},S)$ has occurred may lead to the conclusion that some HI or HO could not have occurred, which implies an information flow from HLU to LLU. All of 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 an arbitrary system *S*, *P* satisfies *S*, written P(S), if and only if every low level equivalent set of *S* satisfies *Q*. Formally,

$$P \text{ is a security Property} \Leftrightarrow$$

$$\exists Q \cdot \forall S \cdot P(S) \Leftrightarrow$$

$$(\forall \tau : traces(S) \cdot Q(LLES(\tau,S)))$$

Definition 10: Event Removing Operator.

Given an event system $S=<E,I,O,T>$ the operation $S\backslash\alpha$, $\alpha \subseteq E$ yields the following system $S'=<E',I',O',T'>$:

$$E' = E\backslash\alpha$$
$$I' = I\backslash\alpha$$
$$O' = O\backslash\alpha$$
$$\text{and } T' = \{ t \mid t|E' \in T \}$$

The event removing operator removes all occurrences of an event from the event system. This operator will be used to describe a condition on properties such that some observations can be made on how the property might emerge.

The composition of two components can be thought of as directing outputs from one component to become inputs at the other. From an operational viewpoint the output event of one component immediately becomes an input to the other component. The output event is indistinguishable from the input event. The following definition of composition is essentially Johnson & Thayer's [4] definition of simple hook-up.

Definition 11: Composition of Components

Given $S_1=<E_1,I_1,O_1,T_1>$ and $S_2=<E_2,I_2,O_2,T_2>$ that satisfy

$$I_1 \cap I_2 = \varnothing$$
$$O_1 \cap O_2 = \varnothing$$
$$(E_1 \setminus (I_1 \cup O_1)) \cap E_2 = \varnothing$$
$$(E_2 \setminus (I_2 \cup O_2)) \cap E_1 = \varnothing$$

then the compostion of S_1 and S_2 produces a new component $S=<E,I,O,T>$ 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 $T = \{ a \in E^* \text{ such that } a|E_1 \in T_1 \wedge a|E_2 \in T_2 \}$

Definition 11 defines composition in general. There are two special cases that we will be discussing bellow, product composition and cascade composition. Product composition requires that $E_1 \cap E_2 = \varnothing$. That is the two components share no events in common. Cascade composition places the restriction that $O_2 \cap I_1 = \varnothing$. Cascade composition prohibits feedback from occuring.

3. Conditions For A Property To Emerge

Consider the interconnection of two components S_1 and S_2 such that the resulting system satisfies a property P. Figure 1 illustrates such a system. The question we wish to address is what property do the components S_1 and S_2 have to satisfy such that the overall system satisfies P. If P is a composable property then a sufficient condition is for S_1 and S_2 to also satisfy P. But, as we will demonstrate, the composabilty of P is not necessary. Furthermore, if P is not a composable property, we are interested in the conclusions that can be made about S_1 and S_2.

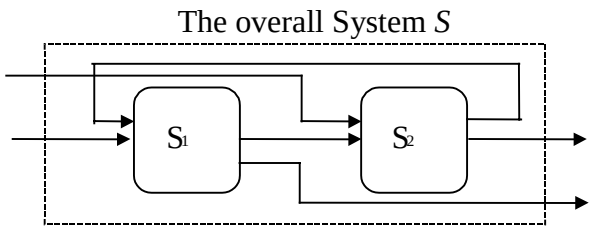

Figure 1: The Composition of S_1 and S_2
The interconnection of components of S_1 and S_2 results in a system S that satisfies a property P.

Without any information about the property P then as we will show nothing can be said about the properties that S_1 and S_2 must satisfy. What we present below is a characterisation of properties. If a property satisfies this characterisation then we can draw certain conclusions about what property S_1 and S_2 must satisfy. Otherwise, no conclusion can be made.

We begin our investigation into emergent properties by considering the product composition of two components. The following theorem states the effects of composing two systems that satisfy different properties.

Theorem 1: Given two components S_1 and S_2 that satisfy properties P_1 and P_2 respectively the product composition of S_1 and S_2 will satisfy the strongest product composable property P such that $P_1 \Rightarrow P$ and $P_2 \Rightarrow P$.

Proof:

Consider the strongest product composable property P that is implied by both P_1 and P_2. Both S_1 and S_2 also satisfy P. Since P is product composable the product composition of S_1 and S_2 satisfies P. ❑

The above theorem requires the property P to be product composable. It might seem strange that we do not take product composability as an axiom. It is, however, possible to define a property that satisfies Definition 9 but is not product composable. Whether such a property is useful is beyond the scope of this paper. All of the security properties presented in the literature, however, are product composable. The above theorem can be used to predict the result of the product composition of two components. For example, the product composition of a system that satisfies

Noninference [6] with a component that satisfies Restrictiveness [7] will yield a system that satisfies Generalized Noninference since this is the strongest product composable property implied by each[1].

Since product composition is the simplest form of composition extending the above theorem to cover cascade composition and composition with feedback would be ideal. The following characterisation of properties will be useful.

Definition 12: Stablity Characterisation.

A property *P* will be called *stable* if and only if for all systems *S*, $\forall\alpha$:power_set(E)$\cdot$P(S)$\Rightarrow$P(S\α).

We say that a property is stable if it satisfies the stability characterisation. If a system satisfies a property and the property satisfies the stability characterisation then the hiding of any combination of events of the system results in a system that still satisfies the property. Are there any such properties? We demonstrate in Appendix A that all security properties that have been presented in the literature satisfy the stability property.

The stability characterisation of a property places some structure on properties. First, a security property is defined as a predicate on the low level equivalent sets of a system (see Definition 9). If a system does not satisfy a stable property then it can be made to satisfy the property by adding traces to the LLES. For example it a system does not satisfy Separability then it can be made to satisfy it by adding the missing traces. For non-stable properties traces might be required to be added or removed (see Example 2 for a property where there might be too many traces in the LLES and the addition of new traces does not help). As is discussed non-stable properties do not inherently violate any obvious information flow properties. Second, the other benefit of stability is as follows. Suppose that a composed system satisfies a stable property *P* and for a given system trace τ a low-level trace τ_1 must be in the low level equivalent set to satisfy the required property. Then for a sub-component S_1 to satisfy the property, $\tau_1|E_1$ must be in the low level equivalent set of $\tau|E_1$. These two points seem a sensible requirement for any useful security property.

Assume that the system in Figure 1 satisfies a stable property *P*. By the definition of a stable property we can conclude $\forall$ $E_1, E_2 \cdot P(S\backslash E_2)$ and $P(S\backslash E_1)$. This follows from the stability condition that any subset of the events can be removed and the property still holds. The following theorem establishes the link between a stable property and the possibility of a composition satisfying a particular property by focusing on the events *C* that communicate between the two components.

Theorem 2: A necessary condition for the composition of two components S_1 and S_2 to yield a system that satisfies a stable property P is $P(S_1\backslash C)$ and $P(S_2\backslash C)$ where C are the communication events between S_1 and S_2.

Proof:

This follows immediately from the definition of a stable property since $\forall$ E_1, E_2 $P(S\backslash E_2)$ and $P(S\backslash E_1)$, and with E_2 = C we get that $S\backslash E_2 = S_1\backslash C$ and with E_1 = C we get that $S\backslash E_1 = S_2\backslash C$. ❑

Theorem 2 implies that one can determine *a priori* if the composition of two components might result in a system that satisfies *P*. The system designer checks to see if $P(S_1\backslash C)$ and $P(S_2\backslash C)$. If this is not true then the composition will not result in a system that satisfies *P*. Unfortunately, stability is not a sufficient condition. Generalized Noninference satisfies the stability definition but as demonstrated in [2] the composition of two Generalized Noninference secure components may or may not satisfy Generalized Noninference. The stability characterisation can be used to hide the communication events between the composed components. Therefore, if there is to be a problem in composing the components then it will occur because of the communication events.

The following example will demonstrate a property that does not satisfy the stability characterisation. It also demonstrates that it is possible for two components to be composed such that the a property of the resulting system is not shared by all of its constituent components.

Example 2: Consider the property EMERGENT:

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

$$\text{EMERGENT(B)} \equiv \exists s\text{:B}\cdot s|HI=<> \wedge \neg s|HO=<>$$

EMERGENT is Generalized Noninference [1] with the added stipulation that the output sequence cannot be empty. EMERGENT, however, is not a stable characterisation of any property because the removing of all high level output events results in a system that does not satisfy EMERGENT. Consider the cascade composition of two components such that S_1 satisfies Noninference and does not satisfy EMERGENT and S_2 satisfies EMERGENT. Such an S_1 exists because Noninference is neither implied by nor implies EMERGENT. This follows because EMERGENT ensures that the high level output sequence is not empty but Noninference requires it to be empty.

It is shown in "On the Composition of Security Properties" [8] that the resulting system satisfies EMERGENT. However, both components did not satisfy

1 We have not proven that Generalized Noninference is the strongest such property but we conjecture that it is. Even if it is not the composition does satisfy Generalized Noninference.

it. Therefore, EMERGENT has emerged under composition. But, $S\backslash E_2$ does not satisfy EMERGENT.

For the security property of a system to satisfy a stable characterisation, the externally visible behaviors of each component must satisfy the characterisation. This implies that it is possible to place a component in a system that does not satisfy the characterisation in any way as long as the component has no externally visible events. For a non-stable property the external visible behavior of the components that satisfy it has no bearing on whether the system satisfies the property.

It might seem that non-stable security properties can be excluded from consideration through information flow arguments. This, however, is not possible. Consider the property EMERGENT. This property implies Generalized Noninference. Therefore, it is no worse (in an information flow sense) than Generalized Noninference. Are there useful non-stable security properties? Probably not. We conjecture that all non-stable security properties imply a stable property that is no worse (as far as information flow is concerned) then the non-stable property.

4. Implications

From the above discussion we see that if composition is to fail it is because of the communication events. With this in mind we can devise some rules to help the system designer. We begin by examining sufficient conditions for the cascade composition of two components to yield a system that satisfies a stable property *P*. We reiterate that stability is required if we are to make any observations about emergence. From the above discussion we know that $P(S_1\backslash C)$ and $P(S_2\backslash C)$ where the C are the communication events. If S_2 satisfies a property P_2 such that $P_2 \Rightarrow P$ and P_2 is cascade composable without any requirement on the communication events[2] and S_1 satisfies P_1 such that $P_1 \Rightarrow P$ then the cascade composition of S_1 and S_2 will satisfy P. This is formalized in the following theorem:

Theorem 3: A sufficient condition for the cascade composition of two components S_1 and S_2 to satisfy a stable property P is if S_1 satisfies P_1 and S_2 satisfies P_2 such that $P_1 \Rightarrow P$ and $P_2 \Rightarrow P$ and the property P_2 is a cascade composable property with no restrictions on communication events.

[2] For example, Noninference is cascade composable but has a requirement for no communication events while Generalized Noninterference has no such requirement.

Proof:

Assume that the cascade composition of S_1 and S_2 does not satisfy *P*. Then for some τ there is at least one missing trace τ_1 from the low level equivalent set of the composed system. From the stability criteria on *P* for S_1 or S_2 to satisfy *P*, $\tau_1|E_1$ and $\tau_2|E_2$ must be in the low level equivalent set for $\tau|E_1$ and $\tau|E_2$ respectively. Since S_1 satisfies P ($P_1 \Rightarrow P$) $\tau_1|E_1$ is in the low level equivalent set 0f S_1. Since S_2 satisfies P ($P_2 \Rightarrow P$) and is cascade composable with no restrictions on communication events $\tau_1|E_2$ is in the low level equivalent set of S_2. This leads to a contradiction. ❑

Theorem 2 can be used in two ways. One way is to determine what the effects of the cascade composition of two components will be. For example, the cascade composition of a Generalized Noninference secure system and a Separability secure system will yield a system that satisfies Generalized Noninference. The other way to use this system is as follows. Given that the system must satisfy a property *P* and that we have a component S_1 that satisfies P_1 ($P_1 \Rightarrow P$) then using stability we know that S_2 must satisfy $P(S_2\backslash C)$, where C are the communication events. From the above theorem we know that a sufficient condition for S_2 is that $P_2 \Rightarrow P$ and P_2 be cascade composable with no restrictions on the communication events.

The above theorem demonstrates how to use stability to get a sufficient condition for the cascade composition of two components to satisfy a given property.

We can extend the above result to composition with feedback.

Theorem 4: A sufficient condition for the composition of two components S_1 and S_2 to satisfy a stable property P is $P_1 \Rightarrow P$ and $P_2 \Rightarrow P$ and the property P_1 is a cascade composable property with no restrictions on communication events and P_2 is a composable property with no restrictions on communication events.

Proof:

The proof is similar to that of Theorem 3. ❑

5. Conclusions

In this paper we have discussed the existence of emergent properties. We presented a condition on properties such that if the property satisfies the condition, predictions can be made about how the property might emerge.

Further, if we know the target property of the composed system and we know the property of one of the

component systems, we can find out if there is a property that the other component could have that will result, after composition, in the target system satisfying the target property. If this property can exist we can find it. Whether the component could be constructed is largely dependent on the interaction of this property we have determined and its functionality.

6. Acknowledgements

We would like to thank The Naval Research Laboratory and Kaman Data Sciences for their support.

7. Bibliography

[1] 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.

[2] A. Zakinthinos and E. S. Lee. "A General Theory of Security Properties and Secure Composition," *Proceedings of the 1997 IEEE Symposium on Research in Security and Privacy*. IEEE Press, June1998.

[3] 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.

[4] 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.

[5] C. A. R. Hoare. "Communicating Sequential Process." London: Prentice-Hall International, UK, LTD., 1985.

[6] Colin O'Halloran. "A Calculus of Information Flow," *Proceedings of the European Symposium on Research in Computer Security*. Toulouse, France. 1990.

[7] 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.

[8] A. Zakinthinos. "On The Composition of Security Properties," Ph.D. Dissertation. University of Toronto, 1997.

Appendix A- The Stability for Various Security Properties

In this Appendix we will demonstrate that the stability requirement is satisfied by most of the security properties presented in the literature. Recall the definition of a stable property:

Definition 12: Stable Property.

A property *P* will be called *stable* if and only if for all systems *S*, $\forall\alpha$:power_set(E)$\cdot$P(S)$\Rightarrow$P(S\α).

Separability

In a Separability secure system all interleavings of high level traces and low level traces are present. Removing an event, be it high level or low level, might reduce the number of traces but all interleavings are still possible.

PSP, GNI and N-Forward Correctability

All of these properties are very similar and hence a similar argument demonstrates that they are stable properties. Consider a system, *S*, that satisfies one of the above properties. Removing a low level event from *S* will result in a system that still satisfies the property since all perturbations of low level traces still have a correction but there are fewer low level traces to consider. Removing a high level input event is not a problem because this can be viewed as a perturbation to the traces *S* . We will now examine removing a high level output event from *S*. We will argue that this results in a system that still satisfies the property. Consider a trace τ of *S* such that a perturbation σ requires some particular high level output event to have a correction τ'. The trace τ' with the high level outputs removed is a trace of the new system and is a correction to the σ perturbation. Therefore all of the properties are stable.

Noninference and Generalized Noninference

Removing low level events and high level input events from a system that satisfies either of these properties will result in a system that still satisfies the property. Also removing high level outputs for a Noninference secure system is acceptable since the property requires the trace with no high level events to be trace of the original system. By the same argument as that given for the PSP, GNI and N-Forward Correctability class of property Generalized Noninference is a stable property.