Causal Security

Miranda Mowbray
Hewlett Packard Pisa Science Centre,
Corso Italia 115, Pisa, Italy
mjfm@hplb.hpl.hp.com

Abstract

This paper gives a new definition of Security, which takes causal information into account. The new definition can be used to determine the security of nondeterministic concurrent systems for which high-level information may be either input into the system during its operation, or inherent in the original state of the system. It is possible to have systems which are secure under this definition which write to audit before performing each transition. The definition satisfies several useful composition properties, including one which gives it some protection from Trojan horse attacks.

1 Introduction

In this paper I will give a new definition of Security, which uses causal information. The main idea is very simple; if there is a link between the states of two objects in the system, then this indicates a breach of security in the system only if the fact that one object has the state that it has *causes* the second object to have a related state.

This observation was made explicitly by John McLean in [8], who used it to simplify a definition of security using just the assumption that the partial order of causality between substates is generated by the causal relations between substates one time tick apart, and that a substate cannot be influenced by anything in the future. This assumption leads to a good approximation of the real causal dependencies in many cases, but causes problems for nondeterministic systems in which there is a certain type of high-level audit. Suppose that, before any transaction is performed, a note of the transaction is first written to a high-level audit file, (so that if the system crashes during the transaction its entire state can be recovered; this is called the *audit-first* requirement in [8]) and that while this audit happens all the low-level objects in the system remain unchanged. Suppose also that there is a reachable state from which two transactions are possible which result in different low-level states. Then according to McLean's definition FM the system is insecure. This is because if c is a computation which reaches the state, does the audit for one of the two transactions, and then does the audited transaction, the final values of the low-level objects after c can be deduced with probability 1 from the history of the high and low-level objects up to and including the audit step, but cannot be deduced with probability 1 from the history of the low-level objects up to and including the audit step. However, it is intuitively clear that the properties of the system described should not in themselves constitute a security violation. Although the low level state after c can be determined by the state of the high level audit file after the audit step, the state of the audit file is not a cause of the low level state after c: instead, both are effects of the decision taken before the audit step.

In John McLean's model the system description does not include any causal information. In contrast, in this paper I assume that all the causal dependencies between the states of different objects before and after a transition of the system are observable in the system view. This extra information is enough for a definition of Security under which systems of the type described above are not automatically insecure.

One of the most common definitions of Information Flow Security is that of Nondeducibility [9]. A system is Nondeducible (under the instantiation described by Sutherland) if there there are no computations of the system in which information flows from high-level inputs to the low-level objects of the system. Nondeducibility is simple and intuitively appealing but suffers from some drawbacks. One of these is that only information which comes from high-level inputs is protected. If a system contains high-level information which is inherent in the initial state of the system, rather than being input into the system during its operation, then Sutherland's instantiation of Nondeducibility cannot protect it. Moreover, any system can be made Nondeducible simply by internalizing all the high level-inputs.

It is possible to protect high-level information which does not come from inputs by using an instantiation of Nondeducibility different from the one recommended by Sutherland, so that information flow is forbidden from *any* high-level object to the low-level objects: however according to this instantiation of Nondeducibility, any system with a high-level audit file is insecure. (See [8] for a discussion.)

There have been several other definitions of security using information flow, for instance [1, 2, 4, 6, 8, 10, 11]. Many of these were motivated by the limitations of Nondeducibility, but most only protect high-level input.

The security definition given in this paper, *Causal Security*, gives results corresponding to intuition for systems with the audit system described above, satis-

fies several useful composition properties, and is able to protect information which is not manifested as high-level input. It is similar to Nondeducibility. A system where the only high-level information is high-level input is Causally Secure if and only if it is Nondeducible.

Causal Security suffers from some of the drawbacks associated with Nondeducibility; for instance, it does not detect possible security loopholes to do with timing or probability. Moreover, the definition of Causal Security is relatively complicated. For a system where the only high-level information is high-level input, or where (for instance) it is important to detect probabilistic attacks, it makes sense to use a different definition.

Causal Security has been used to analyse the security of an implementation of the P_{EDI} protocol [3] for electronic data interchange. This protocol is (like most protocols) large and complicated, so the composition results for Causal Security were crucial in making the analysis tractable. In the implementation, which was for most points the simplest implementation of the protocol, there was some high-level information which was inherent in the original state of the system rather than being high-level input. The protocol requires audit capabilities, and the implementation had the type of high-level audit described above. Although Causal Security was not designed with this protocol in mind, it turned out to be a very useful tool for analysing security for this example, offering a combination of advantages that none of the other definitions mentioned above could match.

The major drawback of analysis using Causal Security as opposed to the other definitions of security based on information flow is that because it uses information on causal dependencies in the system, it requires a greater investment of time in understanding the details of how the system behaves. In practice the need to study the behaviour of the system closely may bring advantages as well as disadvantages: the analysis of the P_{EDI} protocol detected some strange behaviour with regard to the causality in forward notification.

2 The model for the system

This paper will use a system model which is similar to, but not the same as, a model which is used in the design of delay-insensitive circuits [5]. This section describes the system model.

The system consists of a set of objects with associated security levels. The specification of the system says how the values of the objects can change in a transition of the system, as in the specification of a state machine. However the specification gives more information than a state machine specification; it also says for each possible transition which objects cause the transition and which are the objects whose changes of value are effects of the transition.

The system consists of objects indexed over a set I. The possible states of the object O_i, where $i \in I$, are in a nonempty set D_i. There are two types of objects, buffering objects and non-buffering objects. I will illustrate the difference between these two types later. If O_i is a buffering object then $D_i = A_i^*$ for some set A_i. (Throughout this paper, B^* denotes the set of finite strings over B.) The state space D is the Cartesian product of all the D_i. There is a subset $\mathcal{M}$ of D, which is the set of legitimate initial states of the system.

If J is a subset of I, I will write v_J to signify a function from J to $\bigcup_{j \in J} D_j$ satisfying $v_J(j) \in D_j$ for all $j \in J$, and I will write v_j as shorthand for $v_J(j)$.

Each statement in the system specification is a quadruple

$$(J, v_J, K, w_K)$$

where $J, K \subseteq I$. Each quadruple represents a transition of the system; the transition of the system corresponding to the quadruple (J, v_J, K, w_K) is caused by the objects $\{O_j : j \in J\}$ and affects the objects $\{O_k : k \in K\}$. An object may be a cause of a transition which affects it, so $J \cap K$ may be non-empty.

In fact, a system specification will not usually list all the quadruples, but will give schemata to obtain them; for instance, a specification may contain the schema

$$(\{1,2,3\}, v_{\{1,2,3\}}, \{3,i\}, w_{\{3,i\}}) :$$

$$v_2 \neq 0, v_3 = 0, w_3 = v_2, i \geq 6$$

which is shorthand for a set (possibly an infinite set) of quadruples. In order to make some definitions simpler I will assume that the set of quadruples in the system specification always contains the null quadruple

$$(\emptyset, v_\emptyset, \emptyset, w_\emptyset)$$

where $\emptyset$ denotes the empty set.

2.1 Behaviour of the system

Each quadruple

$$Q = (J, v_J, K, w_K)$$

defines a partial function (the associated transition) on D. If $u_I \in D$ then $u_I Q$ is defined if and only if $front(u_j) = v_j$ for all $j \in J$, where $front(u_j)$ is u_j if O_j is a non-buffering object and is the front value in the queue of values u_j if O_j is a buffering object.

The behaviour of an object under the partial function depends on whether it is a buffering object or a non-buffering object. I will describe the behaviour of non-buffering objects first; it is simpler because non-buffering objects have no memory.

If $u_I Q$ is defined and O_i is a non-buffering object then

$$(u_I Q)_i = u_i \text{ if } i \in I \backslash K$$

$$(u_I Q)_i = w_i \text{ if } i \in K$$

For example, if there are two non-buffering objects O_{paper} and $O_{Miranda}$ with appropriate sets of values, I can say in the specification

> If the paper is written, and Miranda is at the office, these facts can cause Miranda to go home.

This rule would be written as the quadruple

$$(\{paper, Miranda\}, v, \{Miranda\}, w_{\{Miranda\}})$$

where $w_{Miranda}$=*home*, and v is the function from $\{paper, Miranda\}$ satisfying v_{paper}=*written*, $v_{Miranda}$=*office*.
Notice that $O_{Miranda}$ is both a cause and subject to an effect of the associated transition, and that the values of all other non-buffering objects (in fact, of all other objects) are left unchanged by the transition.

In this example, the value of O_{paper} is still *written* after the transition, and there is no memory in the state of the system after the transition that Miranda has ever been in the office. I would also like to model objects that have a type of memory, and these are the buffering objects. The existence of buffering objects makes the model more complicated, but gives an advantage: following the ideas in [10], input and output ports can be modelled with buffering objects, and if this is done then it is only necessary to forbid information flow from the *initial* states of objects.

A buffering object has a state which is a queue of values. The immediate behaviour of the object is determined entirely by the front value in the queue. If a buffering object is one of the causes of a transition then the front value in the queue is deleted in the course of the transition. If a buffering object is affected by a transition, then it is affected by having a new value attached to the back of the queue.

Formally speaking, if $u_I Q$ is defined and O_i is a buffering object then

$$(u_I Q)_i = u_i \text{ if } i \in I \backslash (J \cup K)$$

$$(u_I Q)_i = x_i \text{ if } i \in J \backslash K, u_i = x_i.front(u_i)$$

$$(u_I Q)_i = w_i.u_i \text{ if } i \in K \backslash J$$

$$(u_I Q)_i = w_i.x_i \text{ if } i \in J \cap K, u_i = x_i.front(u_i)$$

For example, there can be a buffering object $O_{Authors}$ whose value is a list of the authors of e-mail messages which I have received but not read, in chronological order, and a buffering object O_{Docs} that records a list of documents on which I ought to work. The specification can have the rule

> If the oldest e-mail message which I haven't seen is from Vladi, when I read the message this can cause the addition to the list of documents of a reply to Vladi.

This rule can be written as the quadruple

$$(\{Authors\}, v_{\{Authors\}}, \{Docs\}, w_{\{Docs\}})$$

where $v_{Authors} = Vladi$ and $w_{Docs} = ReplyVladi$.
In a state of the system where the value of $O_{Authors}$ is $Giangi.Cicci.Vladi$ and the value of O_{Docs} is $Competition$, it is possible to apply this rule to obtain a new state in which $O_{Authors}$ has value $Giangi.Cicci$ and O_{Docs} has value $ReplyVladi.Competition$. (The values of the other objects are unchanged.) I have not forgotten the competition, and I have taken $Vladi$ away from the list of authors. Although the value of $O_{Authors}$ changed in the course of the transition, $O_{Authors}$ is not counted as one of the effects of the transition, because the change of value is just a side-effect of the behaviour of a buffering object that is one of the causes of a transition.

Notice that in general the behaviour of a system from a given initial state under a specification like the one described is non-deterministic.

Given a finite string S of quadruples there is a partial function from D to D defined in the obvious way:

> $u_I S.Q$ is defined if and only if both
> (a) $u_I S$ is defined and
> (b) $(u_I S)Q$ is defined.
> Moreover, if $u_I S.Q$ is defined, it is equal to $(u_I S)Q$.

If λ is the empty string then $u_I \lambda$ is defined and equal to u_I for all $u_I \in D$. (Throughout the rest of this paper λ will denote the empty string or queue.)

2.2 Input and output ports

Input and output ports are modelled as buffering objects. An input port is a buffering object and is not an effect of any transition, and an output port is a buffering object and not a cause of any transition. They are assumed to behave in the way described in [10], where the initial state of an input port is the string of all the external inputs at that port, the initial state of an output port is the empty string, and in the course of an evolution of the system the length of the string which is the state of an input port decreases, and the length of the string which is the state of an output port increases.

Suppose the object O_i is an input port. I will assume that O_i has the following behaviour. O_i is a buffering object, and no new values are admitted to the buffer during the course of any evolution of the system. The set D_i is A_i^* where A_i is the set of possible input values to the system. The initial value of the input port will represent the string of inputs to the system, as in [10]. This is a simplification of the general behaviour of input ports, because it assumed that as soon as one input value has been used in a transition, the next value is ready. I will also assume that any state which differs from a valid initial state just by the value of O_i is still a valid initial state. In particular, every string of values in D_i is valid as an initial state of O_i. This means that any strategy by the external environment is valid, as far as this particular type of port is concerned.

Formally speaking,

- For each quadruple (J, v_J, K, w_K) in the specification, $i \notin K$
- If $d_I \in \mathcal{M}$, $d'_j = d_j$ for all $j \in I \backslash \{i\}$, and $d'_i \in D_i$, then $d'_I \in \mathcal{M}$

Output ports are assumed to have a similar behaviour; they are buffering objects O_j for which

- For each quadruple (J, v_J, K, w_K) in the specification, $j \notin J$

- If $d_I \in \mathcal{M}$, then $d_j = \lambda$

3 Definition of causal security

This section defines what it means for a system to be causally secure with respect to a given policy.

3.1 Security policy

A security policy is given by a set of ordered pairs $(H(a), L(a))$ of nonintersecting subsets of I. (The letters H,L stand for High-level and Low-level.) The security policy given by the set $\{(H(a), L(a)) : a \in A\}$ (where A is an indexing set, and for each $a \in A$, $(H(a), L(a))$ is a pair of non-intersecting subsets of I) is satisfied if there is no information flow with a causal link from the initial states of the objects in the set $\{O_i : i \in H(a)\}$ to the set of objects $\{O_i : i \in L(a)\}$ for any $a \in A$.

Since information flow is only forbidden from the *initial* states of the objects in $\{O_i : i \in H(a)\}$, I have assumed that it is impossible to obtain high-level information without using some input from a high-level input port or some information present in the initial state of a high-level object. As McLean points out [8], this is not generally true: for instance, any system which generates cryptographic keys converts low-level input seeds into high-level output. The justification of the assumption is that the low-level users are expected to know not just the specification of the low-level parts of the system, but the specification of the entire system. In the cryptographic key example a low-level user would know the algorithm used by the key generator, although not the initial state of the object which performs the algorithm, if there are several possibilities for this. There is also an assumption that the low-level users have access to as much time and computing power as the high-level users.

In most definitions of security using information flow it is assumed that the objects $\{O_i : i \in H(a) \cup L(a)\}$ are all either input ports or output ports. This assumption is not made here. In fact, Causal Security with respect to a policy satisfying this assumption is equivalent to Nondeducibility.

3.2 The set of causes

In this subsection I will define a set $causes(c_I, S)$ which is the set indexing the objects whose initial states are the causes of the evolution of the system with initial state c_I and transitions S. It is an extension of the idea that if S is given by the quadruple (J, v_J, K, w_K), the causes of S are the objects $\{O_j : j \in J\}$.

Informally, k is in $causes(c_I, S)$ if the fact that O_k initially has value c_k affects the course of the computation indicated by the pair (c_I, S). The formal definition uses some supplementary sets $causes_n(c_I, S)$, where k is in $causes_n(c_I, S)$ if the computation is affected by the n^{th} entry (reading from right to left) of the string c_k, which is the initial value of O_k.

The formal definition of *causes* is as follows. It is the function from pairs (u_I, S) such that S is a finite string of quadruples in the specification and $u_I S$ is defined, to subsets of I, satisfying the following. ($length(u_i)$ denotes the length of the string u_i.)

- $causes(u_I, S) = \bigcup_{n \geq 1} causes_n(u_I, S)$
- $causes_n(u_I, \lambda) = \emptyset$ for all u_I and $n \geq 1$
- If Q is the quadruple (J, v_J, K, w_K), $u_I Q.S$ is defined, and O_i is a non-buffering object, then $i \in causes_n(u_I, Q.S)$ if and only if $n = 1$ and $i \in J \cup (causes_1(u_I Q, S) \backslash K)$
- If Q is the quadruple (J, v_J, K, w_K), $u_I Q.S$ is defined, O_i is a buffering object, and $length(u_i) < n$ then $i \notin causes_n(u_I, Q.S)$
- If Q is the quadruple (J, v_J, K, w_K), $u_I Q.S$ is defined, O_i is a buffering object, and $length(u_i) \geq n \geq 1$, then $i \in causes_n(u_I, Q.S)$ if and only if *either* $n = 1$ and $i \in J$, *or* $i \in (causes_{n-1}(u_I Q, S) \cap J)$, *or* $i \in (causes_n(u_I Q, S) \backslash J)$

The element i of I is in $causes(u_I, S)$ if and only if the action of S on u_I is influenced by the initial state of O_i. It is in $causes_n(u_I, S)$ if and only if the action of S on u_I is influenced by the n^{th} entry (reading from right to left) in the string u_i. Notice that if $u_I S$ is defined, and u'_I is such that $u'_i = u_i$ for all $i \in causes(u_I, S)$, then $u'_I S$ is defined.

If O_i is a buffering object, then it follows from the definitions that i is not a cause of any valid computation starting from a state in which O_i is empty. This is because it is assumed that the causal dependence in the system is achieved through the transfer of data, and an empty buffering object has no data to transfer.

3.3 Definition

What can low-level users see during a sequence of transitions of the system? The answer for this model is that they can see all the values of the low-level objects during the computation. They can see if two low-level objects change their values at the same time, as the result of the same transition. But if there is a transition which does not change any low-level value, they cannot tell whether or not this transition has taken place; from the point of view of a low-level user there is no discernible difference between such a transition and the idle transition (represented by the quadruple $(\emptyset, v_\emptyset, \emptyset, w_\emptyset)$) which does nothing at all. I will write "(c_I, S') simulates (a_I, S) on L", if a user who can only see the values of objects in $\{O_i : i \in L\}$ cannot distinguish the evolution with initial state c_I and sequence of transitions S' from the evolution with initial state a_I and sequence of transitions S. For fixed L, the relation given by simulation on L is an equivalence relation.

The formal definition is as follows. Suppose that there are two specifications with sets of quadruples $Quad1$, $Quad2$, possible initial states $\mathcal{M}1$, $\mathcal{M}2$, and index sets $I1, I2$, and suppose that L is a subset of $I1 \cap I2$. Given a finite string $S = Q_1, \ldots, Q_\ell$ of quadruples in $Quad1$ and a state $a_{I1} \in \mathcal{M}1$ such that $a_{I1} S$ is defined, say that a pair consisting of a state $c_{I2} \in \mathcal{M}2$ and a string $S' = Q'_1, \ldots, Q'_m \in Quad2^*$ *simulates* (a_{I1}, S) on L if and only if $c_{I2} S'$ is defined, and there are $0 = i(0) < i(1) \leq i(2) \leq \ldots \leq i(k) = m$, $i(k+1) =$

$m+1$, such that for each r and s such that $0 \leq r \leq k$ and $i(r) \leq s < i(r+1)$, and for each $\ell \in L$,

$$(c_{I2}Q'_1 \ldots Q'_s)_\ell = (a_{I1}Q_1 \ldots Q_r)_\ell$$

Finally, here is the definition of security which will be used in this paper.

A system is *causally secure* with respect to the security policy $\{(H(a), L(a)) : a \in A\}$ (where A is an indexing set, and for each $a \in A$, $(H(a), L(a))$ is a pair of non-intersecting subsets of I) if and only if it is causally secure with respect to the policy $(H(a), L(a))$ for each $a \in A$.

A system is *causally secure* with respect to the security policy (H, L) if and only if whenever $u_I, d_I \in \mathcal{M}$ and S is a finite string of quadruples of the system such that $u_I S$ is defined,
then there is some $c_I \in \mathcal{M}$ and a string S' of quadruples of the system, such that

- $c_I S'$ is defined
- (c_I, S') simulates (u_I, S) on L
- $c_k = d_k$ for all $k \in causes(c_I, S') \cap H$.

I will write $CS(H, L)$ as shorthand for "causally secure with respect to the security policy (H, L)".

3.4 Causal security and Nondeducibility

A non-causal definition of security based on Nondeducibility would say

> Given a valid initial state d_H of the high-level objects, and a valid evolution of the system, (which may start from any valid initial state, not just d_H) there is another valid evolution of the system for which the initial state of the high-level objects is d_H, such that a low-level user cannot distinguish this new evolution from the original evolution.

On the other hand, Causal Security uses the causal information present in the system model. The difference is that I don't insist that the initial state of all the high-level objects in the simulating computation is d_H. I only insist that the initial state of each object in H *which is a cause of the new computation* is as in the state d_H. The point is that I don't care about the original state of the objects which are not causes, because the values of these objects cannot influence the low-level objects in the course of the computation. An example is an audit file, which is high-level but influences nothing and so does not cause security problems.

So the definition of Causal Security says

> Given a valid initial state d_H of the high-level objects, and a valid evolution of the system, there is a valid evolution of the system for which the initial state of each high-level object O_i which is a cause of the computation is d_i, such that a low-level user cannot distinguish this new evolution from the original evolution.

Clearly, if a system satisfies the non-causal definition of security for some security policy, then it is causally secure with respect to the same policy. There is a tempting, but erroneous, argument that the converse is also true, which goes like this. Suppose that (u_I, S) is a valid computation in a system which is $CS(H, L)$, and d_I is a valid initial state. Then there is some valid computation (c_I, S') which simulates (u_I, S) on L such that $c_k = d_k$ for all $k \in causes(c_I, S')$. For each $i \in I$, let $c'_i = c_i$ if $i \notin H$, $c'_i = d_i$ if $i \in H$. Then $c'_I S'$ is defined, (c'_I, S') simulates (u_I, S) on L, and $c'_H = d_H$. This is true, but it does not follow that the system satisfies the non-causal definition, because in general c'_I is not a possible initial state of the system. In fact there are many systems which are $CS(H, L)$ but which do not satisfy the non-causal definition with respect to the policy (H, L).

However, if all the objects in $H \cup L$ are either input or output ports, then the indexed set c'_I in the argument above *is* a possible initial state of the system, and so the system satisfies the non-causal definition if and only if it is is $CS(H, L)$. Moreover, in this particular case the non-causal definition is equivalent to Nondeducibility. The subtlety of the definition of Causal Security lies in the fact that it can deal with systems where there is high-level information given by the original value of objects which are not input objects.

3.5 Example

Consider the following system, which is a very simplified version of the encryption part of the User Agent in the P_{EDI} implementation.

The system consists of three objects, a buffering object $O_{message}$, an output object O_{output}, and a non-buffering object O_{key}. There is a function *encrypt* from $A_{message} \times A_{key}$ to A_{output}. The quadruples are given by the following schemata.

$$(\emptyset, v_\emptyset, \{message\}, w_{\{message\}})$$

$$(\emptyset, v_\emptyset, \{key\}, w_{\{key\}})$$

$$(\{message, key\}, v_{\{message,key\}}, \{output\}, w_{\{output\}}) :$$

$$w_{output} = encrypt(v_{message}, v_{key})$$

In other words, the User Agent may add a message to the list of messages ready to send, may change the key, or may encrypt the oldest message in the list with the key and send it. The set of initial states consists of all functions $v_{\{message,output,key\}}$ satisfying $v_{message} \in A^*_{message}$, $v_{output} = \lambda$, and $v_{key} \in A_{key}$.

Notice that $O_{message}$ is not an input port. One reason for this is that the list of messages is not "ready input", that is, it may be necessary to wait between the sending of one message and the appearance of the next in the message list. Another reason is that according to the protocol, the User Agent is able to add to the message list (for example, for automatic acknowledgements) without any interaction from the human user. The quadruple describing possible additions to the message list describes both such action by

the User Agent and additions by the user. The fact that the key can be changed in the system at random may seem strange: this is a result of the fact that only one User Agent is being modelled here. In the model of the whole protocol the changes to the key are synchronized with the receiving User Agent, and there is a general result ensuring that the system given by synchronizing the key changes of secure User Agents is secure.

This system is $CS(\{message, key\}, \{output\})$ if and only if for every $x \in A_{output}$ and $m \in A_{message}$, there is some $k \in A_{key}$ such that $encrypt(m, k) = x$. This is seems to be a sensible result.

However, some definitions of security which were developed for systems described just in terms of input and output ports do not perform so well on this example. For example, any system without input ports satisfies Nondeducibility [9], Generalized Noninterference [6], Nondeducibility on Strategies [11], Forward Correctability [4], and Feedback Non-Deducibility on Views [10], so in particular the system above with the encryption function $encrypt(m, k) = m$ is secure according to all these definitions, even though it sends the high-level messages straight out as low-level output. (Obviously, it is silly to even try to apply these definitions to a model where there is high-level information which does not come from high-level inputs.)

Restrictiveness (the state-machine definition given in [6]) looks promising, because its systems model includes transitions which are neither inputs nor outputs. However, the system described above does not satisfy Restrictiveness, no matter which encryption function is used. To show this, pick $m \in A_m$ and $k \in A_k$ and consider the two states v, w of the system, where $v_{message} = m$, $w_{message} = v_{output} = w_{output} = \lambda$, $v_{key} = w_{key} = k$. These two states correspond to the same low-level state. From the state v it is possible to perform a transition giving low-level output $encrypt(m, k)$. There are no transitions of the system from any state which give high-level output. Therefore, for Restrictiveness to hold, it should be possible to perform a transition starting at w, giving low-level output $encrypt(m, k)$. But there are no transitions giving output which are immediately possible from w, so the system is not Restrictive. The problem arises because Restrictiveness requires outputs to be simulated with sequences of outputs, rather than arbitrary sequences with the correct low-level part.

Since this system is nondeterministic, its security cannot be analysed using Noninterference [2] or Bieber and Cuppens' causality [1].

McLean's FM [8] gives a sensible answer for this system, and in fact can detect probabilistic attacks, which Causal Security cannot. However, FM was unsuitable for the analysis of the P_{EDI} implementation because the implementation had the type of high-level audit described in the introduction, and there were reachable states from which transitions with different effects on the low-level objects were possible, and so the implementation was automatically insecure according to FM.

In the rest of this paper, I will describe how to add the audit behaviour described in the introduction to an unaudited system without losing causal security, and I will give several ways of composing causally secure systems to produce a more complicated system which is still secure. The results will be stated without proof. The proofs are generally easy, although sometimes fiddly.

4 Audit files

As mentioned in the introduction, Causal Security behaves sensibly with respect to systems satisfying audit behaviour described there.

This section describes a way to add a high-level audit output to a causally secure system in such a way that the resulting system has the audit behaviour described and retains causal security. This involves adding a special object to the system which ensures that the transitions are taken in the right order, as well as the audit output port.

Suppose that the system System1 is $CS(H, L)$. Define System2 as follows.

- The index set of System2 is $I2 = I1 \cup \{a, b\}$, where $I1$ is the index set of System1; $a \notin I1$ will be the index of the audit file, which will be a buffering object, and $b \notin I1$ will be the index of a non-buffering object which ensures that the audit is done first
- The domain D_a is as in the previous subsection, viz. the set of finite strings whose entries are either possible initial states of System1 or quadruples of System1. The domain D_b is $Quad1 \cup \{e\}$, where $Quad1$ is the set of quadruples of System1, and $e \notin Quad1$. If $i \in I1$ then the domain D_i is as in System1
- The set $\mathcal{M}2$ of possible initial states of System2 is the set of states $u_{I1 \cup \{a,b\}}$ for which $u_b = e$ and u_a is equal to $u_{I1 \cup \{a,b\}} \mid_{I1}$ (the restriction of $u_{I1 \cup \{a,b\}}$ to $I1$), and moreover $u_a \in \mathcal{M}1$, the set of possible initial states of System1
- The set Quad2 of quadruples for System2 is

$$\{(\{b\}, v_{\{b\}}, K \cup \{b\}, w_{K \cup \{b\}}) :$$
$$w_b = e, v_b = Q = (J, v_J, K, w_{K \cup \{b\}} \mid_K) \in Quad1\}$$
$$\cup \{(J \cup \{b\}, v_{J \cup \{b\}}, \{a, b\}, w_{a,b}) :$$
$$v_b = e, w_a = w_b = (J, v_{J \cup \{b\}} \mid_J, K, w_K) \in Quad1\}$$

System2 has the audit behaviour as described in the introduction. It is straightforward to prove that it is $CS(H \cup \{a, b\}, L)$.

5 Composition properties

Although some important security properties are not easily composeable, if a security property *can* be captured with a definition which satisfies some composition results then this makes it much easier to work with. This is because the proof that a large system is secure can be performed in stages, where first small subsystems are proved secure and then composition

rules are used to show that the system built from these subsystems is also secure. In this section I will describe several ways in which a causally secure system can be changed while still retaining some security.

5.1 Parallel composition of systems

Given two causally secure systems with disjoint object sets, the system which is the parallel composition of these two (without communication) is causally secure.

Formally speaking, suppose that the two systems System1, System2 have index sets $I1$, $I2$ (with $I1 \cap I2 = \emptyset$) and sets of quadruples $Quad1$ and $Quad2$. Let the sets of initial states for these be $\mathcal{M}1$ and $\mathcal{M}2$. Define System1 || System2 to be the system such that

- The index set is $I = I1 \cup I2$
- The set of quadruples is $Quad1 \cup Quad2$
- The set of initial states is $\mathcal{M} = \mathcal{M}1 \times \mathcal{M}2$

Notice that the operator || is associative, so that $System1 \parallel System2 \parallel System3$ is well defined.

If H, L are subsets of $I1 \cup I2$, and System1 and System2 are causally secure with respect to the policies $(H \cap I1, L \cap I1)$ and $(H \cap I2, L \cap I2)$ respectively, then System1 || System2 is $CS(H, L)$.

5.2 A feedback property

The following lemma is concerned with a property of the type described in [10]. The idea of the feedback property is that output from an output port in a secure system can be fed back as input to an input port, without causing a breach of security.

For the feedback property described in this section a value is only present in the queue of values to the fed object if it has been released from the feeding object; in other words, the initial state of the fed object is λ. This assumption will be lifted later in this paper, where there is a treatment of "open feedback"; under open feedback a fed object may be fed values from the environment as well as values released from the feeding object.

Clearly the feedback cannot be done arbitrarily. For instance, if a system is $CS(H, L)$ then feeding back output from an object not in L into an object in L may cause a breach of security. Therefore there is a restriction on the security status of the pair of objects between which the feedback takes place. Moreover, the values which are fed back must be acceptable to the object which receives them. This gives the following condition.

Condition on i, j
O_j is an output port and O_i is an input port of a system such that the value sets A_i, A_j satisfy $A_j \subseteq A_i$. Moreover, the system is $CS(H, L)$ and either $i \in H$ or $i, j \in L$.

Given System1 satisfying the condition on i, j, define a new system, $feed(j, i$,System1), as follows.

- The index set is $I \backslash \{j\}$, where I is the index of System1
- The set Quad2 of quadruples is obtained by substituting i for j wherever it occurs in the set Quad1 of quadruples for System1
- The object O_i is a buffering object. The other objects are buffering objects if and only if the object with the same index in System1 is a buffering object
- A state $u_{I \backslash \{j\}}$ is in the set $\mathcal{M}2$ of possible initial states if and only if $u_i = \lambda$ and there is some $v_I \in \mathcal{M}1$ (the set of possible initial states of System1) such that $v_k = u_k$ for all $k \in I \backslash \{i, j\}$

Then $feed(j, i$,System1) is $CS(H \backslash \{j\}, L \backslash \{j\})$.

It is easy to check that if System1 is such that System2 $= feed(j(1), i(1), feed(j(2), i(2)$,System1)) is defined, (where $i(1), i(2), j(1), j(2)$ are all different) then System2 is equal to

$$feed(j(2), i(2), feed(j(1), i(1), System1))$$

For ease of notation I will write $feed(L, System)$, where $L = \{(j(1), i(1)), \ldots, (j(n), i(n))\}$, to denote

$$feed(j(1), i(1), \ldots, feed(j(n), i(n), System)) \ldots).$$

5.3 A hookup property

Suppose that System1 and System2 are systems whose objects are indexed by the disjoint sets $I1$, $I2$ respectively, and whose sets of possible initial states are $\mathcal{M}1$, $\mathcal{M}2$. Let $I(out) = \{j(1), \ldots, j(m)\}$ be a set of output ports of System1 || System2, (in other words, let each $j(r)$ be either an output port of System1 or an output port of System2) and let $I(in) = \{i(1), \ldots, i(m)\}$ be a set of input ports of System1 || System2. Form the hookup system System3 by feeding the output at $j(r)$ for $1 \leq r \leq m$ into $i(r)$. Formally, the System3 is

$$feed(L, System1 \parallel System2)$$

where $L = \{(j(1), i(1)), \ldots, (j(m), i(m))\}$. Let $H1, L1$ and $H2, L2$ be nonintersecting subsets of $I1$ and of $I2$, such that System1 is $CS(H1, L1)$ and System2 is $CS(H2, L2)$. Suppose that for each $1 \leq r \leq m$,

- $A_{j(r)} \subseteq A_{i(r)}$
- Either $i(r) \in H1 \cup H2$, or $i(r) \in L1 \cup L2$ and $j(r) \in L1 \cup L2$

Then it follows directly from the previous two properties that System3 is

$$CS((H1 \cup H2) \backslash I(out), (L1 \cup L2) \backslash I(out))$$

This hookup property is not the same as the composition used in [6], under which data which is fed into an input node must then be used as a cause of a transition. In contrast, in the composition described here it is possible for data to be fed into an input node and then never used. Nondeducibility is not preserved under the composition of [6] but it *is* preserved under the composition described here.

5.4 Two very simple systems

In order to build more complicated systems it is useful to introduce an *interleaver* and a *doubler*, which are very simple systems which can be used for flow control inside larger systems.

An *interleaver* $\bigvee(i(1), i(2), o)$ has input ports $O_{i(1)}$, $O_{i(2)}$, a single output port O_o, and no other objects. The sets A_O, $A_{i(1)}$, and $A_{i(2)}$ are equal. The quadruples for this system are given by the schemata

$$(\{i(1)\}, v_{\{i(1)\}}, \{o\}, w_{\{o\}}) : v_{i(1)} = w_o$$

$$(\{i(2)\}, v_{\{i(2)\}}, \{o\}, w_{\{o\}}) : v_{i(2)} = w_o$$

A *doubler* $\bigwedge(i, o(1), o(2))$ has output ports $O_{o(1)}$, $O_{o(2)}$, a single input port O_i, and no other objects. The sets $A_{o(1)}, A_{o(2)}$ are both equal to A_i. The quadruples for this system are given by the schema

$$(\{i\}, v_{\{i\}}, \{o(1), o(2)\}, w_{\{o(1),o(2)\}}) : v_i = w_{o(1)} = w_{o(2)}$$

The following results hold for these two systems:
$\bigvee(i(1), i(2), o)$ is $CS(\{(H(a), L(a)) : a \in A\})$ if and only if there is no $a \in A$ such that $(H(a), L(a))$ is the pair $\{(\{i(1), i(2)\}, \{o\}), (\{i(1)\}, \{i(2), o\})$, or $(\{i(2)\}, \{i(1), o\})$.
$\bigwedge(i, o(1), o(2))$ is $CS(\{(H(a), L(a)) : a \in A\})$ if and only if there is no $a \in A$ such that $i \in H(a)$ and $L(a) \neq \emptyset$.

5.5 An open feedback property

If System1 is a system such that $feed(j, i, System1)$ is defined, there is no object with index j in $feed(j, i, System1)$, because the object O_j in System1 disappears when its values are fed back. Moreover, the object with index i in $feed(j, i, System1)$ no longer accepts input from the environment, because the only values which it receives are those fed back. An alternative way of turning System1 into a more complicated system with feedback from O_j to O_i is to output all values arriving at O_j as well as feeding them back, and to allow values to reach O_i from the environment as well as from O_j.

Given System1, a system which has this behaviour is System2, where System2 is

$$feed(L, (\bigvee(i(1), i(2), o(3)) \mid\mid System1$$

$$\mid\mid \bigwedge(i(3), o(1), o(2))))$$

where $L = \{(o(2), i(2)), (o(3), i), (j, i(3))\}$, none of $i(1), i(2), i(3), o(1), o(2), o(3)$ are in the index set for System1, and $A_{i(1)} = A_{i(2)} = A_{o(3)} = A_i$, $A_{o(1)} = A_{o(2)} = A_{i(3)} = A_j$.

Now suppose that System1 is $CS(H, L)$ and either $i \in H$ or $i, j \in L$. Let $H2, L2$ be the sets obtained from H, L by adding $i(1)$ and $i(2)$ if H (resp.L) contains i, and replacing j if it occurs by the pair $i(3), o(1)$. It is straightforward, but tedious, to show that System2 is $CS(H2, L2)$.

5.6 Protection against high-level Trojans

Finally, I will describe a composition property which gives some protection against Trojan horse attacks. The connection between composition and protection against Trojans was first noticed by Millen [7]. The sort of attack against which the property gives protection is a machine which can intercept input before it reaches the system, read output from the system, and combine these two to give new input to the system in place of the input from the external environment. This is the type of attack described in [7]. The threat of such an attack is that high-level information may reach low-level objects as the result of this interference. The security condition which I will describe will assign high-level security to all the objects of the Trojan; the idea is that the Trojan should not influence the low-level operation of the system. I will assume that the Trojan can only input to high-level input ports. This ensures that it cannot, for instance, feed high-level output directly back into a low-level input port.

Let System1 be a system with index set I, containing elements i, j such that O_i is an input port and O_j is an output port. Let T be a system with index set J which has an empty intersection with I and contains elements $i(1), i(2), o(3)$ such that the objects $O_{i(1)}$ and $O_{i(2)}$ are input ports in T, $O_{o(3)}$ is an output port, $A_j \subseteq A_{i(2)}$, and $A_{o(3)} \subseteq A_i$. Let $o(1), o(2), i(3)$ be elements not in $I \cup J$ and define $A_{o(1)}$, $A_{o(2)}$, and $A_{i(3)}$ to equal A_j. Then System2 behaves as the composite system consisting of System1 and the Trojan T, where System2 is

$$feed(L, System1 \mid\mid T \mid\mid \bigwedge(i(3), o(1), o(2)))$$

where $L = \{(o(3), i), (j, i(3)), (o(2), i(2))\}$.

Suppose that System1 is $CS(H, L)$, where $i \in H$. Let $H2, L2$ be the sets obtained from H, L by replacing j by $\{o(1), i(3)\}$ wherever it occurs. It can be proved that System2 is $CS((H2 \cup J) \setminus \{o(3)\}, L2)$, as required.

Acknowledgements

I would like to thank the anonymous referee who pointed out several errors in a previous version of this paper, and Fabio Gadducci, who did the analysis of the P_{EDI} protocol.

References

[1] P. Bieber and F.Cuppens, "A Definition of Secure Dependencies using the Logic of Security," in *Proc. Computer Security Foundations Workshop IV*, pp.2-11, 1991.

[2] J. A. Goguen and J. Meseguer, "Security Policies and Security Models," in *Proc. IEEE Computer Society Symposium on Research in Security and Privacy*, pp.11-20, 1982.

[3] R. Hill, *EDI and X.400 using P_{EDI}*, Technology Appraisal Ltd., Middlesex, 1990.

[4] D. M. Johnson and F. J. Thayer, "Security and the Composition of Machines," in *Proc. Computer Security Foundations Workshop, Franconia, NH*, 1988.

[5] Alain J. Martin, "Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits," *UT Year of Programming Institute on Concurrent Programming*, ed. C.A.R. Hoare, publ. Addison-Wesley, 1989.

[6] D. McCullough, "Noninterference and the Composability of Security Properties," in *Proc. IEEE Computer Society Symposium on Research in Security and Privacy*, pp.177-186, 1988.

[7] Jonathan K. Millen, "Hookup Security for Synchronous Machines," IEEE document TH0315-2/90/0000/0084, pp.84-89, 1990.

[8] John McLean, "Security Models and Information Flow," in *Proc. IEEE Computer Society Symposium on Research in Security and Privacy*, pp.180-187, 1990.

[9] D. Sutherland, "A Model of Information," in *Proc. of the 9th National Computer Security Conference, Gaithersburg, MD.*, September 1986.

[10] Vladimiro Sassone and Vijay Varadharajan, "A Unifying Petri Net Model of Non-Interference and Non-Deducibility Security," Technical Report, HP Labs Bristol, submitted to the Journal of Computer Security.

[11] J. Todd Wittbold and Dale M. Johnson, "Information Flow in Nondeterministic Systems," in *Proc. IEEE Computer Society Symposium on Research in Security and Privacy*, pp.144-161, 1990.