8.2. Deterministic Noninterference

The example above suggests an alternative view of security phrased in terms of **interference**. In essence, a system is secure if groups of subjects cannot interfere with one another. In the first example in Section 8.1, the "interference" would be Matt's interfering with Holly's acquiring the CPU for her process. Intuitively, this notion is more inclusive than "writing" and enables us to express policies and models more simply. Gougen and Meseguer [412] used this approach to define security policies.

To begin, we view a system as a state machine consisting of a set $\mathbf{S} = \{ \mathbf{s}_1, \dots \}$ of subjects, a set $\Sigma = \{ \sigma_0, \sigma_1, \dots \}$ of states, a set $\mathbf{O} = \{ \mathbf{o}_1, \dots \}$ of outputs, and a set $\mathbf{Z} = \{ \mathbf{z}_1, \dots \}$ of commands. For notational convenience, we define a set of state transition commands $\mathbf{C} = \mathbf{S} \times \mathbf{Z}$, because in what follows the clearance of the subject executing the command affects the actual command that is performed.

> **Definition 8–1.** A **state transition function T:** $\mathbf{C} \times \Sigma \rightarrow \Sigma$ describes the effect of executing command **c** when in state σ, and an **output function P:** $\mathbf{C} \times \Sigma \rightarrow \mathbf{O}$ describes the output of the machine on executing command **c** in state σ. Initially, the system is in state σ_0.

We do not define any inputs, because either they select the specific commands to be executed or they can be encoded in the set of state transition commands. If the number **x** is to be input, we simply define a command that corresponds to reading **x**. We can encode the initial state as a command; the system begins at the empty state (and the first command moves it to σ_0). This notation simplifies the abstract system.

In this system, the state transition commands produce outputs. The outputs are therefore functions of the transition commands, and thus are functions of the inputs and the initial state. We have also assumed that the system is deterministic, since the state transition functions are functions, and time is not considered. We will relax this restriction later.

EXAMPLE: Consider a machine with two bits of state information, **H** and **L** (for "high" and "low," respectively). The machine has two commands, **xor0** and **xor1**, which exclusive-or both bits with 0 and 1, respectively. There are two users: Holly (who can read high and low information) and Lucy (who can read only low information). The system keeps two bits of state (**H, L**). For future reference, this will be called the **two-bit machine**. For this example, the operation affects both state bits regardless of whether Holly or Lucy executes the instruction. (This is not a requirement of the two-bit machine and will be varied later on.)

The set Σ = { (0, 0), (0, 1), (1, 0), (1, 1) }. The set **S** = { Holly, Lucy }. The set **C** = { **xor0, xor1** }. Table 8-1 shows the result of the state transition function. The output function for Holly is both bits; for Lucy, it is the **L** bit only.

Table 8-1. State Transition Function

Commands	Input states (*H*, *L*)			
	(0, 0)	(0, 1)	(1, 0)	(1, 1)
xor0	(0, 0)	(0, 1)	(1, 0)	(1, 1)
xor1	(1, 1)	(1, 0)	(0, 1)	(0, 0)

Next, let us relate outputs to state. Two observations will make the formulation straightforward. First, **T** is inductive in the first argument, because $\mathbf{T}(\mathbf{c}_0, \boldsymbol{\sigma}_0) = \boldsymbol{\sigma}_1$ and $\mathbf{T}(\mathbf{c}_{i+1}, \boldsymbol{\sigma}_{i+1}) = \mathbf{T}(\mathbf{c}_{i+1}, \mathbf{T}(\mathbf{c}_i, \boldsymbol{\sigma}_i))$. This gives us the notion of applying a sequence of commands to an initial state, so let $\mathbf{C}^*$ be the set of sequences of commands in **C**—that is, $\mathbf{C}^*$ is the transitive closure of **C** under composition. Then $\mathbf{T}^*: \mathbf{C}^* \times \Sigma \to \Sigma$, where

$$\mathbf{c}_s = \mathbf{c}_0, \ldots, \mathbf{c}_n \Rightarrow \mathbf{T}^*(\mathbf{c}_s, \boldsymbol{\sigma}_i) = \mathbf{T}(\mathbf{c}_n, \mathbf{T}(\mathbf{c}_{n-1}, \ldots, \mathbf{T}(\mathbf{c}_0, \boldsymbol{\sigma}_i)\ldots))$$

Second, the output function **P** is also inductive in the second argument. This allows us to define a similar function $\mathbf{P}^*: \mathbf{C}^* \times \Sigma \to \mathbf{O}$, which gives the sequence of outputs resulting from a sequence of commands to a system beginning at an initial state.

Given the assumptions above, the outputs provide a record of the system's functioning. The problem is that some subjects are restricted in the outputs (and actions) they can see. In the first example in Section 8.1, Holly should not have seen Matt's outputs, but Matt could see any outputs from Holly. We make this notion rigorous.

Definition 8–2. Let $\mathbf{T}^*(\mathbf{c}_s, \boldsymbol{\sigma}_i)$ be a sequence of state transitions for a system. Let $\mathbf{P}^*(\mathbf{c}_s, \boldsymbol{\sigma}_i)$ be the corresponding outputs. Then $\mathbf{proj}(\mathbf{s}, \mathbf{c}_s, \boldsymbol{\sigma}_i)$ is the set of outputs in $\mathbf{P}^*(\mathbf{c}_s, \boldsymbol{\sigma}_i)$ that subject **s** is authorized to see, in the same order as those outputs appear in $\mathbf{P}^*(\mathbf{c}_s, \boldsymbol{\sigma}_i)$.

In this definition, each command may produce some output, but subjects with insufficient clearance may not be able to see that output, lest they deduce information about the previous state of the system. The function $\mathbf{proj}(\mathbf{s}, \mathbf{c}_s, \boldsymbol{\sigma}_i)$ is simply the list of outputs resulting from removing the outputs that **s** is not authorized to see.

This captures the notion that **s** may not see all outputs because the security policy may restrict **s**'s access to them. However, **s** may not have knowledge of all commands, either, and so we need a corresponding definition for them.

Definition 8–3. Let $\mathbf{G} \subseteq \mathbf{S}$ be a group of subjects, and let $\mathbf{A} \subseteq \mathbf{Z}$ be a set of commands. Define $\pi_{\mathbf{G}}(\mathbf{c}_s)$ as the subsequence of $\mathbf{c}_s$ obtained by deleting all elements $(\mathbf{s}, \mathbf{z})$ in $\mathbf{c}_s$ with $\mathbf{s} \in \mathbf{G}$. Define $\pi_{\mathbf{A}}(\mathbf{c}_s)$ as the subsequence of $\mathbf{c}_s$ obtained by deleting all elements $(\mathbf{s}, \mathbf{z})$ in $\mathbf{c}_s$ with $\mathbf{z} \in \mathbf{A}$. Define $\pi_{\mathbf{G},\mathbf{A}}(\mathbf{c}_s)$ as the subsequence of $\mathbf{c}_s$ obtained by deleting all elements $(\mathbf{s}, \mathbf{z})$ in $\mathbf{c}_s$ such that both $\mathbf{s} \in \mathbf{G}$ and $\mathbf{z} \in \mathbf{A}$.

This purge function π captures the notion that certain command executions must be invisible to some subjects. Applying the purge function to an output string generates the output string corresponding to those commands that the subject is allowed to see. For a specific system, the desired protection domains would dictate the membership of **G** and **A**.

EXAMPLE: In the two-bit machine, let $\sigma_0 = (0, 1)$. Holly applies the command **xor0**, Lucy the command **xor1**, and Holly the command **xor1** to the state machine. The commands affect both state bits, and both bits are output after each command. We take $\mathbf{c_s}$ to be the sequence (Holly, **xor0**), (Lucy, **xor1**), (Holly, **xor1**). The output is 011001 (where bits are written sequentially, the **High** bit being first in each pair).

proj(Holly, $\mathbf{c_s}$, $\boldsymbol{\sigma_0}$) = 011001

proj(Lucy, $\mathbf{c_s}$,$\boldsymbol{\sigma_0}$) = 101

$\pi_{\text{Lucy}}(\mathbf{c_s}) = \pi_{\text{Lucy},\mathbf{xor1}}(\mathbf{c_s})$ = (Holly, **xor0**), (Holly, **xor1**)

$\pi_{\text{Holly}}(\mathbf{c_s})$ = (Lucy, **xor1**)

$\pi_{\text{Lucy},\mathbf{xor0}}(\mathbf{c_s})$ = (Holly, **xor0**), (Lucy, **xor1**), (Holly, **xor1**)

$\pi_{\text{Holly},\mathbf{xor0}}(\mathbf{c_s}) = \pi_{\mathbf{xor0}}(\mathbf{c_s})$ = (Lucy, **xor1**), (Holly, **xor1**)

$\pi_{\text{Holly},\mathbf{xor1}}(\mathbf{c_s})$ = (Holly, **xor0**), (Lucy, **xor1**)

$\pi_{\mathbf{xor1}}(\mathbf{c_s})$ = (Holly, **xor0**)

Intuitively, if the set of outputs that any user can see corresponds to the set of inputs that that user can see, then the system is secure. The following definition formalizes this as "noninterference."

Definition 8–4. Let **G, G'** $\subseteq$ **S** be distinct groups of subjects and let **A** $\subseteq$ **Z** be a set of commands. Users in **G** executing commands in **A are noninterfering with** users in **G'** (written **A,G** :| **G'**) if and only if, for all sequences $\mathbf{c_s}$ with elements in **C***, and for all **s** $\in$ **G', proj(s,** $\mathbf{c_s}$, $\boldsymbol{\sigma_i}$) = **proj(s**, $\pi_{\mathbf{G,A}}(\mathbf{c_s})$, $\boldsymbol{\sigma_i}$).

If either **A** or **G** is not present, we handle it in the obvious way.

EXAMPLE: Consider the sequence $\mathbf{c_s}$ = (Holly, **xor0**), (Lucy, **xor1**), (Holly, **xor1**) in the two-bit machine with operations affecting both state bits and both bits being output after each command. Take **G** = { Holly }, **G'** = { Lucy }, and **A** = Ø. Then $\pi_{Holly}(\mathbf{c_s})$ = (Lucy, **xor1**), so **proj**(Lucy, $\pi_{Holly}(\mathbf{c_s})$, $\boldsymbol{\sigma_0}$) = 1. This means that the statement { Holly } :| { Lucy } is false, because **proj**(Lucy, $\mathbf{c_s}$, $\boldsymbol{\sigma_0}$) = 101 ≠ **proj**(Lucy, $\pi_{Holly}(\mathbf{c_s})$, $\boldsymbol{\sigma_0}$). Intuitively, this makes sense, because commands issued to change the **H** bit also affect the **L** bit.

EXAMPLE: Modify the set of commands above so that Holly can alter only the **H** bit and Lucy only the **L** bit. Consider the sequence $\mathbf{c_s}$ = (Holly, **xor0**), (Lucy, **xor1**), (Holly, **xor1**). Given an initial state of (0, 0), the output is $0_{\mathbf{H}}1_{\mathbf{L}}1_{\mathbf{H}}$, where the subscripts indicate the security level of the output. Take **G** = { Holly }, **G'** = { Lucy }, and A = Ø; so $\pi_{Holly}(\mathbf{c_s})$ = (Lucy, **xor1**) and **proj**(Lucy, $\pi_{Holly}(\mathbf{c_s})$, $\boldsymbol{\sigma_0}$) = (1). Now we have **proj**(Lucy, $\mathbf{c_s}$, $\boldsymbol{\sigma_0}$) = **proj**(Lucy, $\pi_{Holly}(\mathbf{c_s})$, $\boldsymbol{\sigma_0}$), and { Holly } :| { Lucy } holds. Again, intuition suggests that it should, because no action that Holly takes has an effect on the part of the system that Lucy can observe.

We can now formulate an alternative definition of a security policy. By Definition 4–1, a security policy describes states in which forbidden interferences do not occur (authorized states). Viewed in this light [412], we have:

Definition 8–5. A **security policy** is a set of noninterference assertions.

The set of noninterference relations defined in Definition 8–4 characterize the security policy completely. An alternative, less common but more elegant, approach begins with the notion of a security policy and characterizes noninterference in terms of that definition [858].

Consider a system **X** as a set of protection domains **D** = { $\mathbf{d_1}$, ..., $\mathbf{d_n}$ }. Associated with **X** are states, commands, subjects, and transition commands. Whenever a transition command **c** is executed, the domain in which it is executed is written **dom**(**c**).

Definition 8–5A. Let **r** be a reflexive relation on **D** × **D**. Then **r** defines a security policy for **M**.

The relation **r** defines how information can flow. If $\mathbf{d_i r d_j}$, then information can flow from domain $\mathbf{d_i}$ to domain $\mathbf{d_j}$. Otherwise, it cannot. Because information can flow within a protection domain, $\mathbf{d_i r d_i}$. Note that this definition says nothing about the **content** of the security policy; it merely defines what a "policy" is.

We can define a function π' analogous to the π in Definition 8–3, but the commands in **A** and the subjects in **G** and **G'** are now part of the protection domains, so we express π' in terms of protection domains.

Definition 8–3A. Let **d** ∊ **D**, **c** ∊ **C**, and $\mathbf{c_s}$ ∊ **C***. Then $\pi_{\mathbf{d}}'(\nu) = \nu$, where ν is the empty sequence. If **dom**(**c**)**rd**, then $\pi_{\mathbf{d}}'(\mathbf{c_s c}) = \pi_{\mathbf{d}}'(\mathbf{c_s})\mathbf{c}$. Otherwise, $\pi_{\mathbf{d}}'(\mathbf{c_s c}) = \pi_{\mathbf{d}}'(\mathbf{c_s})$.

This says that if executing **c** will interfere with protection domain **d**, then **c** will be "visible." Otherwise, the resulting command sequence has the same effect as if **c** were not executed.

Given this definition, defining noninterference security is immediate.

Definition 8–4A. Let a system consist of a set of domains **D**. Then it is **noninterference-secure with respect to the policy r** if $\mathbf{P}^*(\mathbf{c}, \mathbf{T}^*(\mathbf{c_s}, \sigma_0)) = \mathbf{P}^*(\mathbf{c}, \mathbf{T}^*(\pi_{\mathbf{d}}'(\mathbf{c_s}), \sigma_0))$.

Rather than defining a projection function (as in Definition 8–2), consider the set of states related by an equivalence relation with respect to a domain of a command.

Definition 8–2A. Let $\mathbf{c} \in \mathbf{C}$ and $\mathbf{dom(c)} \in \mathbf{D}$, and let $\sim^{\mathbf{dom(c)}}$ be an equivalence relation on the states of a system **X**. Then $\sim^{\mathbf{dom(c)}}$ is **output-consistent** if $\sigma_{\mathbf{a}} \sim^{\mathbf{dom(c)}} \sigma_{\mathbf{b}}$ implies $\mathbf{P}(\mathbf{c}, \sigma_{\mathbf{a}}) = \mathbf{P}(\mathbf{c}, \sigma_{\mathbf{b}})$.

Two states are output-equivalent if, for the subjects in **dom(c)**, the projections of the outputs for both states after **c** is applied are the same. This immediately leads to the following lemma.

Lemma 8–1. Let $\mathbf{T}^*(\mathbf{c_s}, \boldsymbol{\sigma}_0) \sim^{\mathbf{d}} \mathbf{T}^*(\pi_{\mathbf{d}}(\mathbf{c_s}), \boldsymbol{\sigma}_0)$ for $\mathbf{c} \in \mathbf{C}$. Then, if $\sim^{\mathbf{d}}$ is output-consistent, **X** is noninterference-secure with respect to the policy **r**.

Proof Take $\mathbf{d} = \mathbf{dom(c)}$ for some $\mathbf{c} \in \mathbf{C}$. Applying Definition 8–2A to the hypothesis of this claim, $\mathbf{P}^*(\mathbf{c}, \mathbf{T}^*(\mathbf{c_s}, \boldsymbol{\sigma}_0)) = \mathbf{P}^*(\mathbf{c}, \mathbf{T}^*(\pi_{\mathbf{dom(c)}}(\mathbf{c_s}), \boldsymbol{\sigma}_0))$. But this is the definition of noninterference-secure with respect to **r** (see Definition 8–4A).

Contrasting this approach with the more common first approach illuminates the importance of the security policy. In the first approach, the security policy was defined in terms of noninterference, but it arose from the conditions that caused the particular set of subjects **G** and the commands **A**. So, in some sense, Definition 8–5 is circular. The second approach eliminates this circularity, because noninterference is characterized in terms of a defined security policy. However, the second approach obscures the relationship among subjects, commands, and noninterference requirements because of the abstraction of "protection domains." The notion of outputs characterizing commands is crucial, because information flow is defined in terms of outputs. So both characterizations have their places.

8.2.1. Unwinding Theorem

The unwinding theorem links the security of sequences of state transition commands to the security of the individual state transition commands. The name comes from "unwinding" of the sequence of commands. This theorem is central to the analysis of systems using noninterference, because it reduces the problem of proving that a system design is multilevel-secure to proving that if the system design matches the specifications from which certain lemmata are derived, then the design is mathematically certain to provide multilevel-security correctly.[2]

[2] This says nothing about the **implementation** of that design, of course. See Part 6, "Assurance."

We follow Rushby's treatment [858]. The next two definitions provide the necessary background.

Definition 8–6. Let **r** be a policy. Then a system **X locally respects r** if **dom(c)** being noninterfering with $\mathbf{d} \in \mathbf{D}$ implies $\boldsymbol{\sigma}_{\mathbf{a}} \sim^{\mathbf{d}} \mathbf{T}(\mathbf{c}, \boldsymbol{\sigma}_{\mathbf{a}})$.

If the command **c** does not have any effect on domain **d** under policy **r**, then the result of applying **c** to the current state should appear to have no effect with respect to domain **d**. When this is true, the system locally respects **r**.

Definition 8–7. Let **r** be a policy and $\mathbf{d} \in \mathbf{D}$. If $\sigma_a \sim^{d} \sigma_b$ implies $T(c, \sigma_a) \sim^{d} T(c, \sigma_b)$, the system **X** is **transition-consistent** under policy **r**.

Transition consistency simply means that states remain equivalent with respect to **d** for all commands **c**.

Lemma 8–2. Let $c_1, c_2 \in C$, $d \in D$, and for some policy **r**, $(dom(c_1), d) \in r$ and $(dom(c_2), d) \in r$. Then $T^*(c_1c_2, \sigma) = T(c_1, T(c_2, \sigma)) = T(c_2, T(c_1, \sigma))$.

Proof See Exercise 2.

Theorem 8–1. Unwinding Theorem. Let **r** be a policy, and let **X** be a system that is output consistent, transition consistent, and locally respects **r**. Then **X** is non-interference secure with respect to the policy **r**.

Proof The goal is to establish that $\sigma_a \sim^{d} \sigma_b$ implies $T^*(c_s, \sigma_a) \sim^{d} T^*(\pi_d'(c_s), \sigma_b)$. We do this by induction on the length of c_s.

Basis. If $c_s = \nu$, $T^*(c_s, \sigma_a) = \sigma_a$ and $\pi_d'(\nu) = \nu$. The claim follows immediately.

Hypothesis. Let $c_s = c_1 \ldots c_n$. Then $\sigma_a \sim^{d} \sigma_b$ implies $T^*(c_s, \sigma_a) \sim^{d} T^*(\pi_d'(c_s), \sigma_b)$.

Step. Consider c_sc_{n+1}. We assume $\sigma_a \sim^{d} \sigma_b$. We must show the consequent. We consider two distinct cases for $T^*(\pi_d'(c_sc_{n+1}), \sigma_b)$.

1. If $(dom(c_{n+1}), d) \in r$, by definition of T^* and definition 8-3A,

   $T^*(\pi_d'(c_sc_{n+1}), \sigma_b) = T^*(\pi_d'(c_s)c_{n+1}, \sigma_b) = T(c_{n+1}, T^*(\pi_d'(c_s), \sigma_b))$.

   As **X** is transition consistent, if $\sigma_a \sim^{d} \sigma_b$, then $T(c_{n+1}, \sigma_a) \sim^{d} T(c_{n+1}, \sigma_b)$. From this and the induction hypothesis,

   $T(c_{n+1}, T^*(c_s, \sigma_a)) \sim^{d} T(c_{n+1}, T^*(\pi_d'(c_s), \sigma_b))$.

   Substituting for the right side from the previous equality,

   $T(c_{n+1}, T^*(c_s, \sigma_a)) \sim^{d} T^*(\pi_d'(c_sc_{n+1}), \sigma_b)$.

   From the definition of T^*, this becomes

   $T^*(c_sc_{n+1}, \sigma_a) \sim^{d} T^*(\pi_d'(c_sc_{n+1}), \sigma_b)$.

   proving the hypothesis.

2. If $(dom(c_{n+1}), d) \notin r$, by definition 8-3A,

   $T^*(\pi_d'(c_sc_{n+1}), \sigma_b) = T^*(\pi_d'(c_s), \sigma_b)$.

   From the induction hypothesis, this means

$T^*(c_s, \sigma_a) \sim^d T^*(\pi_d'(c_s c_{n+1}), \sigma_b)$.

As X locally respects r, $\sigma \sim^d T(c_{n+1}, \sigma)$ for any σ. Then $T(c_{n+1}, \sigma_a) \sim^d \sigma_a$, so

$T(c_{n+1}, T^*(c_s, \sigma_a)) \sim^d T^*(c_s, \sigma_a)$.

Substituting back, this becomes

$T(c_{n+1}, T^*(c_s, \sigma_a)) \sim^d T^*(\pi_d'(c_s c_{n+1}), \sigma_b)$

verifying the hypotheses.

In either case, then, the hypothesis holds, completing the induction step.

Having completed the induction, take $\sigma_a = \sigma_b = \sigma_0$. Then, by Lemma 8–1, if $\sim^d$ is output consistent, **X** is non-interference secure with respect to the policy **r**.

The significance of Theorem 8–1 is that it provides a basis for analyzing systems that purport to enforce a noninterference policy. Essentially, one establishes the conditions of the theorem for a particular set of commands and states with respect to some policy and a set of protection domains. Then noninterference security with respect to **r** follows.

8.2.2. Access Control Matrix Interpretation

no 8.2.2 - 8.2.4

Rushby presented a simple example of the use of the unwinding theorem [858]. The goal is to apply the theorem to a system with a static access control matrix. Our question is whether or not the given conditions are sufficient to provide noninterference security.

We begin with a model of the system. As usual, it is composed of subjects and objects. The objects are locations in memory containing some data. The access control matrix controls the reading and writing of the data. The system is in a particular state; the state encapsulates the values in the access control matrix.

Specifically, let $L = \{ l_1, ..., l_m \}$ be the set of objects (locations) in memory or on disk. Let $V = \{ v_1, ..., v_n \}$ be the set of values that the objects may assume. As usual, the set of states is $\Sigma = \{ \sigma_1, ..., \sigma_k \}$. The set $D = \{ d_1, ..., d_j \}$ is the set of protection domains. We also define three functions for convenience.

1. **value**: $L \times \Sigma \to V$ returns the value stored in the given object when the system is in the given state.
2. **read**: $D \to P(V)$ returns the set of objects that can be observed under the named domain (here, **POW**(**V**) denotes the power set of **V**).
3. **write**: $D \to P(V)$ returns the set of objects that can be written under the named domain.

Let **s** be a subject in the protection domain **d**, and let **o** be an object. The functions represent the access control matrix **A** because the entry corresponding to **A**[**s, o**] contains “read” if $o \in$ **read**(**d**) and contains “write” if $o \in$ **write**(**d**). This also leads to a natural interpretation of the equivalence relation in Definition 8–2A—namely,