8.1. The Problem

Return to the Bell-LaPadula Model for a moment. That model forbids reading of higher-level objects (the simple security condition) and writing to lower-level objects (the *-property). However, writing can take many forms.

EXAMPLE: Suppose two users are sharing a single system. The users are separated, each one having a virtual machine, and they cannot communicate directly with one another. However, the CPU is shared on the basis of load. If user Matt (cleared for SECRET) runs a CPU-intensive program, and user Holly (cleared for CONFIDENTIAL) does not, Matt's program will dominate the CPU. This provides a **covert channel** through which Matt and Holly can communicate. They agree on a time interval and a starting time (say, beginning at noon, with intervals of 1 minute). To transmit a 1-bit, Matt runs his program in the interval; to transmit a 0-bit, Matt does not. Every minute, Holly tries to execute a program, and if the program runs, then Matt's program does not have the CPU and the bit is 0; if the program does not run in that interval, Matt's program has the CPU and the transmitted bit is 1. Although not "writing" in the traditional sense, information is flowing from Matt to Holly in violation of the Bell-LaPadula Model's constraints.

This example demonstrates the difficulty of separating policy from mechanism. In the abstract, the CPU is transmitting information from one user to another. This violates the *-property, but it is not writing in any traditional sense of the word, because no operation that alters bits on the disk has occurred. So, either the model is insufficient for preventing Matt and Holly from communicating, or the system is improperly abstracted and a more comprehensive definition of "write" is needed. This is one problem, and in what follows, exploring it will lead to the notions of noninterference and nondeducibility.

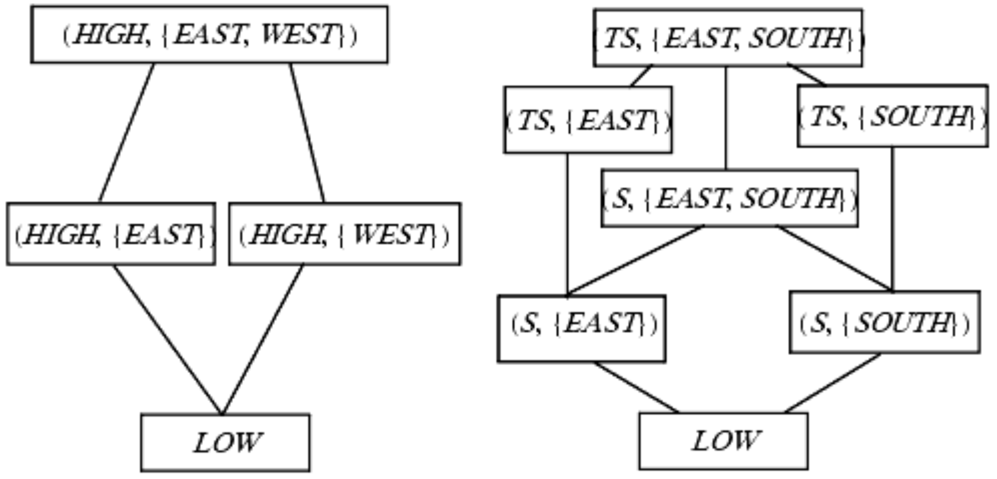
8.1.1. Composition of Bell-LaPadula Models

The techniques of modular decomposition and bottom-up programming are widely used throughout the disciplines of computer science, including computer security. Many standards, such as the Trusted Network Interpretation of the Trusted Computer Security Evaluation Criteria (see Section 21.2), require secure components to be connected to create a secure distributed or networked system. An obvious question is whether or not the composition of two secure systems is itself secure. For our purposes, we assume that the implementation of those systems is precise with respect to the security policy, and we confine ourselves to the issue of composition of security policies. If their composition is not secure, then the composed system is not secure.

Consider two systems with policies that match the Bell-LaPadula Model. These policies can be represented as lattices. The composed system is therefore the composition of the lattices. The relevant issue is the relationship among the labels (security levels and categories). If they are the same, the composition is simply the lattice itself. If they differ, the new lattice must reflect the relationship among the compartments.

EXAMPLE: Consider two systems with policies modeled by the Bell-LaPadula Model. One, system **allie**, has two security levels, LOW and HIGH, and two categories, EAST and WEST. The other, system **son**, has three security levels, LOW, S, and TS, and two categories, EAST and SOUTH. (The two EAST categories have the same meaning, as do the two LOW security levels.) Figure 8-1 shows the lattices of these two systems. The relevant issues are (1) how S and TS compare with HIGH and (2) how SOUTH compares with EAST and WEST.

Figure 8-1. The lattice on the left corresponds to the policy of system allie; the one on the right, to system son.



Assume that HIGH corresponds to a level between S and TS, and that SOUTH is a category disjoint from EAST and WEST. Then the composed lattice has four security levels (LOW, S, HIGH, and TS) and three categories (EAST, WEST, and SOUTH). Drawing the resulting lattice is left as an exercise for the reader.

The security policy of the composite system in the preceding example is a composition of the two security policies of the component systems. If we can change the policies that the components must meet, then composing multiple secure systems to produce a single secure system becomes trivial. However, if we must compose two components that meet a particular policy and show that the resulting composition also meets that same policy, the problem becomes quite difficult. We will explore this surprising fact at length throughout the rest of this chapter.

An interesting question is how to compose systems with different policies to produce a secure policy. Under these conditions, the notion of "security" is not clear: which policy dominates? Gong and Qian [408, 409] suggest the following guiding principles.

Axiom 8–1. Principle of Autonomy: Any access allowed by the security policy of a component must be allowed by the composition of the components.

Axiom 8–2. Principle of Security: Any access forbidden by the security policy of a component must be forbidden by the composition of the components.

The composite system therefore satisfies the security policies of its components because the policies of the components take precedence over the composite. Moreover, a "composition policy" handles the accesses not falling under either principle. If a new access is neither allowed nor forbidden by either principle, it should be disallowed unless it is explicitly permitted by the composition policy.[1]

[1] This differs from Gong and Qian's approach, in which they allow the access unless the composition policy explicitly forbids it, but follows the Principle of Fail-Safe Defaults (see Section 13.2.2).

Gong and Qian [408, 409] show that, given these principles, the problem of determining if an access is secure is of polynomial complexity. Their algorithm is to compute the transitive closure of the allowed accesses under each component's policy and under the composition policy. Then all forbidden accesses in this set are deleted. If the requested access is in the remaining set, it is allowed; otherwise, it is denied.

EXAMPLE: Let **X** be a system in which Bob is forbidden to access Alice's files. Let system **Y** be a system with users Eve and Lilith, each of whom can access each other's files. The composition policy says that Bob can access Eve's files, and Lilith can access Alice's files. The question is, can Bob access Lilith's files?

We write (**a, b**) to indicate that **a** can access **b**'s files and we write **AS**(**x**) to denote the access set of **x**:

$$\mathbf{AS}(\mathbf{X}) = \varnothing$$

$$\mathbf{AS}(\mathbf{Y}) = \{ (\text{Eve}, \text{Lilith}), (\text{Lilith}, \text{Eve}) \}$$

$$\mathbf{AS}(\mathbf{X} \cup \mathbf{Y}) = \{ (\text{Bob}, \text{Eve}), (\text{Lilith}, \text{Alice}), (\text{Eve}, \text{Lilith}), (\text{Lilith}, \text{Eve}) \}$$

The transitive closure of the last set is

$$\mathbf{AS}(\mathbf{X} \cup \mathbf{Y})^{+} = \{ (\text{Bob}, \text{Eve}), (\text{Bob}, \text{Lilith}), (\text{Bob}, \text{Alice}), (\text{Eve}, \text{Lilith}), (\text{Eve}, \text{Alice}), (\text{Lilith}, \text{Eve}) \}$$

Deleting accesses that conflict with the access policies of the components yields

$$\mathbf{AS}(\mathbf{X} \cup \mathbf{Y})^{-} = \{ (\text{Bob}, \text{Eve}), (\text{Bob}, \text{Lilith}), (\text{Eve}, \text{Lilith}), (\text{Eve}, \text{Alice}), (\text{Lilith}, \text{Eve}) \}$$

So Bob can access Lilith's files.

The dropping of the accesses that violate components' restrictions **after** the transitive closure has been computed eliminates accesses that are allowed by the composition policy but forbidden by the components. This is dictated by the principle of security. Without it, Bob could read Alice's files in the composition but not within the component.

Determining the minimum set of accesses that the composition policy must forbid in order to enforce both principles is generally in **NP**.