16.1. Basics and Background

Information flow policies define the way information moves throughout a system. Typically, these policies are designed to preserve confidentiality of data or integrity of data. In the former, the policy's goal is to prevent information from flowing to a user not authorized to receive it. In the latter, information may flow only to processes that are no more trustworthy than the data.

Any confidentiality and integrity policy embodies an information flow policy.

EXAMPLE: The Bell-LaPadula Model describes a lattice-based information flow policy. Given two compartments **A** and **B**, information can flow from an object in **A** to a subject in **B** if and only if **B** dominates **A**.

Let **x** be a variable in a program. The notation $\underline{\mathbf{x}}$ refers to the information flow class of **x**.

EXAMPLE: Consider a system that uses the Bell-LaPadula Model. The variable **x**, which holds data in the compartment (TS, { NUC, EUR }), is set to 3. Then **x** = 3 and $\underline{\mathbf{x}}$ = (TS, { NUC, EUR }).

16.1.1. Entropy-Based Analysis

We now define precisely the notion of information flow. Intuitively, information flows from an object **x** to an object **y** if the application of a sequence of commands **c** causes the information initially in **x** to affect the information in **y**. We use the notion of entropy, or uncertainty (see Chapter 32, "Entropy and Uncertainty"), to formalize this concept.

Let **c** be a sequence of commands taking a system from state **s** to another state **t**. Let **x** and **y** be objects in the system. We assume that **x** exists when the system is in state **s** and has the value $\mathbf{x_S}$. We require that **y** exist in state **t** and have the value $\mathbf{y_t}$. If **y** exists in state **s**, it has value $\mathbf{y_S}$.

Definition 16–1. The command sequence **c** causes a **flow of information from x to y** if $\mathbf{H}(\mathbf{x_S} \mid \mathbf{y_t}) < \mathbf{H}(\mathbf{x_S} \mid \mathbf{y_S})$. If **y** does not exist in **s**, then $\mathbf{H}(\mathbf{x_S} \mid \mathbf{y_S}) = \mathbf{H}(\mathbf{x_S})$.

This definition states that information flows from the variable **x** to the variable **y** if the value of **y** after the commands allows one to deduce information about the value of **x** before the commands were run.

This definition views information flow in terms of the information that the value of **y** allows one to deduce about the value in **x**. For example, the statement

```
y := x;
```

reveals the value of **x** in the initial state, so $\mathbf{H}(x_S \mid y_t) = 0$ (because given the value y_t, there is no uncertainty in the value of x_S). The statement

```
y := x / z;
```

reveals some information about **x**, but not as much as the first statement.

The final result of the sequence **c** must reveal information about the initial value of **x** for information to flow. The sequence

```
tmp := x;
y := tmp;
```

has information flowing from **x** to **y** because the (unknown) value of **x** at the beginning of the sequence is revealed when the value of **y** is determined at the end of the sequence. However, no information flow occurs from **tmp** to **x**, because the initial value of **tmp** cannot be determined at the end of the sequence.

EXAMPLE: Consider the statement

```
x := y + z;
```

Let **y** take any of the integer values from 0 to 7, inclusive, with equal probability, and let **z** take the value 1 with probability 0.5 and the values 2 and 3 with probability 0.25 each. Let **s** be the state before this operation is executed, and let **t** be the state immediately after it is executed. Then $\mathbf{H}(y_S) = \mathbf{H}(y_t) = 3$ and $\mathbf{H}(z_S) = H(z_t) = 1.5$. Once the value of x_t is known, y_S can assume at most three values, so $\mathbf{H}(y_S \mid x_t) < \lg 3$ Ý 1.58. Thus, information flows from **y** to **x**. Similar results hold for $\mathbf{H}(z_S \mid x_t)$; see Exercise 1.

EXAMPLE: Consider a program in which **x** and **y** are integers that may be either 0 or 1. The statement

```
if x = 1 then y := 0;
else y := 1;
```

does not explicitly assign the value of **x** to **y**.

Assume that **x** is equally likely to be 0 or 1. Then $H(\mathbf{x_S}) = 1$. But $\mathbf{H(x_S \mid y_t)} = 0$, because if **y** is 0, **x** is 1, and vice versa. Hence, $\mathbf{H(x_S \mid y_t)} = 0 < \mathbf{H(x_S \mid y_S)} = \mathbf{H(x_S)} = 1$. Thus, information flows from **x** to **y**.

Definition 16–2. An **implicit flow of information** occurs when information flows from **x** to **y** without an explicit assignment of the form **y** := **f**(**x**), where **f**(**x**) is an arithmetic expression with the variable **x**.

The flow of information occurs, not because of an assignment of the value of **x**, but because of a flow of control based on the value of **x**. This demonstrates that analyzing programs for assignments to detect information flows is not enough. To detect all flows of information, implicit flows must be examined.

16.1.2. Information Flow Models and Mechanisms

An information flow policy is a security policy that describes the authorized paths along which that information can flow. Part 3, "Policy," discussed several models of information flow, including the Bell-LaPadula Model, nonlattice and nontransitive models of information flow, and nondeducibility and noninterference. Each model associates a label, representing a security class, with information and with entities containing that information. Each model has rules about the conditions under which information can move throughout the system.

In this chapter, we use the notation $\mathbf{\underline{x} \leq \underline{y}}$ to mean that information can flow from an element of class **x** to an element of class **y**. Equivalently, this says that information with a label placing it in class $\mathbf{\underline{x}}$ can flow into class $\mathbf{\underline{y}}$.

Earlier chapters usually assumed that the models of information flow policies were lattices. We first consider nonlattice information flow policies and how their structures affect the analysis of information flow. We then turn to compiler-based information flow mechanisms and runtime mechanisms. We conclude with a look at flow controls in practice.