Chapter 3
Foundational Results

MARIA: Ay, but you must confine yourself
within the modest limits of order.
— *Twelfth Night*, I, iii, 8–9.

In 1976, Harrison, Ruzzo, and Ullman [874] proved that the security of computer systems was undecidable in the general case and explored some of the limits of this result. In that same year, Jones, Lipton, and Snyder [972] presented a specific system in which security was not only decidable, but decidable in time linear with the size of the system. Minsky [1354] suggested a third model to examine what made the general, abstract case undecidable but at least one specific case decidable. Sandhu [1657] devised a related model extending the decidability results to a large class of systems.

These models explore the most basic question of the art and science of computer security: under what conditions can a generic algorithm determine whether a system is secure? Understanding models and the results derived from them lays the foundations for coping with limits in policy and policy composition as well as applying the theoretical work.

3.1 The General Question

Given a computer system, how can we determine if it is secure? More simply, is there a generic algorithm that allows us to determine whether a computer system is secure? If so, we could simply apply that algorithm to any system; although the algorithm might not tell us where the security problems were, it would tell us whether any existed.

The first question is the definition of "secure." What policy shall define "secure"? For a general result, the definition should be as broad as possible. We use the access control matrix to express our policy. However, we do not provide any special rights such as *copy* or *own*, nor do we apply the principle of attenuation of privilege.

Let R be the set of generic (primitive) rights of the system.

Definition 3–1. When a generic right r is added to an element of the access control matrix that did not contain r initially, that right is said to be *leaked*.[1]

Under this definition, if a system begins with a right r in $A[s, o]$, deletes it, and then adds it back, r has not leaked.

Our policy defines the authorized set of states A to be the set of states in which no command $c(x_1, \ldots, x_n)$ can leak r. This means that no generic rights can be added to the matrix.

We do not distinguish between a *leaking* of rights and an *authorized transfer* of rights. In our model, there is no authorized transfer of rights. (If we wish to allow such a transfer, we designate the subjects involved as "trusted." We then eliminate all trusted subjects from the matrix, because the security mechanisms no longer apply to them.)

Let a computer system begin in protection state s_0.

Definition 3–2. If a system can never leak the right r, the system (including the initial state s_0) is called *safe* with respect to the right r. If the system can leak the right r (enter an unauthorized state), it is called *unsafe with respect to the right r*.

We use these terms rather than *secure* and *nonsecure* because safety refers to the abstract model and security refers to the actual implementation. Thus, a secure system corresponds to a model safe with respect to all rights, but a model safe with respect to all rights does not ensure a secure system.

EXAMPLE: A computer system allows the network administrator to read all network traffic. It disallows all other users from reading this traffic. The system is designed in such a way that the network administrator cannot communicate with other users. Thus, there is no way for the right r of the network administrator over the network device to leak. This system is safe.

Unfortunately, the operating system has a flaw. If a user specifies a certain file name in a file deletion system call, that user can obtain access to any file on the system (bypassing all file system access controls). This is an implementation flaw, not a theoretical one. It also allows the user to read data from the network. So this system is not secure.

The *protection state* of a system consists of the parts of the system state relevant to protection. The *safety question* is: Does there exist an algorithm to

[1] Tripunitara and Li [1890] point out that this differs from the definition in Harrison, Ruzzo, and Ullman [874], which defines "leaked" to mean that r is added to any access control matrix entry in which r was not present *in the immediately previous state*. But the proofs in that paper use Definition 3–1. See Exercise 2 for an exploration of the differences when the stated definition in Harrison, Ruzzo, and Ullman is used.

determine whether a given protection system with initial state s_0 is safe with respect to a generic right r?

3.2 Basic Results

The simplest case is a system in which the commands are mono-operational (each consisting of a single primitive operation). In such a system, the following theorem holds.

> **Theorem 3.1.** [874] There exists an algorithm that will determine whether a given mono-operational protection system with initial state s_0 is safe with respect to a generic right r.
>
> **Proof** Because all commands are mono-operational, we can identify each command by the type of primitive operation it invokes. Consider the minimal length sequence of commands $c_1, \ldots, c_k$ needed to leak the right r from the system with initial state s_0.
>
> Because no commands can test for the absence of rights in an access control matrix entry, we can omit the **delete** and **destroy** commands from the analysis. They do not affect the ability of a right to leak.
>
> Now suppose that multiple **create** commands occurred during the sequence of commands, causing a leak. Subsequent commands check only for the presence of rights in an access control matrix element. They distinguish between different elements only by the presence (or lack of presence) of a particular right. Suppose that two subjects s_1 and s_2 are created and the rights in $A[s_1, o_1]$ and $A[s_2, o_2]$ are tested. The same test for $A[s_1, o_1]$ and $A[s_1, o_2] = A[s_1, o_2] \cup A[s_2, o_2]$ will produce the same result. Hence, all **create** commands are unnecessary except possibly the first (if there are no subjects initially), and any commands entering rights into the new subjects are rewritten to enter the new right into the lone created subject. Similarly, any tests for the presence of rights in the new subjects are rewritten to test for the presence of that right in an existing subject (or, if none initially, the first subject created).
>
> Let $|S_0|$ be the number of subjects and $|O_0|$ the number of objects in the initial state. Let n be the number of generic rights. Then, in the worst case, one new subject must be created (one command), and the sequence of commands will enter every right into every element of the access control matrix. After the creation, there are $|S_0| + 1$ subjects and $|O_0| + 1$ objects, and $(|S_0|+1)(|O_0|+1)$ elements. Because there are n generic rights, this leads to $n(|S_0| + 1)(|O_0| + 1)$ commands. Hence, $k \leq n(|S_0| + 1)(|O_0| + 1) + 1$.

By enumerating all possible states we can determine whether the system is safe. Clearly, this may be computationally infeasible, especially if many subjects,