

Protection



In Chapter 16, we addressed security, which involves guarding computer resources against unauthorized access, malicious destruction or alteration, and accidental introduction of inconsistency. In this chapter, we turn to protection, which involves controlling the access of processes and users to the resources defined by a computer system.

The processes in an operating system must be protected from one another's activities. To provide this protection, we can use various mechanisms to ensure that only processes that have gained proper authorization from the operating system can operate on the files, memory segments, CPU, networking, and other resources of a system. These mechanisms must provide a means for specifying the controls to be imposed, together with a means of enforcement.

CHAPTER OBJECTIVES

- Discuss the goals and principles of protection in a modern computer system.
- Explain how protection domains, combined with an access matrix, are used to specify the resources a process may access.
- Examine capability- and language-based protection systems.
- Describe how protection mechanisms can mitigate system attacks.

17.1 Goals of Protection

As computer systems have become more sophisticated and pervasive in their applications, the need to protect their integrity has also grown. Protection was originally conceived as an adjunct to multiprogramming operating systems, so that untrustworthy users might safely share a common logical name space, such as a directory of files, or a common physical name space, such as memory. Modern protection concepts have evolved to increase the reliability of any complex system that makes use of shared resources and is connected to insecure communications platforms such as the Internet.

We need to provide protection for several reasons. The most obvious is the need to prevent the mischievous, intentional violation of an access restriction by a user. Of more general importance, however, is the need to ensure that each process in a system uses system resources only in ways consistent with stated policies. This requirement is an absolute one for a reliable system.

Protection can improve reliability by detecting latent errors at the interfaces between component subsystems. Early detection of interface errors can often prevent contamination of a healthy subsystem by a malfunctioning subsystem. Also, an unprotected resource cannot defend against use (or misuse) by an unauthorized or incompetent user. A protection-oriented system provides means to distinguish between authorized and unauthorized usage.

The role of protection in a computer system is to provide a mechanism for the enforcement of the policies governing resource use. These policies can be established in a variety of ways. Some are fixed in the design of the system, while others are formulated by the management of a system. Still others are defined by individual users to protect resources they “own.” A protection system, then, must have the flexibility to enforce a variety of policies.

Policies for resource use may vary by application, and they may change over time. For these reasons, protection is no longer the concern solely of the designer of an operating system. The application programmer needs to use protection mechanisms as well, to guard resources created and supported by an application subsystem against misuse. In this chapter, we describe the protection mechanisms the operating system should provide, but application designers can use them as well in designing their own protection software.

Note that *mechanisms* are distinct from *policies*. Mechanisms determine *how* something will be done; policies decide *what* will be done. The separation of policy and mechanism is important for flexibility. Policies are likely to change from place to place or time to time. In the worst case, every change in policy would require a change in the underlying mechanism. Using general mechanisms enables us to avoid such a situation.

17.2 Principles of Protection

Frequently, a guiding principle can be used throughout a project, such as the design of an operating system. Following this principle simplifies design decisions and keeps the system consistent and easy to understand. A key, time-tested guiding principle for protection is the **principle of least privilege**. As discussed in Chapter 16, this principle dictates that programs, users, and even systems be given just enough privileges to perform their tasks.

Consider one of the tenets of UNIX—that a user should not run as root. (In UNIX, only the root user can execute privileged commands.) Most users innately respect that, fearing an accidental delete operation for which there is no corresponding undelete. Because root is virtually omnipotent, the potential for human error when a user acts as root is grave, and its consequences far reaching.

Now consider that rather than human error, damage may result from malicious attack. A virus launched by an accidental click on an attachment is one example. Another is a buffer overflow or other code-injection attack that is successfully carried out against a root-privileged process (or, in Windows,