Lecture 26: April 29

Lecturer: Prashant Shenoy *TA: Vimal Mathew & Tim Wood*

26.1 Protection

An operating system needs a way to provide protection for the many different objects it manages. The operating system must protect both hardware and objects such as memory pages, files, and semaphores. Protecting these resources ensures that malicious users or processes can not hurt the system, as well as ensure that resources can only be used subject to certain system policies. This allows the OS to provide a set of policies which dictate what type of access users have to each resource in the system.

An access right, <object-name, right-set>, defines an object and a list of allowable operations the process can perform on that object. A **domain** defines a list of access rights for a process in the OS. For example, a process may have an access right specifying that it can read and write certain files, while others it can only read.

The UNIX operating system associates a protection domain with each user in the system. This means that the set of objects that can be accessed depends on the identity of the current user. Typically, when a user executes processes they are restricted to the permissions defined by the user's own domain. However, in some cases it is desirable for the domain to switch when a process is run, for example to allow the user to run a privileged process that has additional powers. This is supported in Unix by including two extra pieces of information in the meta data associated with every executable file in the system: the user id that owns the file, and a *setuid* bit which is used as a flag. When the setuid bit of a particular executable file is off, then running the executable makes no change to the protection domain. However, if the setuid bit is on, then running the executable will temporarily change the active user-id (and thus the active protection domain) to be the owner of the file. Once the file finishes being executed, the user-id is reset to the original user.

26.1.1 Access Matrix

The mapping of domains, objects, and valid actions can be viewed as an **access matrix** within the operating system. In this matrix, the rows represent the different domains and the columns refer to the objects in the system. The entry at row_i, $column_j$ defines what actions users in $Domain_i$ can perform on $Object_j$. Different types of objects may support different types of actions: a file may have read/write/execute privileges, while a device such as a printer may indicate whether users have the ability to connect to it and print.

Allowing the rights listed in the access matrix to change over time gives support for **dynamic protection**. The system must support functionality to add, delete, or modify access rights for a given domain and object; naturally, such actions must also be restricted to users within a particular protection domain.

The use of an access matrix separates the protection mechanism from policy. The **mechanism** is the code within the OS that enforces the rules contained within the access matrix, while the **policy** is the list of rules which the system administrators can define.

26.1.2 Access Lists versus Capabilities

The Access Matrix described in the previous section is a form of **access list**. An access list specifies precisely which users have which rights for a given object. In contrast, a **capability** based system is one where each object only specifies a "capability" (think of this as a secret key) that is required to perform a given action. If a process attempts to access an object, the capability system checks whether the process holds the correct capability. This provides a layer of abstraction—the protection rules are no longer based on precisely listing who has access to what, but rather specify what is required in order to gain access to a given object.

An advantage of access list based systems is that it is very easy to adjust which users have permissions for a given object, and such changes can be applied immediately. In a capability based system, verifying that a user has the right permissions can be very fast—it only needs to check that its capability is valid, rather than scan through a long list of all users. However, revoking a capability from a specific user can be more complicated.

26.1.3 Language Based Protection

In most cases, protection is provided by the operating system itself, as it is a trusted entity that has complete access to the system and hardware. However, protection can also be provided at a higher layer, such as within programming languages. Providing an API for managing protection domains within a programming language makes it easier to meet the high level security concerns of programmers. In addition, providing protection within the programming language eliminates any reliance on hardware support, allowing the protection system to be flexibly deployed across a wider range of systems.

In the Java programming language, protection is provided by the Java Virtual Machine (JVM). In Java, each class is assigned a protection domain that defines its permissions. As a result, a Java class loaded from a local binary may be given greater permissions than a class loaded as part of a Java applet from a potentially untrusted website. In order to enforce permissions, the JVM monitors which classes are making access to protected resources. When an action on a protected resource is attempted, the JVM uses **stack inspection** to examine the set of function calls which led to the questionable access. Only if the classes on the call stack actually have the proper permissions will the call be allowed.