CS 537 Notes, Section #29: Protection and Security

The purpose of a protection system is to prevent accidental or intentional misuse of a system.

- Accidents: Problems of this kind are easy to solve (can do things to make the likelihood small).
- Malicious abuse: Problems of this kind are very hard to completely eliminate (cannot leave any loopholes, cannot play on probabilities).

There are three aspects to a protection mechanism:

- **User identification (authentication)**: make sure we know who is doing what.
- **Authorization determination**: must figure out what the user is and is not allowed to do. Need a simple database for this.
- **Access enforcement**: must make sure there are no loopholes in the system.

Even the slightest flaw in any of these areas may ruin the whole protection mechanism.

Authentication

User identification is most often done with *passwords*. This is a relatively weak form of protection.

- A password is a secret piece of information used to establish the identity of a user.
- Passwords should not be stored in a readable form. One-way transformations should be used.
- Passwords should be relatively long and obscure.

Another form of identification: badge or key.

- Does not have to be kept secret.
- Should not be able to be forged or copied.
- Can be stolen, but the owner should know if it is.

Key paradox: key must be cheap to make, hard to duplicate. This means there must be some trick (i.e. secret) that has to be protected.

Once identification is complete, the system must be sure to protect the identity since other parts of the system will rely on it.

Authorization Determination

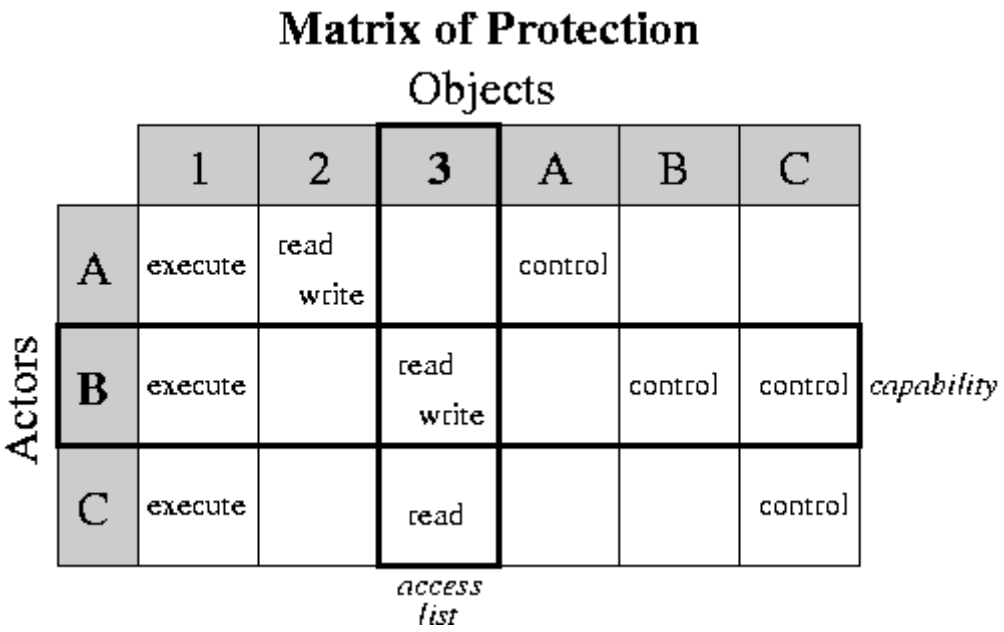
Must indicate who is allowed to do what with what. Draw the general form as an access matrix with one row per user, one column per file. Each entry indicates the privileges of that user on that object. There are two general ways of storing this information: *access lists* and *capabilities*.

Access Lists: with each file, indicate which users are allowed to perform which operations.

- In the most general form, each file has a list of pairs.
- It would be tedious to have a separate listing for every user, so they are usually grouped into classes. For example, in Unix there are three classes: self, group, anybody else (nine bits per file).
- Access lists are simple, and are used in almost all file systems.

Capabilities: with each user, indicate which files may be accessed, and in what ways.

- Store a list of pairs with each user. This is called a *capability list*.
- Typically, capability systems use a different naming arrangement, where the capabilities are the only names of objects. You cannot even name objects not referred to in your capability list.
- In access-list systems, the default is usually for everyone to be able to access a file. In capability-based systems, the default is for no-one to be able to access a file unless they have been given a capability. There is no way of even naming an object without a capability.
- Capabilities are usually used in systems that need to be very secure. However, capabilities can make it difficult to share information: nobody can get access to your stuff unless you explicitly give it to them.



Are the following things access-based or capability-based protection schemes?

- Protection Keys
- Page tables

Access Enforcement

Some part of the system must be responsible for enforcing access controls and protecting the authorization and identification information.

- Obviously, this portion of the system must run unprotected. Thus it should be as small and simple as possible. Example: the portion of the system that sets up memory mapping tables.

- The portion of the system that provides and enforces protection is called the *security kernel*. Most systems, like Unix, do not have a security kernel. As a consequence, the systems are not very secure.

- What is needed is a hierarchy of levels of protection, with each level getting the minimum privilege necessary to do its job. However, this is likely to be slow (crossing levels takes time).
