

Real and Effective IDs

First, in order to really understand how this works, we need to step back for a moment and talk about the Unix user ID concept.

At the lowest level of the operating system, the *kernel*, users and groups aren't identified by names, but numbers. The kernel needs to be fast and robust, and data structures better be small, and moving around strings is anything but efficient. So, each user name and group name is mapped to a unique unsigned number, called user and group ID for short, or uid and gid. This mapping is done via the `/etc/passwd` and `/etc/group` files, respectively. The user and group ID 0 are commonly called root, but that's really just a convention.

Each Unix process has a user ID and a group ID associated with it, and when trying to open a file for writing, for instance, these IDs are used to determine whether the process should be granted access or not. These IDs constitute the *effective* privilege of the process, because they determine what a process can do and what it cannot. Most of the time, these IDs will be referred to as the effective uid and gid.

What happens when you invoke the `passwd` utility is that the effective uid of the process is set to 0, i.e. the uid of the root user. As a result, the program is permitted to modify the `/etc/passwd` file, and can thus replace the encrypted password in your account entry with the new one you just provided.

If you're familiar with the `passwd` utility, you'll know that as a normal user, you're only allowed to modify the password of your own account; it will not let you modify the password of any other account. So this begs the question, how does it know who invoked it?

That's where another pair of user and group ID comes in, called the *real* uid and gid, respectively. These IDs are used to track who a user *really* is, i.e. on what account he or she is logged in. This uid value is not changed when you invoke programs such as `passwd`. So the program simply needs to find out what user name corresponds to its real uid, and refuse to change any other account.

Most of the time, the effective user ID of a process is just the same as the real ones, and there's no point in making a fuss of this minor distinction.

Things start to get interesting when you invoke a `setuid` application, however. Assume you're logging into your normal user account, which has a user ID of 500. Now you invoke a `setuid` root application. Because it's `setuid` root, the operating system will set the effective user ID of the process to that of the root user (0). The real user ID, however, remains unchanged. This allows the application to learn the identity of the user who invoked it, and to continue to access files etc with the privilege of the invoking user.

If you think about a large company where employees have different levels of access to different locations, you could compare the real user ID to the name badges people wear, and the effective user ID to the set of keys they've been given.

I have to confess that for the sake of simplicity, I have so far avoided to talk about an additional bunch of group IDs a process usually lugs around; these are called the supplementary gids. A user can be a member of several groups at the same time, for instance a software engineer working on project Fortytwo may be a member of both the `eng` and `fortytwo` groups at the same time. This set of all groups is transported in an additional group vector associated with each process, called the supplementary group ID; what we've called "the" gid all the time is really the *primary* gid in this case.