Got Root? All About Users and Permissions

Robert Heller
Deepwoods Software
Wendell, MA, USA *

April 17, 2006

Abstract

This article discusses permissions and users, including special users like the root user. Also covered are tools to securely delegate administrative tasks without creating security holes.

Contents

1 Users and Groups: Names and Ids

Under Linux and UNIX, all "users"[1] have a unique id (UID) number and are members of one or more "groups", which also have a a unique id (GID). These numbers are used to decide what access a given user has to the various resources available on the system, including files and directories, input-output devices, memory, and processes. The password file (/etc/passwd) and group (/etc/group) file contains the mapping between user ids and user name and group ids and group names and group membership.

*Copyright (C) 2006 Robert heller.

[1] Both actual human users and non-human users.

There should be a unique user name and user id for each user on a Linux/UNIX system. Users can be collected into groups, particularly if a group of users will be sharing resources, typically files and directories. Since users can be members of more than one group, it is possible to create several overlapping groups for different purposes, such as programming projects or other collaborative activities.

2 Permissions

POSIX defines 12 permission or mode bits, which are arranged a 4 groups of 3. The first 9 define read, write, and execute/search, by the owner of the resource, the group that owns the resource, and everyone else. The last three bits control special permissions and include set user id, set group id, and the sticky bit. All of the permissions are shown below.

Value in Mode	Corresponding Permission
	Other users not in the file's group:
1	Execute
2	Write
4	Read
	Other users in the file's group:
10	Execute
20	Write
40	Read
	The file's owner:
100	Execute
200	Write
400	Read
	Special permissions:
1000	Sticky bit
2000	Set group ID on execution
4000	Set user ID on execution

The first three batches of three are pretty straight forward. For directories, the Execute controls "search" or the permission to get a listing of the directory. The special permissions relate to both directories and executable files. The sticky bit on a directory restricts modifying a file to the file's owner. The set group ID bit on a directory indicates that BSD semantics are to be used for that directory: files created there inherit their group ID from the directory, not from the effective GID of the creating process, and directories created there will also get the set group ID bit set. For a file that does not have the group execution bit set, it indicates mandatory file/record locking.

3 The Root User (UID 0)

On every Linux or UNIX system there is a special user and group id: 0. A process (or user) with a user id (and generally a groups id as well) of 0 has virtually unlimited access to files, devices, and other resources, as well as access to restricted system services. Generally, this user is given the user name of "root". It is *very strongly* recommended that this user name not be used for common everyday activities and should not be used as a regular login, even by the owner or administrator of the computer system. Instead, a normal (non privileged) user should be created for use as a normal login. There are special commands (described in the next section) that are intended for the purposes of raising a user's privilege state as needed to perform system management tasks.

There are also other special user ids and group ids that are used by system background processes (known as daemons). Often these special user and group ids provide these processes with the special permissions they need, usually access to special files and devices that are also owned by these special user and group ids. User ids less than 100 are generally reserved for system user ids.

4 The Su and Sudo Commands

So, if logging into the root account is not recommended, how does one perform system management tasks? There are two common commands that are used to change your user id on a temporary basis and thus allow you to perform system management tasks. The older command is su (set user), which actually allows you create a new process with a different UID, usually UID 0, the root user. This command can either be used to start a fresh shell process or to execute a single shell command as a different user id. The downside to this command is that it requires the password of the user id it will be changing to (the root password if the root user is the target user id) and provides no "graininess" to the access it provides. A newer command, sudo, is much more powerful and provides many advantages over su[2].

The sudo command can be set up so that the primary system administrator can delegate administrative tasks to others in a way that maintains overall system security. Generally, sudo is used to run single commands (with a user id of 0) directly from a normal user's command shell. A user can be authorized to run *any* command or only selected commands. Sudo maintains security by asking for the user's (not the root!) password, and conveniently uses a timeout between times it asks for a password. Sudo also logs its usage for later review.

[2] See the HISTORY file in the sudo distribution or visit http://www.sudo.ws/sudo/history.html for a short history of sudo.

5 Who is This Nobody Person Anyway?

There is another special user id on a typical Linux system, the nobody user. This user is used as the effective user id of daemon processes, particularly of daemons that provide network services, particularly those that might write files. By running as nobody, these daemons will be prevented from writing files where they should not be writing files. This helps protect from network-based attacks. Since the nobody user id, in general, owns no files or directories, it cannot write to anywhere on the file system. It can also only read files that are specifically granted “other” access. In general, you should *never* grant write access to “others”. If it is needful to share write access between different users, you should grant “group” write access and put these users in a common group and change the group ownership of the files in question to this group. The nobody user should never be part of any group but its own group (and nobody else should be part of that group).

6 Conclusions

This article should have provided an overview of users, groups, and permissions and also talked about how to elevate one’s user access as needed to perform system administration tasks. You should be sure to read the Linux documentation (including the *man pages* and *info pages*) for the various commands and the corresponding configuration and data files they use. This includes: the passwd, group, and sudoers data and configuration files; the passwd, useradd, sudo, su, chmod, chown, and chgrp commands.