Users Access Control

The “ownership” concept

- Linux regards files and processes as objects
- Objects have owners. Owners have *almost* unrestricted control over their own objects
- The root account is the system administrator’s. It can take ownership of any object
- Processes control the access to themselves internally. For example, the `passwd` command will a non-root user change his own account. But it won’t let him reset another user’s password
- Filesystems control access to files and directories. Filesystems are the only object type that implements the concept of “groups”.
- Groups can contain a number of users. If the group has some permission on a file or directory, all group members have this same privilege.

Filesystem security model overview

- Users and groups own files and directories
- Only the owner user can change the permissions of his own file or directory
- The filesystem views users and groups as numbers rather than textual names. For example, root is identified by UID of 0, while wheel group is identified by GID 10.
- When an access attempt occurs from a user or a group, the filesystem will use the UID and/or the GID to grant or deny access.
- UID's and GID's are stored in `/etc/passwd` and `/etc/group` respectively. When a command like `ls -l` is used to display ownership information, those files (or the appropriate user database) will be consulted to display the human-friendly name rather than the id.
- You can use the `id` command to display your own UID, and `id -g` to display the group

Process security model overview

- Any process owner can send signals to it (like kill -9 for example). He can also increase it's *nice* value. That is, decrease it's priority on CPU using the `renice` command
- When a process or a command runs in Linux, it has the following user and group id's:
  - **Real:** the account of the owner of this process. It is what defines which files and directories that this process can have access to, and the type of this access.
  - **Effective:** normally, this is the same as the real id's. But sometimes it is changed to enable a non-privileged user to access files that can be only accessed by root. For example, the `passwd` command when run by a normal user to change his own password, the effective id becomes 0 to enable the user to make changes to `/etc/shadow` file. The process will check the real id of the user to grant or deny access accordingly
  - **Saved:** it is used when a process is running with a elevated privileges needs to do some work – temporarily – as a non-privileged account. In this case, it *saves* the privileged id to the suid so that it can use it back as its effect id.

The power of `root`

- The root account has the id of 0.
- A process with the id of 0 can do any operations on any other file or process
- In addition, some tasks can only be performed by the root account like:
  - Setting the machine's hostname or IP address
  - Changing the system's date and time
  - Open network sockets on privileged ports (below port 1024)
- The UID 0 process can even change it's own UID and GID. This happens when a normal user logs into the system. The login process changes it's UID and GID to those of the user. This change cannot be rolled back.

What makes a strong root password?

- Should be at least 8 characters long
- Using a meaningless mix of numbers, special characters, small and capital letters is a good approach but it is hard to remember. As a result, an admin may write it down to paper, he also may enter it slowly, both of which poses a security risk of revealing the password.
- A more modern approach nowadays is to use *passphrases*. That is, a long phrase that means something to you, and that is unpredictable. For example: *I hate SUSE Linux 10* may be used like this `IhateSUSELinux10!` it's long, has special characters, contains numbers as well as upper and lower case letters. Yet, it is easy for you and only you to remember. Who might think that you hate SUSE Linux 10? You get the idea
- The root password should be changed every 3 months at most.
- It's advisable to store root passwords. However, maximum security measures should be applied to the place/software they're stored in

Should I login as root?

- The short answer is no, it's not a good idea for the following reasons:
  - You lose the user accountability: who did what, and when
  - You give a hacker one step forward: instead of having to break a normal user's password first, then the root password, you leave him with only one password to crack: the root one
  - Most administrators do not apply password locking on root passwords, because this may completely lock the system, with no direct solution to unlock the root account. This gives the intruder all the time needed to try to guess the password through trial and error
- The recommended approach is to give administrators normal, unprivileged accounts. When the root power is needed, they either su to root, or use the sudo command

Using the `su` command

- The su command is short for substitute user
- You can use it to change your current login session to a different user session or to root. Provided that you have the appropriate password.
- Root can su to any user without specifying the password
- It has two forms:
  - su *user*: does not load the user's environment
  - su – *user*: loads the user's environment.
- You will remain in the alternate user session until you type `exit` or press `CTRL-D`.

Delegating root powers with `sudo`

- The sudo command is used to specify specific commands to be run as another user or, typically, as root
- The usage syntax is as follows: `sudo` *`command`* for root and `sudo -u` and `sudo -g` for running the command as the user or group respectively
- In order for it to work, the user has to be in the `/etc/sudoers`
- The `/etc/sudoers` is edited using the command `visudo`. It ensures that the file contains no errors, and that no one else is editing the file at the same time.
- The user is prompted for his own password before allowing the command to run.
- A use that has just used sudo can continue running commands without being prompted for password for 5 minutes
- All sudo commands are logged into a log file. You can use syslog to forward the logs to a central log host

Working with `/etc/sudoers`

- The line that adds privileges to a user has the following format: `user HOSTS=(USERS:GROUPS) COMMAND1,COMMAND2…`
- You can also use command aliases. Many of them are already defined in the `/etc/sudoers` file
- The HOSTS part is used so that the `sudouers` file can be shared across multiple machines
- A users' group can be denoted by %. For example: `%wheel` is the wheel group
- Optionally, you can add `NOPASSWD : ALL || commands` to instruct sudo not to ask for passwords for specific commands, or for all commands.
- If the commands you are delegating for users include those which can spawn other processes (like vi, vim, and less), you should add `NOEXEC:command` to avoid letting the use open a complete shell with the sudoed account

System accounts

- They are user accounts used to execute system services
- The idea is to use them instead of using the root account for such tasks
- They are protected from being used for login, by placing an asterisk instead of the password hash in the shadow file. Also by setting the default shell to `/bin/false` or `/bin/nologin`
- Their UID's are generally under 100.
- You should use these guidelines when creating a custom user account for a service or an application