Search

Using sudo to delegate permissions in Linux

This article explores some legitimate use cases for the sudo command.

Posted: April 20, 2020 | David Both (Sudoer)

*Photo by **Gratisography** from **Pexels***

In my previous article, "Real sysadmins don't sudo," I discussed the really horrible misuse of `sudo` by some distributions. In this article, which is partially excerpted from Chapter 11 of my book, "Using and Administering Linux, Zero to SysAdmin, Volume 1: Getting started," I explore a couple of valid use cases for `sudo`.

Use case 1: Remote file copy

I recently wrote a short Bash program to copy some MP3 files from a USB thumb drive on one network host to another network host. The files are copied to a specific directory on the server that I run for an organization to which I belong, from where they can be downloaded and played.

More Linux resources

- Advanced Linux Commands Cheat Sheet for Developers
- Get Started with Red Hat Insights
- Download Now: Basic Linux

Skills Assessment

My program does a few other things, like changing the name of the files before they are copied so that they are automatically sorted by date on the web page. It also deletes all of the files on the USB drive after verifying that the transfer has taken place correctly. This nice little program has a few options such as `-h` to display help, `-t` for test mode and a couple of others.

My program, wonderful as it is, needs to run as root in order to perform its primary functions. Unfortunately, this organization has only a couple of people besides myself who have any interest in administering our audio and computer systems, which puts me in the position of finding semi-technical people to train to login to the computer we use to perform the transfer and run this little program.

It is not that I cannot run the program myself, but I am not always there for various reasons such as travel and illness. Even when I am present, as the "Lazy SysAdmin," I like to have others do my work for me. So I write scripts to automate those tasks and use `sudo` to anoint a couple of users to run the scripts. Many Linux commands require the user to be root in order to run. This protects the system against accidental damage such as that caused by my own stupidity and intentional damage by a user with malicious intent.

Do do that sudo that you do so well

The `sudo` program is a handy tool that allows me as a sysadmin with root access to delegate responsibility for all or a few administrative tasks to other users of the computer as I see fit. It allows me to perform that delegation without compromising the root password and thus maintain a high level of security on the host.

Let's assume, for example, that I have given regular user, "ruser," access to my Bash program, `myprog`, which must be run as root in order to perform part of its functions. First, the user logs in as ruser with their own password. The user then uses the following command to run `myprog`.

```
sudo myprog
```

The `sudo` program checks the `/etc/sudoers` file and verifies that ruser is permitted to run `myprog`. If so, `sudo` requests that the user enter their own password – not the root password. After ruser enters their own password, the program is run. `sudo` also logs the facts of the access to `myprog` with the date and time the program was run, the complete command, and the user who ran it. This data is logged in `/var/log/security`.

I find that having the log of each command run by `sudo` to be helpful in training. I can see who did what and whether they actually entered the command correctly.

I have done this to delegate authority to run a single program to myself and one other user. However, `sudo` can be used to do so much more. It can allow the sysadmin to delegate authority for managing network functions or specific services to a single person or to a group of trusted users. It allows these functions to be delegated while protecting the security of the root password.

Configuring the sudoers file

As a sysadmin, I can use the `/etc/sudoers` file to allow users or groups of users access to a single command, defined groups of commands, or all commands. This flexibility is key to both the power and the simplicity of using `sudo` for delegation.

I have copied the entire `sudoers` file in **Figure 1** from the host on which I am using it in order to deconstruct it for you. I found it very confusing the first time I encountered it. Hopefully, it won't be quite so obscure for you by the time we get through. I do like that Red Hat-based distributions tend to have default configuration files with lots of comments and examples to provide guidance. This does make things easier as much less Googling is necessary.

Do not use your standard editor to modify the `sudoers` file. Use the `visudo` command because it is designed to enable any changes as soon as the file is saved and you exit from the editor. It is possible to use editors besides `vi` in

Host aliases

The Host Aliases section is used to create groups of hosts on which commands or command aliases can be used to provide access. The basic idea is that this single file will be maintained for all hosts in an organization and copied to `/etc` on each host. Some hosts, such as servers can thus be configured as a group to allow some users access to specific commands such as the ability to start and stop services like HTTPD, DNS, networking, the ability to mount filesystems, and so on.
IP addresses can be used instead of hostnames in the host aliases.

```
## Sudoers allows particular users to run various commands as
## the root user, without needing the root password.
##
## Examples are provided at the bottom of the file for collections
## of related commands, which can then be delegated out to particular
## users or groups.
##
## This file must be edited with the 'visudo' command.

## Host Aliases
## Groups of machines. You may prefer to use hostnames (perhaps using
## wildcards for entire domains) or IP addresses instead.
# Host_Alias FILESERVERS = fs1, fs2
# Host_Alias MAILSERVERS = smtp, smtp2

## User Aliases
## These aren't often necessary, as you can use regular groups
## (ie, from files, LDAP, NIS, etc) in this file - just use %groupname
## rather than USERALIAS
# User_Alias ADMINS = jsmith, mikem
User_Alias AUDIO = dboth, user

## Command Aliases
## These are groups of related commands...

## Networking
# Cmnd_Alias NETWORKING = /sbin/route, /sbin/ifconfig, /bin/ping, /sbin/dhclient, /usr/bin/net,
/sbin/iptables, /usr/bin/rfcomm, /usr/bin/wvdial, /sbin/iwconfig, /sbin/mii-tool

## Installation and management of software
# Cmnd_Alias SOFTWARE = /bin/rpm, /usr/bin/up2date, /usr/bin/yum

## Services
# Cmnd_Alias SERVICES = /sbin/service, /sbin/chkconfig

## Updating the locate database
# Cmnd_Alias LOCATE = /usr/bin/updatedb

## Storage
# Cmnd_Alias STORAGE = /sbin/fdisk, /sbin/sfdisk, /sbin/parted, /sbin/partprobe, /bin/mount,
/bin/umount

## Delegating permissions
# Cmnd_Alias DELEGATING = /usr/sbin/visudo, /bin/chown, /bin/chmod, /bin/chgrp

## Processes
# Cmnd_Alias PROCESSES = /bin/nice, /bin/kill, /usr/bin/kill, /usr/bin/killall

## Drivers
# Cmnd_Alias DRIVERS = /sbin/modprobe

# Defaults specification
#
# Refuse to run if unable to disable echo on the tty.
#
Defaults !visiblepw
Defaults env_reset
Defaults env_keep =  "COLORS DISPLAY HOSTNAME HISTSIZE KDEDIR LS_COLORS"
Defaults env_keep += "MAIL PS1 PS2 QTDIR USERNAME LANG LC_ADDRESS LC_CTYPE"
Defaults env_keep += "LC_COLLATE LC_IDENTIFICATION LC_MEASUREMENT LC_MESSAGES"
Defaults env_keep += "LC_MONETARY LC_NAME LC_NUMERIC LC_PAPER LC_TELEPHONE"
Defaults env_keep += "LC_TIME LC_ALL LANGUAGE LINGUAS _XKB_CHARSET XAUTHORITY"
```

```
## systems).
## Syntax:
##
## user MACHINE=COMMANDS
##
## The COMMANDS section may have other options added to it.
##
## Allow root to run any commands anywhere
root ALL=(ALL) ALL

## Allows members of the 'sys' group to run networking, software,
## service management apps and more.
# %sys ALL = NETWORKING, SOFTWARE, SERVICES, STORAGE, DELEGATING, PROCESSES, LOCATE, DRIVERS

## Allows people in group wheel to run all commands
%wheel  ALL=(ALL) ALL

## Same thing without a password
# %wheel ALL=(ALL) NOPASSWD: ALL

## Allows members of the users group to mount and unmount the
## cdrom as root
# %users ALL=/sbin/mount /mnt/cdrom, /sbin/umount /mnt/cdrom

## Allows members of the users group to shutdown this system
# %users  localhost=/sbin/shutdown -h now

## Read drop-in files from /etc/sudoers.d (the # here does not mean a comment)
#includedir /etc/sudoers.d

################################################################################
# Added by David Both, 11/04/2017 to provide limited access to myprog #
################################################################################
#
AUDIO guest1=/usr/local/bin/myprog
```

Figure 1: A default sudoers file with my modifications in bold.

User aliases

The next group of configuration samples is user aliases. This allows root to sort users into aliased groups so that an entire group can be provided access to certain root capabilities. For the little program I wrote, I added the following alias to this section which defines the alias `AUDIO` and assigns two users to that alias.

```
User_Alias AUDIO = dboth, ruser
```

It is possible, as stated in the `sudoers` file itself, to simply use Linux groups defined in the `/etc/groups` file instead of aliases. If you already have a group defined there that meets your needs, such as “audio”, use that group name preceded by a `%` sign like so: `%audio` when assigning commands available to groups later in the `sudoers` file.

Command aliases

Further down the `sudoers` file is a section with command aliases. These aliases are lists of related commands such as networking commands or commands required to install updates or new RPM packages. These aliases allow the sysadmin to easily allow access to groups of commands.

There are a number of aliases already set up in this section that make it easy to delegate access to specific types of commands. We don’t need any of them for our use case.

Environment defaults

Command section

This section is the main part of the `sudoers` file. Everything necessary can be done without all of the aliases by adding enough entries here. The aliases just make it a whole lot easier by using the aliases already defined to tell `sudo` who can do what on which hosts. The examples are self-explanatory once you understand the syntax in this section. So let's look at the syntax that we find in the command section. In **Figure 2** we have a generic entry for our user, ruser.

```
ruser ALL=(ALL) ALL
```

Figure 2: A generic entry that says that ruser can run any program on any host as any user.

The first ALL in the line indicates that this rule applies to all hosts. The second ALL allows ruser to run commands as any other user. By default, commands are run as root user but ruser can specify on the `sudo` command line that a program is run as any other user. The last ALL means that ruser can run all commands without restriction. This entry would make ruser effectively into root. We won't be using this entry because we do not want this user to have all of those capabilities.

Note that there is an entry for root as shown in **Figure 3**. This entry allows root to have all-encompassing access to all commands on all hosts.

```
root ALL=(ALL) ALL
```

Figure 3: An entry that says that toot can run any program on any host as any user.
But, of course, I had to try this out so I commented out the line in **Figure 3** and, as root, tried to run `chown` without `sudo`. That did work – much to my surprise. Then I used `sudo chown` and that failed with the message, "root is not in the sudoers file. This incident will be reported." This means that root can run everything as root, but nothing when using the `sudo` command. This would prevent root from running commands as other users via the `sudo` command, but root has plenty of ways around that restriction.

The entry in **Figure 4** is the one I added to control access to `myprog`. It specifies that users who are listed in the AUDIO group, as defined near the top of the `sudoers` file, have access to only the one program, `myprog`, on one host, guest1.

```
AUDIO guest1=/usr/local/bin/myprog
```

Figure 4: This is the entry I added to allow users who are part of the AUDIO group access to `myprog` on the host, guest1.

Note that the syntax of the line in **Figure 4** specifies only the host on which this access is to be allowed and the program. It does not specify that the user may run the program as any other user.

Bypassing passwords

Figure 5 illustrates using NOPASSWORD to allow the users specified in the group AUDIO to run `myprog` without the need for entering their passwords.

```
AUDIO guest1=NOPASSWORD : /usr/local/bin/myprog
```

Figure 5: This hypothetical entry would allow users who are part of the AUDIO group access to `myprog` without the need to enter their passwords.

I did not do this for my program because I believe that users with `sudo` access must stop and think about what they are doing and this may help a bit with that. I just used the entry for my little program as an example.

Wheel

to the group there for this to work. The `%` sign preceding the group name means that `sudo` should look for that group in the `/etc/group` file.

```
%wheel ALL = (ALL) ALL
```

Figure 6: This entry says that all users who are members of the wheel group as defined in the `/etc/group` file can run all commands on any host.

This is a good way to delegate full root access to multiple users without providing the root password. Just adding a user to the wheel group gives them access to full root powers. It also provides a means to monitor their activities via the log entries created by `sudo`. Some distributions such as Ubuntu add users' IDs to the wheel group in `/etc/group` which allows them to use the `sudo` command to use all privileged commands.

Use case 2: The presentation

We usually think of presentations in terms of slick slides created by LibreOffice Impress or PowerPoint, animated transitions, and graphics, but that need not be the case. On March 3 of this year, I presented an extended session at Open Source 101 in Columbia South Carolina, entitled, "Using and Configuring Bash."

Now, if you are attending a long session about using Bash, why would you want to see a lot of fancy slides? Personally, I want to see Bash in use. So I created a Bash script that became my presentation. I am not a fast or accurate typist so this also allowed me to preprogram all of the commands I wanted to demonstrate into the Bash script so I would only need to press the `Enter` key to move to the next slide containing text or to issue a command. This worked very well for me and the attendees seemed to like it as well because it was a really good use case for Bash scripts.

This is relevant because one and only one of the commands in this 1500 line Bash shell script required root privilege. It would be incredibly insecure to login as root and run the entire program. What if there were to be an error that deleted or mangles some important system-level files? As a sysadmin and a Bash programmer, I am not perfect and errors happen.

So the answer was to run the program as the "student" user and use the `sudo` command as a prefix to the one command that required privileged access. Even in the middle of my presentation, typing in the password for the student user was easy. I also used this in my presentation as an example of one appropriate use of `sudo`.

The setup for this use case is the same as for the previous case, just with different user names and command names.

Final thoughts

I have used `sudo` in these cases for a very limited objective – providing one or two users with access to a single command. I accomplished this with two lines if you ignore my own comments. Delegating the authority to perform specific tasks to users who do not have root access is simple and can save you as a sysadmin a good deal of time. It can also significantly enhance security by providing access to specific commands only when necessary.

The `sudo` command also generates log entries that can help detect problems. The `sudoers` file offers a plethora of capabilities and options for configuration. Check the `man` files for `sudo` and `sudoers` for the down and dirty details.

Topics: Linux

David Both

David Both is an Open Source Software and GNU/Linux advocate, trainer, writer, and speaker who lives in Raleigh