How To Edit the Sudoers File

digitalocean.com/community/tutorials/how-to-edit-the-sudoers-file

Justin Ellingwood, Brian Boucheron September 17, 2013

Introduction

Privilege separation is one of the fundamental security paradigms implemented in Linux and Unix-like operating systems. Regular users operate with limited privileges in order to reduce the scope of their influence to their own environment, and not the wider operating system.

A special user, called **root**, has *super-user* privileges. This is an administrative account without the restrictions that are present on normal users. Users can execute commands with super-user or **root** privileges in a number of different ways.

In this article, we will discuss how to correctly and securely obtain **root** privileges, with a special focus on editing the `/etc/sudoers` file.

We will be completing these steps on an Ubuntu 20.04 server, but most modern Linux distributions such as Debian and CentOS should operate in a similar manner.

This guide assumes that you have already completed the initial server setup discussed here. Log into your server as regular, non-**root** user and continue below.

Note: This tutorial goes into depth about privilege escalation and the `sudoers` file. If you just want to add `sudo` privileges to a user, check out our *How To Create a New Sudo-enabled User* quickstart tutorials for Ubuntu and CentOS.

There are three basic ways to obtain **root** privileges, which vary in their level of sophistication.

Logging In As Root

The simplest and most straightforward method of obtaining **root** privileges is to directly log into your server as the **root** user.

If you are logging into a local machine (or using an out-of-band console feature on a virtual server), enter `root` as your username at the login prompt and enter the **root** password when asked.

If you are logging in through SSH, specify the **root** user prior to the IP address or domain name in your SSH connection string:

```
1.
   ssh root@server_domain_or_ip
```

If you have not set up SSH keys for the **root** user, enter the **root** password when prompted.

Logging in directly as **root** is usually not recommended, because it is easy to begin using the system for non-administrative tasks, which is dangerous.

The next way to gain super-user privileges allows you to become the **root** user at any time, as you need it.

We can do this by invoking the `su` command, which stands for “substitute user”. To gain **root** privileges, type:

```
su
```

You will be prompted for the **root** user’s password, after which, you will be dropped into a **root** shell session.

When you have finished the tasks which require **root** privileges, return to your normal shell by typing:

```
exit
```

The final, way of obtaining **root** privileges that we will discuss is with the `sudo` command.

The `sudo` command allows you to execute one-off commands with **root** privileges, without the need to spawn a new shell. It is executed like this:

```
sudo command_to_execute
```

Unlike `su`, the `sudo` command will request the password of the *current* user, not the **root** password.

Because of its security implications, `sudo` access is not granted to users by default, and must be set up before it functions correctly. Check out our *How To Create a New Sudo-enabled User* quickstart tutorials for Ubuntu and CentOS to learn how to set up a `sudo`-enabled user.

In the following section, we will discuss how to modify the `sudo` configuration in greater detail.

What is Visudo?

The `sudo` command is configured through a file located at `/etc/sudoers`.

Warning: Never edit this file with a normal text editor! Always use the `visudo` command instead!

Because improper syntax in the `/etc/sudoers` file can leave you with a broken system where it is impossible to obtain elevated privileges, it is important to use the `visudo` command to edit the file.

The `visudo` command opens a text editor like normal, but it validates the syntax of the file upon saving. This prevents configuration errors from blocking `sudo` operations, which may be your only way of obtaining **root** privileges.

Traditionally, `visudo` opens the `/etc/sudoers` file with the `vi` text editor. Ubuntu, however, has configured `visudo` to use the `nano` text editor instead.

If you would like to change it back to `vi`, issue the following command:

```
1.
   sudo update-alternatives --config editor
```

```
Output

There are 4 choices for the alternative editor (providing /usr/bin/editor).

  Selection    Path                Priority   Status
------------------------------------------------------------
* 0            /bin/nano            40        auto mode
  1            /bin/ed             -100       manual mode
  2            /bin/nano            40        manual mode
  3            /usr/bin/vim.basic   30        manual mode
  4            /usr/bin/vim.tiny    10        manual mode

Press <enter> to keep the current choice[*], or type selection number:
```

Select the number that corresponds with the choice you would like to make.

On CentOS, you can change this value by adding the following line to your `~/.bashrc`:

```
1.
   export EDITOR=`which name_of_editor`
```

Source the file to implement the changes:

```
1.
   . ~/.bashrc
```

After you have configured `visudo`, execute the command to access the `/etc/sudoers` file:

```
1.
   sudo visudo
```

You will be presented with the `/etc/sudoers` file in your selected text editor.

I have copied and pasted the file from Ubuntu 20.04, with comments removed. The CentOS `/etc/sudoers` file has many more lines, some of which we will not discuss in this guide.

/etc/sudoers

```
Defaults        env_reset
Defaults        mail_badpass
Defaults
secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bi
n"

root    ALL=(ALL:ALL) ALL

%admin ALL=(ALL) ALL
%sudo   ALL=(ALL:ALL) ALL

#includedir /etc/sudoers.d
```

Let's take a look at what these lines do.

Default Lines

The first line, `Defaults env_reset`, resets the terminal environment to remove any user variables. This is a safety measure used to clear potentially harmful environmental variables from the `sudo` session.

The second line, `Defaults mail_badpass`, tells the system to mail notices of bad `sudo` password attempts to the configured `mailto` user. By default, this is the **root** account.

The third line, which begins with `Defaults secure_path=...`, specifies the `PATH` (the places in the filesystem the operating system will look for applications) that will be used for `sudo` operations. This prevents using user paths which may be harmful.

User Privilege Lines

The fourth line, which dictates the **root** user's `sudo` privileges, is different from the preceding lines. Let's take a look at what the different fields mean:

- `root ALL=(ALL:ALL) ALL` The first field indicates the username that the rule will apply to (**root**).

- `root ALL=(ALL:ALL) ALL` The first “ALL” indicates that this rule applies to all hosts.
- `root ALL=(ALL:ALL) ALL` This “ALL” indicates that the **root** user can run commands as all users.
- `root ALL=(ALL:ALL) ALL` This “ALL” indicates that the **root** user can run commands as all groups.
- `root ALL=(ALL:ALL) ALL` The last “ALL” indicates these rules apply to all commands.

This means that our **root** user can run any command using `sudo`, as long as they provide their password.

Group Privilege Lines

The next two lines are similar to the user privilege lines, but they specify `sudo` rules for groups.

Names beginning with a `%` indicate group names.

Here, we see the **admin** group can execute any command as any user on any host. Similarly, the **sudo** group has the same privileges, but can execute as any group as well.

Included /etc/sudoers.d Line

The last line might look like a comment at first glance:

/etc/sudoers

```
. . .

#includedir /etc/sudoers.d
```

It *does* begin with a `#`, which usually indicates a comment. However, this line actually indicates that files within the `/etc/sudoers.d` directory will be sourced and applied as well.

Files within that directory follow the same rules as the `/etc/sudoers` file itself. Any file that does not end in `~` and that does not have a `.` in it will be read and appended to the `sudo` configuration.

This is mainly meant for applications to alter `sudo` privileges upon installation. Putting all of the associated rules within a single file in the `/etc/sudoers.d` directory can make it easy to see which privileges are associated with which accounts and to reverse credentials easily without having to try to manipulate the `/etc/sudoers` file directly.

As with the `/etc/sudoers` file itself, you should always edit files within the `/etc/sudoers.d` directory with `visudo`. The syntax for editing these files would be:

```
sudo visudo -f /etc/sudoers.d/file_to_edit
```

How To Give a User Sudo Privileges

The most common operation that users want to accomplish when managing `sudo` permissions is to grant a new user general `sudo` access. This is useful if you want to give an account full administrative access to the system.

The easiest way of doing this on a system set up with a general purpose administration group, like the Ubuntu system in this guide, is actually to add the user in question to that group.

For example, on Ubuntu 20.04, the `sudo` group has full admin privileges. We can grant a user these same privileges by adding them to the group like this:

```
sudo usermod -aG sudo username
```

The `gpasswd` command can also be used:

```
sudo gpasswd -a username sudo
```

These will both accomplish the same thing.

On CentOS, this is usually the `wheel` group instead of the `sudo` group:

```
sudo usermod -aG wheel username
```

Or, using `gpasswd`:

```
sudo gpasswd -a username wheel
```

On CentOS, if adding the user to the group does not work immediately, you may have to edit the `/etc/sudoers` file to uncomment the group name:

```
sudo visudo
```

/etc/sudoers

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

How To Set Up Custom Rules

Now that we have gotten familiar with the general syntax of the file, let's create some new rules.

How To Create Aliases

The `sudoers` file can be organized more easily by grouping things with various kinds of "aliases".

For instance, we can create three different groups of users, with overlapping membership:

/etc/sudoers

```
. . .
User_Alias              GROUPONE = abby, brent, carl
User_Alias              GROUPTWO = brent, doris, eric,
User_Alias              GROUPTHREE = doris, felicia, grant
. . .
```

Group names must start with a capital letter. We can then allow members of `GROUPTWO` to update the `apt` database by creating a rule like this:

/etc/sudoers

```
. . .
GROUPTWO        ALL = /usr/bin/apt-get update
. . .
```

If we do not specify a user/group to run as, as above, `sudo` defaults to the **root** user.

We can allow members of `GROUPTHREE` to shutdown and reboot the machine by creating a "command alias" and using that in a rule for `GROUPTHREE`:

/etc/sudoers

```
. . .
Cmnd_Alias              POWER = /sbin/shutdown, /sbin/halt, /sbin/reboot,
/sbin/restart
GROUPTHREE      ALL = POWER
. . .
```

We create a command alias called `POWER` that contains commands to power off and reboot the machine. We then allow the members of `GROUPTHREE` to execute these commands.

We can also create “Run as” aliases, which can replace the portion of the rule that specifies the user to execute the command as:

/etc/sudoers

```
. . .
Runas_Alias             WEB = www-data, apache
GROUPONE        ALL = (WEB) ALL
. . .
```

This will allow anyone who is a member of `GROUPONE` to execute commands as the `www-data` user or the `apache` user.

Just keep in mind that later rules will override earlier rules when there is a conflict between the two.

How To Lock Down Rules

There are a number of ways that you can achieve more control over how `sudo` reacts to a call.

The `updatedb` command associated with the `mlocate` package is relatively harmless on a single-user system. If we want to allow users to execute it with **root** privileges *without* having to type a password, we can make a rule like this:

/etc/sudoers

```
. . .
GROUPONE        ALL = NOPASSWD: /usr/bin/updatedb
. . .
```

`NOPASSWD` is a “tag” that means no password will be requested. It has a companion command called `PASSWD`, which is the default behavior. A tag is relevant for the rest of the rule unless overruled by its “twin” tag later down the line.

For instance, we can have a line like this:

/etc/sudoers

```
. . .
GROUPTWO        ALL = NOPASSWD: /usr/bin/updatedb, PASSWD: /bin/kill
. . .
```

Another helpful tag is `NOEXEC`, which can be used to prevent some dangerous behavior in certain programs.

For example, some programs, like `less`, can spawn other commands by typing this from within their interface:

`!command_to_run`

This basically executes any command the user gives it with the same permissions that `less` is running under, which can be quite dangerous.

To restrict this, we could use a line like this:

/etc/sudoers

```
. . .
username        ALL = NOEXEC: /usr/bin/less
. . .
```

Miscellaneous Information

There are a few more pieces of information that may be useful when dealing with `sudo`.

If you specified a user or group to "run as" in the configuration file, you can execute commands as those users by using the `-u` and `-g` flags, respectively:

1. `sudo -u run_as_user command`
2. `sudo -g run_as_group command`

For convenience, by default, `sudo` will save your authentication details for a certain amount of time in one terminal. This means you won't have to type your password in again until that timer runs out.

For security purposes, if you wish to clear this timer when you are done running administrative commands, you can run:

1. `sudo -k`

If, on the other hand, you want to "prime" the `sudo` command so that you won't be prompted later, or to renew your `sudo` lease, you can always type:

1. `sudo -v`

You will be prompted for your password, which will be cached for later `sudo` uses until the `sudo` time frame expires.

If you are simply wondering what kind of privileges are defined for your username, you can type:

```
1.
   sudo -l
```

This will list all of the rules in the `/etc/sudoers` file that apply to your user. This gives you a good idea of what you will or will not be allowed to do with `sudo` as any user.

There are many times when you will execute a command and it will fail because you forgot to preface it with `sudo`. To avoid having to re-type the command, you can take advantage of a bash functionality that means “repeat last command”:

```
1.
   sudo !!
```

The double exclamation point will repeat the last command. We preceded it with `sudo` to quickly change the unprivileged command to a privileged command.

For some fun, you can add the following line to your `/etc/sudoers` file with `visudo`:

```
1.
   sudo visudo
```

/etc/sudoers

```
. . .
Defaults        insults
. . .
```

This will cause `sudo` to return a silly insult when a user types in an incorrect password for `sudo`. We can use `sudo -k` to clear the previous `sudo` cached password to try it out:

```
1.
   sudo -k

2.
   sudo ls
```

```
Output

[sudo] password for demo:    # enter an incorrect password here to see the results
Your mind just hasn't been the same since the electro-shock, has it?
[sudo] password for demo:
My mind is going. I can feel it.
```

Conclusion

You should now have a basic understanding of how to read and modify the `sudoers` file, and a grasp on the various methods that you can use to obtain **root** privileges.

Remember, super-user privileges are not given to regular users for a reason. It is essential that you understand what each command does that you execute with **root** privileges. Do not take the responsibility lightly. Learn the best way to use these tools for your use-case, and lock down any functionality that is not needed.

Spin up a real linux environment on a hosted virtual machine in seconds with DigitalOcean Droplets! Simple enough for any user, powerful enough for fast-growing applications or businesses.