# KIT304 Server Administration and Security Assurance

# Tutorial 3

## Goal

In today's tutorial you will continue looking at user groups, and then move on to looking at file ownership and permissions.

## Introduction

In the previous tutorial you explored practical skills around creating users and groups. Users, with authentication, enable us to express ownership and access rights. Groups enable us to share ownership, or grant access rights to a set of users. File access permissions are the most obvious way of seeing how access rights are applied to system objects. As you have seen in previous units, and will see again today, these are commonly expressed as read, write and execute, or **rwx** permissions. If a user doesn't have permission to undertake a given task, such as read a file, then the task can't be undertaken.

At times in this unit you have used commands that require a higher level of permission than a "standard" user account normally has. These examples have involved system files (like `/etc/shadow`) that require administrator privileges to be accessed. When you execute the `su` command and enter the root password, you become the administrator of the system and have full access. Your access rights change from that of a user to that of an administrator.

What you may not realise is that many Unix programs let users update sensitive system files, such as `/etc/shadow`, which they can't normally access, and without using the `su` command to do so. The `passwd` program is an example of this. It isn't the program that enables this access, but rather special permissions that are set on it. When such a program is executed it uses a special permission that elevates it temporarily to root privilege, and once it's complete, it terminates, and the user is left running at their original permission. The special permission that provides this capability is the `setuid` bit, which is expressed via the letter **s** in place of the usual **x** or **-** when listing a file's permissions. For example:

```
[student@localhost ~]$ ls -al /usr/bin/passwd
-rwsr-xr-x. 1 root root 27832 Jun 10  2014 /usr/bin/passwd
[student@localhost ~]$
```

Notice the **s** (shown in red above). This indicates that the executable will run with the permissions of the owner (root in this case) rather than with the permissions of the user executing it.

The **setuid** bit can be set on any executable by the root user through the **`chmod`** command, which you will explore later in this tutorial.

The **`sudo`** command is another tool that can provide root access without using the root password. At its simplest level, this requires that the user be a member of a special group (the **`wheel`** group in CentOS, or the **`sudoers`** or **`admin`** in other Unix distributions) to run commands as root. **`sudo`** is actually much more general than this, and can be configured to allow any user or group of users to run commands as another user.

## Activity 1 – More on root access

In today's tutorial you will again be connecting to CentOS from Kali. While you could run all of the commands you will be exploring today on CentOS directly, you will be using multiple user accounts, and for that, it's usually simpler to use multiple **`ssh`** sessions from another system.

1. Start up both **CentOS** and **Kali**, and configure both virtual machines so that they have IP addresses. If you have forgotten how to do this since the last tutorial, there is an appendix at the end of this document summarising the relevant steps.

2. Add three users to the **CentOS** system using the same approach you used in the previous tutorial (that is, use the terminal and **`ssh`** across to CentOS, become the root user and so on). These three users will be used for specific purposes over the next few tasks, so write them down here so you remember which user will be used for which task:

    a) a user to give **`sudo`** access to ____________________________
    b) a user to be network admin ____________________________
    c) a user to delete ____________________________

3. For the user (a), give them a password. Then, create a new terminal window on Kali, and use it to **`ssh`** across to the CentOS VM as that user. Once that connection is set up, use it to try to delete user (c) (do not use the **`su`** command).

    It shouldn't work, and this shouldn't surprise you as user (a) has only just been created and doesn't have any special permissions. Now attempt the deletion of user (c) again but this time precede the command with **`sudo`**.

    Again it shouldn't work, but it will attempt the process by first asking for authentication of the user (a) account – this is because **`sudo`** authenticates the user before checking whether or not that user is permitted to run that command. Since user (a) is not yet permitted to delete users, **`sudo`** displays an error message, and then indicates that you will be reported. To view that report have a look at the file **`/var/log/secure`** (which you can only do as the **root** user).

4. The configuration information for **`sudo`**, that defines which users can run which commands with special privileges is **`/etc/sudoers`**. To edit this file you use the special command **`visudo`**. Why does this special command need to exist?

   ______________________________________________________________________

   ______________________________________________________________________

> ➢ **`visudo`** is based on the **`vi`** editor. Be sure you understand the basics of editing files in **`vi`**, as covered in the first tutorial.

As the root user on CentOS, run **`visudo`** and read the comments spread throughout the file. It enables you to define which accounts have root privileges, and which groups have admin privileges. It flexibly allows full privileges to be given to some users, while only allowing specific commands to run by others. Note particularly the command aliases that are pre-defined but commented out, and specifically the **`NETWORKING`** command alias in the file – you will use this shortly.

Close to the bottom of the file you will find the section that is of the most interest today, which is the part where the privilege rules are defined. There you should see the following two lines (not consecutively, but a few lines apart)

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

The first line declares, in order, that the root user can execute from **`ALL`** terminals, acting as **`ALL`** (any) users, and run **`ALL`** (any) command. Likewise, the **`%wheel`** line declares, in order, that the members of the group **`wheel`** can execute from **`ALL`** terminals, acting as **`ALL`** users running **`ALL`** commands. These rules basically state that **root** or any member of the **wheel** group can do anything that they want on this system.

Up until this point you have become the root user by using the **`su`** command. But if you add someone to the **wheel** group, they can get root access through the **`sudo`** command using *their own password*, and without knowing the root password. This is a common technique to give users administrative rights to a system without sharing the root password with them.

5. Quit **`visudo`** without saving any edits (**`:q`**). Using the last tutorial as a guide, add the user (a) that you created back in step 2 to the **`wheel`** group. Confirm that they have been added to the group.

   What is the command to add them? ______________________________

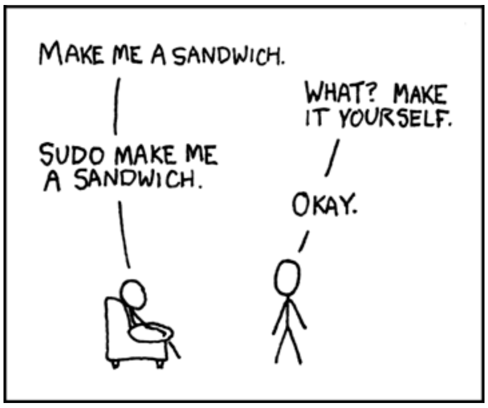
   What is the command to check that they are now in the group?

______________________________________________________________________________

6. Go back to your **Kali** VM terminal window where user (a) is currently connected to CentOS via **ssh**. Try again to delete the account created in (c).

   It still won't work because group membership is read by the shell at login, and not updated during your session. Type `exit` to leave the **ssh** session, and then create a new `ssh` connection session and attempt again to delete account (c), this time prefixing the command with `sudo`.

   This time you should see success. The user (a) that you created now has root access without needing to know the root password. They just need to prefix their commands with `sudo` to run them as the root user.



https://www.xkcd.com/149/

7. Next you are going to give the user (b) some root privileges, but not complete root access to the whole system. As you should have seen earlier in the `sudoers` file, there are command aliases that group certain commands together. These allow us to give a user root access to sets of commands, and only those commands.

   On the CentOS VM give user (b) a password, open a new terminal window (in Kali), and create another `ssh` session to CentOS as user (b). This user is a normal user and can't access all commands in the system. To prove this, and to illustrate another feature you may not have known about, try the following options on a command you are familiar with:

   ```
   ping ipAddressofKali -c 10 -i 0
   ```

   Why doesn't this work?

______________________________________________________________________________

8. Returning to your CentOS terminal session where you have root access, edit the `sudoers` file using the special editing tool described earlier. This time remove the **#** comment marker from in front of the `NETWORKING` command alias, and then add the following below the wheel group:

```
%networking	ALL=(ALL)	NETWORKING
```

**Note**: The three fields are **tab** separated, so it doesn't look as nicely aligned as you might expect.

Save and quit the file (`:x`). Using the previous tutorial as a guide, create a group called `networking` and add user (b) to it. Return to your ssh session to Kali as user (b). You may try the `ping` command again (prefixed with `sudo`), although you should recall that you need to establish a new terminal session for `sudo` to recognise the new permissions.

Try to use `sudo` as account (b) to create a new user. Does it work? Why/why not?

________________________________________________________________________

## Activity 2 – File Permissions

At this point you now have had a good amount of exposure to, and hopefully confidence in, creating and modifying user accounts. Realistically, in the real world, the kinds of things that you have explored are on the less common side of an account's life cycle. Accounts get created, perhaps some minor change are made to them, but then the bulk of what occurs from that point onward is the user interacting with the system, creating and sharing files, and executing commands. The remainder of your tutorial today is going to focus in on this. How are permissions set? How do you share items? What can the user do themselves, and what does the administrator need to facilitate?

1. Create two users on CentOS, which will be referred to here as **alpha** and **bravo**, and assign both a password. Then on Kali create two new terminal windows, and create an `ssh` session through to CentOS for each of them.

2. Both **alpha** and **bravo**, upon connection, are placed in their empty home directories. Using one account (e.g., **alpha**) create a file using `touch` and a directory using `mkdir`. If this is the first time you have ever seen these commands have a quick read of the relevant man pages.

3. Using `ls` can you view the file permissions of the items you created? What flag do you need?

   ________________________________________________________________________

   What visible differences are there in the listing between the file and the directory?

   ________________________________________________________________________

   ________________________________________________________________________

4. The permissions are expressed as three triplets of `rwx` – one set for the owner of the file, one for the group (by default the group of the owner) and finally everyone else (other). These

permissions can be changed with the `chmod` tool. Read up on that command now before you start to use it.

5. Assume that **alpha** has a file and would like **bravo** to be able to read it. Currently, this isn't possible – try to access `alpha`'s file from `bravo's` ssh session using the `cat` command). What `chmod` command, in symbolic mode, can you use to give **bravo** read access?

   ______________________________________________________________________

   If you were to remove that same permission, what would the command be? (You don't need to run it, or if you do, re-add the permission afterwards)

   ______________________________________________________________________

6. Now that **alpha** has given read access to **bravo**, attempt to open the file as **bravo**.

   You should find that **bravo** still can't open it, even with the full path being entered. The reason is that **bravo** can't get into the home directory of **alpha** because **alpha**'s home directory permissions prevent this. Can **bravo** change the permissions on **alpha**'s home directory to gain access? Why?

   ______________________________________________________________________

7. As **alpha**, use the `chmod` to change the permissions on their home directory. What is the command that will make **all** of this user's files readable to all other users, using numeric mode (rather than symbolic mode)?

   ______________________________________________________________________

   Did this enable **bravo** to read the files in **alpha**'s home directory? Is there anything you need in addition to read permission?

   ______________________________________________________________________

   ______________________________________________________________________

8. Granting **bravo** access to all of the files of **alpha** is not typically a good solution – a better approach is to use **group memberships** and **group permissions** to provide access to just the files that need to be shared. That way, you can add members to the group to grant them access to a resource. In addition, it's more common in real-world systems to only provide access to the specific resources that need to be shared, not (as in our simple example) to every file in **alpha**'s home directory.

9. Use **`chmod`** to reset the permissions of **alpha**’s home directory back to what they were originally so that **bravo** can no longer access **alpha**’s files (the permissions are **`700`** if you’re changing them numerically).

   Next, you’ll re-try the approach using groups rather than opening up access to everyone.

10. Create a new group (which we’ll call **`team`** here, but you can call it anything you want) and add both **alpha** and **bravo** to it. Can either of the users create this group themselves?

    ______________________________________________________________________

    What commands did you use to make the users members of the new group?

    ______________________________________________________________________

    After changing each user’s group membership, you will need to re-establish their **`ssh`** connection from Kali to CentOS to pick up the new group membership – do this for both users before proceeding.

11. To change a folder’s *group ownership*, use the **`chgrp`** command. What command would **alpha** use to change the group ownership of all of the files in their own home directory to allow all members of the **team** group to access it?

    ______________________________________________________________________

12. If you do a directory listing of **`/home`**, you should see that **alpha**’s home directory now shows the group owner is **team** (or whatever you called it) which **bravo** is a member of. But **bravo** still cannot access the files inside **alpha**’s home directory because although you’ve assigned a group owner, no *group permissions* have been set. The directory listing should show that **alpha**’s home directory group access permissions are `---`, which means that members of the **team** group still can’t read, write or search that directory. You can fix this with the **`chmod`** command. Using numeric mode, what **`chmod`** command would you apply to grant read and search access to the group owner?

    ______________________________________________________________________

    With that done, **bravo** should now be able to read the file **alpha** created back at step 2, but they won’t have write permission because you didn’t grant write access just now.

13. The final tool to explore today enables you to change the ownership of a file or directory. Imagine **alpha** and **bravo** work together, and then **bravo** leaves the company. **alpha** now might need access to all the files of **bravo**. One option is for the admin to make **alpha** the owner of all of **bravo**’s files. The tool to do this is **`chown`**, which can only be run by the root user. What

command and flags would you use to change **bravo**'s home directory, and everything inside it, so that it was owned by **alpha**?

________________________________________________________________________

Run this command and verify that **alpha** now owns all of **bravo**'s files. As a point of interest, what happens when **bravo** connects again via **ssh**?

________________________________________________________________________

## Conclusion

In today's tutorial you've built upon the foundations of the previous tutorial in relation to user and group creation, and moved into permissions. We haven't discussed security through these activities, focussing instead on gaining skills, but this is an area in which security is vitally important. Some of your readings and future lectures, and some future tutorials, will focus more on the security aspect.

By this point you should be starting to gain confidence in using these commands and being able to quickly obtain information about tools as needed (with `man` or `info`). It's important to understand that it's not necessary to memorise all of the flags a command can use, but it's important to know generally what a tool can do, and how to quickly learn the details when needed.

## Skills to Master

To successfully complete this tutorial, you must have mastered the following skills:

- understanding file and directory permissions, and how to change them
- understanding file and directory user and group ownership, and how to change them
- manipulating group memberships and file and directory permissions to allow groups of users to access shared resources
- granting privileged access to users via the `sudoers` file
- granting privileged access to a specific subset of commands via the `sudoers` file
- using `sudo` to run commands with elevated privilege

Anything identified in this skills section could be part of an assessment item in this module.

## Appendix – Networking

To configure the CentOS IP address:

- Click on the **Power** icon menu at the upper right of the screen, and then click on the **Ethernet (ens33) Off** entry to expose a submenu. Choose the **Wired Settings** option.
- In the network settings panel that opens, press the "gear" icon to the right of the **Ethernet (ens33) 1000 MB/s** connection and then click on the **IPv4** tab.
- Change **IPv4 Method** from *Automatic (DHCP)* to *Manual*.
- Set the IP address and subnet mask appropriately, and set the gateway to 0.0.0.0.
- Click **Apply**
- Turn the Wired Connection **Off** and then **On** again to force the changes to be applied.

Check that the settings are now in use by starting a terminal window, and entering the command `ifconfig ens33` (the settings you entered should be visible in the output).

To configure the Kali IP address:

- open a terminal window (using the second icon from the top of the quick launch item on the left hand side of screen) and type the following

  ```
  ip a add 192.168.1.Y/24 dev eth0
  ```

  Choosing your own value for ***Y***, ensuring that it's different to the value of ***X*** that you used for CentOS. Check that it has been correctly assigned by entering the command `ifconfig`, or `ip a`.