

Change permissions

Hi there! In the previous video, you learned how to check permissions for a user. In this video, we're going to learn about changing permissions.

When working as a security analyst, there may be many reasons to change permissions for a user. A user may have changed departments or been assigned to a different work group. A user might simply no longer be working on a project that requires certain permissions. These changes are necessary in order to protect system files from being accidentally or deliberately altered or deleted.

Let's explore a related command that helps control this access. In this video, we'll learn about `chmod`. `chmod` changes permissions on files and directories. The command `chmod` stands for change mode.

There are two modes for changing permissions, but we'll focus on symbolic. The best way to learn about how `chmod` works is through an example. I know this has a lot of details, but we'll break this down. Also, please keep in mind that, like many Linux commands, you don't have to memorize the information and can always find a reference.

With `chmod`, you need to identify which file or directory you want to adjust permissions for. This is the final argument, in this case, a file named: `access.txt`. The first argument, added directly after the `chmod` command, indicates how to change permissions. Right now, this might seem hard to interpret, but soon we'll understand why this is called symbolic mode.

Previously, we learned about the three types of owners: user, group, and other. To identify these with `chmod`, we use `u` to represent the user, `g` to represent the group, and `o` to represent other. In this particular example, `g` indicates we will make some changes to group permissions, and `o` to permissions for other. These owner types are separated by a comma in this argument.

But do we want to add or take away permissions? Well, for this, we use mathematical operators. So, the plus sign after `g` means we want to add permissions for group. The minus sign after `o` means we want to take them away from other. And the last question is: what kind of changes? We've already learned that `r` represents read permissions, `w` represents write permissions, and `x` represents execute permissions.

So in this case, the w indicates that we're adding write permissions to the group, and r indicates that we are taking away read permissions from other.

This is still very complex.

But now that we've broken it down, perhaps it doesn't seem quite so much like a foreign language. And remember, you don't have to memorize this all.

Let's give this new command a try.

We'll start out in the logs sub-directory.

If we use the ls -l command, it will output the permissions for the file.

It shows the permissions for the only file in this directory: access.txt.

Previously, we learned how to read these permissions.

The second through fourth characters indicate that the user has read and write permissions.

The fifth through seventh characters show the group only has read permissions.

And the eighth to tenth characters show that other only has read permissions.

We need to adjust these permissions.

We want to ensure analysts in the security group have write permission, but takeaway read permissions from the owner-type other, so we add write permissions for group and remove read permissions for other.

Let's run ls -l again. This shows a change in the permissions for access.txt.

Notice how in the middle segment of permissions for the group, w has been added to give write permissions.

And another change is that the r has been removed in the last segment, indicating that read permissions for other have been removed.

As mentioned earlier, these hyphens indicate a lack of permissions.

Now, other is lacking all permissions.

Though it requires practice,
working in Linux becomes more natural with time.

I'm glad you're learning a little
more about how to use Linux.

Revision #1

Created 7 July 2023 11:17:36 by naruzkurai

Updated 7 July 2023 11:20:19 by naruzkurai