Permission commands

Previously, you explored file permissions and the commands that you can use to display and change them. In this reading, you'll review these concepts and also focus on an example of how these commands work together when putting the principle of least privilege into practice.

Reading permissions

In Linux, permissions are represented with a 10-character string. Permissions include:

- **read**: for files, this is the ability to read the file contents; for directories, this is the ability to read all contents in the directory including both files and subdirectories
- **write**: for files, this is the ability to make modifications on the file contents; for directories, this is the ability to create new files in the directory
- **execute**: for files, this is the ability to execute the file if it's a program; for directories, this is the ability to enter the directory and access its files

These permissions are given to these types of owners:

- **user**: the owner of the file
- **group**: a larger group that the owner is a part of
- **other**: all other users on the system

Each character in the 10-character string conveys different information about these permissions. The following table describes the purpose of each character:

Character	Example	Meaning
1st	**d**rwxrwxrwx	file type • *d* for directory • **-** for a regular file
2nd	d**r**wxrwxrwx	read permissions for the user • *r* if the user has read permissions • **-** if the user lacks read permissions

Character	Example	Meaning
3rd	dr**w**xrwxrwx	write permissions for the user • *w* if the user has write permissions • **-** if the user lacks write permissions
4th	drw**x**rwxrwx	execute permissions for the user • *x* if the user has execute permissions • **-** if the user lacks execute permissions
5th	drwx**r**wxrwx	read permissions for the group • *r* if the group has read permissions • **-** if the group lacks read permissions
6th	drwxr**w**xrwx	write permissions for the group • *w* if the group has write permissions • **-** if the group lacks write permissions
7th	drwxrw**x**rwx	execute permissions for the group • *x* if the group has execute permissions • **-** if the group lacks execute permissions
8th	drwxrwx**r**wx	read permissions for other • *r* if the other owner type has read permissions • **-** if the other owner type lacks read permissions
9th	drwxrwxr**w**x	write permissions for other • *w* if the other owner type has write permissions • **-** if the other owner type lacks write permissions

Character	Example	Meaning
10th	drwxrwxrw**x**	execute permissions for other • *x* if the other owner type has execute permissions • **-** if the other owner type lacks execute permissions

Exploring existing permissions

You can use the *ls* command to investigate who has permissions on files and directories. Previously, you learned that *ls* displays the names of files in directories in the current working directory.

There are additional options you can add to the *ls* command to make your command more specific. Some of these options provide details about permissions. Here are a few important *ls* options for security analysts:

- *ls -a*: Displays hidden files. Hidden files start with a period (.) at the beginning.
- *ls -l*: Displays permissions to files and directories. Also displays other additional information, including owner name, group, file size, and the time of last modification.
- *ls -la*: Displays permissions to files and directories, including hidden files. This is a combination of the other two options.

Changing permissions

The **principle of least privilege** is the concept of granting only the minimal access and authorization required to complete a task or function. In other words, users should not have privileges that are beyond what is necessary. Not following the principle of least privilege can create security risks.

The *chmod* command can help you manage this authorization. The *chmod* command changes permissions on files and directories.

Using chmod

The *chmod* command requires two arguments. The first argument indicates how to change permissions, and the second argument indicates the file or directory that you want to change permissions for. For example, the following command would add all permissions to

login_sessions.txt:

chmod u+rwx,g+rwx,o+rwx login_sessions.txt

If you wanted to take all the permissions away, you could use

chmod u-rwx,g-rwx,o-rwx login_sessions.txt

Another way to assign these permissions is to use the equals sign (=) in this first argument. Using = with *chmod* sets, or assigns, the permissions exactly as specified. For example, the following command would set read permissions for *login_sessions.txt* for user, group, and other:

chmod u=r,g=r,o=r login_sessions.txt

This command overwrites existing permissions. For instance, if the user previously had write permissions, these write permissions are removed after you specify only read permissions with =.

The following table reviews how each character is used within the first argument of *chmod*:

Character	Description
u	indicates changes will be made to user permissions
g	indicates changes will be made to group permissions
o	indicates changes will be made to other permissions
+	adds permissions to the user, group, or other
-	removes permissions from the user, group, or other
=	assigns permissions for the user, group, or other

Note: When there are permission changes to more than one owner type, commas are needed to separate changes for each owner type. You should not add spaces after those commas.

The principle of least privilege in action

As a security analyst, you may encounter a situation like this one: There's a file called *bonuses.txt* within a compensation directory. The owner of this file is a member of the Human Resources department with a username of *hrrep1*. It has been decided that *hrrep1* needs access to this file. But, since this file contains confidential information, no one else in the *hr* group needs access.

You run *ls -l* to check the permissions of files in the compensation directory and discover that the permissions for *bonuses.txt* are *-rw-rw----*. The group owner type has read and write permissions that do not align with the principle of least privilege.

To remedy the situation, you input *chmod g-rw bonuses.txt*. Now, only the user who needs to access this file to carry out their job responsibilities can access this file.

Key takeaways

Managing directory and file permissions may be a part of your work as a security analyst. Using *ls* with the *-l* and *-la* options allows you to investigate directory and file permissions. Using *chmod* allows you to change user permissions and ensure they are aligned with the principle of least privilege.

Revision #1
Created 5 July 2023 11:48:40 by naruzkurai
Updated 6 July 2023 17:33:12 by naruzkurai