UNDERSTANDING LINUX FILE PERMISSIONS

Although there are already a lot of good security features built into Linux-based systems, one very important potential vulnerability can exist when local access is granted - - that is file permission based issues resulting from a user not assigning the correct permissions to files and directories. So based upon the need for proper permissions, I will go over the ways to assign permissions and show you some examples where modification may be necessary.

Basic File Permissions

Permission Groups

Each file and directory has three user based permission groups:

- **owner** - The Owner permissions apply only the owner of the file or directory, they will not impact the actions of other users.
- **group** - The Group permissions apply only to the group that has been assigned to the file or directory, they will not effect the actions of other users.
- **all users** - The All Users permissions apply to all other users on the system, this is the permission group that you want to watch the most.

Permission Types

Each file or directory has three basic permission types:

- **read** - The Read permission refers to a user's capability to read the contents of the file.
- **write** - The Write permissions refer to a user's capability to write or modify a file or directory.
- **execute** - The Execute permission affects a user's capability to execute a file or view the contents of a directory.

Viewing the Permissions

You can view the permissions by checking the file or directory permissions in your favorite GUI File Manager (which I will not cover here) or by reviewing the output of the ***\"ls -l\"*** command while in the terminal and while working in the directory which contains the file or folder.

The permission in the command line is displayed as: _***rwxrwxrwx 1 owner:group***

1. User rights/Permissions

    a) The first character that I marked with an underscore is the special permission flag that can vary.

    b) The following set of three characters (rwx) is for the owner permissions.

    c) The second set of three characters (rwx) is for the Group permissions.

d) The third set of three characters (rwx) is for the All Users permissions.

2. Following that grouping since the integer/number displays the number of hardlinks to the file.
3. The last piece is the Owner and Group assignment formatted as Owner:Group.

Modifying the Permissions

When in the command line, the permissions are edited by using the command ***chmod***. You can assign the permissions explicitly or by using a binary reference as described below.

Explicitly Defining Permissions

To explicity define permissions you will need to reference the Permission Group and Permission Types.

The Permission Groups used are:

- **u** - Owner
- **g** - Group
- **o** or **a** - All Users

The potential Assignment Operators are + (plus) and - (minus); these are used to tell the system whether to add or remove the specific permissions.

The Permission Types that are used are:

- **r** - Read
- **w** - Write
- **x** - Execute

So for an example, lets say I have a file named file1 that currently has the permissions set to **_rw_rw_rw,** which means that the owner, group and all users have read and write permission. Now we want to remove the read and write permissions from the all users group.

To make this modification you would invoke the command: ***chmod a-rw file1***
To add the permissions above you would invoke the command: ***chmod a+rw file1***

As you can see, if you want to grant those permissions you would change the minus character to a plus to add those permissions.

Using Binary References to Set permissions

Now that you understand the permissions groups and types this one should feel natural. To set the permission using binary references you must first understand that the input is done by entering three integers/numbers.

A sample permission string would be **chmod 640 file1**, which means that the owner has read and write permissions, the group has read permissions, and all other user have no rights to the

file.

The first number represents the Owner permission; the second represents the Group permissions; and the last number represents the permissions for all other users. The numbers are a binary representation of the rwx string.

- ***r** = 4*
- ***w** = 2*
- ***x** = 1*

You add the numbers to get the integer/number representing the permissions you wish to set. You will need to include the binary permissions for each of the three permission groups.

So to set a file to permissions on file1 to read _***rwxr***_____, you would enter ***chmod 740 file1***.

Owners and Groups

I have made several references to Owners and Groups above, but have not yet told you how to assign or change the Owner and Group assigned to a file or directory.

You use the chown command to change owner and group assignments, the syntax is simple ***chown owner:group filename**,* so to change the owner of file1 to user1 and the group to family you would enter ***chown user1:family file1**.*

Advanced Permissions

The special permissions flag can be marked with any of the following:

- _ - no special permissions
- ***d*** - directory
- ***l*** - The file or directory is a symbolic link
- ***s*** - This indicated the setuid/setgid permissions. This is not set displayed in the special permission part of the permissions display, but is represented as a **s** in the read portion of the owner or group permissions.
- ***t*** - This indicates the sticky bit permissions. This is not set displayed in the special permission part of the permissions display, but is represented as a **t** in the executable portion of the all users permissions

Setuid/Setgid Special Permissions

The setuid/setguid permissions are used to tell the system to run an executable as the owner with the owner\'s permissions.

Be careful using setuid/setgid bits in permissions. If you incorrectly assign permissions to a file owned by root with the setuid/setgid bit set, then you can open your system to intrusion.

You can only assign the setuid/setgid bit by explicitly defining permissions. The character for

the setuid/setguid bit is **s**.

So do set the setuid/setguid bit on file2.sh you would issue the command ***chmod g+s file2.sh***.

Sticky Bit Special Permissions

The sticky bit can be very useful in shared environment because when it has been assigned to the permissions on a directory it sets it so only file owner can rename or delete the said file.

You can only assign the sticky bit by explicitly defining permissions. The character for the sticky bit is **t**.

To set the sticky bit on a directory named dir1 you would issue the command ***chmod +t dir1***.

When Permissions Are Important

To some users of Mac- or Windows-based computers you don't think about permissions, but those environments don't focus so aggressively on user based rights on files unless you are in a corporate environment. But now you are running a Linux-based system and permission based security is simplified and can be easily used to restrict access as you please.

So I will show you some documents and folders that you want to focus on and show you how the optimal permissions should be set.

- ***home directories*** - The users\' home directories are important because you do not want other users to be able to view and modify the files in another user\'s documents of desktop. To remedy this you will want the directory to have the **drwx_____ (700)** permissions, so lets say we want to enforce the correct permissions on the user user1\'s home directory that can be done by issuing the command **chmod 700 /home/user1**.
- ***bootloader configuration files*** - If you decide to implement password to boot specific operating systems then you will want to remove read and write permissions from the configuration file from all users but root. To do you can change the permissions of the file to 700.
- ***system and daemon configuration files*** - It is very important to restrict rights to system and daemon configuration files to restrict users from editing the contents, it may not be advisable to restrict read permissions, but restricting write permissions is a must. In these cases it may be best to modify the rights to 644.
- ***firewall scripts*** - It may not always be necessary to block all users from reading the firewall file, but it is advisable to restrict the users from writing to the file. In this case the firewall script is run by the root user automatically on boot, so all other users need no rights, so you can assign the 700 permissions.