Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Linux - Local System Security/60cc9673cfaf5f6628f2d410

TUTORIAL

Linux - Local System Security

Topics

Linux Security

The Linux kernel allows properly authenticated users to access files and applications. While each user is identified by a unique integer (the user id or UID), a separate database associates a username with each UID. Upon account creation, new user information is added to the user database and the user's home directory must be created and populated with some essential files. Command line programs such as useradd and userdel as well as GUI tools are used for creating and removing accounts. For each user, the following seven fields are maintained in the /etc/passwd file:

Field Name

Details

Remarks

Username :

- User login name
- Should be between 1 and 32 characters long

Password

- User password (or the character **x** if the password is stored in the **/etc/shadow** file) in encrypted format
- Is never shown in Linux when it is being typed; this stops prying eyes

User ID (UID)

- Every user must have a user id (UID)
- UID 0 is reserved for root user
- UID's ranging from 1-99 are reserved for other predefined accounts
- UID's ranging from 100-999 are reserved for system accounts and groups
- Normal users have UID's of 1000 or greater

Group ID (GID)

- The primary Group ID (GID); Group Identification Number stored in the **/etc/group** file
- Is covered in detail in the chapter on Processes

User Info

- This field is optional and allows insertion of extra information about the user such as their name
- For example: **Dr Girdhar Gopal**

Home Directory

- The absolute path location of user's home directory
- For example: **/home/gopal**

Shell

- The absolute location of a user's default shell
- For example: **/bin/bash**

root Account

root is the most privileged account on a Linux/UNIX system. This account has the ability to carry out all facets of system administration, including adding accounts, changing user passwords, examining log files, installing software, etc. Utmost care must be

taken when using this account. It has no security restrictions imposed upon it.

When you are signed in as, or acting as root, the shell prompt displays '#' (if you are using bash and you haven't customized the prompt). This convention is intended to serve as a warning to you of the absolute power of this account.

Operations Requiring root Privileges

root privileges are required to perform operations such as:

- Creating, removing and managing user accounts
- Managing software packages
- Removing or modifying system files
- Restarting system services.

Regular account users of Linux distributions might be allowed to install software packages, update some settings, use some peripheral devices, and apply various kinds of changes to the system. However, root privilege is required for performing administration tasks such as restarting services, manually installing packages and managing parts of the filesystem that are outside the normal user's directories.

Operations Not Requiring root Privileges

A regular account user can perform some operations requiring special permissions; however, the system configuration must allow such abilities to be exercised.

SUID (Set owner User ID upon execution—similar to the Windows "run as" feature) is a special kind of file permission given to a file. SUID provides temporary permissions to a user to run a program with the permissions of the file owner (which may be root) instead of the permissions held by the user.

su and sudo

Sometimes we have to use another user account without logging out. There are two commands which enable you to do so : su and sudo.

Su (switch user) is a command to switch temporarily to another user account. You have to provide the user account name and password if you are not root user. In case of root user you need only the name of the account to switch to. Its syntax is: -

```
user1@home:~$ su accountname
```

if you are logged in as user1 and wants to login as Gopal for some time, you can use on your terminal as

```
user1@home:~$ su Gopal
```

It will ask for a password and if password matched, your prompt will change to Gopal user till you log out again.

When using su, you continue to use your own environment variables and profile. If you want to use the account's user environment, put a dash (-) between the su and the account name:

```
user1@home:~$ su - Gopal
```

If you type the su command with no account name (with or without the -), you are attempting to log in to the root account and will be asked for the root password. When you have completed the tasks requiring the switched account type exit, you are returned to your original account (and environment, if applicable).

sudo (superuser do) : It enables the superuser, or root administrator, to delegate commands that can be run by others. So if any command require the sudo privileges, then without logging out and logging in as root user, you can prefix sudo to that command and that command will be executed as a root user. It is generally required when you install/uninstall something or make some configuration changes. To open a file in /etc directory we can use:

```
user1@home:~$ sudo cat /etc/passwd
```

It will ask for root password, and if password match, the command will execute as root user and then you come back to your original

account privileges. So its a temporary switch for a single command, whereas su is a temporary switch for some time till you type exit.

Listed below are the differences between the two commands:

su

- When elevating privilege, you need to enter the root password. Giving the root password to a normal user should never, ever be done.
- Once a user elevates to the root account using su, the user can do anything that the root user can do for as long as the user wants, without being asked again for a password.
- The command has limited logging features.

sudo

- When elevating privilege, you need to enter the user's password and not the root password.
- Offers more features and is considered more secure and more configurable. Exactly what the user is allowed to do can be precisely controlled and limited. By default the user will either always have to keep giving their password to do further operations with sudo, or can avoid doing so for a configurable time interval.
- The command has detailed logging features.

sudo user

All the demonstrations created have a user configured with sudo capabilities to provide the user with administrative (admin) privileges when required. sudo allows users to run programs using the security privileges of another user, generally root (superuser). The functionality of sudo is similar to that of "run as Administrator" in Windows.

If your system does not already have sudo set up and enabled, you need to do the following steps:

- You will need to make modifications as the administrative or superuser, root. While sudo will become the preferred method of doing this, we do not have it set up yet, so we will use su (which

we will discuss later in detail) instead. At the command line prompt, type su and press Enter. You will then be prompted for the root password, so enter it and press Enter. You will notice that nothing is printed; this is so others cannot see the password on the screen. You should end up with a different looking prompt, often ending with '#'. For example:

```
$ su Password:
#
```

- Now, you need to create a configuration file to enable your user account to use sudo. Typically, this file is created in the /etc/sudoers.d/ directory with the name of the file the same as your username. For example, for this demo, let's say your username is "codequotient". After doing step 1, you would then create the configuration file for "codequotient" by doing this:

```
# echo "codequotient ALL=(ALL) ALL" > /etc/sudoers.d/codequotient
```

- Finally, some Linux distributions will complain if you do not also change permissions on the file by doing:

```
# chmod 440 /etc/sudoers.d/codequotient
```

That should be it. When using sudo, by default you will be prompted to give a password (your own user password) at least the first time you do it within a specified time interval. It is possible (though very insecure) to configure sudo to not require a password or change the time window in which the password does not have to be repeated with every sudo command.

Passwords

How Passwords Are Stored

The system verifies authenticity and identity using user credentials. Originally, encrypted passwords were stored in the /etc/passwd file, which was readable by everyone. This made it rather easy for

passwords to be cracked. On modern systems, passwords are actually stored in an encrypted format in a secondary file named /etc/shadow. Only those with root access can modify/read this file.

Password Algorithm

Protecting passwords has become a crucial element of security. Most Linux distributions rely on a modern password encryption algorithm called SHA-512 (Secure Hashing Algorithm 512 bits), developed by the U.S. National Security Agency (NSA) to encrypt passwords.

The SHA-512 algorithm is widely used for security applications and protocols. These security applications and protocols include TLS, SSL, PHP, SSH, S/MIME and IPSec. SHA-512 is one of the most tested hashing algorithms.

For example, if you wish to experiment with SHA-512 encoding, the word "test" can be encoded using the program sha512sum to produce the SHA-512 form.

```
$ echo –n test | sha512sum
```

Good Password Practices

IT professionals follow several good practices for securing the data and the password of every user.

- Password aging is a method to ensure that users get prompts that remind them to create a new password after a specific period. This can ensure that passwords, if cracked, will only be usable for a limited amount of time. This feature is implemented using chage, which configures the password expiry information for a user.
- Another method is to force users to set strong passwords using Pluggable Authentication Modules (PAM). PAM can be configured to automatically verify that a password created or modified using the passwd utility is sufficiently strong. PAM configuration is implemented using a library called pam_cracklib.so, which can also be replaced by pam_passwdqc.so for more options.

File/Directory Permissions

The permissions are part of Linux security. The basic building blocks for permissions are read, write, execute, user, group and others. Permission behaves differently for files and directories: -

read(r)

Directory - Grants the capability to read the contents of the directory or subdirectories.

Files - Grants the capability to view the file.

write(w)

Directory - Grants the capability to create, modify, or remove files or subdirectories.

Files- Grants write permissions, allowing an authorized entity to modify the file.

execute(x)

Directory - Grants the capability to enter the directory.

Files - Allows the user to “run” the file as a program.

We can see the permission of a file using ls command with -l option. The output of this command is as sample: -

```
user1@home:~$ ls -l
total 16
-rw-r--r--     1     home     home     30     Nov 25 13:16
abc.txt
drwxr-xr-x     4     home     home     4096     Nov 25 13:15
project1
drwxr-xr-x     4     home     home     4096     Nov 25 13:15
project2
```

```
-rw-r--r--     1     home     home   47     Nov 25 13:16
second.txt
```

First 10 characters shows the type of file and file permissions. From 2nd to 10th character permissions are divided into 3 groups. Each group contains three permissions in the order read, write and execute. First group represent the permissions of file owner. Second group represent the permissions of the user group to which the file belongs. Third group represents the permissions of all other users of system.

So, the following line of output will be represented as below: -

```
-rw-r--r--     1     home     home   47     Nov 25 13:16
second.txt
rw-    : Group1 (Owner) : The owner has read and write permissions
r--    : Group2 (Group) : The other users from same group has
read permissions only
r--    : Group3 (Others) : All other users has read permissions
only
```

Changing permissions:

If you want to change the permissions of a particular file/folder, you can use *chmod* command. This command can work in various modes. We can apply some operators for all three groups for all three permissions.

Operator Meaning Example Result

+ Add a permission chmod u+x Add execute permission to owner.

- Remove a permission chmod o-r Remove the read permission from other users

= Set the permissions chmod g=rwx Set read, write and execute permissions to group

So if we use following commands we will get the changed outputs as below: -

```
user1@home:~$ chmod ug+x abc.txt
user1@home:~$ ls -l
total 16
-rwxr-xr--     1     home     home     30     Nov 25 13:16
abc.txt
drwxr-xr-x     4     home     home     4096     Nov 25 13:15
project1
drwxr-xr-x     4     home     home     4096     Nov 25 13:15
project2
-rw-r--r--     1     home     home   47     Nov 25 13:16
second.txt

user1@home:~$ chmod g-x abc.txt
user1@home:~$ ls -l
total 16
-rwxr--r--     1     home     home     30     Nov 25 13:16
abc.txt
drwxr-xr-x     4     home     home     4096     Nov 25 13:15
project1
drwxr-xr-x     4     home     home     4096     Nov 25 13:15
project2
-rw-r--r--     1     home     home   47     Nov 25 13:16
second.txt

user1@home:~$ chmod o=rx abc.txt
user1@home:~$ ls -l
total 16
-rwxr--r-x     1     home     home     30     Nov 25 13:16
abc.txt
drwxr-xr-x     4     home     home     4096     Nov 25 13:15
project1
drwxr-xr-x     4     home     home     4096     Nov 25 13:15
project2
-rw-r--r--     1     home     home   47     Nov 25 13:16
second.txt
```

Another way to set the permissions is with absolute numbers. We can use an absolute number with some arithmetic which denotes a particular combinations of permissions. The arithmetic can be defined as follows for each group.

Number	Specification	Permissions

0	No permissions	---
1	Execute permission	--x
2	Write permission	-w-
3	Write and Execute permissions	-wx
4	Read permission	r--
5	Read and Execute permissions	r-x
6	Read and Write permission	rw-
7	Read, Write and Execute permissions	rwx

Now for each place User, Group and others we can place any number from above and use with chmod command. For example.

- To give Read, Write permissions to User, Read permission to group and no permission to others we can use as for User(Read + Write = 6), For Group(Read = 4), For Others(No = 0), So we can use 640 with chmod as below

```
user1@home:~$ chmod 640 abc.txt
```

- To give Read, Write and Execute permissions to User, Read and Execute permission to group and others we can use as for User(Read + Write + Execute= 7), For Group(Read + Execute = 5), For Others(Read + Execute = 5), So we can use 755 with chmod as below

```
user1@home:~$ chmod 755 abc.txt
```

So, setting permissions to files is easy with chmod command.

chown: To change the owner of a file/folder

We may also change the owner of a file/directory using chown command. The chown utility changes the owner of a file and/or the group the file is associated with. Only root can change the owner of a file.

```
chown username file-list
```

The owner is the username or numeric user ID of the new owner. The file-list is a list of the pathnames of the files whose ownership and/or group association you want to change. The group is the group name or numeric group ID of the new group that the file is associated with.

Argument Meaning

owner The new owner of file-list; the group is not changed

owner:group The new owner and new group association of file-list

owner: The new owner of file-list; the group association is changed to that of the new owner's login group

:group The new group associated with file-list; the owner is not changed

umask:

When you create a file, it has a default set of permissions. On most systems, any file you create will have read and write permissions for you, and no permissions for your group or other users. But this behavior is configurable. The default permission scheme is controlled by the umask command.

Like chmod, umask takes a numerical value as its argument. Unlike chmod, however, the value given to umask represents the permissions that are to be denied. That is, the umask value gives permission to everything except that which is specified.

For example, suppose that you want to give all newly created files the permission mode 644, which gives the owner read and write permission, and read permission to both group and all. Simply take that number and subtract it from 666, the default octal base for files, to get the proper argument for umask :

```
umask 022
```

This command will cause all new files to be created with permission mode 644. Another example, suppose that you want to give all newly created files the permission mode 666, which gives the read and write permission to user, group and all. Simply take that number and subtract it from 666, the default octal base for files, to get the proper argument for umask :

```
umask 000
```

This command will cause all new files to be created with permission mode 666. The effect of umask is limited to the shell in which it’s invoked. To make the effect persist, put it in your .profile file.