Lab 3.2.2.4 - Navigating the Linux Filesystem and Permission Settings

This lab has been updated for use on NETLAB+.
www.netdevgroup.com

Objectives

In this lab, you will use familiarize yourself with Linux filesystems.

Part 1: Exploring Filesystems in Linux

The 1Linux filesystem is one of its most popular features. While Linux supports many different types of filesystems, this lab focuses on the **ext** family, one the most common filesystems found on Linux.

Step 1: Access the command line.

Launch the **CyberOps Workstation** VM. Log on to the **CyberOps Workstation** VM as the `analyst`, using the password `cyberops` and open a **terminal** window.

Step 2: Display the filesystems currently mounted.

Filesystems must be *mounted* before they can be accessed and used. In computing, *mounting a filesystem* means to make it accessible to the operating system. Mounting a filesystem is the process of linking the physical partition on the block device (hard drive, SSD drive, pen drive, etc.) to a directory, through which the entire filesystem can be accessed. Because the aforementioned directory becomes the root of the newly mounted filesystem, it is also known as *mounting point.*

a. Use the **lsblk** command to display all block devices:

```
[analyst@secOps ~]$ lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sda 8:0 0 5.9G 0 disk
└─sda1 8:1 0 5.9G 0 part /
sdb 8:16 0 1G 0 disk
└─sdb1 8:17 0 1023M 0 part
sr0 11:0 1 1024M 0 rom
```

The output above shows that the *CyberOps Workstation* VM has three block devices installed: *sr0*, *sda* and *sdb*. The tree-like output also shows partitions under *sda* and *sdb*. Conventionally, */dev/sdX* is used by *Linux* to represent hard drives, with the trailing number representing the partition number inside that device. Computers with multiple hard drives would likely display more */dev/sdX* devices. If *Linux* was running on a computer with four hard drives for example, it would show them as */dev/sda*, */dev/sdb*, */dev/sdc* and */dev/sdd*, by default. The output implies that *sda* and *sdb* are hard drives, each one containing a single partition. The output also shows that *sda* is a *5.9GB* disk while *sdb* has *1GB*.

Note: *Linux* often displays USB flash drives as */dev/sdX* as well, depending on their firmware type.

b. Use the **mount** command to display more detailed information on the currently mounted filesystems in the *CyberOps Workstation* VM.

```
[analyst@secOps ~]$ mount
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
sys on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
dev on /dev type devtmpfs (rw,nosuid,relatime,size=1030408k,nr_inodes=218258,mode=755)
run on /run type tmpfs (rw,nosuid,nodev,relatime,mode=755)
/dev/sda1 on / type ext4 (rw,relatime,data=ordered)
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,mode=755)
<output omitted>
```

Many of the filesystems above are out of scope of this course and irrelevant to the lab. Let's focus on the *root filesystem*, the filesystem stored in */dev/sda1*. The root filesystem is where the *Linux* operating system itself is stored; all the programs, tools, configuration files are stored in root filesystem by default.

c. Run the **mount** command again, but this time, use the pipe **|** to send the output of mount to **grep** to filter the output and display only the root filesystem:

```
[analyst@secOps ~]$ mount | grep sda1
/dev/sda1 on / type ext4 (rw,relatime,data=ordered)
```

In the filtered output above, mount shows us that the root filesystem is located in the first partition of the *sda* block device (*/dev/sda1*). We know this is the root filesystem because of the mounting point used: "/" (the slash symbol). The output also tells us the type of formatting used in the partition, *ext4* in this case. The information in between parentheses relates to the partition mounting options.

d. Issue the following two commands below on the *CyberOps Workstation* VM:

```
[analyst@secOps ~]$ cd /
[analyst@secOps /]$ ls -l
```

What is the meaning of the output? Where are the listed files physically stored?

Where the command changes from directory to root directory. The physical stored in /dev/sda1 file system is mounted on the root directory (/).

Why is */dev/sdb1* not shown in the output above?

It is not showing the output because /dev/sdb1 was not currently mounted.

Step 3: Manually mounting and unmounting filesystems

The **mount** command can also be used to mount and unmount filesystems. As seen in *Step 1*, the *CyberOps Workstation* VM has two hard drives installed. The first one was recognized by the kernel as */dev/sda* while the second was recognized as */dev/sdb*. Before a block device can be mounted, it must have a mounting point.

a. Use the **ls -l** command to verify that the directory **second_drive** is in the analyst's home directory.

```
[analyst@secOps /]$ cd ~
```

```
[analyst@secOps ~]$ ls –l
total 28
drwxr-xr-x 3 analyst analyst 4096 Aug 16 15:15 cyops_folder2
drwxr-xr-x 2 analyst analyst 4096 Sep 26  2014 Desktop
drwx------ 3 analyst analyst 4096 Jul 14 11:28 Downloads
drwxr-xr-x 8 analyst analyst 4096 Jul 25 16:27 lab.support.files
drwxr-xr-x 2 analyst analyst 4096 Mar  3 15:56 second_drive
```

Note: If the directory *second_drive* does not exist, use the `mkdir second_drive` command to create it.

```
[analyst@secOps ~]$ mkdir second_drive
```

Note: Depending on the state of your VM, your listing will most likely have different files and directories.

b. Use **ls -l** again to list the contents of the newly created *second_drive* directory.

```
[analyst@secOps ~]$ ls -l second_drive/
total 0
```

Notice that the directory is empty.

c. Use the **mount** command to mount **/dev/sdb1** on the newly created **second_drive** directory. The syntax of mount is: `mount [options] <device to be mounted> <mounting point>`.

```
[analyst@secOps ~]$ sudo mount /dev/sdb1 ~/second_drive/
[sudo] password for analyst:
```

No output is provided which means the mounting process was successful.

d. Now that the **/dev/sdb1** has been mounted on **/home/analyst/second_drive**, use `ls -l` to list the contents of the directory again.

```
[analyst@secOps ~]$ ls -l second_drive/
total 20
drwx------ 2 root    root    16384 Mar  3 10:59 lost+found
-rw-r--r-- 1 root    root      183 Mar  3 15:42 myFile.txt
```

Why is the directory no longer empty? Where are the listed files physically stored?

After the mount of /home/analyst/second_drive becomes entry to the file system then it will physically store in /dev/sdb1.

e. Issue the **mount** command with no options again to display detailed information about the **/dev/sdb1** partition. As before, use the **grep** command to display only the **/dev/sdX** filesystems:

```
[analyst@secOps ~]$ mount | grep sd
/dev/sda1 on / type ext4 (rw,relatime,data=ordered)
/dev/sdb1 on /home/analyst/second_drive type ext2
(rw,relatime,block_validity,barrier,user_xattr,acl)
```

f. Unmounting filesystems is just as simple. Make sure you change the directory to something outside of the mounting point and use the **umount** command as shown below:

```
[analyst@secOps ~]$ sudo umount /dev/sda1
[sudo] password for analyst:
[analyst@secOps ~]$[analyst@secOps ~]$ ls -
l second_drive/
total 0
```

Part 2: File Permissions

Linux filesystems have built-in features to control the ability of the users to view, change, navigate, and execute the contents of the filesystem. Essentially, each file in filesystems carries its own set of permissions, always carrying a set of definitions about what users and groups can do with the file.

Step 1: Visualize and Change the File Permissions.

a. Navigate to **/home/analyst/lab.support.files/scripts/**.

```
[analyst@secOps ~]$ cd lab.support.files/scripts/
```

b. Use the **ls -l** command to display file permissions.

```
[analyst@secOps scripts]$ ls -l
total 60
-rwxr-xr-x 1 analyst analyst  190 Jun 13 09:45 configure_as_dhcp.sh
-rwxr-xr-x 1 analyst analyst  192 Jun 13 09:45 configure_as_static.sh
-rwxr-xr-x 1 analyst analyst 3459 Jul 18 10:09 cyberops_extended_topo_no_fw.py
-rwxr-xr-x 1 analyst analyst 4062 Jul 18 10:09 cyberops_extended_topo.py
-rwxr-xr-x 1 analyst analyst 3669 Jul 18 10:10 cyberops_topo.py
-rw-r--r-- 1 analyst analyst 2871 Apr 28 11:27 cyops.mn
-rwxr-xr-x 1 analyst analyst  458 May  1 13:50 fw_rules
-rwxr-xr-x 1 analyst analyst   70 Apr 28 11:27 mal_server_start.sh
drwxr-xr-x 2 analyst analyst 4096 Jun 13 09:55 net_configuration_files
-rwxr-xr-x 1 analyst analyst   65 Apr 28 11:27 reg_server_start.sh
-rwxr-xr-x 1 analyst analyst  189 Dec 15  2016 start_ELK.sh
-rwxr-xr-x 1 analyst analyst   85 Dec 22  2016 start_miniedit.sh
-rwxr-xr-x 1 analyst analyst   76 Jun 22 11:38 start_pox.sh
-rwxr-xr-x 1 analyst analyst  106 Jun 27 09:47 start_snort.sh
-rwxr-xr-x 1 analyst analyst   61 May  4 11:45 start_tftpd.sh
```

Consider the **cyops.mn** file as an example. Who is the owner of the file? How about the group?

Owner: analyst; Group: analyst.

The permission for **cyops.mn** are **–rw-r--r--**. What does that mean?

Here -rw-r—r— not displayed in cyops.mn. From the above content.

- The owner of the file can read and write but it will not execute it (-rw).
- Members of the analyst group other than the owner can only read the file(-r-) writing is not allowed.

c. The *touch* command is very simple and useful. It allows for the quick creation of an empty text file. Use the command below to create an empty file in the **/mnt** directory:

```
[analyst@secOps scripts]$ touch /mnt/myNewFile.txt
touch: cannot touch '/mnt/myNewFile.txt': Permission denied
```

Was the file created? List the permissions, ownership and content of the **/mnt** directory and explain what happened. Record the answer in the lines below.

```
[analyst@secOps ~]$ ls -l /mnt
total 4
drwxr-xr-x 2 root root 4096 Mar 3 11:13 second_drive
```

With the addition of *-d* option, it lists the permission of the parent directory.

```
[analyst@secOps ~]$ ls -ld /mnt
drwxr-xr-x 3 root root 4096 Mar  3 15:43 /mnt
```

The permissions of /mnt directory is owned by the root user,with all the permissons drwxr-xr-x.it is the way only that root user is allowed to right /mnt folder.

What can be done for the *touch* command shown above to be successful?

The touch command can be excuted with permission of /mnt directory can be modified.

d. The *chmod* command is used to change the permissions of a file or directory. As before, mount the **/dev/sdb1** partition on the **/home/analyst/second_drive** directory created earlier in this lab:

```
[analyst@secOps scripts]$ sudo mount /dev/sdb1 ~/second_drive/
```

e. Change to the **second_drive** directory and list the contents of it:

```
[analyst@secOps scripts]$ cd ~/second_drive
[analyst@secOps second_drive]$ ls -l
total 20
drwx------ 2 root    root    16384 Mar  3 10:59 lost+found
-rw-r--r-- 1 root    root      183 Mar  3 15:42 myFile.txt
```

What are the permissions of the **myFile.txt** file?

-rw-r-r-

f. Use the **chmod** command to change the permissions of **myFile.txt**.

```
[analyst@secOps second_drive]$ sudo chmod 665 myFile.txt
[analyst@secOps second_drive]$ ls -l
total 20
drwx------ 2 root    root    16384 Mar  3 10:59 lost+found
-rw-rw-r-x 1 root    root      183 Mar  3 15:42 myFile.txt
```

Did the permissions change? What are the permissions of *myFile.txt*?

-rw-rw-r-x

The *chmod* command takes permissions in the octal format. In that way, a breakdown of the *665* is as follows:

6 in octal is 110 in binary. Assuming each position of the permissions of a file can be 1 or 0, 110 means rw- (read=1, write=1 and execute=0).

Therefore, the *chmod 665 myFile.txt* command changes the permissions to:

Owner: rw- (6 or 110 in octal)

Group: rw- (6 or 110 in octal)

Other: r-x (5 or 101 in octal)

What command would change the permissions of *myFile.txt* to *rwxrwxrwx*, granting any user in the system full access to the file?

sudo chmod 777 myFile.txt

g. The *chown* command is used to change ownership of a file or directory. Issue the command below to make the analyst user the owner of the **myFile.txt**:

```
[analyst@secOps second_drive]$ sudo chown analyst myFile.txt
[sudo] password for analyst:
[analyst@secOps second_drive]$ ls -l
total 20
drwx------ 2    root root    16384 Mar  3 10:59 lost+found
-rw-rw-r-x 1 analyst root      183 Mar  3 15:42 myFile.txt
[analyst@secOps second_drive]$
```

Note: To change the owner and group to analyst at the same time, use the *sudo chown analyst:analyst myFile.txt* format.

h. Now that analyst is the file owner, try appending the word 'test' to the end of **myFile.txt**.

```
[analyst@secOps second_drive]$ echo test >> myFile.txt
[analyst@secOps second_drive]$ cat myFile.txt
```

Was the operation successful? Explain.

Yes, the operation is done successfully. The permission allows the owner and user in the analyst group to make changes to the file. Here analyst is the owner of the file and the permissions are still set to 665.

Step 2: Directory and Permissions

Similar to regular files, directories also carry permissions. Directories, however, have an extra bit in the permissions.

a. Change back to the **/home/analyst/lab.support.files** directory and issue the `ls -l` command to list all the files with details:

```
[analyst@secOps second_drive]$ cd ~/lab.support.files/
[analyst@secOps lab.support.files]$ ls -l
total 580
-rw-r--r-- 1 analyst analyst    649 Jun 28 18:34 apache_in_epoch.log
-rw-r--r-- 1 analyst analyst    126 Jun 28 11:13 applicationX_in_epoch.log
drwxr-xr-x 4 analyst analyst   4096 Aug  7 15:29 attack_scripts
-rw-r--r-- 1 analyst analyst    102 Jul 20 09:37 confidential.txt
-rw-r--r-- 1 analyst analyst   2871 Dec 15  2016 cyops.mn
-rw-r--r-- 1 analyst analyst     75 May 24 11:07 elk_services
```

```
-rw-r--r-- 1 analyst analyst    373 Feb 16 16:04 h2_dropbear.banner
-rw-r--r-- 1 analyst analyst    147 Mar 21 15:30 index.html
-rw-r--r-- 1 analyst analyst    255 May  2 13:11 letter_to_grandma.txt
-rw-r--r-- 1 analyst analyst  24464 Feb  7  2017 logstash-tutorial.log
drwxr-xr-x 2 analyst analyst   4096 May 25 13:01 malware
-rwxr-xr-x 1 analyst analyst    172 Jul 25 16:27 mininet_services
drwxr-xr-x 2 analyst analyst   4096 Feb 14  2017 openssl_lab
drwxr-xr-x 2 analyst analyst   4096 Aug  7 15:25 pcaps
drwxr-xr-x 7 analyst analyst   4096 Sep 20  2016 pox
-rw-r--r-- 1 analyst analyst 473363 Feb 16 15:32 sample.img
-rw-r--r-- 1 analyst analyst     65 Feb 16 15:45 sample.img_SHA256.sig
drwxr-xr-x 3 analyst analyst   4096 Jul 18 10:10 scripts
-rw-r--r-- 1 analyst analyst  25553 Feb 13  2017 SQL_Lab.pcap
```

Compare the permissions of the *malware* directory with the *mininet_services* file. What is the difference between their permissions?

It is not displayed in the screen as from the image the starting letter is started with d of the permissions for the malware directory.

The letter ‘d’ indicates that that specific entry is a directory and not a file. Another difference between file and directory permissions is the execution bit. If a file has its execution bit turned on, it means it can be executed by the system. Directories are different than files with the execution bit set (a file with the execution bit set is an executable script or program). A directory with the execution bit set specifies whether a user can enter that directory.

The *chmod* and *chown* commands work for directories in the same way they work for files.

Part 3: Symbolic Links and other Special File Types

You have now seen some of the different file types in *Linux*. The first character in each file listing in an *ls –l* command shows the file type. The three different types of files in *Linux* including their sub-types and characters are:

Regular files (-) including:

- Readable files – text files
- Binary files - programs
- Image files
- Compressed files

Directory files (d)

- Folders

Special Files including:

- **Block files (b)** – Files used to access physical hardware like mount points to access hard drives.
- **Character device files (c)** – Files that provide a serial stream of input and output. tty terminals are examples of this type of file.
- **Pipe files (p)** – A file used to pass information where the first bytes in are the first bytes. This is also known as FIFO (first in first out).
- **Symbolic Link files (l)** – Files used to link to other files or directories. There are two types: symbolic links and hard links.-
- **Socket files (s)** – These are used to pass information from application to application in order to communicate over a network.

Step 1: Examine file types.

a. Change to the **/home/analyst** directory and create a space.txt file. Use the `ls -l` command to display the files. Notice the first characters of each line are either a "–" indicating a file or a "d" indicating a directory

```
[analyst@secOps lab.support.files]$ cd ~
[analyst@secOps ~]$ touch space.txt
[analyst@secOps ~]$ ls -l
total 28
drwxr-xr-x 2 analyst analyst 4096 Sep 26  2014 Desktop
drwx------ 3 analyst analyst 4096 Jul 14 11:28 Downloads
drwxr-xr-x 8 analyst analyst 4096 Jul 25 16:27 lab.support.files
drwxr-xr-x 3 analyst analyst 4096 Mar  3 18:23 second_drive
-rw-r--r-- 1 analyst analyst    0 Aug 16 13:38 space.txt
```

b. Produce a listing of the **/dev** directory. Scroll to the middle of the output and notice how the block files begin with a "b", the character device files begin with a "c" and the symbolic link files begin with an "l":

```
[analyst@secOps ~]$ ls -l /dev/
<output omitted>
crw-rw-rw-  1 root tty       5,   2 May 29 18:32 ptmx
drwxr-xr-x  2 root root           0 May 23 06:40 pts
crw-rw-rw-  1 root root      1,   8 May 23 06:41 random crw-
rw-r--  1 root root     10,  56 May 23 06:41 rfkill
lrwxrwxrwx  1 root root           4 May 23 06:41 rtc -> rtc0
crw-rw----  1 root audio   253,   0 May 23 06:41 rtc0
brw-rw----  1 root disk      8,   0 May 23 06:41 sda
brw-rw----  1 root disk      8,   1 May 23 06:41 sda1
brw-rw----  1 root disk      8,  16 May 23 06:41 sdb
brw-rw----  1 root disk      8,  17 May 23 06:41 sdb1
drwxrwxrwt  2 root root          40 May 28 13:47 shm
crw-------  1 root root     10, 231 May 23 06:41 snapshot
drwxr-xr-x  2 root root          80 May 23 06:41 snd
brw-rw----+ 1 root optical  11,   0 May 23 06:41 sr0
lrwxrwxrwx  1 root root          15 May 23 06:40 stderr -> /proc/self/fd/2
lrwxrwxrwx  1 root root          15 May 23 06:40 stdin -> /proc/self/fd/0
lrwxrwxrwx  1 root root          15 May 23 06:40 stdout -> /proc/self/fd/1
crw-rw-rw-  1 root tty       5,   0 May 29 17:36 tty
crw--w----  1 root tty       4,   0 May 23 06:41 tty0
<output omitted>
```

c. Symbolic links in *Linux* are like shortcuts in *Windows*. There are two types of links in *Linux*: symbolic links and hard links. The difference between symbolic links and a hard links is that a symbolic link file points to the name of another file and a hard link file points to the contents of another file. Create two files by using echo:

```
[analyst@secOps ~]$ echo "symbolic" > file1.txt
[analyst@secOps ~]$ cat file1.txt
symbolic
[analyst@secOps ~]$ echo "hard" > file2.txt
[analyst@secOps ~]$ cat file2.txt
hard
```

d. Use `ln -s` to create a symbolic link to **file1.txt**, and `ln` to create a hard link to **file2.txt**:

```
[analyst@secOps ~]$ ln -s file1.txt file1symbolic
[analyst@secOps ~]$ ln file2.txt file2hard
```

e. Use the `ls -l` command and examine the directory listing:

```
[analyst@secOps ~]$ ls -l
total 40
drwxr-xr-x 2 analyst analyst 4096 Sep 26  2014 Desktop
drwx------ 3 analyst analyst 4096 Jul 14 11:28 Downloads
lrwxrwxrwx 1 analyst analyst    9 Aug 17 16:43 file1symbolic -> file1.txt
-rw-r--r-- 1 analyst analyst    9 Aug 17 16:41 file1.txt
-rw-r--r-- 2 analyst analyst    5 Aug 17 16:42 file2hard
-rw-r--r-- 2 analyst analyst    5 Aug 17 16:42 file2.txt
drwxr-xr-x 8 analyst analyst 4096 Jul 25 16:27 lab.support.files
drwxr-xr-x 3 analyst analyst 4096 Mar  3 18:23 second_drive
-rw-r--r-- 1 analyst analyst  254 Aug 16 13:38 space.txt
```

Notice how the file *file1symbolic* is a symbolic link with an **l** at the beginning of the line and a pointer -> to *file1.txt*. The *file2hard* appears to be a regular file, because in fact it is a regular file that happens to point to the same inode on the hard disk drive as *file2.txt*. In other words, *file2hard* points to the same attributes and disk block location as *file2.txt*.

f. Change the names of the original files: **file1.txt** and **file2.txt**, and notice how it affects the linked files.

```
[analyst@secOps ~]$ mv file1.txt file1new.txt
[analyst@secOps ~]$ mv file2.txt file2new.txt

[analyst@secOps ~]$ cat file1symbolic
cat: file1symbolic: no such file or directory

[analyst@secOps ~]$ cat file2hard
hard
```

Notice how *file1symbolic* is now a broken symbolic link because the name of the file that it pointed to *file1.txt* has changed, but the hard link file *file2hard* still works correctly because it points to the inode of *file2.txt* and not its name which is now *file2new.txt*.

What do you think would happen to *file2hard* if you opened a text editor and changed the text in *file2new.txt*?

Changing the content from one file will be change the contents in the other file because both points to the inode of the HDD.

Reflection

File permissions and ownership are two of the most important aspects of Linux. They are also a common cause of problems. A file that has the wrong permissions or ownership set will not be available to the programs that need to access it. In this scenario, the program will usually break and errors will be encountered.