Linux Lab 4 - Commands, Help, Paths and Executables

Reading

Read Chapter 5 in “The Linux Command Line,” pp 39 - 47 in the printed version or pp 42 - 53 in the pdf. Feel free to run any of the examples in a terminal on your Ubuntu/CentOS VM.

Part 1 - Getting Assistance

When you type a command at the terminal, like `cd` or `gedit`, the code that you are executing can come from several places; these are listed at the top of page 40 in the reading. Commands may act differently and have differing amounts of assistance, depending on where they come from.

Shell built-ins

Our VM uses BASH (/bin/bash) as its default shell for logins. Another common shell for users who access a variety of distributions and don’t always know what distribution they will be on (attackers and penetration testers) is the Bourne shell, or sh (/bin/sh.) The Bourne shell (and variants) is small shell that strictly follows the POSIX standard and works pretty much the same everywhere. You can determine the shell you are using with the command

```
echo $SHELL
```

Here is some practice in accessing help for shell built-ins.

1) One of the commands we have been using frequently is cd. Use the `type` command to see what kind of command it is.
   ```
   type cd
   ```
2) Assistance for shell built-ins is usually found by typing `help [command name]`. Try that now on a command that you know is a shell built-in.
3) Shell built-ins may not support the help options `-h` or `--help`. You may just get an invalid option message and usage instructions, which are the most abbreviated kind of assistance. Try typing `[command name] -h` and `[command name] --help` for a command you know is a shell built-in.
4) Shell built-ins may not have manual pages. Try running `man [command name]` on a command you know is a shell built-in.

Executable programs

This category includes compiled binary programs, as well as scripts written in Python, Perl, BASH, etc. Two things they have in common are that their code is kept in a file on the file system, and that file is marked with the executable permission. The amount of assistance available varies greatly depending on who wrote the code. Formally published code generally has several varieties of help; a script you wrote last year and neglected to document may have no help at all.

5) Use the `type` command to verify that `cp` and `gedit` are files. Then use `ls -l /bin/cp` and `ls -l /usr/bin/gedit` to examine the executable bits in their file permissions. Besides the x in the rwx line, your terminal will use green to signify that `cp` and `gedit` are executable.
6) Try the assistance options `-h` or `--help` for `cp` or `gedit` as well as the `man` command. You should find extensive assistance.

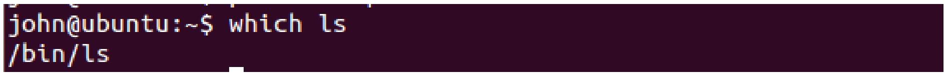
Alias

An alias is a string that often calls an existing command with convenient options already selected, although it may be group of commands as well. An alias can reduce our typing load, but it can also be used by an attacker to fool us into executing something we do not want. It is good to know what aliases are in your environment. The assistance you get with an alias will be the same as that available with the application the alias calls. While we are here though, let's look a little deeper.

7) When we use `ls` for listing directories, we're running a command, aren't we? See if that is true by running
   `type ls`
   It turns out that we've been executing an alias all along. The `which` command will give you the alias information as well as the absolute path to the binary file for `ls`. Run `ls -la` on your current directory or a directory that has sub-directories or executables, so you see the color output. Then run `ls -la` using the absolute path (remember the absolute path starts with "/"). You should be able to tell the difference.
8) The command `alias` will show you what aliases are active in your environment. Try it now. You should see some handy aliases for ls. One will list hidden files (almost all files) and the other will give you a long listing of all files (the exact results will differ between CentOS and Ubuntu.) Remember them, as they will save typing. If you are on another distribution that does not have those aliases, you can easily create them. For example, Fedora has a nice alias to list only hidden files, `l.` You can make your own hidden files alias in Ubuntu by entering the command
   `alias l.='ls -d .* '`.

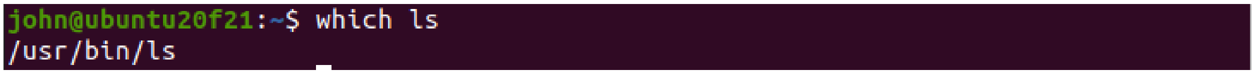
Part 2 - Paths, or where is the executable?

We have already seen that there may be more than one way to execute code (`ls` the alias, and `/bin/ls` the application), and there may be several copies of an application in our file system. The `type` command is helpful, and it has a cousin called which. The `which` command gives us the absolute path to the file that would be executed if we type a command, assuming there is no alias for that command. For instance, in Ubuntu 18.04 ls is found in /bin/ls.



```
john@ubuntu:~$ which ls
/bin/ls
```

In Ubuntu 20.04 it is found in /usr/bin.

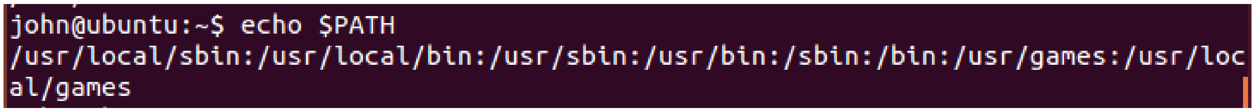


```
john@ubuntu20f21:~$ which ls
/usr/bin/ls
```

Suppose there are several copies of ls in several different directories? Which one runs? The answer is in the PATH environment variable. To see the contents of PATH, execute

`echo $PATH`

The "$" tells BASH that we want the contents of the PATH variable; otherwise it will just repeat whatever we typed.



```
john@ubuntu:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games
```

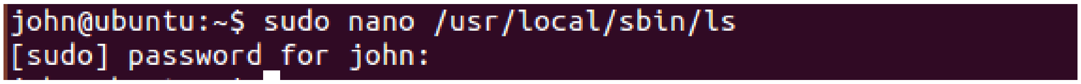
When you type “ls” the shell checks each directory in the path, in order, looking for a file named “ls.” It executes the first one it finds.

9) Look for a file called “ls” in the path directories. First try the command
   `ls /usr/local/sbin/ls`
   Nothing…so ls is not in the first directory of the path. Now try each directory in the path until you find ls.
10) Now, let’s have some fun. Use `sudo nano` (~~or `sudo gedit` if you like~~) to create a file in `/usr/local/bin` with a filename “ls” and content like this:

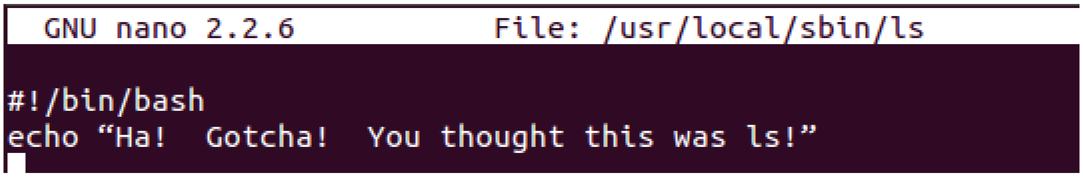
```
#!/bin/bash
echo 'Ha!  Gotcha!  You thought this was ls!'
```

Note: Use single quotes, not double quotes.

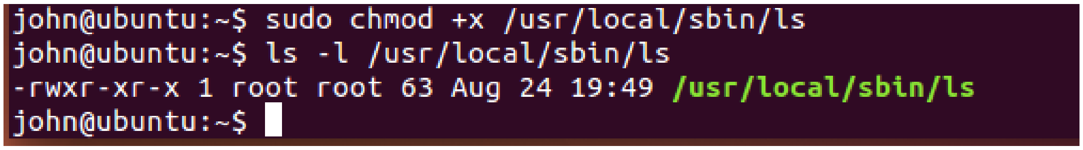
Then make the file executable with
`sudo chmod +x /usr/local/sbin/ls`



```
john@ubuntu:~$ sudo nano /usr/local/sbin/ls
[sudo] password for john:
```



```
  GNU nano 2.2.6              File: /usr/local/sbin/ls

#!/bin/bash
echo “Ha!  Gotcha!  You thought this was ls!”
```



```
john@ubuntu:~$ sudo chmod +x /usr/local/sbin/ls
john@ubuntu:~$ ls -l /usr/local/sbin/ls
-rwxr-xr-x 1 root root 63 Aug 24 19:49 /usr/local/sbin/ls
john@ubuntu:~$
```

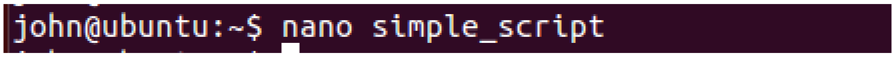
<u>Then exit your terminal, start a new terminal and try to use ls. What happens and why? Fix the problem by deleting the bogus ls file you created, or else ls will not work any more.</u>

11) Once you are done, fix your VM by using:
    `sudo rm /usr/local/sbin/ls`

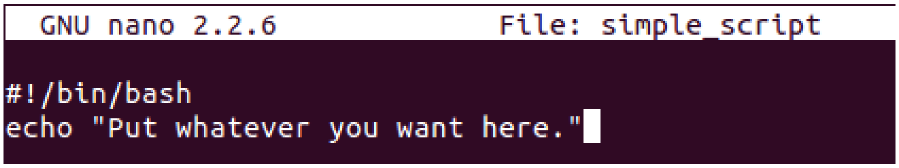
Part 3 - Executing from the current directory

If you examine your environment variable, PATH (`echo $PATH`), you will see that it does not include the “.” character, which signifies the current directory. If it did, an attacker could do the same attack we demonstrated in step 10 without needing to use sudo and your password; they could put a fake “ls” in your home directory, for example. However, you will often wish to execute scripts you’ve written from the same directory that holds them. Here is how to do it.

12) Change directories to your home directory if you aren’t there already. Create a simple script using nano or gedit (no sudo needed this time.)

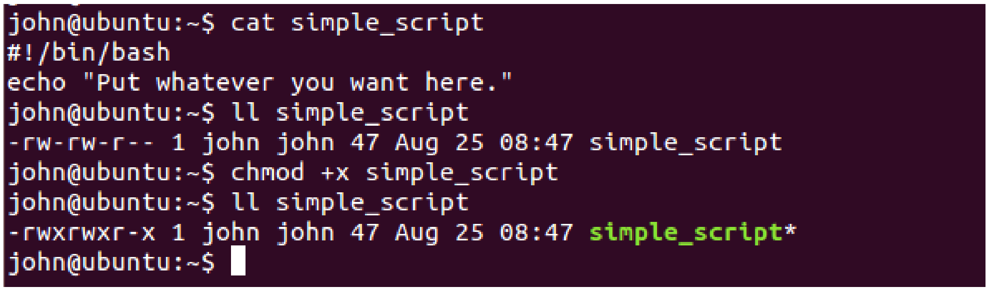


```
john@ubuntu:~$ nano simple_script
```



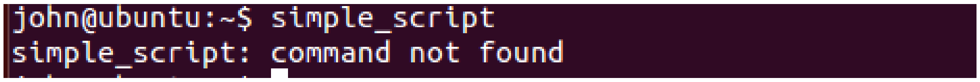
```
  GNU nano 2.2.6                File: simple_script

#!/bin/bash
echo "Put whatever you want here."
```

13) Here I used `cat` to verify that my script was correct and used `chmod` to make it executable.



```
john@ubuntu:~$ cat simple_script
#!/bin/bash
echo "Put whatever you want here."
john@ubuntu:~$ ll simple_script
-rw-rw-r-- 1 john john 47 Aug 25 08:47 simple_script
john@ubuntu:~$ chmod +x simple_script
john@ubuntu:~$ ll simple_script
-rwxrwxr-x 1 john john 47 Aug 25 08:47 simple_script*
john@ubuntu:~$ 
```

14) Try to execute the script by typing its name. It should fail. You may notice that even tab-complete does not work.



```
john@ubuntu:~$ simple_script
simple_script: command not found
```

15) The reason it fails is that the current directory is not in the PATH variable. It will execute if you explicitly specify the path to the script, however. Any of the following should work--try them.

```
./simple_script                         (this is a relative path)
/home/[your username]/simple_script     (this is an absolute path)
~/simple_script                         (~ is a shortcut for your
home directory)
```

Obviously, "./" is easier so most people use that.

Hand in

1) In step 10, why did your script run instead of the `ls` executable?
2) What are the two steps you need to perform when you want to execute a script you've written from your current directory?