CITS 3004
Cybersecurity

THE UNIVERSITY OF WESTERN AUSTRALIA

Lab 1: Linux

Task 1: Lab setup

Written by Jin Hong

A) RUNNING THE VM ON YOUR OWN LAPTOP (PREFERRED METHOD)

This is the simplest and easiest setup to do labs for CITS3004, due to its portability and speed (given your laptop has sufficient computing power). Steps are as follows:

1. Install VirtualBox from their main site (last tested version: 6.0.18) - https://www.virtualbox.org/
2. Download the SeedUbuntu Image from here (16.04!) - https://seedsecuritylabs.org/lab_env.html
3. From the same link in step (2) above, there is a manual to install the VM, follow those instructions.
4. Done!

B) RUNNING THE VM USING AZURE

To begin our labs, we first need to setup the environment which will be used for the labs in the coming weeks. We will be using the Ubuntu image (provided) as our main environment, which has been developed by the SEED Labs, Syracuse University [1]. We will cover some of the lab materials from the SEED lab as well. By now, you should be given an access to the Azure cloud, which will connect you to a Windows 10 VM. The interface should look as in Figure 1. This means you can access your VM from **anywhere on any computer**!

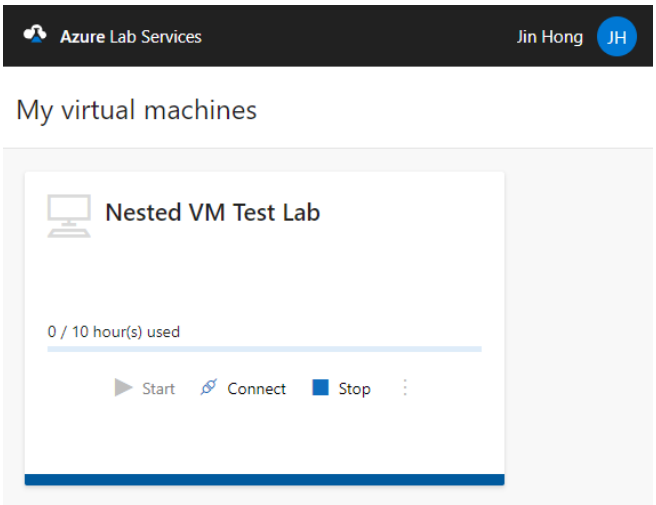

Figure 1: Azure Lab Services: Access to the VM page

Press "Start" to boot your VM. Once the VM is booted, you should be able to "Connect", which will give you a remote access to the VM. This VM is unique to you (i.e. you can customise it as you want, you have full control). Once you access the VM via remote access, you can launch Hyper-V and mount the SEEDUbuntu image.

NOTE: you have **LIMITED** hours you can access the VM. Please make sure to "**STOP**" the VM once you finished using it, otherwise your quota will be used up. If you find the given quota insufficient to complete your labs, please contact Jin to discuss increasing your quota.

Follow these steps to setup the SEEDUbuntu VM on your Host VM.

1. You do this by creating a new virtual machine.
2. Make sure to select Generation 1 in "Specify Generation" step.
3. You should at least allocate 1024MB of memory, and you can push this up to 2048MB or higher (Your VM has 4 cores and 8MB ram).
4. Select "Default Switch" for networking.
5. Select "Use an existing virtual disk" and select SEEDUbuntu image.

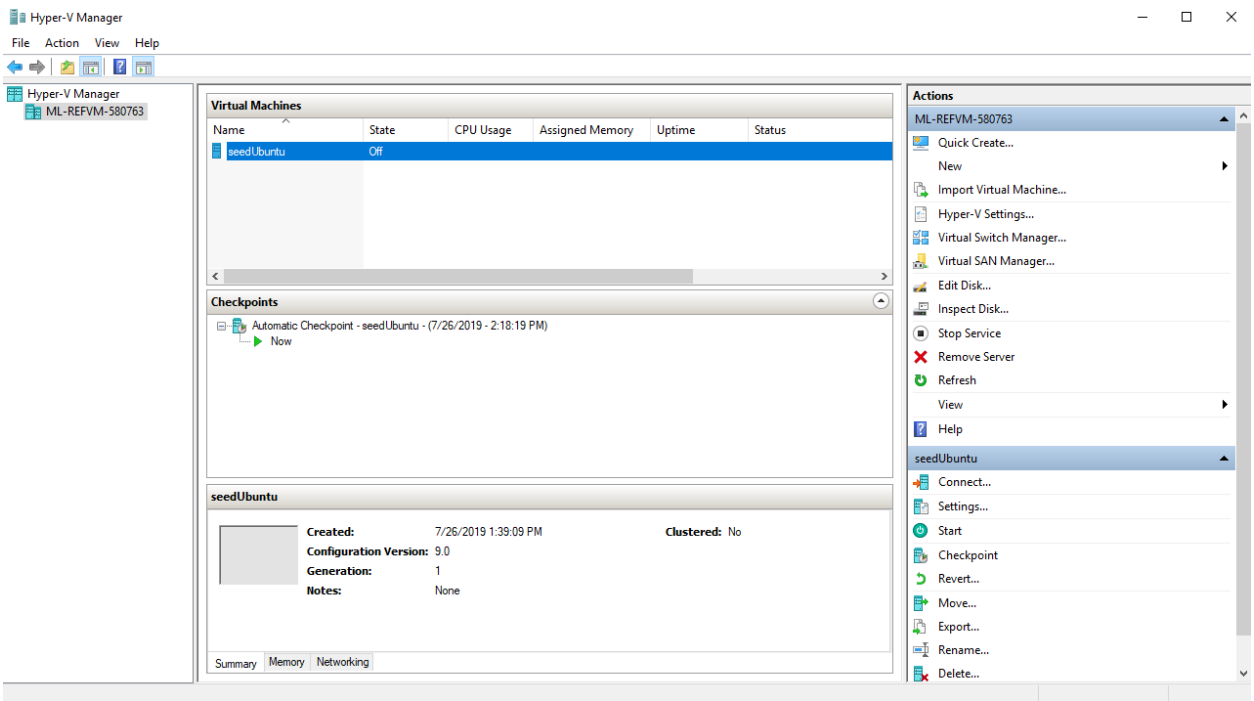

Figure 2: Launching the VMWare

Figure 2 shows the Hyper-V state after you successfully setup the SEEDUbuntu VM. You can follow these steps to add more VMs in Hyper-V later on. To setup a shared folder to move files, follow instructions on [10].

Figure 3: SEEDUbuntu start page

IMPORTANT: Once you finish the lab, make sure that you always **TURN OFF** the VM by clicking the menu (top right hand corner) -> Shut Down -> Shut Down. This ensures that you can boot the VM on any other host computers in the lab. If you ever break the VM image, you can always setup a new VM using the SEEDUbuntu image. Unless you saved your file on the local host, your files may be lost **permanently**.

You can also setup the environment on your own computer to do the labs. If you would like to setup using the VirtualBox, you can follow the instructions given in [4].

C) USER ACCOUNTS

Two accounts are given by default:

User ID	Password	Comment
seed	dees	Already given a root privilege, but need to use sudo command to use it
root	seedubuntu	Root account, but you have to login as a normal user first and use su to login to the root account.

CITS 3004
Cybersecurity

Task 2: Linux Lab

Written by Jin Hong using online materials

The idea of Linux is simple-To take the idea of UNIX, which is to have a collection of programs and modules that work cohesively together-and make a free, open and community driven operating system that can be edited and customised to your heart's content.

Please note, you get **3 attempts per lab**!

A) DISTRIBUTIONS

The base system of Linux comes in many different distributions which contain different packages and features written by different groups. These are referred to as distros for short, and they have a wide variety of different uses, purposes, system, features, and fan-bases. This guide will attempt to be distro-independent, however, a few of the more popular distros are listed:

- Ubuntu
- Debian
- Fedora
- Linux Mint
- Red Hat
- CentOS
- Arch Linux
- Gentoo

B) PACKAGE MANAGER AND REPOSITORIES

Each distribution comes with a package manager, which handles software installed on the system and has a number of remote repositories from which it gets its software. For example, Ubuntu uses a package manager called aptitude (apt for short). You can type:

```
sudo apt-get install firefox
```

to install the Mozilla Firefox web browser. It will search the remote repositories (listed in /etc/apt/-sources.list) for the required packages and instructions to install them and once found, it will install the software on your computer for you. The package manager can also update and remove software, and manage your local package database. This is one of the brilliant things of a package manager: you can run a single command and you've installed new software - You can run a single command, and update all your packages, etc.

Note: apt-get is the package manager for Ubuntu. Fedora uses `dnf` (previously `yum`), Arch uses `pacman`, and Debian uses `aptitude`, etc. For instance:

```
yum install firefox
```

will install Firefox on a Fedora machine. You can also install multiple package managers on any one distro, but as they say*: too many cooks spoil the broth*.

Answer Q1 ~ Q4 on LMS

CITS 3004
Cybersecurity

C) THE TERMINAL

C1) Commands

Firstly you should get familiar with the man pages, which are essentially the manual, and will display help pages on almost all commands.

```
Usage: man COMMAND
```

Where COMMAND is replaced with whatever command you want help on. Press 'q' to exit a manual page. Alternatively, most commands will allow you to add --help on the end to get their own personal help pages. Here is a list of the essential commands that you should become familiar with (and you can use the manual to learn how to use them):

```
cd ------ # Change directory
ls ------ # Show contents of directory
echo ---- # Print text to the screen
cat ----- # Display contents of a file
nano ---- # Edit a file via the command line
mv ------ # Move (or rename) a file
cp ------ # Copy a file
rm ------ # Remove (delete) a file
```

These are some extra commands which aren't totally essential but are certainly helpful:

```
mkdir --- # Make a new directory
rmdir---- # Delete a directory
grep ---- # Search for specific text within text
pwd ----- # Print working (current) directory
whoami -- # Display user
ps ------ # Display running processes
pstree -- # Display a tree showing running processes and processes
they started
top ----- # Display most intensive running programs
who ----- # Display logged on users
w ------- # Display logged on users
which --- # Display the path to a command's binary
df ------ # Disk space free
du ------ # Disk space used
passwd -- # Change user password (not to be confused with pwd)
more ---- # Display text one screenful at a time
less ---- # Display text one screenful at a time
wc ------ # Word/letter/byte count
id ------ # Display the uid, gid and groups of a user
su ------ # Switch user
tty ----- # Display which tty you are on
```

CITS 3004
Cybersecurity

C2) Shortcuts

Every key pressed ends a character to the terminal, and you can send different characters by holding down keys like [Ctrl] or [Alt]. This is how the shell can tell what key is pressed, and thus, allow shortcuts to be defined. Some of the more useful keyboard shortcuts are defined:

- Up arrow or Down arrow

  Scroll through typed commands
- Home or End

  Move to the start or end of a line, respectively
- Tab

  Autocomplete a file name, directory name or command name.
- Ctrl + C

  End a running process
- Ctrl + D

  End an End-Of-File (EOF) character (usually ends a process or signifies the end of input data)
- Ctrl + Z

  Send the currently running process to the background
- Ctrl + L

  Clear the screen, same as running the clear command

C3) Piping and redirection

There are a number of little quirks that the shell has that gives it more functionality. Piping takes the stdout of the left program and connects it (i.e. *pipes* it) into stdin of the right program with the pipe operator `|`. For example:

```
# Count number of words in helloworld.txt
cat helloworld.txt | wc -w
```

Redirection directs data in and out of files, i.e.

```
# Redirect stdout to file
echo "Hello world" > helloworld.txt
# Redirect stdout to the end of a file
echo "world." >> hello.txt
# Redirect a file to stdin
more < helloworld.txt
```

CITS 3004
Cybersecurity

C4) Wildcards

The shell uses a number of special characters called wildcards, similar to regular expressions or regex, which can be used to manipulate what is being dealt with on the command line. The standard wildcards are thus:

`*` Match 0 or more characters. For example, `rm *.txt` will delete all files that end in `.txt`, and `cp somedirectory/* .` will copy all files from `somedirectory' to the current directory.

`?` Match any single character. For example, `cp example.? somedir` will copy all files named `example' with a single character extension, into the directory `somedir'

`[]` Match any single character in the square brackets. You can even specify a range, i.e. `rm m[a-e]m` will delete any files starting and ending with `m`, and with any letter between `a' and `e' in between. `rm m[abc]m` will delete files `mam', `mbm', `mcm'.

`{}` Match any item in the braces. For example, `cp {*.doc,*.pdf} ~` copies any files with the extension `.doc' or `.pdf' to the home directory.

C5) Conditional execution

You can chain commands together on one line by separating them with a semicolon `;'.

```
cmd1; cmd2; cmd3
```

However, every program returns a number back to the OS once it has finished running to tell if it completed successfully or not, and we can use this to chain execute commands conditionally.

To run a command if and only if the last command completed successfully, we use &&:

```
user@MY-PC:~$ mkdir foo && cd foo && echo "hooray" > somefile
user@MY-PC:~/foo$ cat somefile
hooray
user@MY-PC:~/foo$
```

To run a command if and only if the last command failed, we use ||:

```
user@MY-PC:~$ ls foo || cd foo || mkdir foo && ls -ld foo
ls: cannot access foo: No such file or directory
bash: cd: foo: No such file or directory
drwxrwxr-x 2 user user 4096 May 24 00:57 foo
```

CITS 3004
Cybersecurity

THE UNIVERSITY OF WESTERN AUSTRALIA

C6) Processes

Every program that runs, runs in virtual memory as a process, even the shell. You can list the currently running processes with the command top. When you run a command, the terminal session runs it on its process, waits for it to complete, then regains control once the command is finished. So, if you were to close the terminal window while a command was running, that would stop the command. Since this can be inconvenient, we can `fork' the command into its own process to run in the background, and still use the shell while it runs (which is useful for commands that take a long time). To do this, we end the command with a single ampersand &. For example:

```
user@MY-PC:~$ (sleep 15; date) & date
[1] 12186
Thu May 24 02:06:14 AWST 2018
user@MY-PC:~$
user@MY-PC:~$ Thu May 24 02:06:29 AWST 2018

[1]+ Done                    ( sleep 15; date )
user@MY-PC:~$
```

`(sleep 15; date)` is sent to the background and returns the process ID (PID), then the next date is run and the shell is returned. After sleeping for 15 seconds, the date sent to the background outputs and the shell reports that the command completed.

There is a command, `nohup` (no hangup), which prevents a program from being forcefully terminated under normal circumstances. We can combine this with & to run programs that need to run uninterrupted for long periods of time.

Another way to list the processes running is with `ps`, and then end them with `pkill` or `killall` (kill by process name), or with `pkill` (kill by process ID), or even with the keyboard shortcut [Ctrl] + [C] as mentioned above. Check the man page, as well as [8] for more options.

You can also check what is running in the background and foreground with `bg` and `fg`, respectively.

Answer Q5 and Q6 on LMS

D) FILE SYSTEM STRUCTURE

The file system is structured as a tree that flows down from the root directory, which is simply represented as /. Below shows an example listing of a system's root directory using the ls command:

```
user@MY-PC:~$ ls /
bin    etc         initrd.img.old  lost+found   proc   selinux   usr
boot   fixdm       lib             media        root   srv       var
cdrom  home        lib32           mnt          run    sys       vmlinuz
dev    initrd.img  lib64           opt          sbin   tmp       vmlinuz.old
```

CITS 3004
Cybersecurity

The standard path is listed as all the directories to a file, separated by the / character. You can also use `..` to represent the folder that the current folder is in, `.` to represent the current directory, and ~ to represent your home directory. For example

```
# Move two folders up, then into dir1 and then dir2, then back into
dir1, then back into dir2
cd ../../dir1/dir2/../dir2
# Get a listing of the current directory
ls .
# Change into your home directory
cd ~
```

Linux will automatically complete a command or filename if you are part-way through typing it; all you have to do is hit the [Tab] key. Press [Tab] enough times and it will list possible suggestions based on what you currently have typed in the terminal.

D1) File operations

There are a number of useful programs that allow us to do file manipulation. To list some of the main operations:

cp Copy a file from one location to the other.

mv Move a file from one location to the other. Note, this is also used to rename files{you just `move' the file to the directory it is already in but as a new name, for example `mv foo bar` would rename the file from `foo' to `bar', assuming you didn't have a directory named `bar', in which case, the file would be moved to that directory instead.

rm Delete a file. Note that you can also delete empty directories this way, and you can delete a directory and its subdirectories by using `rm -r`. However, be **VERY** careful: if you were to run `rm -rf /`, you would erase every file on your whole computer, because it would delete the root directory and then every file and subdirectory below it and it wouldn't stop because the `f' in `-rf' means `force'. Use `rm -rf` with extreme caution, or even use `rmdir`, which removes a directory.

grep `grep` stands for Global Regular Expressions Parser, and can search through text for a match. For example, `grep foo bar` searches the file bar for the string foo, and you can also use `ls -l | grep "foo"`, which searches the file listing for a file called foo. When combined with `sed` and `awk`, you can do almost anything string related.

D2) $PATH and the environment

Variables in the shell are defined using the export command, and when variables are used, they start with a `$`.

```
user@MY-PC:~$ export FOO="This is a string"
user@MY-PC:~$ echo $FOO
This is a string
```

You can see a list of the set environment variables by typing set by itself into the terminal.

Linux uses a global terminal variable to find programs. This is the `$PATH` variable and it consists of a list of file paths to search in for a specified program, in order, separated by the colon (:). For example, a listing of my path:

```
user@MY-PC:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

This would mean that if I were to use the command `ls`, it would search for a binary file called 'ls' in /usr/local/sbin, then in /usr/local/bin, and so on until it found it. Then it would be executed. Because these are searched in order, if you were to prepend a directory path to the front of $PATH with your own copy of `ls` in it, and then run the ls command, then your copy would be run instead.

This is sometimes used as an exploit by modifying the user's $PATH variable so that a path containing malicious binaries with the same names as common commands is on the front. When the user goes to run these commands, then the malicious binaries are run instead.

Answer Q5 ~ Q12 on LMS

E) USERS AND PERMISSIONS

Every user has a user ID (uid) and a group ID (gid). Each user also has a list of groups they are a part of which give them the permissions that are assigned to those groups. You can see this by using the 'id' command:

```
user@MY-PC:~$ id
uid=1000(user)gid=1000(user)groups=1000(user),4(adm),24(cdrom)
,27(sudo),29(audio),30(dip),44(video),46(plugdev),109(lpadmin)
,119(pulse),124(sambashare)
```

E1) sudo and root

Now for the most powerful user on Linux: The root user. Root's uid and gid are both 0.

```
user@MY-PC:~$ id root
uid=0(root) gid=0(root) groups=0(root)
```

The root account can do anything, while other accounts would be denied due to the lack of permissions required. Root is the first account created on a newly installed Linux distro, and it is generally encouraged that you do not use the root account unless you absolutely have to because since root can do anything, then there's no stopping you from accidentally deleting something important.

This is where the `sudo` command comes in (a.k.a "super-user do"). You can use `sudo` to execute commands that require elevated privileges without having to actually switch to root.

Say you want to edit the hostname file, which contains the name of your computer, but, by default, you need elevated privileges to edit it. You would type:

```
sudo nano /etc/hostname
```

to which it asks you for your password, and then opens `nano` with the extra privileges provided by `sudo`. There is a `sudoers` file which contains a list of users who can use `sudo`, and what privileges they get from using it.

E2) File permissions

Linux inherits its file permissions system from Unix. You can use the command `ls -l` to display the permissions of a file or files:

```
user@MY-PC:~/junk$ ls -l
total 8
-rw-rw-r-- 1 user user 9 Apr 25 21:28 junk1
drwxrwxr-x 2 user user 4096 Apr 25 21:29 other_junk
```

The first string consists of a sequence of letters, which represent the permissions on the file. The two names refer to the owner and group the file belongs to, respectively.

Lets take the file, `junk1`, as our example. The first character is the file type. This is a `d' if the file is a directory (like `other_junk`). The next part should be read as three sets of permissions

```
-rw-rw-r--
( d )( u )( g )( o )
( - )(rw-)(rw-)(r--)
```

where the first set, `u`, refers to the permissions for the user who owns the file. The next set, `g`, refers to the permissions for the group that the file belongs to. The final set, `o`, refers to the permissions for any other user. Each set uses `rwx' to specify the permission to (r)ead, (w)rite or e(x)ecute, or - if that permission is not set. Let's take a look at the folder `other_junk`.

```
drwxrwxr-x
```

This is a directory, the user has read/write/execute access, users belonging to the group of the file have read/write/execute access, and everyone else has read/execute access, but not write access.

CITS 3004
Cybersecurity

THE UNIVERSITY OF WESTERN AUSTRALIA

E3) Changing permissions

If you want to change a file's permissions, you can use `chmod`, meaning \change mode". There are two ways to do this: using u/g/o and +/- r/w/x:

```
chmod o+x junk1     # Add execute permission to others
chmod og+wx junk1   # Add execute and write permissions to other and group
chmod +x junk1      # Make junk1 executable for the user
chmod g-w junk1     # Remove write permissions from junk1 for group
etc...
```

or with a number that represents permissions, called a bitmask. This will set all the permissions at once for you.

```
user@MY-PC:~/junk$ chmod 755 junk1
user@MY-PC:~/junk$ ls -l
total 8
-rwxr-xr-x 1 user user 9 Apr 25 21:28 junk1
drwxrwxr-x 2 user user 4096 Apr 25 21:29 other_junk
```

The bit mask is three digits (sometimes four digits) between the numbers 0 and 7. Each digit represents what the read/write/execute permissions would be in binary. Take a look:

```
0 = 000 = ---
1 = 001 = --x
2 = 010 = -w-
3 = 011 = -wx
4 = 100 = r--
5 = 101 = r-x
6 = 110 = rw-
7 = 111 = rwx
```

So, as a few examples,

```
777 = rwxrwxrwx
755 = rwxr-xr-x
132 = --x-wx-w-
564 = r-xrw-r--
000 = ---------
etc...
```

So when we set our file junk1 to 755 earlier, we set it to rwxr-xr-x, which is a pretty good permission set on your average file. Realistically, you will usually always have your own user permissions set to rwx or rw-, otherwise you are just inconveniencing yourself. You can also use `chown` to change ownership of a file.

Answer Q13 ~ Q16 on LMS

Extra

You can visit [5] for more UNIX tutorials and resources.

You can also visit [11] for more basic introduction to the Unix Shell.

You can try out some of the basic Linux command quiz at [6].

Further recommended reading [7].

A comprehensive Linux tutorial at [9].

References

1. "User Manual of the Pre-built Ubuntu 16.04 Virtual Machine", Seed Labs, Syracuse University, 2018, url: http://www.cis.syr.edu/~wedu/seed/Documentation/Ubuntu16_04_VM/Ubuntu16_04_VM_Manual.pdf
2. "Introduction to using Linux" Cody Harrington, University of Canterbury 2013
3. "Linux Lab: Exercises" Cody Harrington, University of Canterbury, 2013
4. "Run SEED VM on VirtualBox: User Manual" Seed Labs, Syracuse University, 2018, url: http://www.cis.syr.edu/~wedu/seed/Labs_16.04/Documents/SEEDVM_VirtualBoxManual.pdf
5. "UNIX tutorial for Beginners" M. Stonebank, Surrey University 2001, url: http://www.ee.surrey.ac.uk/Teaching/Unix/
6. Linux commands quiz, url: https://www.sporcle.com/games/sporcilicious/common_linux_commands
7. "The Unix Programming Environment, 2nd edition" Brian W. Kernighan and Rob Pike
8. https://www.lifewire.com/list-and-kill-processes-using-the-pgrep-and-pkill-4065112
9. https://0xax.gitbooks.io/linux-insides/?fbclid=IwAR1UOzoLvB-OKfCpLcxB0JMy-6GBkKVRZnSdgxydoW8jJLgjAX9BiKHKzf8
10. Shared Folder in Hyper V: https://linuxhint.com/shared_folders_hypver-v_ubuntu_guest/
11. The Unix Shell: http://swcarpentry.github.io/shell-novice/