WEEK 4 – TASK T4.1P

Pass Task

Release Date: 29 July, Due Date: 5 August, End Date: 12 August

Learning Outcomes

In this task, you will learn more about Unix/Linux Access Control. You will get hands-on experience of configuring and interpreting permissions. This will complement the theoretical discussion about Access Control in Week 3.

Instructions

An **answer sheet template** is available on OnTrack as a `**Resources**'. Please download the answer sheet and fill it with your answers. To upload on OnTrack, you need to convert the answer sheet template document to **PDF**. MS Word includes built-in PDF conversation capability.

All 8 questions and their sub-questions of this task must be attempted. If screenshots are required, please ensure that text in screenshots is readable.

Remember that troubleshooting technical problems is part of learning in this field. You must patiently work through issues and solve these. Tasks are not step-by-step guide. You need to be in the driver seat and learn concepts by doing – as you would when you start your future job (many times even your future supervisor doesn't know the answer to problems you face). After patent troubleshooting and research, if you need help:

Help is always available in SIT182. Please go to **Discussions** and ask your questions about this task in **Task 4.1P**. All students are encouraged to participate and help peers with their questions. Helping others is a great way to learn and think about aspects you may have overlooked. You can also seek help from tutors during online and face-to-face pracs. Please do not raise your questions through Teams, OnTrack, or Email.

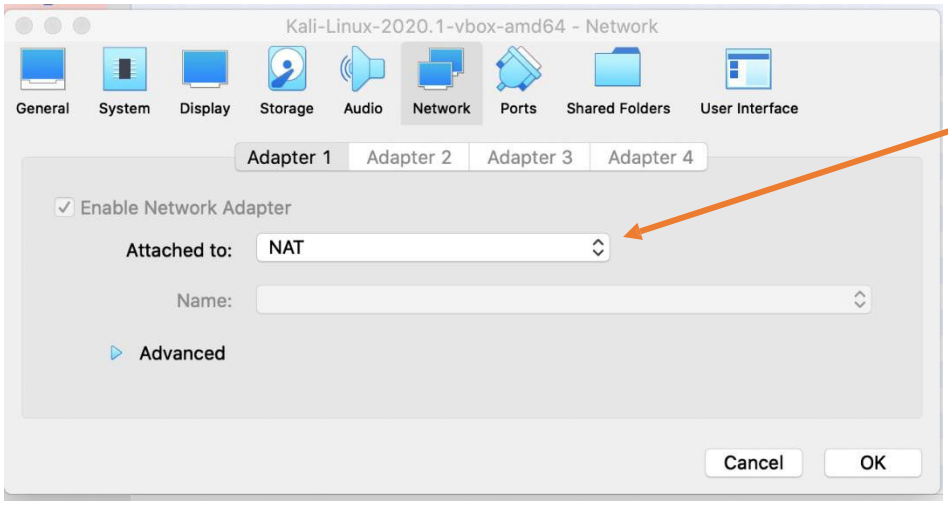
References In cyber security, our preferred referencing style is **IEEE** – however, you are allowed to use any Deakin approved referencing style in this unit. Please refer to unit site > Content > Referencing - Hints & Tips for more information.

This task involves working on a set of challenges. You will also need to answer a set of follow up questions listed in this task sheet to ensure the learning outcomes of the task are met.

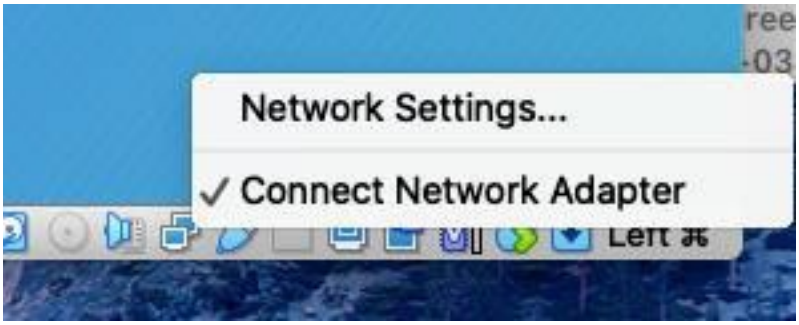
To access challenges that you need to complete open Kali VM from VirtualBox. We will use the same Kali VM that you used for Task 2.1P.

Note: If you are using cyber lab PCs, you will need to import Kali VM into VirtualBox from Drive D > VM >SIT182>Archive folder. Just like you did for Task 2.1P.

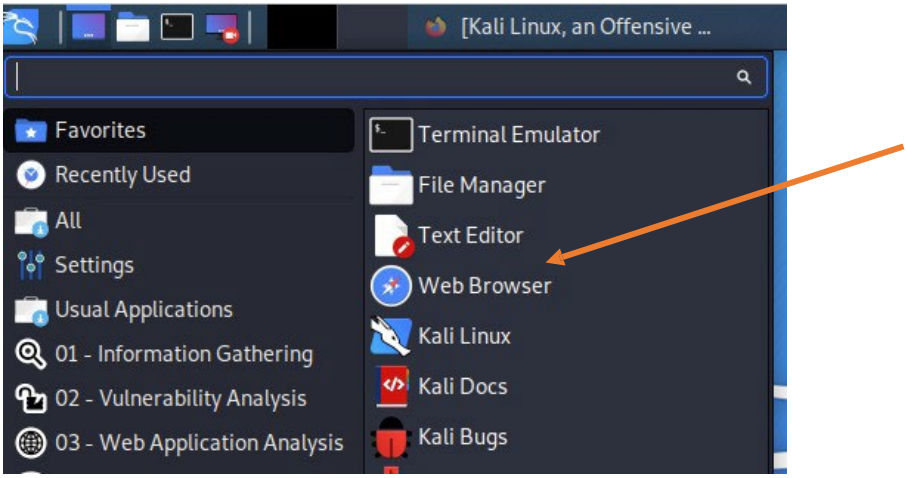
First, ensure that Kali is connected to the Internet. Check the Settings of Kali VM and ensure Network Adapter is connected either through NAT or Bridged Adapter.



Run Kali VM (remember that username and password is **kali**). Ensure Network Adapter is connected using the toolbar at the bottom of VM window.



At this point you should have access to the Internet from Kali. In Kali, access Web Browser.



Notice

Please note that the Kali VM provided in Task 2.1P is Docker-enabled. Unless you are using your own version of Kali, you will need to install Docker CE from here before starting the task.

To get started, you need to download a ZIP file in your Kali VM. You can use any of the links below:

1. https://drive.google.com/file/d/18_o7qIZ0H7Y93-adOLxlh9IMi_uBATxN/view?usp=sharing

2. Access_Control.zip

Copy and Unzip/Extract the downloaded ZIP file to the Desktop in Kali (Kali-Linux-2020.1-vbox-amd64). Then, open the README.txt file in the "Access Control" folder that is extracted. Follow the instructions provided and then start working on the challenges. Remember that all challenges will need to be executed in your Kali VM (not your host OS).

Note: If you get a docker daemon error while building the image using the commands in README.txt file, use the below command which you need to run in a separate terminal and keep it running.

Open one Terminal and run the below command.

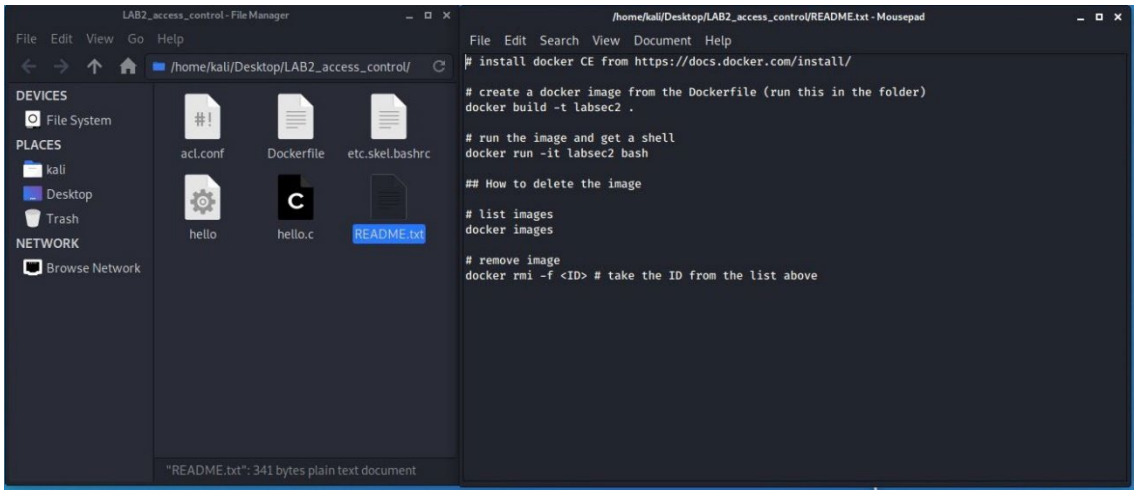
sudo dockerd

```
kali@kali:~$ sudo dockerd
[sudo] password for kali:
INFO[2024-04-18T00:27:35.377308752-04:00] Starting up
INFO[2024-04-18T00:27:35.381305503-04:00] libcontainerd: started new containerd process  pid=12
64
INFO[2024-04-18T00:27:35.381814251-04:00] parsed scheme: "unix"                           module=
grpc
INFO[2024-04-18T00:27:35.381830011-04:00] scheme "unix" not registered, fallback to default sch
eme  module=grpc
INFO[2024-04-18T00:27:35.381855004-04:00] ccResolverWrapper: sending update to cc: {[{unix:///v
ar/run/docker/containerd/containerd.sock 0  <nil>}] <nil>}  module=grpc
INFO[2024-04-18T00:27:35.381868572-04:00] ClientConn switching balancer to "pick_first"  module
=grpc
INFO[2024-04-18T00:27:35.553222766-04:00] starting containerd                             revisio
```

The important README files of Unix

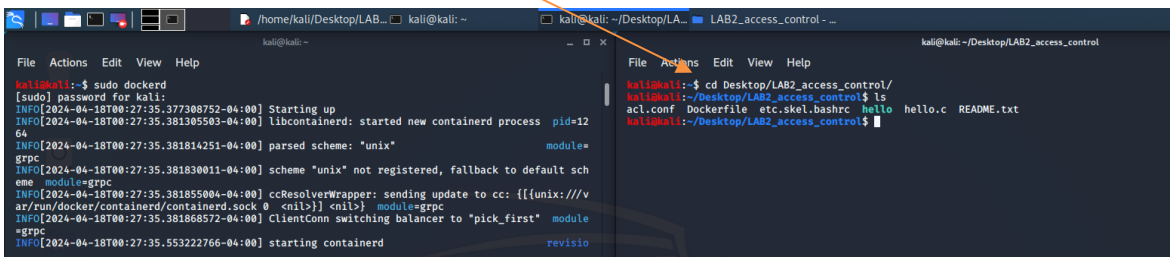
A README file contains information about other files in a directory or archive of computer software. A form of documentation, it is usually a simple plain text file called Read Me, READ.ME, README.TXT, or README.md (for a text file using markdown markup). The file's name is generally written in uppercase letters. On Unix-like systems in particular this makes it easily noticed – both because lowercase filenames are more common, and because traditionally the ls command sorts and displays files in ASCII-code order, so that uppercase filenames appear first.

The README.txt file is shown in the below figure.



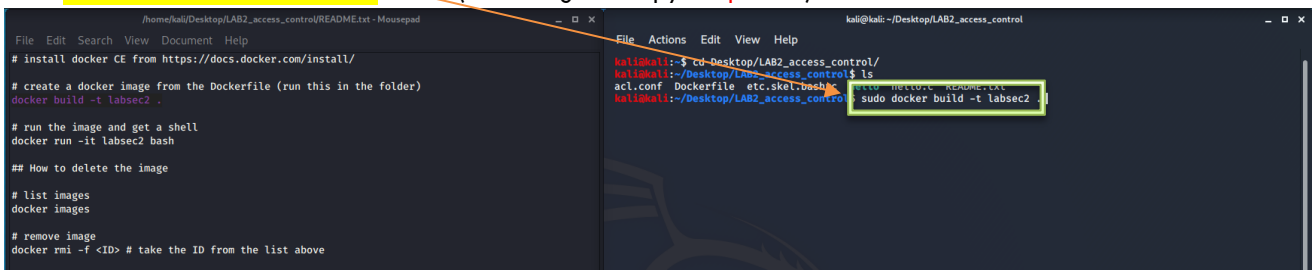
Open another terminal. We need to change the working directory to the unzipped file at the Desktop.

Cd Desktop/LAB2_access_control

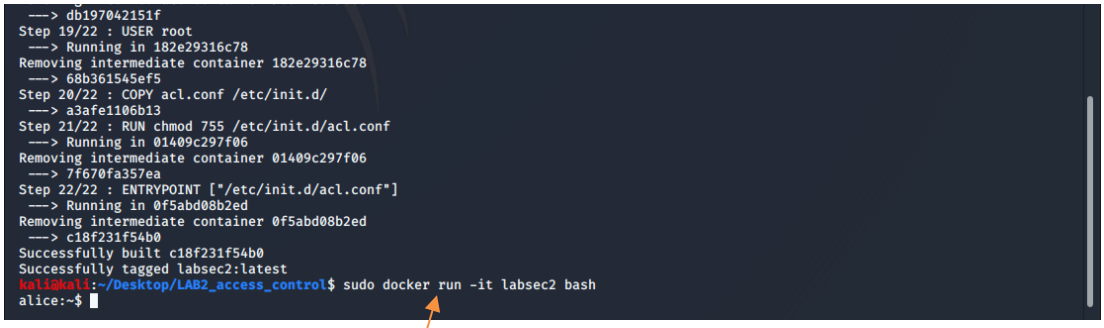


Create a docker image from the Dockerfile.

sudo docker build -t labsec2 . (Don't forget to copy the period.)



Wait for a few minutes, until you see the success figure below.



Then run the image and get a shell.

sudo docker run -it labsec2 bash

Once the alice shell is accessible, you may start the Challenges.

Question 1:

Answer the below questions.

a. What is a Docker? How it is different from a Virtual Machine?

b. Include a screenshot confirming that you have managed to create the docker image, build it and get an "alice" shell (*by following the readme file in Access Control folder*).

Start from Challenge 1.

Challenge 1: Unix permissions

In Unix, the kernel is the program that has unrestricted access to the whole machine. All other programs (subjects) run as a specific identity and their access to files and devices (objects) is mediated by the kernel.

Help Video

To complete this challenge, you need to ensure that you follow the content covered and practice with commands. To help further, a help video for Linux Access Control is available on the Unit Site from Content > Help Point (OnTrack) > OnTrack: Week 4 Task Resources. If you watch the video and work through the examples of each challenge, completing the challenges should be easy.

An alternative good resource is "The Linux Command Line" book by William Shotts, which is available for free as PDF. You can find it here.

User and group id

Access decisions are made based on the userid/groupid associated with the program.

If the user is root (userid = 0), access is always granted by the kernel.

Users have a primary group which usually has same id and name as the user id, but they may belong to several additional groups. By joining an existing group, a user inherits the permissions it grants.

Command id displays information about user and group id:

```
alice:~$ id
uid=1000(alice) gid=1000(alice) groups=1000(alice),1003(student)
```

Explanation:

- **uid** is the **user id,** for alice it is number 1000.
- **gid** is the **primary group id,** for alice it is the same as uid, i.e., 1000. This group is the one assigned at login and is used when files are created (see below)
- **groups** lists all the **groups** alice belongs to: **alice**(the default group) and **student**

In the docker you have three users (**alice, bob, carol**), plus root. Use **su** command to switch from one user to the other. You start as **alice**. Passwords for **bob, carol** and **root** are the same as the username:

```
alice:~$ su bob
Password:
bob:/home/alice$ exit
exit
alice:~$
```

su bob starts a shell as **bob**. With exit you go back to **alice** shell.

Unix permissions

Using the ls -l command we can display the Unix permissions set to a file or a directory:

```
alice:~$ touch myfile # creates an empty file named myfile
alice:~$ ls -l myfile
total 0
-rw-r--r-- 1 alice alice 0 Oct  3 08:08 myfile
```

Explanation:

- The fields displayed from left to right are:
  - file permissions **-rw-rw-r--,**
  - number of links **1,**
  - owner name **alice,**
  - owner group **alice,** (the primary group is used when creating a new file)
  - file size **0,**
  - time of last modification **Oct 3 08:08,** and
  - file/directory name myfile
- Apart from the first - (which represents the type of the file), file permission rw-rw-r-- is made of 3 triadsdefining the permissions granted to the owner, to the group and to all the other users, respectively. Each permission triad is commonly made up of the following characters:
  - **r**: the file can be **read** / the directory's contents can be **shown**
  - **w**: the file can be **modified** / the directory's contents can be **modified**
  - **x**: the file can be **executed** / the directory can be **traversed**
  - **s**: the file is **SUID** if s is found in the user triad (**SGID** if s is in the group triad). Implies x. Enables the file to **run with the privileges of its owner** (or group).

Example 1

In the following example file **rootfile1** is owned by **root** and has group **student.** It gives **read** and **write** permissions to **root** and only read permission to **student:**

```
alice@3545200f0b11:~$ ls -l  rootfile1
-rw-r----- 1 root student 39 Oct  3 08:26 rootfile1

alice@3545200f0b11:~$ id
uid=1000(alice) gid=1000(alice) groups=1000(alice),1003(student)

alice@3545200f0b11:~$ cat rootfile1     # read access
This file is readable by student group

alice@3545200f0b11:~$ cat > rootfile1   # write access
bash: rootfile1: Permission denied
```

Explanation:

- **rw-r-----** gives read/write permissions to owner (root) and only read permission to group student
- **cat rootfile1** prints the file content (read access) and this is allowed since alice belong to group student
- **cat > rootfile1** writes from stdin into the file (write access) and this is not allowed since student group permissions are r--

Example 2

File rootfile2 has the same permissions as **rootfile1** but group is **root,** so it can only be read by **root:**

```
alice@3545200f0b11:~$ ls -l rootfile2
-rw-r----- 1 root root 35 Oct  3 08:26 rootfile2

alice@3545200f0b11:~$ cat rootfile2   # read access
cat: rootfile2: Permission denied

alice@3545200f0b11:~$ cat > rootfile2 # write access
bash: rootfile2: Permission denied
```

Question 2:

Change directory to /tmp/ using commands you learned in Task 2.1P. Now, find the file that is accessible to alice. This file contains the password.

Now answer the below questions:

a. What does the "sudo" or "su" command do?
b. Include the screenshots of all commands that you have used to complete the Challenge 1. List the password that you obtained in Challenge 1.

Challenge 2: Managing Permissions

Changing permissions and ownership

Helpful slides

You can access these slides which could also help you complete Challenge 2.

Unix permissions can be altered using the **chmod** command, whilst the **owner** and **group** can be set using **chown**. Notice that non-root users can change the group (to one they belong to) but not the ownership.

```
alice:~$ ls -l myfile
-rw-rw-r-- 1 alice alice 0 Oct  3 10:28 myfile

alice:~$ chmod 600 myfile

alice:~$ ls -l myfile
-rw------- 1 alice alice 0 Oct  3 10:28 myfile

alice:~$ chown alice:student myfile

alice:~$ ls -l myfile
-rw------- 1 alice student 0 Oct  3 10:28 myfile

alice:~$ chown alice:bob myfile
chown: changing ownership of 'myfile': Operation not permitted

alice:~$ chown bob:alice myfile
chown: changing ownership of 'myfile': Operation not permitted
```

Explanation

- **chmod 600 myfile** changes permissions to **rw-------- 600** is interpreted as an octal number, each digit corresponding to the three permission bits, i.e., **6** is **rw-** and **0** is **---**
- **chown alice:student myfile** changes to group to **student** : succeeds since **alice** is in group **student**
- **chown alice:bob myfile** tries to change to group to **bob** : fails since **alice** is not in group **bob**
- **chown bob:alice myfile** tries to change owner to **bob** : fails since only **root** can change ownership

Question 3:

Answer the below questions:

a. What does the command "ls -l" do?
b. In your own words (i.e., no direct quotes), what does `Chmod' command do in Unix/Linux?
c. What is the command to set -rwxr-xr-x permissions to myfile? (make sure to include the exact command including any spaces).

Challenge 3: Access Control Lists

Access Control Lists (ACLs) provide an additional, fine grained permission mechanism for files and directories. With ACLs it is possible to define different permissions on a per-user/per-group basis. They have higher priority over Unix permissions.

To provide an example, we add an ACL to file **forcarol** so that it an be accessed **read**/**write** by **carol**.

```
alice:~$ ls -l forcarol
-rw------- 1 alice alice 59 Oct  3 15:04 forcarol

alice:~$ setfacl -m "u:carol:rw-" forcarol  # gives rw- permission to carol on file forcarol

alice:~$ ls -l forcarol
-rw-rw----+ 1 alice alice 59 Oct  3 15:04 forcarol



alice:~$ getfacl forcarol      # displays the ACL for file forcarol
# file: forcarol
# owner: alice
# group: alice
user::rw-
user:carol:rw-
group::---                     # here we have the new entry in the ACL
mask::rw-
other::---
```

Notice that **ls** -l shows a **+** symbol after permission to represent the fact that the permissions are not the ones represented by **-rw-rw—-** . In fact, rw does not refer to group **alice** but to user carol.

Now we can check that carol has access while bob does not:

```
alice:~$ su carol                  # become carol
Password: carol

carol:/home/alice$ cat forcarol      # read access
This file should be readable/writable by carol but not bob

carol:/home/alice$ su bob            # become bob
Password: bob

bob:/home/alice$ cat forcarol        # read access is forbidden to bob
cat: forcarol: Permission denied
```

The mask entry reported by the **getfacl** output is called the effective rights mask. This entry limits the **effective rights** that can be granted to the user and group ACL. By tightening the mask, we can, for example, revoke the write permission previously granted to the user carol.

```
alice:~$ setfacl -m 'm::r--' forcarol     # change the mask

alice:~$ getfacl forcarol
# file: forcarol
# owner: alice
# group:  alice
user::rw-
user:carol:rw-          #effective:r--
group::---
mask::r--
other::---

alice:~$ su carol
Password: carol

carol:/home/alice$ cat forcarol     # read access is OK
This file should be readable/writable by carol but not bob

carol:/home/alice$ cat > forcarol   # write access has been revoked
bash: forcarol: Permission denied
```

To remove an ACL from a file one can use the -b option:

```
alice:~$ ls -l forcarol
-rw-rw----+ 1 alice alice 59 Oct  3 15:04 forcarol

alice:~$ setfacl -b forcarol

alice:~$ getfacl forcarol
# file: forcarol
# owner: alice
# group: alice
user::rw-
group::---
other::---
```

Question 4:
Look for a file in **/tmp/** that is accessible by carol. It contains the password.
Include the screenshots of all commands that you have used to complete the Challenge 3. List the password that you obtained in Challenge 3.

Challenge 4: SUID and Capabilities

SUID permission

When permission s appears in place of x (executable) the program will be run with the privileges of the owner. In some cases, this in necessary to get appropriate access rights. Some system utilities, for

example, have SUID root permission as the need administrative privileges to run.

alice_shell program tries to set the uid of alice (1000) and start a shell: thus, the shell should belong to user alice. Let's see what happens if you runs this program as bob, without setting SUID permission:

```
alice:~$ ls -al alice_shell
-rwxr-xr-x 1 alice alice 8520 Oct  3 17:29 alice_shell

alice:~$ su bob
Password: bob

bob:/home/alice$ ./alice_shell
Trying to set uid 1000 (alice)
Failed to set alice uid. No permissions?
```

The program fails when trying to set **alice** uid, as it has no permissions to do so: **bob** is running the program so the program runs with **bob's** privileges.

We now set **SUID** permission. This can be done by prepending a 4 before the octal number specifying the actual permissions:

```
alice:~$ chmod 4755 alice_shell

alice:~$ ls -al alice_shell
-rwsr-xr-x 1 alice alice 8520 Oct  3 17:29 alice_shell

alice:~$ su bob
Password: bob

bob:/home/alice$ ./alice_shell
Trying to set uid 1000 (alice)
Starting a shell as 1000!

alice:~$ whoami
alice
```

Explanation:

- **rwsr-xr-x** shows the s representing **SUID** permission: the file is executable by any user and, when executed, it will run with the owner privileges (alice). The program name also appears coloured in red.
- When bob runs the program, it successfully set **uid 1000** (alice) and starts a new shell. Notice that the shell is started as user alice (as shown by the output of **whoami**)!

Example: ping

ping is a system utility that can be used to "ping" hosts. It requires root access in order to get raw access to the network which, in turn, is necessary to generate ICMP protocol packets (used by ping).

```
alice:~$ ls -al /bin/ping
-rwsr-xr-x 1 root root 64424 Jun 28 11:05 /bin/ping
```

Notice the SUID permission (s). Program ping will get root privileges when run by any user.

Privilege drop

SUID root can be very dangerous, since a vulnerability in the program might allow an adversary to get root access to the system. A standard mitigation technique is called privilege drop, and is summarised below:

1. Program is run SUID root;
2. Program performs privileged accesses, as soon as possible;
3. Once privileged accesses have been performed, the program drops the root privileges
4. The program goes on with unprivileged accesses

In this way, a vulnerability would give root privileges only during 2.

Capabilities

Linux Capabilities offer the possibility of assigning a subset of root permissions to specific processes. This can be also achieved via SUID root (s) permission, but executing the whole program as root is more risky in case of vulnerabilities, as discussed above.

Let's remove SUID root permission to ping (we need to log as root):

```
alice:~$ su
Password: root
root:/home/alice# ls -l /bin/ping
-rwsr-xr-x 1 root root 64424 Jun 28 11:05 /bin/ping

root:/home/alice# chmod 755 /bin/ping

root:/home/alice# exit
exit

alice:~$ ping localhost
ping: socket: Operation not permitted
```

Now **ping** is not working anymore as it need root permission to have raw network access. We can add this specific capability to ping as follows:

```
alice:~$ su
Password: root

root:/home/alice# setcap cap_net_raw+ep /bin/ping

root:/home/alice# getcap /bin/ping
/bin/ping = cap_net_raw+ep

root:/home/alice# exit
exit

alice:~$ ping localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.029 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.086 ms
^C
--- localhost ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1006ms
rtt min/avg/max/mdev = 0.029/0.057/0.086/0.029 ms
```

setcap cap_net_raw+ep /bin/ping adds the **cap_net_raw** capability to ping which can now be run by user alice without the necessity of SUID root permission.

Question 5:

In a paragraph (up to 200 words) summarize what you understood about SUID permission and capabilities as covered in Challenge 4. This needs to be in your own words (i.e., no direct quotes).

The following questions are common in interviews for cyber related positions. If you spent time going through the challenges and understanding Linux permission, then with a little extra research you should be able to answer them easily. Remember, doing your own research is an important skill for your future career in this field. If you are finding it difficult to answer these questions, practice a bit more in the terminal and watch the help video for Linux access control. Then, try again.

Question 6:

a. **Is the following statement True or False**? `sticky bit is a special permission that can be assigned to a file'.
b. **Is the following statement True or False**? `An executable file has SUID permission set. When the file is executed on the system, the user who runs the file becomes the file's temporary owner'
c. You just created a new script file named myapp.sh. However, when you try to run it from the command prompt, the bash shell generates an error that says -bash: ./myapp.sh: Permission denied. Which command will fix this problem?
d. A file named sit182.txt has a mode of rw-r--r--. If arash is not the file's owner and is not a member of the group that owns this file, what can he do with it?
e. A file named ontrack.ppt has a mode of rw-r--r--. If chang-tsun is the file's owner, what can he do with it?

Question 7:

a) If you wanted to have a data file that you could read or write, but don't want anyone else to see, the permission would be ..... (answer using the 9-bit e.g. -r--r--r--)
b) If the file is owned by the user, the ..... permission determine the access. (fill the blank either with OWNER/GROUP/OTHER)
c) If the group of the file is the same as the user's group, the ..... determine the access. (fill the blank either with OWNER/GROUP/OTHER)
d) If the user is not the file owner, and is not in the group, then the …… is used. (fill the blank either with OWNER/GROUP/OTHER)

Question 8:

Reflection point – What did you learn that was new to you? How did you manage to learn about Unix permissions to complete this task? Did you primarily use the Help Video and textbook provided or used your own resources?