COMP2700: Solutions to Lab 5 Exercises

Exercise 1.

1. ```
   $ chown alice:tutors tutorsonly.txt
   $ chmod 640 tutorsonly.txt
   ```

2. ```
   $ chown alice:tutors courses.txt
   $ chmod 664 courses.txt
   ```

3. ```
   $ chown alice:tutors feedbacks.txt
   $ chmod 640 feedbacks.txt
   ```

4. Since members of the `other` users have no read or write access to `feedbacks.txt`, they can't directly modify that file. This has to be done via an SUID program, `addfeedbacks` in this case. We need to turn `addfeedbacks` into an SUID program owned by `alice`, so when launched, the process for `addfeedbacks` will have alice's uid as the effective uid, and will therefore inherits alice's permission to `feedbacks.txt` (read & write). We will need to modify the SUID bit in the `special mode` group of permission bits. In addition, we need to make sure that members of `tutors` (other than `alice`) are not allowed to call `addfeedbacks`. This can be done by changing the group owner to `tutors`, and assign no permissions to the group.

   ```
   $ chmod 4705 addfeedbacks
   ```

Exercise 2.

In Unix/Linux, to traverse to a directory, a user needs to have the execute permission for that directory, whereas the read permission is needed to see the content of that directory. To be able to read the content of a file in a directory, a user first needs the permission to traverse to the location of the file, so they need to have the execute permission for the directory. The user does not need the read permission for the directory to access the files in the directory, but without the read permission for the directory, the user would not be able to list the files in that directory, so the user would need to know exactly the file name they want to access. So the minimum permission needed for the directory is the execute permission.

Exercise 3.

1. To find the process id, run the program `whatsmyid` in a terminal, and open another terminal and run the `ps` command. There are various options to achieve the same thing, here's one example
   ```

```
$ ps -eo pid,command | grep whatsmyid
```

In this case the first column of the output contains the process id (`pid`) and the second column contains the name of the program that process is running. The `grep` command filters the program name so we only print commands that contain `whatsmyid`. In this case, the pid is in the first row in the output. The process pid is not fixed, so you may see different pids for different runs of the program.

To send SIGTERM, use the `kill` command, using the numerical value of the signal (15 in this case) and the process id. For example, if the process id is 1807 (it may differ in your session)

```
$ kill -15 1807
```

You can use also use the keyword SIGTERM directly if you don't know the numerical value of the signal:

```
$ kill -SIGTERM 1807
```

This will terminate the process gracefully.

2. Perform the same steps as above to find the process id of `stubborn`. This process tries to ignore all signals sent to it, in particular the SIGTERM signal, so you won't be able to terminate it using SIGTERM. Instead, we will use SIGKILL -- this signal can't be ignored even if the process tries to. Allowing a process to ignore all signals (including SIGKILL) could allow a malicious process to perform denial of service, e.g., consuming resources of the system without the system being able to terminate it. Note that SIGSTOP will suspend the process, but it does not kill the process. You can bring it back by using the command `fg` (in the same terminal where the process was launched).

Exercise 4.

This depends on whether the user has a way to either list a file in the target directory (read permission), or to change into that directory (execute permission). A symbolic link (or soft link, as it is sometimes called) is essentially an `alias` for a path to a file, so for a user to ascertain the existent of a file pointed to by a symbolic link, the user would need to be able to at least traverse to the directory containing that file.

If the current user can traverse to the target directory, then it is obviously possible for the user to determine the existent of a file in that directory.

If the current user has no access at all to the target directory, e.g., when the target directory is `/root` and the current user is `alice`, then resolving any symbolic link to any file in `/root` (e.g., via `cat`), whether the file exits or not, will return the same response (`permission denied`). For example, suppose /root/file1 exists but not /root/file2. Here's the output of an attempt by `alice` to link to /root/file1 and /root/file2:

```
alice@comp2700_lab:~$ ln -s /root/file1
alice@comp2700_lab:~$ ln -s /root/file2
alice@comp2700_lab:~$ cat file1
cat: file1: Permission denied
alice@comp2700_lab:~$ cat file2
cat: file2: Permission denied
```

In this case `alice` will not be able to tell whether or not any of the files exist in /root.

Exercise 5.

There are at least two ways to find files which are hard-linked. One way is to use the `find` command with the `-samefile` option, e.g., to find all files that are hard-linked to file1,

```
$ cd ~/lab5/links
$ find . -samefile file1
```

You would need to do this for all files in the subdirectories as well, so it may take a few more similar commands.

Alternatively, you can also use `ls` with the option `-i` to print the inodes of the files in the directory:

```
$ ls -i -1 *
```

The command `ls -i -1 *` will list all files (recursively), print their inodes (the option `-i`), list one file per line (the option `-1`). Files that have the same inode are hard linked.

Exercise 6.

Run the `passwd` command as `admin2700` and leave it at the `Current password:` prompt. Open another terminal and run:

```
$ ps -u root -o ruser:10,euser,pid,comm | grep passwd
admin2700  root         1587 passwd
```

It shows that the real user of the `passwd` command is `admin2700` but the effective user is `root`.

Exercise 7.

By setting the environment variable USER to anything other than `charlie` and run the SUID program:

```
$ su charlie
$ cd /home/alice/lab5/suid
```

```
$ env USER=notcharlie ./filter
```

Note that the suid program `filter` looks for `not_for_charlie.txt` in the current directory, so you'd need to change to `/home/alice/lab5/suid` first before running `filter`. The problem with this SUID program (`filter`) is that it relies on an attacker-controlled value (the environment variable `USER`) in its access control decision.

Exercise 8a.

1. This can be found using the df command (among others):

   ```
   $ df
   Filesystem     1K-blocks    Used Available Use% Mounted on
   udev              969212       0    969212   0% /dev
   tmpfs             203100    1072    202028   1% /run
   /dev/sda2       65733164 6526700  55834996  11% /
   tmpfs            1015488       0   1015488   0% /dev/shm
   tmpfs               5120       0      5120   0% /run/lock
   tmpfs            1015488       0   1015488   0% /sys/fs/cgroup
   /dev/loop1         68864   68864         0 100% /snap/lxd/21835
   /dev/loop2         63488   63488         0 100% /snap/core20/1587
   /dev/loop0         63488   63488         0 100% /snap/core20/1593
   /dev/loop3         48128   48128         0 100% /snap/snapd/16292
   /dev/loop4         69504   69504         0 100% /snap/lxd/22753
   /dev/loop5         44672   44672         0 100% /snap/snapd/14978
   tmpfs             203096       0    203096   0% /run/user/1000
   ```

   The last column (`Mounted on`) shows the directory for each mounted file system. For the root directory `/`, we can see that it corresponds to the device `/dev/sda2`. Note that you may see a different device, depending on the configuration of your system. The above is an example from the lab VM.

2. To see the permissions associated to the block device /dev/sda1, we can use the ls command:

   ```
   alice@comp2700-lab:~$ ls -l /dev/sda2
   brw-rw---- 1 root disk 8, 2 Aug 15 05:21 /dev/sda2
   ```

   The device /dev/sda2 is owned by root user, and group ‘disk’. The permissions for the group is ‘rw-’, so anyone who is in the ‘disk’ group will be able to read and write to the device directly. Recall from the lecture that the information about members of a group is located in the file `/etc/group`. So to find members of the group disk, simply display the file /etc/group (which can be read by anyone in the system). The following command uses ‘grep’ to display only the relevant group:

   ```
   $ grep disk /etc/group
   disk:x:6:alice
   ```

So the group contains alice as its member. So alice is the only non-root user who has direct access to the block device /dev/sda2.

3. The file /mnt/root has the following permission:

```
alice@comp2700-lab:~$ ls -l /mnt/test.txt
-rw------- 1 root root 6 Aug 17 02:46 /mnt/test.txt
```

It is readable only by the root user.

For alice to access the `/mnt/text.txt` file, it cannot be done through the normal operation at the root file system level; one must access directly the underlying block device, i.e., /dev/sda2. This can be done by copying directly the content of the blocks in the device that are associated with the file `/mnt/test.txt`. But this would require us to find out the exact inode and the physical blocks where the data is located. But fortunately, the `debugfs` tool provides a more user-friendly way to access the file directly by its path:

```
alice@comp2700-lab:~$ debugfs /dev/sda2
debugfs 1.45.5 (07-Jan-2020)
debugfs:  cat mnt/test.txt
hello
```

The debugfs command launches a separate shell, and using the `cat mnt/test.txt` we can display directly the shadow file.

Exercise 8b.

This is very similar to Ex. 8a. See the solution for 8a for overall explanation. Below we show only the relevant commands:

1. Use `df` to find the mount points of disks:

```
$ df
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/root       30298176 4828260  25453532  16% /
devtmpfs         2005616       0   2005616   0% /dev
tmpfs            2010108       0   2010108   0% /dev/shm
tmpfs             402024    1072    400952   1% /run
tmpfs               5120       0      5120   0% /run/lock
tmpfs            2010108       0   2010108   0% /sys/fs/cgroup
/dev/loop2         48128   48128         0 100% /snap/snapd/16292
/dev/loop1         69504   69504         0 100% /snap/lxd/22753
/dev/loop0         63488   63488         0 100% /snap/core20/1611
/dev/loop3         63488   63488         0 100% /snap/core20/1587
/dev/sdb15        106858    5321    101537   5% /boot/efi
```

```
/dev/sda1        20464208       32  19399320    1% /mnt
tmpfs              402020        0    402020    0% /run/user/1000
```

We see that `/mnt/` is associated with `/dev/sda1`. Note that in the Azure Labs VM, the root directory `/` is mounted from a device called `/dev/root` (this is not a standard device, but a configuration specific to this Azure Lab VM).

2. To see the permissions associated to the block device /dev/sda1, we can use the ls command:

```
$ ls -l /dev/sda1
brw-rw---- 1 root disk 8, 1 Aug 16 23:49 /dev/sda1
```

Again as in Exercise 8a, we notice that the group owner is `disk` with read-write access, and `alice` is a member of this group, so will also have read-write access.

3. For this, just as in Exercise 8a, we use the `debugfs` program to access the disk `/dev/sda1` directly.

```
alice@ML-RefVm-692768:~$ debugfs /dev/sda1
debugfs 1.45.5 (07-Jan-2020)
debugfs:  cat test.txt
hello
```

Exercise 9 (*).

Run debugfs in write mode:

```
$ debugfs -w /dev/sda2
```

Then use the modify_inode command to modify the inode of /etc/shadow

```
debugfs 1.44.1 (24-Mar-2018)
debugfs:  modify_inode /etc/shadow
                          Mode    [0100640]
```

Edit the first entry, which should be

```
 Mode    [0100640]
```

Change it to `0100666`. Quit debugfs and restart the OS. When you log back in, you can verify that the permission of /etc/shadow has been changed to

```
-rw-rw-rw- 1 root shadow 2039 Jul 24  2019 /etc/shadow
```

Exercise 10 (*).

We can use the `-perm` option, followed by a permission mask to select bits that we are interested in. In this case, since we are only interested in the owner SUID bit, we use the mask `4000`, which sets the SUID bit and leaves other bits unset:

```
$ find / -user root -perm -4000
```

Another, more user-friendly way to specify the mask is to use the symbolic names `u+s`:

```
$ find / -user root -perm  -u+s
```

Note that you may notice a lot of "Permission denied" errors as some directories are not accessible by the current user. You can supress these errors by re-directing these errors to the `/dev/null` device:

```
$ find / -perm -u+s 2>/dev/null
```