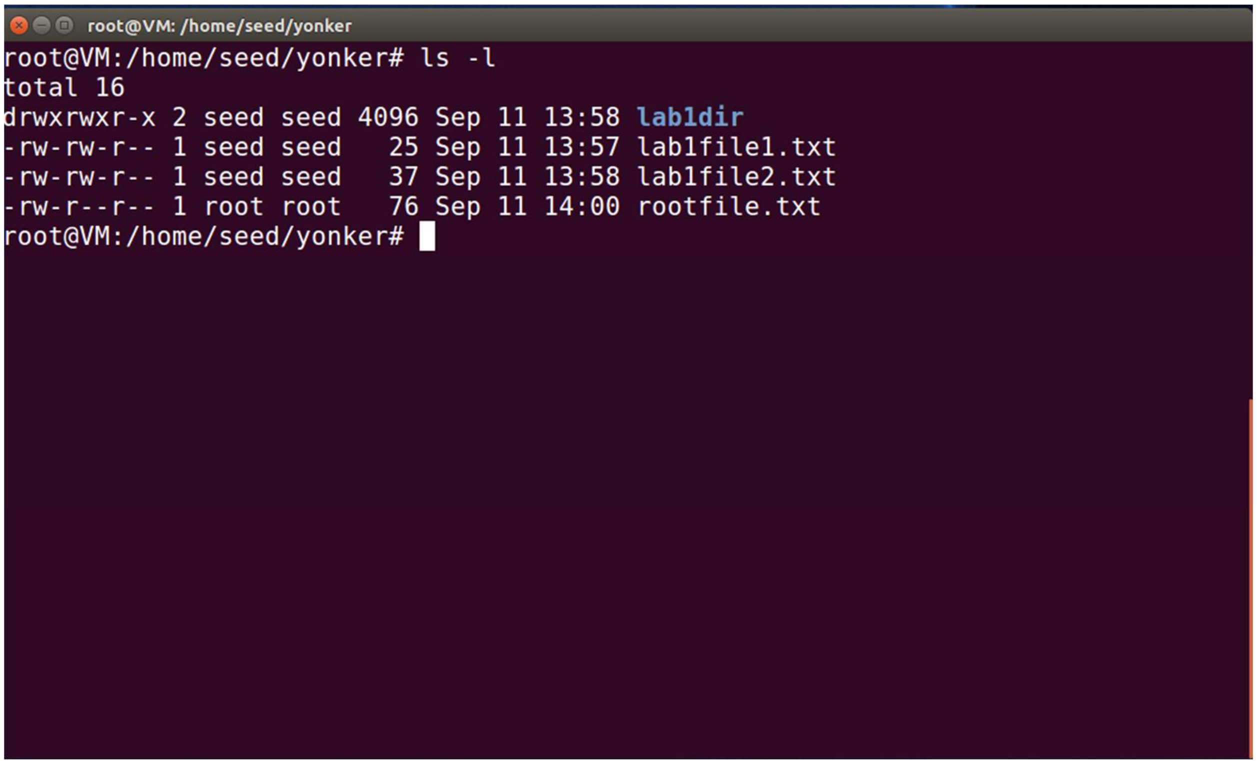
Lab 1 – Access Control in Linux
Gunnar Yonker

Task 1:

(1)

a.



```
root@VM:/home/seed/yonker# ls -l
total 16
drwxrwxr-x 2 seed seed 4096 Sep 11 13:58 lab1dir
-rw-rw-r-- 1 seed seed   25 Sep 11 13:57 lab1file1.txt
-rw-rw-r-- 1 seed seed   37 Sep 11 13:58 lab1file2.txt
-rw-r--r-- 1 root root   76 Sep 11 14:00 rootfile.txt
root@VM:/home/seed/yonker# 
```

b. lab1dir – owner and group has “rwx” so they have read, write, and execute permissions. Others have “r-x”, read and execute permissions. File size is 4096 bytes.

lab1file1.txt and lab1file2.txt – owner and group have “rw-“ so they have read and write permissions. Others have “r—” so they have read only permissions. File sizes are 25 bytes and 37 bytes respectively.

rootfile.txt – owner has “rw-“(read and write), group has “r—“(read only), and others have “r—“(read only). File size is 76 bytes.

“lab1dir”, “lab1file1.txt”, and “lab1file2.txt” are owned by user “seed’ and group “seed”.

“rootfile.txt” is owned by user “root” and group “root”.

Any user on the system can read “lab1file1.txt”, “lab1file2.txt”, and “rootfile.txt”. However, only the respective owners (“seed for the first two and “root” for the third) can write or modify these files. The directory “lab1dir” is readable, writeable, and accessible by both the owner and the group members. Others can read and access the directory but cannot write to it. The “rootfile.txt”, being owned by “root”, means that normal users won’t be able to delete or modify the file unless they have root privileges.

Lab 1 – Access Control in Linux
Gunnar Yonker

(2) The seed user is able to read the content of the file since the file has read permissions for others. However, when trying to edit the file and then save it the action is denied. There is an error message when you try to save the changes. In vim, it is E45 which is when a user is trying to write to a file without the right permissions.

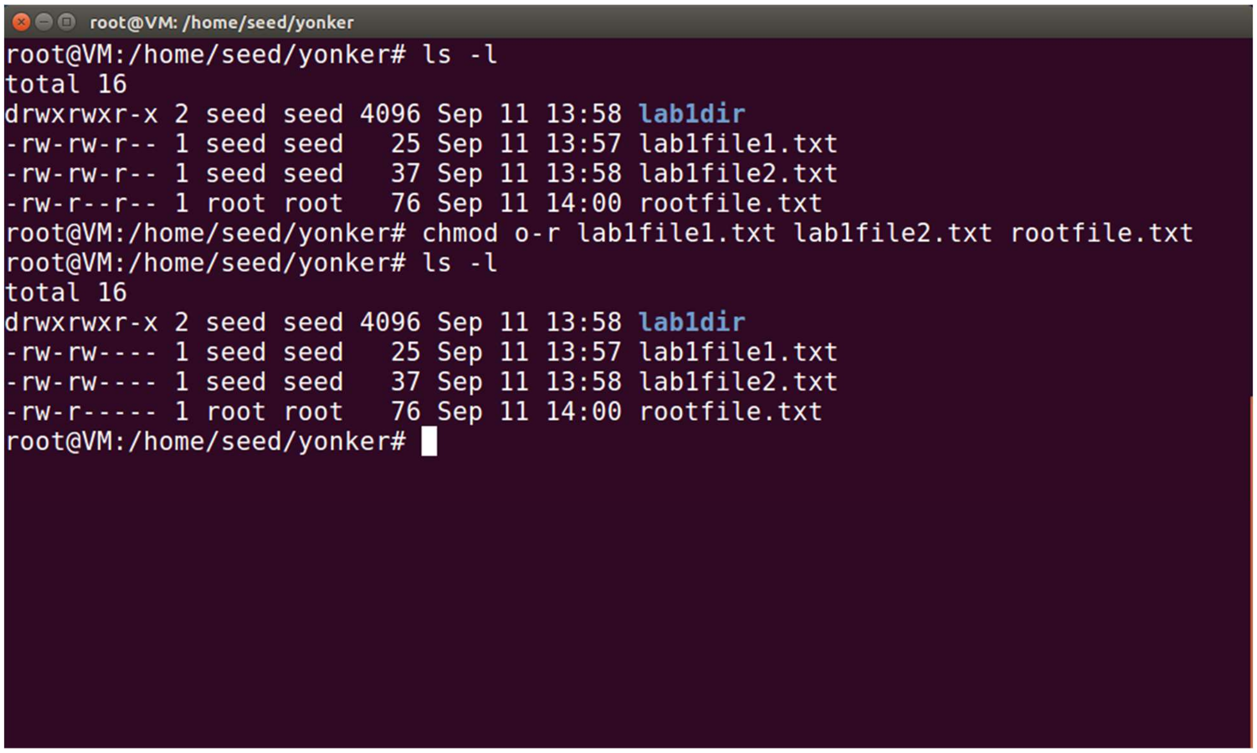
Linux uses a combination of Discretionary Access Control (DAC) and Mandatory Access Control (MAC). The permission system on files and directories is part of the DAC mechanism. The permissions are broken down into owner, group, and others. This granular control allows system administrators and users to specify who can do what with a file or directory. The access controls effectively prevent unauthorized modifications to files but can allow a user to still be able to read the file without being able to change it's contents. There is also flexibility with the files being readable by anyone on the system, but only writeable by the owner or a specific group; The structure is also relatively straightforward and easy to understand who has access to a file or directory and what they can do with it. The system itself is effective and flexible, but with misconfigured permissions there is still the possibility of security vulnerabilities.

(3) The Linux system, by design, follows the "Least privilege" principle. This principle dictates that a user or program should only have the minimum privileges necessary to perform its function thereby limiting the potential for misuse or unintentional harm. When a user account is created, it is given very limited permissions by default. Critical system files and directories are typically inaccessible to regular users, ensuring that they cannot inadvertently or maliciously alter system configurations or disrupt system operations. For example, administrative tasks generally require root privileges, which are not granted to normal users. Most services and daemons running on the system are often configured to run as non-root users, which limits the potential damage they can cause if they are compromised.

Lab 1 – Access Control in Linux
Gunnar Yonker

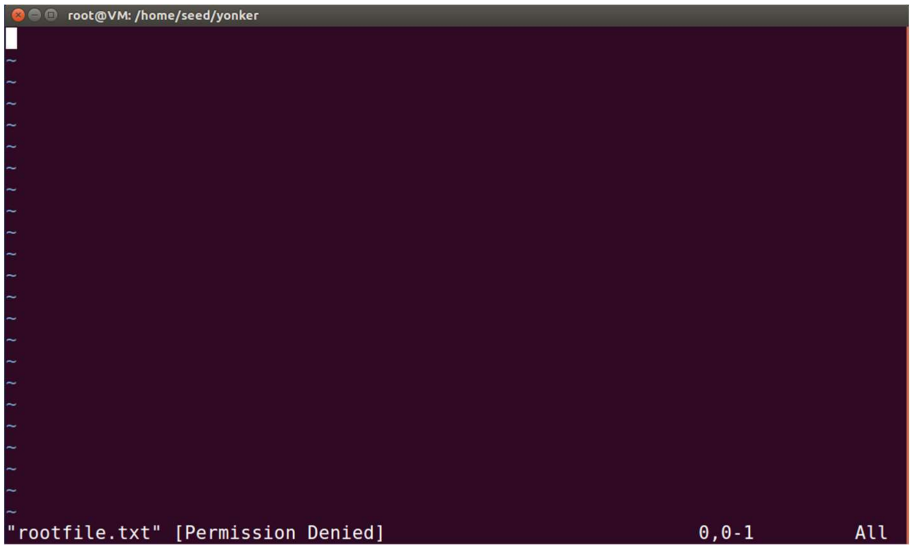
Task 2:

(1)



```
root@VM: /home/seed/yonker
root@VM:/home/seed/yonker# ls -l
total 16
drwxrwxr-x 2 seed seed 4096 Sep 11 13:58 lab1dir
-rw-rw-r-- 1 seed seed   25 Sep 11 13:57 lab1file1.txt
-rw-rw-r-- 1 seed seed   37 Sep 11 13:58 lab1file2.txt
-rw-r--r-- 1 root root   76 Sep 11 14:00 rootfile.txt
root@VM:/home/seed/yonker# chmod o-r lab1file1.txt lab1file2.txt rootfile.txt
root@VM:/home/seed/yonker# ls -l
total 16
drwxrwxr-x 2 seed seed 4096 Sep 11 13:58 lab1dir
-rw-rw---- 1 seed seed   25 Sep 11 13:57 lab1file1.txt
-rw-rw---- 1 seed seed   37 Sep 11 13:58 lab1file2.txt
-rw-r----- 1 root root   76 Sep 11 14:00 rootfile.txt
root@VM:/home/seed/yonker# 
```

As the seed user, now if I try to open the “rootfile.txt” file with vim it shows a blank file and displays “Permission Denied”. This is because the read permission for other has been removed.



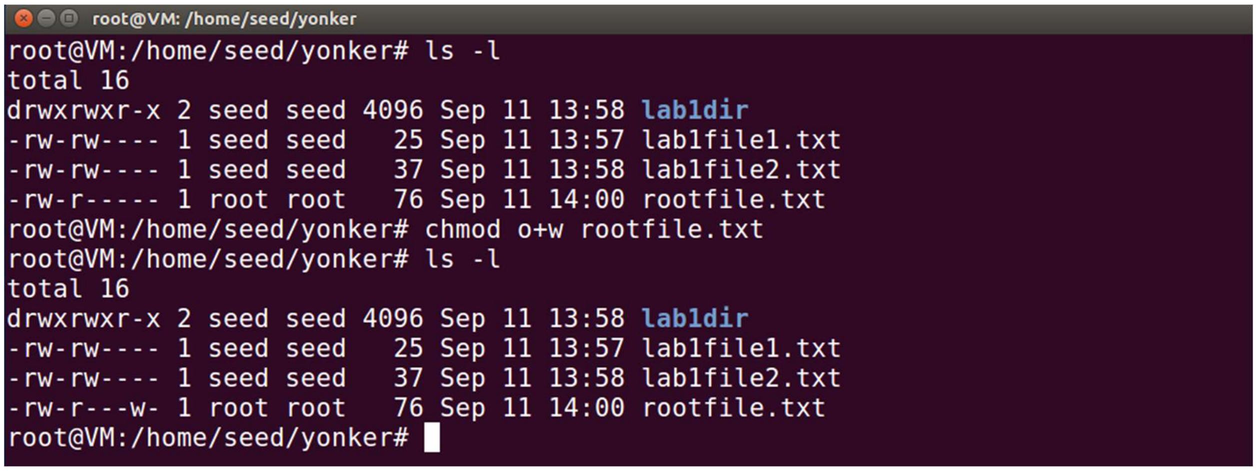
```
root@VM: /home/seed/yonker

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"rootfile.txt" [Permission Denied]                        0,0-1         All
```

The principle behind this behavior is with how Linux evaluates file permissions. When a user tries to access a file, Linux first checks owner permissions, then if not it checks group permissions, and if neither of those are a match it will default to the other permissions. In this case, seed is not the owner or a member of the group root so the other permissions apply. Since other permissions cannot read, write, or execute the file, when I try to open it as seed the permission is denied and the contents are not shown.

Lab 1 – Access Control in Linux
Gunnar Yonker

(2) This permission change needs to be done as the root user before switching back to the seed user and attempting to write to the file.



root@VM: /home/seed/yonker

```
root@VM:/home/seed/yonker# ls -l
total 16
drwxrwxr-x 2 seed seed 4096 Sep 11 13:58 lab1dir
-rw-rw---- 1 seed seed   25 Sep 11 13:57 lab1file1.txt
-rw-rw---- 1 seed seed   37 Sep 11 13:58 lab1file2.txt
-rw-r----- 1 root root   76 Sep 11 14:00 rootfile.txt
root@VM:/home/seed/yonker# chmod o+w rootfile.txt
root@VM:/home/seed/yonker# ls -l
total 16
drwxrwxr-x 2 seed seed 4096 Sep 11 13:58 lab1dir
-rw-rw---- 1 seed seed   25 Sep 11 13:57 lab1file1.txt
-rw-rw---- 1 seed seed   37 Sep 11 13:58 lab1file2.txt
-rw-r---w- 1 root root   76 Sep 11 14:00 rootfile.txt
root@VM:/home/seed/yonker# 
```

When switching back to the seed user and attempting to write to the rootfile.txt file, looking at the permissions I would be able to write to the file, but not read its current contents. When opening the file using vim, the content is not displayed because there are no read permissions, however I am able to add new contents such as a new sentence and then save the file. This effectively overwrites the file's content without seeing it. When switching back to the root user and opening the file, the contents are confirmed to be overwritten even though the seed user was never able to read the original contents. This situation shows the importance of understanding Linux permissions. Granting only write permission without read permission allows users to modify and overwrite the file's content blindly, they are unable to view the current content, but they can replace it. This demonstrates how granular and flexible the permission system can be. It is crucial for system administrators to be cautious and deliberate when assigned such permissions to avoid unintended consequences such as in this example.

Lab 1 – Access Control in Linux
Gunnar Yonker

Task 3:

There are two different solutions that I can think of that can be taken so that the new user, smitha, could be able to read the rootfile.txt file with the reading permission.

Method 1: User Group (Either using sudo or running as root user)

Once smitha was created using the useradd command we could then create a group using the following command:

sudo groupadd readrootfile

Then we could add smitha to the group:

sudo usermod -aG readrootfile smitha

Next, change the file group's ownership:

sudo chown :readrootfile rootfile.txt

Lastly, set the group read permission:

sudo chmod g+r rootfile.txt

This would allow the group readrootfile to have read permissions of the rootfile.txt while still maintaining the original owner being the root user. The group ownership of the file would be readrootfile, which smitha is now part of. Since the group permissions on the file are set to read, smitha will now have reading permissions of the rootfile.txt and be able to open and read the file. This also ensures that the other permissions can still be set to what they previously were.

Method 2: Making the File Readable for Other (Either using sudo or running as root user)

This method is less secure, but does technically accomplish the objective at hand of allowing the user smitha to read the rootfile.txt file. For this method the only command needed would be:

sudo chmod o+r rootfile.txt

As previously mentioned, this would now allow the user smitha to read the rootfile.txt file as the other permissions would be set to read. However, this means that all other users are also able to read this file as well. Depending on the goal of letting smitha read the file, this could be a solution, but is a much less secure solution than method 1 mentioned above.

In conclusion, Linux follows the "Least privilege" design principle in a few different ways. When a new user is created, they are given their own isolated permissions. They can't access files of other users unless explicit permissions are set. There is also granular control with groups because they provide a mechanism to grant multiple users the same set of permissions. This avoids giving direct permissions to individual users, ensuring controlled access (Method 1 vs Method 2). Lastly, there are default restrictions. By default, files and directories created are not world writeable. Permissions need to be explicitly set, ensuring that there aren't unintentional permissions granted. Linux offers tools and mechanisms that embody the "Least privilege" design principle by providing system administrators with the flexibility to implement access control as broadly or as granularly as required. The default configurations also tend to be conservative, ensuring that security isn't inadvertently compromised.