**Task 1: Add Groups and Users**

1. Making the three groups

A screenshot of a computer

Description automatically generated

Verify groups were made successfully:

A screenshot of a computer

Description automatically generated

2. Add users

A screenshot of a computer screen

Description automatically generated

Assign users to groups, two users in each group and each user in only one group each:

A screenshot of a computer screen

Description automatically generated

Verify that the users were successfully added to the groups:

A screenshot of a computer screen

Description automatically generated

Employee: user1 and user2

Manager: user3 and user4

Board: user5 and user6

**Task 2: Role Based Access Control**

1. File Design:

Employee Files: These files are related to general information, common for every employee. Example created by user1

Files: “employee\_info.txt” and “employee\_info2.txt”

Manager Files: These files are related to managerial tasks, like scheduling or budgets. Example created by user3.

Files: “manager\_task1.txt” and “manager\_task2.txt”

Board Files: These files containing sensitive company information or long-term plans. Example created by user5.

Files: “board\_plan1.txt” and “board\_plan2.txt”

**Access Rights Setting:**

**Employee Files:** Here is an example of how to ensure that the owner user1 can read, write, and execute while the group “Employee” can only read the files. Others cannot access. So user2 can read the file, but not write or execute the file. Similarly, user3-6 cannot read, write, or execute the file.

sudo chown user1:Employee employee\_info\*.txt

sudo chmod 740 employee\_info\*.txt

**Manager Files:** Here is an example of how to ensure that owner user3 can read, write, and execute while the group “Manager” can only read the files. Others cannot access. So user4 can read the file, but not write or execute the file. Similarly, user1-2 and user5-6 cannot read, write, or execute the file.

sudo chown user3:Manager manager\_task\*.txt

sudo chmod 740 manager\_task\*.txt

**Board Files:** here is an example of how to ensure that owner user5 can read, write, and execute while the group “Board” can only read the files. Others cannot access. So user6 can read the file, but not write or execute the file. Similarly, user1-4 cannot read, write, or execute the file.

sudo chown user5:Board board\_plan\*.txt

sudo chmod 740 board\_plan\*.txt

2. The root user in Linux, also known as the superuser, has unrestricted access to all files and commands. It is not possible to prevent root from accessing any files, as root inherently has the permission to override or change permissions. This highlights how important it is that root access is restricted, because if compromised, the whole system is vulnerable.

For the seed user (if it’s not a root user or equivalent), you can use similar permissions as discussed above to restrict access. However, if seed has sudo privileges, it can override these permissions to still access the files. Without sudo privileges, seed can be restricted.

3. Implementation and Testing:

Using the above permission examples and files, I will carry out some testing with the users. First is a screenshot showing the example files outlined above.

user1, user3, and user5 will be tested as the owner of their respective files.

user2, user4, and user6 will be tested as the group but not owner of the files for their respective groups as shown previously.

File setup:

A screenshot of a computer screen

Description automatically generated

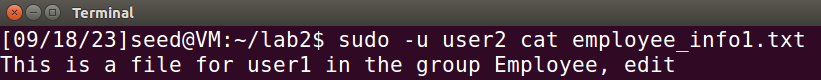
**Employee Group Test:**

user1 access to employee\_info1.txt, shows access to file and the user is able to write to the file:

A screenshot of a computer

Description automatically generated

user2, in the Employee group, is able to read the file but not write or execute:





A screenshot of a computer

Description automatically generated

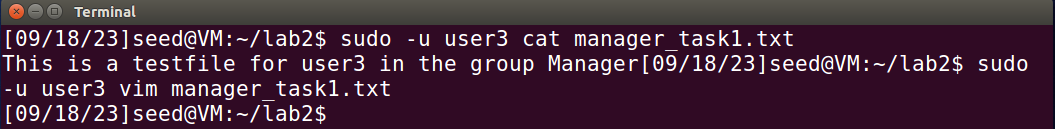
user3-6, not in Employee group and not the owner, are unable to read, write, or execute the file:

A screenshot of a computer screen

Description automatically generated

**Manager Group Test:**

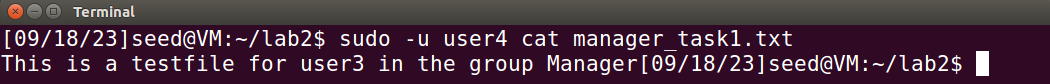
user3 access to manager\_task1.txt, shows access to file and the user is able to write to the file:

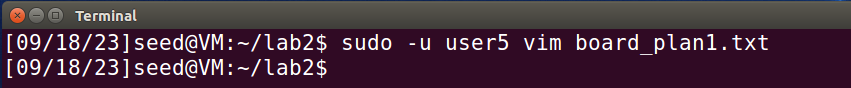


A screenshot of a computer

Description automatically generated

user4, in the Manager group, is able to read the file but not write or execute:





A screenshot of a computer

Description automatically generated

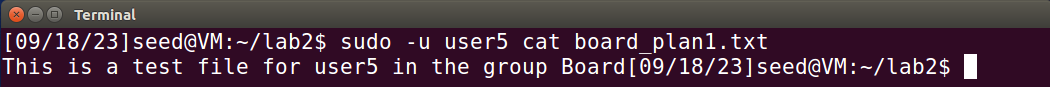
user1-2 and user5-6, not in Manager group and not the owner, are unable to read, write, or execute the file:

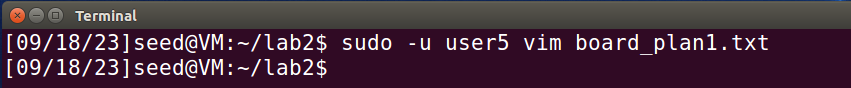
A screenshot of a computer screen

Description automatically generated

**Board Group Test:**

user5 access to board\_plan1.txt, shows access to file and the user is able to write to the file:

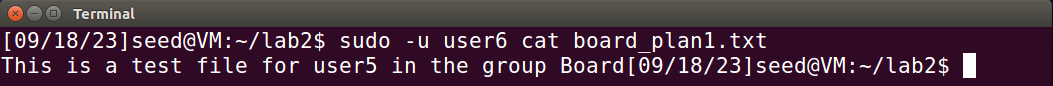


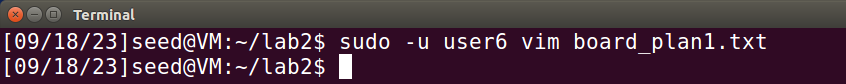


A screenshot of a computer

Description automatically generated

user6, in the Board group, is able to read the file but not write or execute:





A screenshot of a computer

Description automatically generated

user1-4, not in Board group and not the owner, are unable to read, write, or execute the file:

A screenshot of a computer screen

Description automatically generated

**Root is able to access files, because root has unrestricted access:**

**A screenshot of a computer

Description automatically generated**

**seed is able to access files using sudo in this example, if sudo is not possible, then seed would not work:**

**A screenshot of a computer program

Description automatically generated**

**Verify File Ownership and Permissions:**

A screenshot of a computer screen

Description automatically generated

In these test scenarios it shows that the RBAC design and implementation works as intended.

**Task 3: Multi-level Security**

1. Hierarchy: Board > Manager > Employee

**File Design:**

Employee Files: Previously created by user1

Files: “employee\_info1.txt” and employee\_info2.txt”

Manager Files: Previously created by user3

Files: “manager\_task1.txt” and “manager\_task2.txt”

Board Files: Previously created by user5

Files: “board\_plan1.txt” and “board\_plan2.txt”

In order to accomplish this in stock Linux, since stock Linux doesn’t natively support a hierarchical or multi-level access control system readily, I will be using ACL to make this possible.

sudo apt install acl

Employee Group Permissions:

* Owner: Read, write, execute
* Group Member: Full access rights to any file in Employee Group

Manager Group Permissions:

* Owner: Read, write, execute
* Group Member: Full access rights to any file in Employee and Manager Groups

Board Group Permissions:

* Owner: Read, write, execute
* Group Member: Full access rights to any file in Employee, Manager, and Board Groups

**Access Rights Setting:**

Employee Files:

sudo chown user1:Employee employee\_info\*.txt

sudo chmod 770 employee\_info\*.txt

sudo setfacl -m g:Employee:rwx employee\_info\*.txt

sudo setfacl -m g:Manager:rwx employee\_info\*.txt

sudo setfacl -m g:Board:rwx employee\_info\*.txt

This ensures that group members of Board, Manager, and Employee have full access rights for the given Employee group files. All users will have full access rights on these files.

Manager Files:

sudo chown user3:Manager manager\_task\*.txt

sudo chmod 770 manager\_task\*.txt

sudo setfacl -m g:Manager:rwx manager\_task\*.txt

sudo setfacl -m g:Board:rwx manager\_task \*.txt

This ensures that group members of Board and Manager have full access rights for the given Manager group files. The following users will have full access rights on these files: user3, user4, user5, and user6.

Board Files:

sudo chown user5:Board board\_plan\*.txt

sudo chmod 770 board\_plan\*.txt

sudo setfacl -m g:Board:rwx board\_plan\*.txt

This ensures that the group members of Board have full access rights for the given Board group files. The following users will have full access rights on these files: user5 and user6.

Even though this method is functional and accomplishes the goal of the multi-level security design, there are still some obstacles to overcome. The obstacles that are encountered here is that while this works, it can get very complicated if there are many files and roles. It’s not as intuitive or easy to manage as a true multi-level security model. Since it is more complicated as well, ensuring that the proper permissions are established is crucial, otherwise there can be more vulnerabilities and lapses in security. This may work in a situation where there are few roles and files, but scaling this process up for a large number of files and roles isn’t feasible.

**2. Access Checking and Testing:**

**File Setup:**

**A screenshot of a computer

Description automatically generated**

**ACL settings for an example file from each group:**

All employee\_info\*.txt files should look like this:

A screenshot of a computer program

Description automatically generated

All manager\_task\*.txt files should look like this:

A screenshot of a computer

Description automatically generated

All board\_plan\*.txt files should look like this:

A screenshot of a computer

Description automatically generated

With permissions properly set in this scenario, we can move to checking/testing if they work as intended. The goals are as follows:

* user1 and user2 have full access rights to the employee\_info1.txt file, and no access rights to the other group files.
* user3 and user4 have full access rights to the employee\_info1.txt file and manager\_task1.txt file, no access rights to the Board group files.
* user5 and user6 have full access rights to files from all groups. Ex. employee\_info1.txt, manager\_task1.txt, and board\_plan1.txt.

**Check access for user1 (Employee, owner):**

sudo -u user1 cat employee\_info1.txt # Should work

sudo -u user1 cat manager\_task1.txt # Should not work

sudo -u user1 cat board\_plan1.txt # Should not work

**Test:**

Read passed:

A screenshot of a computer

Description automatically generated

Write passed (checked using vim):

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen with a purple background

Description automatically generated

A computer screen with a purple background

Description automatically generated

**Check access for user2 (Employee, not owner):**

sudo -u user2 cat employee\_info1.txt # Should work

sudo -u user2 cat manager\_task1.txt # Should not work

sudo -u user2 cat board\_plan1.txt # Should not work

**Test:**

Read passed:

A screenshot of a computer screen

Description automatically generated

Write passed (checked using vim):

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen with a purple background

Description automatically generated

A computer screen with a dark background

Description automatically generated

**Check access for user3 (Manager, owner):**

sudo -u user3 cat employee\_info1.txt # Should work

sudo -u user3 cat manager\_task1.txt # Should work

sudo -u user3 cat board\_plan1.txt # Should not work

**Test:**

Read passed:

A screenshot of a computer

Description automatically generated

Write passed (checked using vim):

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen with a purple background

Description automatically generated

**Check access for user4 (Manager, not owner):**

sudo -u user4 cat employee\_info1.txt # Should work

sudo -u user4 cat manager\_task1.txt # Should work

sudo -u user4 cat board\_plan1.txt # Should not work

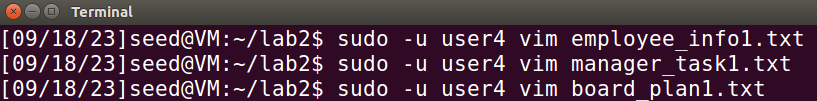
**Test:**

Read passed:

A screenshot of a computer screen

Description automatically generated

Write passed (checked using vim):



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**A computer screen with a purple background

Description automatically generated**

**Check access for user5 (Board, owner):**

sudo -u user5 cat employee\_info1.txt # Should work

sudo -u user5 cat manager\_task1.txt # Should work

sudo -u user5 cat board\_plan1.txt # Should work

**Test:**

Read passed:

A screenshot of a computer

Description automatically generated

Write passed (checked using vim):

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Check access for user6 (Board, not owner):**

sudo -u user6 cat employee\_info1.txt # Should work

sudo -u user6 cat manager\_task1.txt # Should work

sudo -u user6 cat board\_plan1.txt # Should work

**Test:**

Read passed:

A screenshot of a computer

Description automatically generated

Write passed (checked using vim):

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

**Results:**

The multi-level security implementation outlined above was proven to correctly work and accomplished the goals that were established for testing. The hierarchy was successfully implemented where the Board group was able to have full rights for files belonging to the Board, Manager, and Employee groups. The Manager group was able to have full rights for files belonging to the Manager and Employee groups. Lastly, the Employee group was able to have full rights for files belonging to the Employee group. In my scenario, .txt files were used so execute was not applicable to test. However, based on the permissions established and the results of the test, it is reasonable to conclude that if execute was applicable that it would also function correctly in the permission hierarchy.