A logo of a state

AI-generated content may be incorrect.

**Name**: Francisco Izaguirre

**Course**: CSCI 4334

**Date**: 04/29/2025

**Instructor**: Wendy A. Sanchez Gomez

**Lab 06 – Linux Security Basics**

***Introduction:***

This lab focused on understanding Linux file permissions and how they contribute to system security. Using basic terminal commands, I explored how ownership, groups, and permission bits determine access rights for files. I worked with tools like ls, chmod, and umask to observe how permissions are structured and how they can be modified. I also practiced modifying file permissions using both symbolic and numeric modes and observed how different umask values affect the default access level of newly created files. The objective was to gain hands-on experience with managing file access, interpreting permission symbols, and applying best practices for protecting sensitive data in a multi user environment. By interacting directly with permission settings, I developed a better understanding of how the system enforces access rules for different users and groups.

***Experimental Procedure:***

Step 1 & 2) First, we open up a terminal on Linux and tried using the file command to determine the type of file and its data. Then we used the less command to attempt to read the contents of a text file in one page. Unfortunately, we did not have permissions to use the less command with “shadow”. However, we were able to determine that shadow was a standard file with the file command.

A screenshot of a computer

AI-generated content may be incorrect.

A computer screen shot of a code

AI-generated content may be incorrect.

This shows us how Linux distinguishes file types and how sensitive system files such as “shadow”, are protected. It also shows us how not all files can be accessed by regular users even if they exist.

Step 3) In order to access certain files, we must have control over its access either through groups who are given access or by being the owner of the file. In order to find out our identity, we can use the “id” command to display our user identity and group memberships. We see that our user ID is 1000 as well as my group ID.

A screenshot of a computer code

AI-generated content may be incorrect.

This shows us that each user has a unique user ID and is associated with a primary group ID. The user’s ability to access or modify files depends heavily on these ID’s.

Step 4) The gid and uid information comes from the /etc/passwd and /etc/group files. In order to examine the contents of these two files we will use “cat”. In the passwd file, we were able to find several entries that contain information such as my username, uid, gid, and home directory. In both files, the root was in the first line while my user was on the lines that contained “usr” or “adm”. A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

This shows us that these two files define the identity and group used by Linux for access control. Therefore, when we ran the “id” command from step 3, the system pull this the information from these files. This is important to know as knowing where this data is stored is key to managing users and permissions.

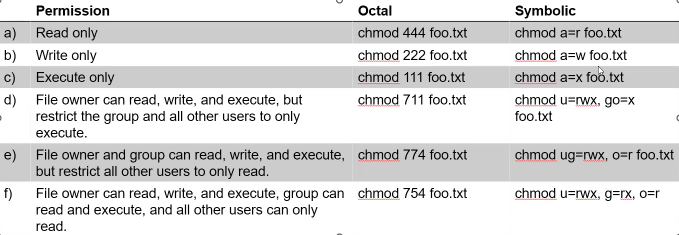
Step 5 & 6) Now we will explore how accessing rights to files and directories are defined in terms of read access, write access, and execution access. First, we will create a file named “foo.txt” by using “> foo.txt”. Then we will use the ls command to view the file permissions. The ls command lists detained information about the file. We see that by default the foo.txt file had read & write access for the owner, read and write access for the group, and read only access for the world. This is shown with the “rw-rw-r” text after the ls command. Then we modified permissions using the “chmod” command through octal mode. We used 6 (r-w) for the owner and 0 (no permissions) for the group and world. Then we used ls to confirm the changes. The new text showed “rw-------“, which confirms our change.

A screen shot of a computer program

AI-generated content may be incorrect.

This shows us how Linux represents permissions visually and lays the foundation for manipulating and understanding permissions. This also shows us how to fully private a file and make it accessible to only the owner.

Step 7) After seeing how we were able to change the permissions for the foo.txt, we will try all these other options (a-f) while also converting the chart from text to octal and symbolic.



This lays the foundation of how permissions work and how we can manipulate them to function how we need. It also shows us two different ways in which we can set these settings up (octal & symbolic).

Step 8) Now we will explore the “umask” command, which controls the default permissions given to a file or directory when it’s created. First, we deleted the existing foo.txt file to start with a clean slate. Then we used “umask” to check the current mask value, which was 0002. Then we created a new empty foo.txt file and used the ls command to check the new file permissions. We see that the command worked as the default is “rw-rw-r”.

A screenshot of a computer program

AI-generated content may be incorrect.

This shows us how umask works (subtracts permissions from the default) and it illustrates how umask controls the default permissions for new files. The file is writable by owner and group, but not writable by others due to the 2 umask.

Step 9) Next, we will repeat the last step but instead set the mask ourselves. First, we removed the foo.txt to start with a clean slate and then set the umask to 0000, which apply to new files. Then, we created a new foo.txt file and used the ls command to ensure our umask changed successfully. We see that the 0000 means setting the owner, group, and world to read and write. Since the umask was set to 0000, no permissions are subtracted.

A screenshot of a computer program

AI-generated content may be incorrect.

This shows us how changing the umask directly affects security and access levels for new files.

Step 10) Now we will repeat the previous step again but instead with 0022 and 0777. Once again, before each umask set, we remove the foo.txt file to start with a new slate. Then we set the umask and create the new foo.txt file and use the ls command to ensure that we properly set the umask. 0022 translated to “rw-r—r—“, while 0777 translated to “-------“ or no permissions for anyone.

This shows us how different mask umask values mask specific permission bits and determine the default access level for new files. This helps reinforce why umask is a key element of Linux file security.

***Questions:***

1. **What is the difference between chmod and umask?** chmod changes a file’s mode while umask sets the default file permissions. (Slide 13)
2. **What is the most convenient umask value to protect the files?**

The most convenient umask value to ensure protection of files would be 0077 as this would mean there are no permissions for anyone but the owner.

1. **Is there a way to set the permissions to a specific value without using command line?**

Yes, you can set file permissions without using the command line by using a graphical file manager on Linux.

1. **Explain the seven fields of each entry in /etc/passwd.**

**Field 1** (Username), used when user logs in. It should be between 1 and 32 characters in length. **Field 2** (Password), an x character indicates that encrypted and salted password is stored in /etc/shadow file. **Field 3** (User ID), each user is assigned a user ID (UID). UID 0 (zero) is reserved for root and UIDs 1-99 are reserved for other predefined accounts. Further UID 100-999 are reserved by system for administrative and system accounts/groups. **Field 4** (Group ID), the primary group ID (stored in /etc/group file). **Field 5** (User ID Info), the comment field. It allow you to add extra information about the users such as user’s full name, phone number etc. This field use by finger command. **Field 6** (Home directory), the absolute path to the directory the user will be in when they log in. If this directory does not exist, then user directory becomes /. **Field 7** (Command/shell), the absolute path of a command or shell (/bin/bash). Typically, this is a shell. (Slide 19 & 20)

***Conclusion:***

The main goal of this lab was to explore how Linux controls file access through its permission system. By using commands like chmod and umask, I was able to modify and analyze permission settings, learning how different configurations affect who can read, write, or execute a file. I also examined how user and group information is stored and used by the system to enforce access restrictions. Throughout the steps, I saw how default permissions are influenced by the umask value and how permission changed can either restrict or open access depending on the situation. Overall, this lab helped reinforce the importance of file security and gave me hands on experience with tools that are commonly used in real world system administration.

**References:**

Sanchez, Wendy. CYBI 3345 – *Operating Systems and Security*

*CSCI 4334 – Operating Systems Security and Protection*. 29 Apr. 2025. (PowerPoint).