Author: Alexander Martinez

Title: Security Analysis of File Permissions and User Access in Linux Operating Systems

1. Experiment aim:

**Student’s answer:**

This experiment aims to explore the functionalities of the chmod command for setting file permissions, understand the impact of permissions on user access, and analyze the capabilities of user groups and file access logging in a Linux environment.

2. Theoretical background:

**Student’s answer:**

**File Permissions**: In Linux, each file and directory is associated with a set of permissions that determine how users can access and modify them. These permissions are categorized into three levels:

* Owner: Permissions granted to the file owner.
* Group: Permissions granted to the group the file belongs to.
* Others: Permissions granted to all other users on the system.

**chmod Command**: This command allows modifying file permissions using a specific syntax: chmod <permission\_code> <file\_name>. The permission code is a combination of numbers and letters representing specific access levels:

* r (4): Read permission
* w (2): Write permission
* x (1): Execute permission
* + (add): Add permission
  + (remove): Remove permission
* = (set): Set permission

**User Groups**: Groups are collections of users who share specific access privileges. Granting permissions to a group allows efficient management of access for multiple users.

**File Access Logging**: Linux offers the ability to track file access attempts through logging functionalities. This enables monitoring user activity and detecting potential security breaches.

This report will provide practical examples and explanations for each exercise, demonstrating how the theoretical concepts are applied in a real-world scenario.

3. Research:

**Ex. 1. Exploring chmod Commands Scenario:**

1. Create a new directory named "SecureFiles" in your home directory.

2. Inside "SecureFiles," create three text files: "file1.txt," "file2.txt," and "file3.txt."

3. Set the following permissions:

* "file1.txt" should be readable, writable, and executable by the owner, and readable by others.
* "file2.txt" should be readable and writable by the owner only.
* "file3.txt" should be readable, writable, and executable by the owner and the group.

**Student’s answer:**

**# Scenario: Create a new directory named "SecureFiles" in your home directory.**

**mkdir ~/SecureFiles**

**# Scenario: Inside "SecureFiles," create three text files: "file1.txt," "file2.txt," and "file3.txt."**

**touch ~/SecureFiles/file1.txt ~/SecureFiles/file2.txt ~/SecureFiles/file3.txt**

**# Scenario: Set the following permissions:**

**# "file1.txt" should be readable, writable, and executable by the owner, and readable by others.**

**chmod 600 ~/SecureFiles/file1.txt # Incorrect command, does not include execute permission for the owner.**

**# "file2.txt" should be readable and writable by the owner only.**

**chmod 777 ~/SecureFiles/file2.txt # Incorrect command, grants excessive permissions to everyone.**

**# "file3.txt" should be readable, writable, and executable by the owner and the group.**

**chmod 555 ~/SecureFiles/file3.txt # Incorrect command, provides incorrect permissions to the owner and the group.Questions:**

What chmod commands did you use to set the specified permissions?  
**Student’s answer:**

For "file1.txt": chmod u+rwx,o+r ~/SecureFiles/file1.txt

For "file2.txt": chmod u+rw ~/SecureFiles/file2.txt

For "file3.txt": chmod ug+rwx ~/SecureFiles/file3.txt

How do the permissions of each file affect user access?  
**Student’s answer:**

For "file1.txt": The owner has read, write, and execute permissions, while others have only read permissions.

For "file2.txt": The owner has read and write permissions, while others have no permissions.

For "file3.txt": Both the owner and the group have read, write, and execute permissions, while others have no permissions.

**Ex. 2.  User Groups and File Access Scenario:**

1. Create a new user named "TestUser" on your Linux system.
2. Add "TestUser" to a group named "SecureGroup."
3. Ensure that "SecureGroup" has read and write access to all files inside the "SecureFiles" directory.
4. Log in as "TestUser" and attempt to modify "file3.txt" inside the "SecureFiles" directory.

**Student’s answer:**

# Scenario: Create a new user named "TestUser" on your Linux system.

sudo useradd TestUser

# Scenario: Add "TestUser" to a group named "SecureGroup."

sudo usermod -aG SecureGroup TestUser

# Scenario: Ensure that "SecureGroup" has read and write access to all files inside the "SecureFiles" directory.

chmod -R g-rwx ~/SecureFiles # Incorrect command, removes all group permissions instead of granting read and write access.

# Scenario: Log in as "TestUser" and attempt to modify "file3.txt" inside the "SecureFiles" directory.

su - TestUser

echo "Additional content" >> ~/SecureFiles/file3.txt

**Questions:**

How did you add "TestUser" to the "SecureGroup"?

**Student’s answer:**

Used the command sudo usermod -aG SecureGroup TestUser to add the user "TestUser" to the group "SecureGroup."

Why was "TestUser" unable to modify "file3.txt"? What permissions were missing?  
**Student’s answer:**

"TestUser" was unable to modify "file3.txt" because the write permissions for the group were not granted. The command chmod -R g+rw ~/SecureFiles was used to ensure read and write access for the group "SecureGroup" to all files inside the "SecureFiles" directory.

**Ex. 3. Logging and Monitoring File Access Scenario:**

1. Enable file access logging for the "SecureFiles" directory.
2. Create a log file to record all file access events.
3. Access "file1.txt" from another user account, and check the log file for the recorded event.

**Student’s answer:**

# Scenario: Enable file access logging for the "SecureFiles" directory.

sudo auditctl -w ~/SecureFiles -p rwx # Incorrect command, invalid option "rwx" for file access logging.

# Scenario: Create a log file to record all file access events.

sudo touch /var/log/file\_access.log

sudo chmod 444 /var/log/file\_access.log # Incorrect command, sets read-only permissions for all users.

# Scenario: Access "file1.txt" from another user account and check the log file for the recorded event.

# (Assuming another user account is named "AnotherUser")

su - AnotherUser

cat ~/SecureFiles/file1.txt

exit

cat /var/log/file\_access.log # Incorrect command, attempts to view the log file directly without proper permissions.

**Questions:**

How did you enable file access logging for the directory?  
**Student’s answer:**

Used the command sudo auditctl -w ~/SecureFiles -p rwxa to enable file access logging for the "SecureFiles" directory.

What information is logged when accessing "file1.txt" from another user account?  
**Student’s answer:**

The log file /var/log/file\_access.log will record events associated with accessing "file1.txt," capturing details such as the username, timestamp, and the type of access (read, write, execute). The exact information can be extracted from the log file for further analysis

4. Conclusions:

**Student’s answer:**

This exploration of file access control in Linux has demonstrated the effectiveness of the chmod command for setting granular permissions, the role of user groups in managing access privileges, and the value of file access logging for monitoring user activity.

Chmod empowers granular control: By manipulating permissions, we can define exactly how users and groups can interact with files, ensuring data confidentiality and integrity.

User groups streamline access management: Assigning users to relevant groups simplifies permission management, granting specific access levels to multiple users efficiently.

File access logging enhances security: By recording access attempts, we gain valuable insights into user behavior, enabling the detection of potential security breaches and unauthorized access attempts.

Understanding these concepts and implementing appropriate file access control measures are crucial for safeguarding sensitive data and maintaining a secure Linux environment.