Author: John Adams

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:**

# Create the directory "SecureFiles"

mkdir SecureFiles

# Create three text files inside "SecureFiles"

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

# Set permissions for file1.txt (owner: rwx, group: r--, others: r--)

chmod 744 SecureFiles/file1.txt

# Set permissions for file2.txt (owner: rw--, group: --, others: --)

chmod 600 SecureFiles/file2.txt

# Set permissions for file3.txt (owner: rwx, group: rw--, others: --)

chmod 660 SecureFiles/file3.txt

**Questions:**

What chmod commands did you use to set the specified permissions?

**Student’s answer:**

chmod 744 SecureFiles/file1.txt: This command sets the permissions for file1.txt as follows:

* Owner: read (4), write (2), and execute (1) - all permissions granted (7)
* Group: read (4) - only read permission granted
* Others: read (4) - only read permission granted

chmod 600 SecureFiles/file2.txt: This command sets the permissions for file2.txt as follows:

* Owner: read (4) and write (2) - read and write permissions granted (6)
* Group: no permissions (0)
* Others: no permissions (0)

chmod 660 SecureFiles/file3.txt: This command sets the permissions for file3.txt as follows:

* Owner: read (4), write (2), and execute (1) - all permissions granted (6)
* Group: read (4) and write (2) - read and write permissions granted (6)
* Others: no permissions (0)

How do the permissions of each file affect user access?

**Student’s answer:**

file1.txt: The owner can read, write, and execute the file. Other users and the group can only read the file. They cannot modify or execute it.

file2.txt: Only the owner can read and write the file. Other users and the group have no access to the file, meaning they cannot read, write, or execute it.

file3.txt: The owner has full access (read, write, and execute). Additionally, the group members can also read and write the file, but they cannot execute it. Others have no access

**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:**

# Create a new user named "TestUser" (use sudo with caution)

sudo useradd TestUser

# Add "TestUser" to the group "SecureGroup" (replace "your\_group" with the actual group name)

sudo usermod -a -G your\_group TestUser

# Grant read and write permissions to "SecureGroup" for the "SecureFiles" directory

chmod g+rw SecureFiles

**Questions:**

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

**Student’s answer:**

The command used is: sudo usermod -a -G your\_group TestUser

Why was "TestUser" unable to modify "file3.txt"? What permissions were missing?

**Student’s answer:**

"TestUser" was unable to modify "file3.txt" because the group "SecureGroup" lacked the execute permission for the directory "SecureFiles". While the group had read and write permissions, attempting to modify a file requires traversing the directory structure, which needs the execute permission.

To resolve this issue, the command chmod g+rw SecureFiles was used. This grants read (r) and write (w) permissions to the group (g) for the directory "SecureFiles". This allows members of the "SecureGroup", including "TestUser", to navigate the directory and access the files within it, enabling them to modify "file3.txt".

**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:**

# Install auditd if not already present (adjust package name if necessary)

sudo apt install auditd

# Start the auditd service

sudo systemctl start auditd

# Configure auditd to log file access events for "SecureFiles"

sudo auditctl -w /home/<username>/SecureFiles -p rwxa -f success,failure

# Create a log file (replace "access.log" with your desired filename)

touch access.log

# Access "file1.txt" from another user account

# Check the log file for recorded events (use tools like "ausearch" or "tail -f access.log")

sudo ausearch /home/<username>/SecureFiles -f

# OR

sudo tail -f access.log

**Questions:**

How did you enable file access logging for the directory?

**Student’s answer:**

1. Install auditd: This daemon is responsible for system auditing, including file access events.
2. Start the auditd service: This ensures the daemon is running and capturing logs.
3. Configure auditd: The auditctl command is used to define specific rules for what gets logged. In this case, the command:
4. -w /home/<username>/SecureFiles: Specifies the directory to be monitored ("SecureFiles").
5. -p rwxa: Defines the types of events to be logged (read, write, execute, and attribute changes).
6. -f success,failure: Logs both successful and failed access attempts.

What information is logged when accessing "file1.txt" from another user account?

**Student’s answer:**

The specific information logged may vary depending on the system configuration, but it typically includes details like:

* **Timestamp:** Time of the access attempt.
* **Subject:** User who attempted to access the file.
* **Object:** Path to the accessed file ("/home/<username>/SecureFiles/file1.txt").
* **Action:** Type of access attempted (e.g., "read").
* **Result:** Whether the access was successful or failed.

By checking the log file (e.g., using ausearch or tail), you can analyze the recorded events to understand user activity and identify any potential security concerns.

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.