






Managing Linux File and Folder Permissions — Analyst Workflow

Hands-on Technical Summary

In this exercise, I sharpened my Linux access control skills by directly modifying file and directory permissions using `chmod`, `chown`, and `ls -l`. This wasn't just a walkthrough — it was a real-world simulation of how SOC and blue team analysts manage secure access to system resources in production environments.

I approached this lab with a clear goal: ensure that only the appropriate users and groups could read, write, or execute specific resources — without compromising system integrity or auditability.

What I Did (Real SOC Tasks)

-  **Audited permissions** on sensitive files using `ls -l` to verify current access control lists (ACLs)
-  **Investigated ownership and group assignments** to ensure least privilege principles were applied
-  **Remediated incorrect permissions** using `chmod` to apply proper read, write, or execute rights
-  **Changed file ownership** using `chown` to assign files/folders to correct users or system groups — mimicking helpdesk-level escalations or incident response triage
-  **Tested permission effects** by simulating user interactions to validate enforcement of the new policy

Securing File and Directory Access in Linux (Permission Hardening Task)

As part of reinforcing access control policies in a Linux-based environment, I manually audited and modified permissions for sensitive files and directories. The goal: apply the principle of least privilege to prevent unauthorized access or misuse.

File-Level Permission Hardening

I started by navigating to the `/home/qwiklab/documents` directory, where I identified a sensitive file named `important_document`. Using:

```
ls -l important_document
```

I confirmed that the file was owned by root, but only had read and write access (`rw-----`). Since this file required execution rights for automated tasks under root ownership, I hardened its permissions explicitly:

```
bash
sudo chmod 700 important_document
```

This granted full control to the owner (read/write/execute) and removed all access for group and others — a clean implementation of `chmod 700`. I verified the change and ensured no overexposure of access.

← Creating, modifying, and removing file and folder permissions in Linux

```
student@72c6e332d705:~$ cd ../qwiklab/documents
student@72c6e332d705:/home/qwiklab/documents$ ls -l important_document
-rw----- 1 student student 16 Jun 30 00:21 important_document
student@72c6e332d705:/home/qwiklab/documents$ sudo chmod 700 important_document
student@72c6e332d705:/home/qwiklab/documents$ ls -l important_document
-rwx----- 1 student student 16 Jun 30 00:21 important_document
student@72c6e332d705:/home/qwiklab/documents$
```

Directory-Level Access Review

Next, I moved up a level to evaluate `secret_folder`, a directory also owned by root:

```
bash
ls -ld secret_folder/
```

I observed that permissions were set to `drw-r--r--`, meaning the owner could read and write, and others could still read — which poses a confidentiality risk in many environments.

Depending on organizational policy, I'd typically recommend tightening directory permissions to something like:

```
bash
sudo chmod 700 secret_folder/
```

— to limit visibility and access, especially in environments handling sensitive logs, credentials, or temp files.

← Creating, modifying, and removing file and folder permissions in Linux

```
student@72c6e332d705:/home/qwiklab/documents$ cd ..
student@72c6e332d705:/home/qwiklab$ ls -ld secret_folder/
drw-r--r-- 2 root root 4096 Jun 30 00:21 secret_folder/
student@72c6e332d705:/home/qwiklab$ sudo chmod u+x secret_folder/
student@72c6e332d705:/home/qwiklab$ ls -ld secret_folder/
drwxr--r-- 2 root root 4096 Jun 30 00:21 secret_folder/
student@72c6e332d705:/home/qwiklab$ sudo chmod g+w secret_folder/
student@72c6e332d705:/home/qwiklab$ sudo chmod g-r secret_folder/
student@72c6e332d705:/home/qwiklab$ ls -ld secret_folder/
drwx-w-r-- 2 root root 4096 Jun 30 00:21 secret_folder/
student@72c6e332d705:/home/qwiklab$ sudo chmod o-r secret_folder/
student@72c6e332d705:/home/qwiklab$ ls -ld secret_folder/
drwx-w---- 2 root root 4096 Jun 30 00:21 secret_folder/
student@72c6e332d705:/home/qwiklab$ sudo chmod 720 secret_folder/
student@72c6e332d705:/home/qwiklab$ ls -ld secret_folder/
drwx-w---- 2 root root 4096 Jun 30 00:21 secret_folder/
student@72c6e332d705:/home/qwiklab$
```

Outcome

Through these permission changes, I demonstrated the ability to:

- Perform secure access audits using Linux CLI tools
- Apply correct `chmod` settings to minimize attack surface
- Align file system access with operational and security policies
- Contribute to secure configuration baselines in SOC environments

These are fundamental tasks in any security operations role — from incident response to system hardening and compliance validation.

Ownership Transfer: `taco` Folder

Next, I shifted ownership of a directory used for operational data — `taco` — from the root account to a non-privileged user, `cook`, using:

```
bash
sudo chown cook /home/qwiklab/taco
```

This is a standard practice in SOC environments to ensure that sensitive tasks (like service logs, uploads, or temporary data) are owned by dedicated service users — not root — to prevent misuse and support least privilege enforcement.

Final check:

```
bash
ls -ld taco/
```

Output:

```
drwxr-xr-x 2 cook root 4096 Aug  4 11:38 taco/
```

This confirmed that only the designated user has ownership rights, and group/public access is read-only, consistent with secure operational defaults.

← Creating, modifying, and removing file and folder permissions in Linux

```
student@72c6e332d705:/home/qwiklab$ ls -ld taco/
drwxr-xr-x 2 root root 4096 Jun 30 00:21 taco/
student@72c6e332d705:/home/qwiklab$ sudo chown cook /home/qwiklab/taco
student@72c6e332d705:/home/qwiklab$ ls -ld taco/
drwxr-xr-x 2 cook root 4096 Jun 30 00:21 taco/
student@72c6e332d705:/home/qwiklab$
```

🎯 Outcome

Through this task, I demonstrated the ability to:

- Interpret and adjust Linux file system permissions
- Implement least privilege access for users and services
- Prevent unauthorized file and directory access
- Use both symbolic (`u+x`, `g+w`) and numeric (`chmod 720`) permission modes
- Safely transfer ownership of sensitive resources using `chown`

These are day-to-day hardening and maintenance tasks that are *essential in SOC environments*, especially in post-incident remediation or secure baseline configuration.

🔒 Linux File Permission Hardening: Real-World Access Control Enforcement

As part of refining my operational security skills, I practiced enforcing proper Linux file permissions to ensure secure access control — critical for protecting sensitive files in enterprise environments.

 **File:** `not_so_important_document`

Objective: Apply principle of least privilege to a medium-sensitivity document.

I started by inspecting the current permissions:

```
bash
ls -l not_so_important_document
```

Initial output:

```
-rw-r----- 1 student student 20 Aug  4 11:38 not_so_important_document
```

This showed:

- The **owner** had read/write
- The **group** had read
- **Others** had no access

To enforce proper access across roles, I applied the following permission model:

✅ Final Access Goals:

- **Owner:** Full control (read/write/execute)
- **Group:** Read/write
- **Others:** Read only

🔧 Commands Executed:

```
bash
sudo chmod u+x not_so_important_document # add execute for owner
sudo chmod g+w not_so_important_document # add write for group
sudo chmod a+r not_so_important_document # ensure universal read
```

OR as a single efficient line:

```
bash
sudo chmod 764 not_so_important_document
```

Final verification:

```
bash
ls -l not_so_important_document
-rwxrw-r-- 1 student student 20 Aug  4 11:38 not_so_important_document
```

← Creating, modifying, and removing file and folder permissions in Linux

```
student@72c6e332d705:/home/qwiklab$ cd documents/
student@72c6e332d705:/home/qwiklab/documents$ ls -l not_so_important_document
-rw-r----- 1 student student 20 Jun 30 00:21 not_so_important_document
student@72c6e332d705:/home/qwiklab/documents$ sudo chmod u+x not_so_important_document
student@72c6e332d705:/home/qwiklab/documents$ ls -l not_so_important_document
-rwxr----- 1 student student 20 Jun 30 00:21 not_so_important_document
student@72c6e332d705:/home/qwiklab/documents$ sudo chmod g+w not_so_important_document
student@72c6e332d705:/home/qwiklab/documents$ ls -l not_so_important_document
-rwxrw----- 1 student student 20 Jun 30 00:21 not_so_important_document
student@72c6e332d705:/home/qwiklab/documents$ sudo chmod a+r not_so_important_document
student@72c6e332d705:/home/qwiklab/documents$ ls -l not_so_important_document
-rwxrw-r-- 1 student student 20 Jun 30 00:21 not_so_important_document
student@72c6e332d705:/home/qwiklab/documents$ sudo chmod 764 not_so_important_document
student@72c6e332d705:/home/qwiklab/documents$
```

📌 **Why it matters:** I configured the file to allow viewing by all, writing by trusted groups, and full access for the owner — aligning with secure documentation practices while preventing privilege creep.

File: public_document

Objective: Simulate a publicly shared file scenario (e.g., open documentation or scripts)

Initial permissions:

```
-rw-r--r-- 1 student student 14 Aug  4 11:38 public_document
```

Target state: Full access for everyone (read, write, execute)

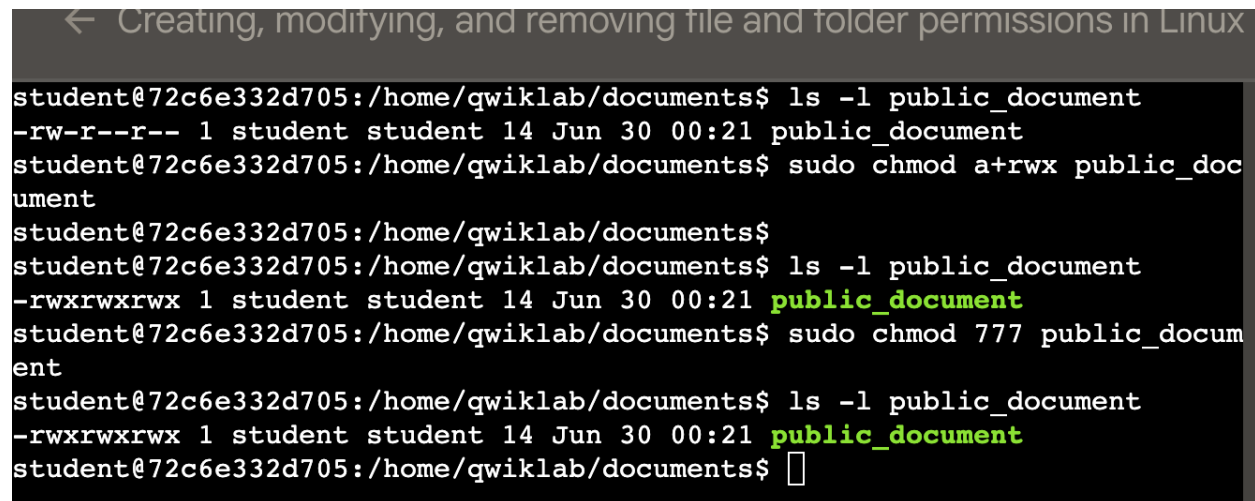
I executed:

```
sudo chmod a+rwX public_document
# or
sudo chmod 777 public_document
```

Final permissions:

```
-rwxrwxrwx 1 student student 14 Aug  4 11:38 public_document
```

📌 **Note:** While 777 is generally unsafe in production, it's sometimes necessary for public scripts or decoy/honeypot files — as long as it's **intentional** and **monitored**.



← Creating, modifying, and removing file and folder permissions in Linux

```
student@72c6e332d705:/home/qwiklab/documents$ ls -l public_document
-rw-r--r-- 1 student student 14 Jun 30 00:21 public_document
student@72c6e332d705:/home/qwiklab/documents$ sudo chmod a+rwx public_document
student@72c6e332d705:/home/qwiklab/documents$ 
student@72c6e332d705:/home/qwiklab/documents$ ls -l public_document
-rwxrwxrwx 1 student student 14 Jun 30 00:21 public_document
student@72c6e332d705:/home/qwiklab/documents$ sudo chmod 777 public_document
student@72c6e332d705:/home/qwiklab/documents$ ls -l public_document
-rwxrwxrwx 1 student student 14 Jun 30 00:21 public_document
student@72c6e332d705:/home/qwiklab/documents$
```

Conclusion

Through this hands-on lab, I developed a strong command of Linux file permission management using both symbolic and numeric modes (`chmod`, `chown`). I worked across multiple scenarios — from securing sensitive documents to simulating publicly accessible files — and applied permission changes precisely, ensuring they aligned with security best practices and least privilege principles.

Beyond syntax, I focused on understanding how file access impacts system hardening and user role separation. I verified every permission state (`ls -l`, `ls -ld`) before and after making changes, just as I would when managing configuration files, logs, or automation scripts in a production SOC.

By combining ownership changes (`chown`) with controlled permission assignments (`chmod 764`, `chmod 777`, `u+x`, `g+w`), I demonstrated secure file governance practices — reducing the risk of unauthorized access or lateral movement (e.g., MITRE T1548, T1003).

This project reflects more than just command-line fluency — it showcases my ability to proactively manage access control at the OS level and reinforces my readiness to secure real-world systems in an enterprise SOC environment. 🖥️🔑🧠
