lab-01-linux-permissions — Instructor Key

Lab 01: Linux Permissions & Access Control

Instructor Key & Teaching Guide

Lab Duration: 45 minutes
Difficulty Level: Intermediate
Prerequisites: Basic command-line navigation, file system concepts

---

Learning Objectives Summary

By completing this lab, students will:

1. Diagnose permission and ownership problems from `ls -la` output
2. Apply `chmod` using both numeric (755) and symbolic (u+x) notation
3. Use `chown` to modify file ownership and group assignment
4. Execute security validation to identify world-writable files
5. Troubleshoot “Permission denied” errors in application contexts

---

Classroom Delivery Schedule & Lecture Outline

For instructors presenting a pre-lab lecture or interactive whiteboard briefing, reference the companion **`lecture_outline.md`** included in this lab package.

Suggested 75-Minute Class Period Schedule:

1. **00:00–00:05:** Incident Hook (The $10,000 `chmod 777` production outage) & scenario overview
2. **00:05–00:15:** Whiteboard lecture: 10-character permission string & octal binary matrix (4/2/1 weights)
3. **00:15–00:25:** Live CLI demo: File vs. directory execute bit differences and sticky bit enforcement
4. **00:25–01:10:** Hands-on lab execution (Students work through `student_handout.md`)
5. **01:10–01:15:** Post-lab debrief & formative reflection review

---

Setup Instructions for Instructors

Environment Provisioning

Option 1: Docker Container (Recommended)

```
docker run -it --name lab01-student -v "$PWD":/lab debian:12 bash
# Inside container:
bash /lab/setup.sh
```

`setup.sh` installs `sudo` itself if it is missing, so no manual prep is required. If the container has no network, setup prints a warning and students should substitute `su -c '<command>' appuser` for `sudo -u appuser <command>`.

Option 2: VM or Dedicated Lab System

```
sudo bash setup.sh
```

Verification:

```
ls -la /opt/secure_app/
# Should show broken permissions (644 on run.sh, root ownership,
# and a world-writable 777 logs/ directory with no sticky bit)
```

Pre-flight check (run this before class):

```
docker run --rm -v "$PWD":/lab debian:12 bash /lab/verify_lab.sh; echo "exit=$?"
```

This provisions, re-provisions (idempotency), asserts the baseline is actually broken, runs the full solution, and checks rubric/handout alignment. Exit code must be `0`.

Solution Walkthrough

Task 1: Investigate Current Permissions

Step 1.1 Expected Student Output:

```
drwxr-x--- 4 root root    4096 Oct  6 17:00 .
-rw-r--r-- 1 root root     287 Oct  6 17:00 run.sh
drwxr-xr-x 2 root root    4096 Oct  6 17:00 data
```

Completed Observation Table:

File/Directory	Owner	Group	Permissions (Symbolic)	Permissions (Numeric)
`/opt/secure_app/`	root	root	drwxr-x—	750
`/opt/secure_app/run.sh`	root	root	-rw-r–r–	644
`/opt/secure_app/data/`	root	root	drwxr-xr-x	755

Checkpoint Question 1 Answer:

```
1. The file is owned by root, but appuser is trying to execute it (ownership problem)
2. The file does not have execute permission (644 = rw-r--r--, missing 'x' bit)
```

Common Student Errors: - Only identifying one problem instead of both - Confusing ownership with permissions - Stating “wrong permissions” without specificity

---

Task 2: Fix Executable Permissions

Command:

```
sudo chmod 755 /opt/secure_app/run.sh
```

Expected Output After Verification:

```
-rwxr-xr-x 1 root root 287 Oct  6 17:00 /opt/secure_app/run.sh
```

Checkpoint Question 2 Answer:

```
No, the permission change alone does not solve the problem. The file is still owned by
root, so appuser cannot execute it even though it has execute permissions. The
ownership must also be changed.
```

Common Student Errors: - Assuming `chmod 777` is the correct solution (“it works but violates security”) - Not understanding the difference between permissions and ownership - Using symbolic notation incorrectly (`chmod +x` instead of `chmod u+x,g+x,o+x`)

Teaching Moment:
Emphasize that permissions define what actions are allowed, but ownership defines who can perform those actions. Both must align for functionality.

Task 3: Fix Ownership

Command:

```
sudo chown appuser:appteam /opt/secure_app/run.sh
```

Expected Output After Verification:

```
-rwxr-xr-x 1 appuser appteam 287 Oct  6 17:00 /opt/secure_app/run.sh
```

Test Execution Output:

```
=== Secure App Starting ===
User: appuser
Groups: appuser appteam
Timestamp: Mon Oct  6 17:30:42 UTC 2026
/opt/secure_app/run.sh: line 8: /opt/secure_app/data/startup.log: Permission denied
```

Analysis:
The script now runs but fails when attempting to write to the data directory. This demonstrates the cascading nature of permission problems.

Common Student Errors: - Using `chown appuser` without specifying group (leaves group as root) - Confusing order of `chown` syntax (should be `user:group`, not `group:user`) - Forgetting `sudo` prefix (results in “Operation not permitted”)

Task 4: Fix Data Directory Permissions

Commands:

```
sudo chown appuser:appteam /opt/secure_app/data
sudo chmod 2775 /opt/secure_app/data
```

Expected Output After Verification:

```
drwxrwsr-x 2 appuser appteam 4096 Oct  6 17:00 /opt/secure_app/data
```

Expected Terminal Diff:

```
- drwxr-xr-x 2 root    root    4096 data     (baseline: 755, root-owned, no setgid)
+ drwxrwsr-x 2 appuser appteam 4096 data     (solved:   2775, group-owned, setgid)
```

Why 2775 and not 775:
`775` alone is enough for this lab's single service account, but it is the wrong answer operationally. Without setgid, a second `appteam` member who creates a file in `data/` produces a file owned by their primary group, and the rest of the team silently loses access. The setgid bit forces group inheritance. Demonstrate it live:

```
$ sudo -u appuser touch /opt/secure_app/data/probe
$ ls -l /opt/secure_app/data/probe
-rw-r--r-- 1 appuser appteam 0 Oct  6 17:00 probe     # group = appteam, inherited
```

Remove the setgid bit (`chmod 775`) and repeat — the group becomes `appuser`. That side-by-side is the clearest 60 seconds in the lab.

Reading the mode string:
Lowercase `s` in the group field (`rws`) = setgid set **and** group-execute set — correct. Uppercase `S` (`rwS`) = setgid set but group-execute missing — almost always a student error, and the directory will not be traversable.

Final Test Execution Output:

```
=== Secure App Starting ===
User: appuser
Groups: appuser appteam
Timestamp: Mon Oct  6 17:32:15 UTC 2026
=== Secure App Running Successfully ===
```

Log File Verification:

```
$ cat /opt/secure_app/data/startup.log
Application started at Mon Oct  6 17:32:15 UTC 2026
```

Common Student Errors: - Setting directory permissions to 755 instead of 775 (group cannot write) - Only changing permissions without changing ownership - Using 777 permissions ("it works but violates security best practices")

Teaching Moment:
Directories require execute permission (`x`) for traversal. Without it, users cannot `cd` into the directory or access files within it, even if file permissions allow it.

Task 5: Security Review

World-Writable Files Check:

```
$ find /opt/secure_app -type f -perm -002
# (No output = correct — no world-writable files)
```

If Students Find World-Writable Files:
This indicates they used `chmod 777` or similar. Explain that world-writable files allow any user on the system to modify critical application files, enabling privilege escalation or data tampering.

Checkpoint Question 3 Answer:

```
The parent directory is set to 750 (not 755) to restrict access to the owner and group
only. The 'others' category has no permissions, preventing non-privileged users from
browsing the application directory. This follows the principle of least privilege.
```

Common Student Errors: - Stating “security” without explaining the specific threat model - Not understanding the significance of the “others” permission category

---

Task 6: Alternative Syntax Practice

Answers:

Numeric to Symbolic: 1. `chmod 644 file.txt` → `chmod u=rw,g=r,o=r file.txt` 2. `chmod 700 script.sh` → `chmod u=rwx,g=,o= script.sh` 3. `chmod 755 /opt/app` → `chmod u=rwx,g=rx,o=rx /opt/app`

Symbolic to Numeric: 1. `chmod u=rwx,g=rx,o=r file.txt` → `chmod 754 file.txt` 2. `chmod u=rw,g=r,o= private.key` → `chmod 640 private.key`

Grading Note:
Award partial credit if students demonstrate understanding of the permission bits (rwx = 7, r-x = 5, etc.) even if symbolic syntax is imperfect.

Reflection Questions - Model Answers

Question 1: Why is `chmod 777` a security risk?

Model Answer:

```
chmod 777 grants read, write, and execute permissions to all users (owner, group, and
others). This means any user on the system can modify or delete the file, or execute it
with their own privileges. In a multi-user environment, this allows attackers with low-
privileged accounts to:

- Tamper with application code (backdoor injection)
- Delete critical files (denial of service)
- Execute malicious scripts from within the application context

The principle of least privilege dictates that permissions should be as restrictive as
possible while still allowing legitimate functionality.
```

Grading Rubric (5 points): - Explains world-writable risk (2 pts) - Identifies specific attack vectors (2 pts) - References least-privilege principle (1 pt)

Question 2: Difference between `chown appuser` and `chown appuser:appteam`

Model Answer:

```
- `chown appuser /opt/secure_app/run.sh` changes ONLY the owner to appuser. The group
ownership remains unchanged (still root in this lab).
- `chown appuser:appteam /opt/secure_app/run.sh` changes BOTH the owner to appuser AND
the group to appteam.

In this lab, the second form is required because the appteam group needs access to the
file. If only the owner is changed, group members still cannot access it under the
'group' permission category.
```

Grading Rubric (3 points): - Correctly distinguishes owner-only vs. owner+group (2 pts) - Explains relevance to lab scenario (1 pt)

Question 3: Real-world scenario for `chmod 750`

Model Answer:

```
A web application directory (/var/www/myapp) where:

- The owner is the application service account (www-data) — needs full access (rwx)
- The group is the web server group (www-data) — needs read/execute for serving files
(r-x)
- Other users should have NO access (---) to prevent unauthorized users from browsing
application source code, configuration files, or database credentials

Example: `chmod 750 /var/www/myapp/config/` protects sensitive configuration files from
non-privileged users while allowing the web server to read them.
```

Grading Rubric (4 points): - Provides realistic scenario (1 pt) - Correctly maps permission bits to user categories (2 pts) - Explains security benefit (1 pt)

Common Troubleshooting Scenarios

Student Issue 1: “I ran chmod 755 but it still says permission denied”

Diagnosis: Student changed permissions but not ownership.

Solution:

```
# Check current ownership
ls -l /opt/secure_app/run.sh
# If owner is still root:
sudo chown appuser:appteam /opt/secure_app/run.sh
```

Teaching Moment: Permissions and ownership are independent. Both must be correct.

Student Issue 2: “I used chmod 777 and it works, why is that wrong?”

Diagnosis: Student prioritized functionality over security.

Solution:
Demonstrate vulnerability:

```
# As a low-privileged user:
echo "malicious code" >> /opt/secure_app/run.sh
```

Teaching Moment: In production, attackers exploit overly permissive settings. Always use minimum-necessary permissions.

Student Issue 3: “The application runs but doesn’t create the log file”

Diagnosis: Data directory permissions incorrect (755 instead of 775).

Solution:

```
# Check data directory permissions
ls -ld /opt/secure_app/data
# Should be 775 (drwxrwxr-x), not 755

sudo chmod 775 /opt/secure_app/data
```

Teaching Moment: Group write permission (w) is required for group members to create files.

Extension Activities for Advanced Students

Challenge 1: Implement Sticky Bit

```
# Apply sticky bit to data directory
chmod +t /opt/secure_app/data

# Verify
ls -ld /opt/secure_app/data
# Should show 't' in place of 'x' for others: drwxrwxr-t
```

Purpose: Prevents users from deleting files they don't own (useful for shared directories like /tmp).

Challenge 2: Use ACLs for Granular Control

```
# Install ACL utilities
apt-get install acl

# Grant specific user read-only access
setfacl -m u:auditor:r /opt/secure_app/run.sh

# Verify
getfacl /opt/secure_app/run.sh
```

Purpose: Demonstrates advanced access control beyond traditional user/group/other model.

Challenge 3: Audit Permissions Across System

```
# Find all SUID/SGID executables (potential privilege escalation vectors)
find /usr/bin -type f \( -perm -4000 -o -perm -2000 \) -ls
```

Purpose: Introduces security auditing and privilege escalation concepts (relevant for Security+ exam).

Grading Rubric Summary

Lab Completion Checklist (60 points): - Task 1: Correct observation table (10 pts) - Task 2: Correct chmod command and verification (10 pts) - Task 3: Correct chown command and

verification (10 pts) - Task 4: Correct data directory configuration (15 pts) - Task 5: Security review completed (5 pts) - Task 6: Syntax conversion exercises (10 pts)

Reflection Questions (40 points): - Question 1: Security risks of 777 (15 pts) - Question 2: chown syntax differences (10 pts) - Question 3: Real-world 750 scenario (15 pts)

Total: 100 points

Automated Validation

Instructor Verification Command:

```
sudo bash test_solution.sh
```

Expected Output:

```
=== Lab 01 Solution Validation ===
[Remediation output...]
[✓] PASS: run.sh has execute permission
[✓] PASS: run.sh owned by appuser
[✓] PASS: run.sh group is appteam
[✓] PASS: run.sh has 755 permissions
[✓] PASS: data directory owned by appuser
[✓] PASS: data directory group is appteam
[✓] PASS: data directory has 775 permissions
[✓] PASS: data directory is group-writable
[✓] PASS: parent directory is accessible
[✓] PASS: no world-writable files exist
[✓] PASS: appuser can execute run.sh
[✓] PASS: startup.log created successfully
[✓] PASS: appteam can write to data directory

=== Validation Summary ===
Tests Passed: 13
Tests Failed: 0

=== ✓ ALL TESTS PASSED ===
```

If Validation Fails:
Review `test_solution.sh` output to identify which specific test failed, then guide student to that configuration item.

Time Management Tips

Suggested Pacing: - Task 1 (Investigation): 10 minutes - Task 2 (Permissions): 5 minutes - Task 3 (Ownership): 5 minutes - Task 4 (Data Directory): 10 minutes - Task 5 (Security Review): 5 minutes - Task 6 (Syntax Practice): 5 minutes - Reflection Questions: 10 minutes (can be homework)

If Students Finish Early:
Assign extension activities (sticky bit, ACLs, system audit).

If Students Are Behind:
Prioritize Tasks 1-4 for functionality. Tasks 5-6 and reflection questions can be completed as homework.

Certification Exam Alignment

CompTIA Linux+ (XK0-005)

Domain 2.0: Security - Objective 2.3: Summarize file and directory permissions and ownership - Permission modes (numeric, symbolic) - Ownership (chown, chgrp) - Special permissions (sticky bit, SUID, SGID)

Sample Exam-Style Question:

```
A Linux administrator receives a report that an application script located at /opt/app/
start.sh
fails to execute for the service account "appuser". The output of `ls -l /opt/app/
start.sh` shows:
-rw-r--r-- 1 root root 1024 Oct 6 12:00 start.sh

Which of the following commands should the administrator run to resolve the issue?
(Choose TWO)

A. chmod 755 /opt/app/start.sh
B. chown appuser /opt/app/start.sh
C. chgrp appuser /opt/app/start.sh
D. chmod 777 /opt/app/start.sh
E. setfacl -m u:appuser:rwx /opt/app/start.sh

Answer: A, B
```

CompTIA A+ Core 2 (220-1102)

Domain 2.0: Security - Objective 2.6: Explain common methods for securing mobile and embedded devices - Access control models (Mandatory, Discretionary, Role-based) - Principle of least privilege

Connection to Lab:
This lab demonstrates discretionary access control (DAC) where file owners can modify permissions. Contrast with MAC (mandatory access control) used in SELinux.

Additional Teaching Resources

Video Demonstrations: - Recording a live walkthrough of the lab can help visual learners - Use `asciinema` to record terminal sessions: `asciinema rec lab01_demo.cast`

Visualization Tools: - Draw permission bit diagrams on whiteboard (rwx = 421 binary representation) - Create flow chart: "Permission Denied Error → Check Ownership → Check Permissions → Check Parent Directory"

Analogies: - **Ownership = Keys to a house** (who has access) - **Permissions = Locks on rooms** (what actions are allowed) - **World-writable = Leaving front door unlocked** (anyone can enter)

Revision History

Version	Date	Changes	Author
1.0	2026-10-06	Initial release	LabForge Curriculum Team

Instructor Feedback

After teaching this lab, please document:

- Average completion time
- Most common student errors
- Suggested improvements to handout clarity
- Extension activities attempted

Submit feedback to: curriculum@labforgestudio.com