lab-01-linux-permissions — Student Handout

Lab 01: Linux Permissions & Access Control

Estimated Time: 45 minutes
Difficulty: Intermediate
Certifications: CompTIA Linux+, A+ Core 2

Scenario Briefing

Your Role: Junior Linux Administrator (First Day)

Situation:
The development team deployed a new application to production last night. This morning, the support team reports the application won't start. Initial investigation shows the service account (`appuser`) cannot execute the startup script at `/opt/secure_app/run.sh`.

Your Mission:
Diagnose and correct the file permissions and ownership to restore functionality while maintaining security best practices.

A previous administrator also "fixed" the shared log directory with `chmod 777`. You will need to undo that.

Required Knowledge

Before starting this lab, you should understand:

- Basic Linux command line navigation
- File ownership concepts (user, group, others)
- Permission types (read, write, execute)
- Special permission bits: setuid, setgid, and the sticky bit

---

Before You Begin

All commands in this lab are run **as root**. If `sudo` is unavailable in your container, substitute `su -c '<command>' appuser` anywhere you see `sudo -u appuser <command>`.

Lab Environment

Test Command: Try running the application as `appuser`:

```
sudo -u appuser /opt/secure_app/run.sh
```

Expected Current Behavior: Permission denied error

Expected Final Behavior: Application starts successfully and writes to data directory

---

Task 1: Investigate Current Permissions

Step 1.1: Check the detailed permissions and ownership of the application directory:

```
ls -la /opt/secure_app/
```

Expected Output:

```
total 16
drwxr-x--- 4 root root    4096 Oct  6 17:00 .
drwxr-xr-x 3 root root    4096 Oct  6 17:00 ..
drwxr-xr-x 2 root root    4096 Oct  6 17:00 data
drwxrwxrwx 2 root root    4096 Oct  6 17:00 logs
-rw-r--r-- 1 root root     331 Oct  6 17:00 run.sh
```

Step 1.2: Record your observations in the table below:

File/Directory	Owner	Group	Permissions (Symbolic)	Permissions (Numeric)
`/opt/secure_app/`	root	root	drwxr-x—	750
`/opt/secure_app/run.sh`				
`/opt/secure_app/data/`				
`/opt/secure_app/logs/`				

Checkpoint Question 1:
Why can't `appuser` execute `/opt/secure_app/run.sh`? Identify **two** problems.

```
Your Answer:

1.

2.
```

Task 2: Fix Executable Permissions

Requirement: The `run.sh` script must be executable by its owner.

Step 2.1: Make the script executable using numeric notation:

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

Step 2.2: Verify the change:

```
ls -l /opt/secure_app/run.sh
```

Expected Output:

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

Step 2.3: Try running as `appuser` again:

```
sudo -u appuser /opt/secure_app/run.sh
```

Checkpoint Question 2:
Did the permission change solve the problem completely? Why or why not?

```
Your Answer:
```

Task 3: Fix Ownership

Requirement: The `appuser` account must own the executable.

Step 3.1: Change ownership of the script:

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

Step 3.2: Verify ownership:

```
ls -l /opt/secure_app/run.sh
```

Expected Output:

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

Step 3.3: Test execution again:

```
sudo -u appuser /opt/secure_app/run.sh
```

Expected Behavior: Script starts but fails when trying to write to `/opt/secure_app/data/`

Task 4: Fix Data Directory Permissions

Requirement: Members of `appteam` group must be able to write to the data directory.

Step 4.1: Change ownership of the data directory:

```
chown appuser:appteam /opt/secure_app/data
```

Step 4.2: Set permissions to allow group write access **and** set the setgid bit so that every file created here inherits the `appteam` group regardless of who creates it:

```
chmod 2775 /opt/secure_app/data
```

The leading `2` is the setgid bit. Without it, a file created by a different `appteam` member would be owned by that user's primary group, and the rest of the team would lose access to it.

Step 4.3: Verify permissions:

```
ls -ld /opt/secure_app/data
```

Expected Output:

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

Note the `s` in the group field (`rws`) — that is the setgid bit. A capital `S` would mean setgid is set but the group execute bit is not, which is almost always a mistake.

Step 4.4: Run the final test:

```
sudo -u appuser /opt/secure_app/run.sh
```

Expected Output:

```
=== Secure App Starting ===
User: appuser
Groups: appuser appteam
Timestamp: [current timestamp]
=== Secure App Running Successfully ===
```

Step 4.5: Verify the log file was created:

```
ls -l /opt/secure_app/data/startup.log
cat /opt/secure_app/data/startup.log
```

Task 5: Fix the Shared Log Directory (Sticky Bit)

Requirement: Every member of `appteam` must be able to drop log files into `/opt/secure_app/logs`, but **no member may delete another member's logs**. No user outside the team should have any access at all.

Step 5.1: Look at what the previous administrator left behind:

```
ls -ld /opt/secure_app/logs
```

Expected Output:

```
drwxrwxrwx 2 root root 4096 Oct  6 17:00 /opt/secure_app/logs
```

Checkpoint Question 3:
`777` does let the team write. Name **two** distinct things an attacker (or a careless teammate) could do in this directory that they should not be able to do.

```
Your Answer:

1.

2.
```

Step 5.2: Give the directory to the team and remove world access:

```
chown root:appteam /opt/secure_app/logs
```

Step 5.3: Apply group write plus the sticky bit:

```
chmod 1770 /opt/secure_app/logs
```

The leading `1` is the sticky bit. On a directory it means: you may only delete or rename a file here if you own that file. This is the same mechanism that protects `/tmp` (which is `1777`).

Step 5.4: Verify:

```
ls -ld /opt/secure_app/logs
```

Expected Output:

```
drwxrws--T 2 root appteam 4096 Oct  6 17:00 /opt/secure_app/logs
```

Wait — do you see `s` or `T` where you expected them? Read the mode carefully. The trailing `T` is the sticky bit with no world-execute permission.

Step 5.5: Prove the sticky bit works. There is a root-owned `audit.log` in that directory. Try to delete it as `appuser`:

```
sudo -u appuser rm -f /opt/secure_app/logs/audit.log
ls -l /opt/secure_app/logs/audit.log
```

Expected Behavior: The removal is refused (`Operation not permitted`) and `audit.log` is still there — even though `appuser` has write permission on the directory.

Step 5.6: Confirm `appuser` can still write its own logs:

```
sudo -u appuser touch /opt/secure_app/logs/appuser.log
ls -l /opt/secure_app/logs/
```

Checkpoint Question 4:
Write the symbolic-notation command that sets the sticky bit without changing any other permission bit on the directory.

```
Your Answer:
```

Task 6: Security Review

Step 6.1: Check for anything world-writable that is not protected by a sticky bit (a security violation):

```
find /opt/secure_app -perm -002 ! -perm -1000
```

Expected Output: (empty)

Note that this search covers directories as well as files. Searching only with `-type f` would have missed the `777` logs directory entirely — which is exactly the kind of gap that lets a misconfiguration survive an audit.

Step 6.2: Lock the parent directory to the team:

```
chown root:appteam /opt/secure_app
chmod 750 /opt/secure_app
ls -ld /opt/secure_app
```

Expected Output:

```
drwxr-x--- 4 root appteam 4096 Oct  6 17:00 /opt/secure_app
```

Step 6.3: Verify that a user outside the team is shut out:

```
useradd -r -s /bin/bash -m outsider
sudo -u outsider ls /opt/secure_app
```

Expected Behavior: `Permission denied`

Checkpoint Question 5:
Why is the parent directory (`/opt/secure_app`) set to `750` instead of `755`? What security principle does this enforce?

```
Your Answer:
```

Task 7: Alternative Syntax Practice

Challenge: Rewrite the following numeric permissions in symbolic notation:

1. `chmod 644 file.txt` → Symbolic: _____
2. `chmod 700 script.sh` → Symbolic: _____
3. `chmod 755 /opt/app` → Symbolic: _____
4. `chmod 2775 /opt/app/data` → Symbolic: _____
5. `chmod 1770 /opt/app/logs` → Symbolic: _____

Challenge: Convert symbolic to numeric:

1. `chmod u=rwx,g=rx,o=r file.txt` → Numeric: _____
2. `chmod u=rw,g=r,o= private.key` → Numeric: _____
3. `chmod u=rwx,g=rwxs,o= shared/` → Numeric: _____
4. `chmod u=rwx,g=rwx,o=rwx,+t /tmp` → Numeric: _____

Challenge: Read the mode string. For each, state whether the special bit is actually usable or represents a misconfiguration:

1. `drwxrws---` → _____
2. `drwxrwS---` → _____
3. `drwxrwx--T` → _____

Reflection Questions

Question 1:
In a production environment, why is `chmod 777` considered a security risk even if it "fixes" permission errors?

```
Your Answer:
```

Question 2:
What is the difference between `chown appuser /opt/secure_app/run.sh` and `chown appuser:appteam /opt/secure_app/run.sh`?

```
Your Answer:
```

Question 3:
Describe a real-world scenario where you would use `chmod 750` instead of `chmod 755` for a directory.

```
Your Answer:
```

Question 4:
`/tmp` on every Linux system is mode `1777`. Explain what would go wrong on a multi-user server if an administrator ran `chmod 777 /tmp`, removing the sticky bit.

```
Your Answer:
```

Question 5:
The setgid bit on a directory and the setgid bit on an executable file do completely different things. Describe both.

Your Answer:

Lab Completion Checklist

- [ ] `appuser` can execute `/opt/secure_app/run.sh` without errors
- [ ] Application successfully writes to `/opt/secure_app/data/startup.log`
- [ ] `/opt/secure_app/data` is `2775` and new files inherit the `appteam` group
- [ ] `/opt/secure_app/logs` is `1770`, owned `root:appteam`
- [ ] `appuser` can create logs but cannot delete the root-owned `audit.log`
- [ ] Nothing world-writable remains (except where a sticky bit protects it)
- [ ] A user outside `appteam` cannot traverse `/opt/secure_app`
- [ ] Ownership follows least-privilege principle
- [ ] All checkpoint questions answered

Instructor Verification: Submit this handout with completed answers for grading.

Troubleshooting Tips

Problem: “Permission denied” when running `chmod` or `chown`
Solution: These commands require root privileges. Use `sudo` prefix.

Problem: “Operation not permitted” when changing ownership
Solution: Only root can change file ownership. Verify you’re using `sudo`.

Problem: Application still can’t write to data directory
Solution: Check both ownership AND permissions. Both must be correct.

Additional Resources

- `man chmod` — Manual page for chmod command
- `man chown` — Manual page for chown command
- CompTIA Linux+ Exam Objectives (XK0-005)
- CompTIA A+ Core 2 Exam Objectives (220-1102)