

Users and Access Control

Notes

Security conversations go wrong when access control is treated as a set of bits rather than as a set of trust decisions.

“Permissions are wrong.”

“Just chmod it.”

“Add them to the group for now.”

Those statements do not tell a team who can reach what, what actions are possible from there, or whether the access model aligns with the system's intentions.

The goal here is to make access control readable in plain language. Users and groups are the identities. Permissions are the recorded rules. Linux will enforce whatever is recorded.

The core idea

Access control is the system's record of trust decisions.

If the record is too generous, the system will do the wrong thing reliably. Most OS-layer risk is not advanced exploitation. It is ordinary access decisions that drift.

Common drift patterns:

- Someone adds a user to an influential group “temporarily,” and it never gets reversed.
- A directory becomes group-writable to unblock work and stays that way.
- A shared identity accumulates authority because attribution is inconvenient.

If you forget everything else, access control is trust, written down.

The two questions to ask

When you see a permissions-related finding, translate it into plain language:

1. Who can reach this?
2. What can they do from there?

If you cannot write the sentence “These identities can do X to this thing,” you do not yet understand the risk.

If you forget everything else, translate bits into people and actions.

Users

A user is a named identity. Processes run as a user.

Users matter because:

- A process inherits the authority of the user it runs as.
- Files and directories have an owning user.
- Audit and accountability depend on actions being tied to a real identity.

If you forget everything else, a user is who the system thinks is acting.

Groups

A group is a named set of users.

Groups matter because they are the main way access spreads:

- One group membership change can expand reach across many paths.
- Shared directories, shared tools, and shared workflows often rely on group permissions.

If you forget everything else, group membership is a trust boundary.

Permissions in Linux

Linux uses three scopes on every file and directory:

- Owner (u) is the owning user
- Group (g) is the owning group
- Other (o) is everyone else

Each scope has three bits:

- `r` is read
- `w` is write
- `x` is execute

What `r`, `w`, `x` mean for files:

- `r`: read file contents
- `w`: modify file contents, including truncation
- `x`: execute the file

What `r`, `w`, `x` mean for directories:

- `r`: list directory entries, meaning see filenames

- **w**: create, delete, rename entries, subject to rules
- **x**: traverse the directory, meaning enter it and access items inside if allowed

Common surprise: directory execute (meaning traverse) often matters more than directory read (meaning list).

If you forget everything else: directory execute means “can pass through,” not “can run.”

Octal permissions

Each bit has a numeric value:

- **r** = 4
- **w** = 2
- **x** = 1

Add them to get one digit per scope, in order of owner, group, and other.

Examples:

- 644 means owner **rw-**, group **r--**, other **r--**
- 755 means owner **rw-x**, group **r-x**, other **r-x**

If you forget everything else, octal is compressed to r, w, and x for owner, group, and other.

Permissions Table

r	w	x	Octal	String	Meaning for files	Meaning for directories
0	0	0	0	---	cannot read, write, or execute	cannot list, enter, or modify
0	0	1	1	--x	can execute	can enter if name is known
0	1	0	2	-w-	can modify or truncate	entries can be modified only if execute is also present
0	1	1	3	-wx	can write and execute	can enter and modify entries, no listing
1	0	0	4	r--	can read	can list names only
1	0	1	5	r-x	can read and execute	can list and enter, no modification

r	w	x	Octal	String	Meaning for files	Meaning for directories
1	1	0	6	rw-	can read and write	Names can be listed, entries can be modified only if execute is also present
1	1	1	7	rwx	can read, write, and execute	can list, enter, and modify

Examples

The Shared Export Folder

A customer support team exports contact lists each week to reconcile tickets and follow up with clients. The exports are placed in a shared directory so multiple staff can help during busy periods. Over time, the group attached to that directory grows as new staff rotate in. No one revisits whether everyone still needs access.

Customer contact export

→ Routine staff login

→ Staff member is in a broad group with read access

→ Private customer emails become visible to an unauthorized person.

This is confidentiality because the system continues to function, and no data is changed or destroyed. The failure is that information becomes visible to someone who should not have access to it. The observable damage is loss of privacy and breach of trust around customer contact data.

The Editable Rules File

A reporting service calculates totals using a rules file stored alongside other shared resources so multiple teams can adjust pricing logic quickly. To avoid delays, the directory is made group-writable. The service keeps running normally and continues producing reports.

Report rule file

→ Routine maintenance workflow

→ Non-owner can write the rules directory because it is group-writable

→ Reports contain incorrect totals that appear legitimate.

This is primarily integrity because the system remains available and produces output, but the output is no longer correct or trustworthy. The observable damage is incorrect reports that look valid.

The Shared Output Directory

A scheduled job generates status files that downstream systems depend on. To simplify troubleshooting, the output directory is writable by many users. Over time, the directory fills with ad-hoc files, and occasional manual cleanup removes items the job expects to find.

Job output directory

- Scheduled job runs

- Broad write access enables deletion or exhaustion

- Job fails repeatedly, and required outputs are missing or late.

This is availability because the primary observable harm is failure to deliver a required function on time. The system cannot produce outputs when needed, even if the data was not stolen or altered.

The Shared Admin Account

To speed up operations, several staff members share a single administrative account. That account owns service scripts and configuration files and has access to customer data. Changes are made quickly, but attribution is lost.

Service scripts and customer data

- Routine operational work uses a shared admin identity

- Shared identity has broad read and write authority without attribution

- Incorrect service changes occur, private data can be viewed, and containment requires disruptive shutdowns

This is primarily integrity because the most stakeholder-visible failure is that service behaviour and outputs become incorrect due to unauthorized or unreviewed changes. The same weak spot also enables confidentiality harm by allowing the shared authority to read sensitive data, and availability harm because containment often requires pausing services when you cannot bound what else the shared identity could have changed. Cascading issues still retain a primary classification for prioritization, but their defining feature is that a single access condition can cause multiple types of damage.

What to notice

- Users and groups express human trust decisions. Permissions encode those decisions.
- Every access finding reduces to who can reach something and what they can do once they get it.
- Group membership changes can widen reach across many paths at once.
- Shared identities erase accountability and expand blast radius.
- You should understand users and groups before you debate permission values.