Linux Basics

Study Guide

Linux is an operating system you can name. The chapter's job is to make you able to anchor any security claim to a specific host, a real running process, the paths it touches, and the interfaces that trigger or influence it, starting from two invariant questions and treating placement and persistence as evidence.

What you must remember

The anchor model

Host → Process → Path → Interface

If you can reliably produce these four anchors, you can turn abstract statements into inspectable host facts and make risk chains reviewable.

What each word means

Component	Meaning
Host	The specific machine where the observation applies, not a role or function.
Process	What is actually executing on that host, and whether it persists over time.
Path	The files and directories the process reads from and writes to, including config, logs, secrets at rest, and output.
Interface	How the process is started, triggered, or influenced, including service managers, schedulers, IPC, file ingestion points, and network listeners.

The two invariant questions

1. What is running?
2. What does it touch?

"What it touches" includes both paths and interfaces.

The three tests

An OS-level claim is ready for review only if it passes:

- Specific Host Test: The claim names a concrete host you could revisit.
- Runtime Truth Test: The claim identifies a real process or recurring trigger that exists now.
- Boundary Test: The claim names the path or interface where influence crosses a boundary.

Rule: If a claim fails a test, rewrite only the failing part.

CIA, used correctly

CIA is outcome vocabulary, not an explanation.

- Confidentiality: The wrong party saw information.
- Integrity: Information or behaviour is incorrect.
- Availability: A required function fails when needed.

Rule: Apply CIA only after the damage is clear in a risk chain.

Placement is evidence

Location carries meaning. When a binary, config, secret, log, or dataset is not where you expect it to be, the mismatch is often the story.

Unusual placement often indicates:

- drift from previous deployments
- troubleshooting artifacts that never got removed
- a process running outside the intended lifecycle
- multiple competing sources of truth for configuration

Service vs activity

A recurring mistake is confusing activity with presence.

- One-off command: runs, does work, exits.
- Service: persists, restarts, creates durable behaviour.

Rule: If a claim depends on ongoing behaviour, there should be persistence somewhere, such as a long-lived process, a scheduler, or a stable interface.

The friction rule

Good security makes the wrong thing expensive and the right thing cheap.

As a target: increase the cost of unauthorized use by about 100× while increasing authorized routine work by less than 1.1×.

The core process to practice

Use this sequence until it becomes automatic:

1. Read for hosts, runtime, and boundaries. Identify candidate hosts, processes, paths, and interfaces.
2. Choose one host. Name it precisely and treat it as the unit of place.
3. State what is running. Identify the process or recurring trigger that creates behaviour.
4. List what it touches. Name the key paths it reads and writes and the interfaces that can influence it.
5. Use placement and persistence as evidence. Ask what is usual on this host role and what deviates.
6. Map to a risk chain. Convert the observation into Asset → Entry Point → Weak Spot → Damage.
7. Run the anchor tests. Fix only the failing component. Repeat.
8. Classify with CIA. Pick the primary harm and justify it from the damage.
9. (Optional) Improve friction. Propose one change that raises resistance without disrupting everyday work.

Common traps to watch for

- Treating Linux as a blur and using role names instead of hostnames.
- Describing what should be running instead of what is running.
- Using tool output as a conclusion instead of as evidence to interpret.
- Confusing a path name with an explanation.
- Ignoring persistence and missing scheduled or supervised execution.
- Calling something reachable without naming the boundary and trigger.
- Jumping to permission fixes before describing host, process, path, and interface.

Readiness questions

Answer these without looking anything up. If you cannot answer one, it tells you exactly what to practice next.

Model recall

1. Write Host → Process → Path → Interface from memory.

2. In one sentence each, define Host, Process, Path, and Interface.

Identification skill

3. Given a short description, can you name a host as a specific machine rather than a role?
4. Can you name a process as a runtime fact, not a general service label?
5. Can you name a concrete path, such as a file or directory, that could be checked today?
6. Can you name an interface as a trigger or boundary, not a vague reachable statement?

Anchor test discipline

7. For each anchor test, what would a failing answer look like?
8. When a claim fails one test, can you rewrite only the failing component and leave the rest intact?

Risk chain connection

9. Explain the difference between a Linux anchor set and a complete risk chain.
10. Given a host-level observation, can you produce Asset → Entry Point → Weak Spot → Damage without using attack terms?

Friction thinking

11. What is one example of a change that reduces unintended reachability without adding much friction to regular operation?
12. What is one example of a change that sounds like hardening but does not change the relevant path or interface?

Practice questions

Answer these in writing. Aim for short, precise answers. If you get stuck, that is the point: it tells you what part of the process still needs practice.

Definitions and recall

1. Write Host → Process → Path → Interface from memory.
2. Define Host in one sentence. Include the word “specific.”
3. Define Process in one sentence. Include the word “running.”
4. Define Path in one sentence. Include the word “files.”
5. Define Interface in one sentence. Include the word “trigger.”
6. In one sentence, explain why placement is evidence.

Extraction from descriptions

7. When reading a description, what are the two invariant questions you answer first?
8. What is a sign that you have chosen a host description that is too broad?
9. What is a sign that your process description assumes runtime behaviour you have not observed?
10. What is a sign that your path description is not checkable today?
11. What is a sign that your interface description is missing the boundary and trigger?

Writing a reviewable claim

12. Write a sentence template you can use to name a host without describing its purpose.
13. Write a sentence template you can use to state a process as a runtime truth.
14. Write a sentence template you can use to state a path that captures what the process touches.
15. Write a sentence template you can use to state an interface as a trigger across a boundary.
16. What is the minimum information your anchor set must contain to be reviewable by someone else?

Anchor test practice

17. Give an example of a host statement that would fail the Specific Host Test.
18. Give an example of a runtime statement that would fail the Runtime Truth Test.
19. Give an example of an interface statement that would fail the Boundary Test.
20. If only the Boundary Test fails, what part are you allowed to rewrite?
21. Why is rewriting only the failing component important?

CIA connection

22. A damage statement says that secrets can be read by identities that should not access them. Which CIA category is primary, and why?
23. A damage statement says that a legacy process can generate reports through an undocumented path while still appearing legitimate. Which CIA category is primary, and why?
24. A damage statement says that a scheduled cleanup deletes working state and report generation intermittently fails. Which CIA category is primary, and why?
25. What is one reason it can be misleading to list multiple CIA categories without choosing a primary one?

Confidence check

26. Which anchor (Host, Process, Path, Interface) do you personally find hardest to make concrete, and why?

27. When you revise an anchor set, what is the one thing you check first to see whether the revision actually improved it?