07 — Permissions & Sandboxing

🕒 state of the art 2026-07 · revised 2026-07-25 · 📖 ~9 min read

Learning objectives

By the end of this chapter, you should be able to:

1. **Distinguish** the two layers of defense — policy (what the agent may request) and containment (what the process can actually do);
2. **Design** permissions along two orthogonal dimensions (sandbox mode × approval policy);
3. **Apply** the "lethal trifecta" and the "rule of two" as checklists for toolset review and session architecture;
4. **Implement** a `PermissionPolicy` as pure domain (testable without an LLM (Large Language Model)) + non-disableable sensitive paths (step 6);
5. **Evaluate** a real harness for its *blast radius* — what leaks if the injection wins?

The problem

An agent with a shell is a user with a shell: it can delete files, exfiltrate credentials, make network calls. The control mechanisms answer two distinct threats: the **mistake** (the model does something destructive by accident) and the **attack** (prompt injection convinces the model to act against the user). It is the dimension of greatest divergence among the harnesses — a sign that the industry has not yet converged, though it is converging fast.

Two levels, frequently conflated: **permissions** (policy: approval, allowlists, modes) and **sandbox** (containment: OS-imposed limits, even when policy fails).

Scientific foundations

- **The threat, defined** — *Not what you've signed up for* (Greshake et al., arXiv 2302.12173): indirect injection — instructions planted in data the agent is going to read — is the vector that no traditional code vulnerability captures.
- **The map of defenses** — the layered attack-surface survey (arXiv 2604.23338) and the agentic security survey (arXiv 2510.06445) organize threats and defenses; the computer-using agents survey (arXiv 2505.10924) focuses on those who have a shell.

(Full bibliography: `livro/bibliografia.md`.)

Industry sources

- **Making Claude Code more secure with sandboxing** (Anthropic): containment on OS primitives (bubblewrap/Seatbelt), workspace-only writes, **network denied by default** — and egress goes through a proxy that runs *outside* the sandbox and enforces a per-domain allowlist. The network boundary is a separate, privileged component, not a bypassable in-process check.
- **How we contain Claude across products** (Anthropic): three regimes (ephemeral gVisor, OS sandbox + approval, sealed VM with credentials outside the guest) and the central thesis — **hard, deterministic boundaries before probabilistic model defenses**. Honest detail: the egress proxy itself broke twice — treat your proxy as the most fragile component, not the most trusted.
- **Agent approvals & security** (OpenAI Codex): the matrix of **two orthogonal axes** — sandbox mode (`read-only`/`workspace-write`/`danger-full-access`) × approval policy (`untrusted`/`on-request`/`on-failure`/`never`), with `on-failure` firing the prompt only *after* the sandbox blocks. The most copyable design pattern on the market.
- **Agents Rule of Two** (Meta AI): an agent should not satisfy more than two of the three — processing untrusted input, accessing sensitive data, changing state/communicating externally — in the same session. A *session architecture* criterion, not a substitute for defense-in-depth.
- **The lethal trifecta** (Simon Willison): private data + untrusted content + external communication = exfiltration. Use it as a **toolset checklist**: which corner does each new tool close? The counterpoint: announced defenses fall when "the attacker moves last".
- **Attacks on OpenClaw** (The Hacker News): the real-world case — one-click RCE (CVE-2026-25253), plaintext credentials, injection planted in an email signature/calendar invite/issue. The vector was not the model, it was the **harness**: secrets in the same space as the tools + unlimited untrusted input.
- **See also**: the living collections Awesome Harness Engineering — Permissions & Authorization and Awesome Harness Engineering — Security, Sandbox & Permissions gather more consultable resources for this dimension (patterns, articles, and implementations), curated by problem.

The state of the art

1. Two orthogonal dimensions, not a slider

The old mental model ("YOLO ↔ ask about everything") is dead. The consensus is to separate **maximum physical capability** (sandbox) from **when to escalate to a human** (approval policy) — independently configurable. Codex is the canonical example (mode × policy, with `on-failure`). And there are two containment paradigms the benchmark separated:

- **OS containment** (the process *cannot*): Seatbelt + bubblewrap/seccomp + Landlock in Codex; 6 Seatbelt profiles + Docker in gemini-cli; fail-closed WASM + per-tenant Docker in IronClaw;
- **authority architecture** (the loop *cannot reach*): IronClaw makes the loop structurally incapable of acting without the kernel — type-unforgeable trust class, approvals as per-invocation leases, verified by dependency tests. No harness fully combines the two yet — it is the open frontier.

2. Policy without containment is a bet on the model's obedience

The cross-cutting lesson of the benchmark: harnesses with elegant policy but no OS sandbox (opencode, ohmo) are betting the model obeys. Three cheap, exportable defenses the state of the art has consolidated: **non-disableable sensitive paths** (OpenHarness's `SENSITIVE_PATH_PATTERNS` — denies `.ssh`, credentials, `.kube/config` before any user rule, explicitly against injection); **structural shell parsing** before judging (gemini-cli's policy engine understands redirections and wrappers; software-agent-sdk's `defense_in_depth` detects compositions like fetch-to-exec via AST); and **credentials outside the process** (injected at the egress edge, never in the tools' space — IronClaw, and the direct lesson of the OpenClaw case).

3. Prompt injection is treated as unsolvable — the effort migrated to blast radius

The 2026 consensus, from the model to the vendors: you cannot reliably "detect" injection. The work migrated to **designing sessions that never accumulate the trifecta** (rule of two as the criterion for when to break context), **isolating credentials** (keychain on the host, sealed VM) and **controlling egress** (per-domain allowlist via an external proxy). In the personal-agent category, the third-party vector earned its own defense: **deny-by-default contact pairing/allowlisting** (OpenClaw, ohmo) and a **`non-main` sandbox** for every session that is not the owner's. And the emerging honesty norm: publishing the gate's **false-negative rates** (Claude Code's auto mode is discussed with numbers in both directions) instead of asserting binary security.

> **EXECUTIVE SUMMARY**
>
> What is most modern: the two orthogonal dimensions; the two containment paradigms (OS × authority) and the finding that nobody has combined them; and the migration from "detect injection" to "reduce blast radius" (trifecta/rule-of-two as checklists, credentials outside the process, controlled egress). **What to steal:** non-disableable sensitive paths; structural shell parsing; `on-failure` (approve only after the block); contact pairing; publishing the gate's false-negative rate.

Hands-on — harness-zero, step 6

Step 6 introduces the `PermissionPolicy` as **pure domain**: a function `(ação, contexto) → allow | ask | deny` that knows nothing of LLMs or chat — testable in isolation (it is the "isolated domain" DDD names, and the test runs without a network). You implement: the three verdicts (allow/ask/deny), the non-disableable sensitive paths, and **inline approval in the chat** (the front end pauses and asks — the visible manifestation of the policy). Completeness exercise: rule evaluation ships ready; you add minimal parsing of a shell command before judging it.

Check your understanding

1. A harness has only an approval policy, with no OS sandbox. Which class of attack can it not contain, and why? (Policy without containment; the model can be persuaded.)
2. You are about to add an email-sending tool to an agent that already reads GitHub issues and has access to the private repository. Apply the lethal trifecta. (It closes the third corner → exfiltration possible.)

3. Why can `on-failure` (approve only after the block) be better than `on-request` (approve before each action)? (Friction × coverage; the sandbox filters out what never needs a human.)

Appendix A — How each repository handles permissions and sandboxing

> Per-harness evidence, with paths — online supplement, expanded each round.

gemini-cli (round 1) ⭐ policy engine + OS sandbox

`packages/core/src/policy/policy-engine.ts`: prioritized rules with **structural shell parsing** (`parseCommandDetails`, `stripShellWrapper`, redirection detection), rules in TOML; 4 `ApprovalMode`s; **6 Seatbelt profiles** (`sandbox-macos-*.sb`) + Docker/Podman with proxy; **trusted folders** gatekeeping hooks/agents.

OpenHarness (round 1) ⭐ sensitive paths

`permissions/checker.py`: path rules, denied commands, 3 modes; **hardcoded, non-disableable** **`SENSITIVE_PATH_PATTERNS`** (`.ssh`, `.aws/credentials`, `.gnupg`, `.kube/config`) against injection; sandbox via `sandbox-runtime`/Docker with a domain allowlist; `trust_env=False` in the web tools (anti-SSRF).

opencode (round 1) — policy without containment

`permission/`: rulesets with wildcards (`allow | ask | deny`, last-match-wins, default `ask`), approval via `Deferred` + event; **subagents derive restricted permissions**; **no OS sandbox in the core** (containers only in enterprise).

Codex CLI (round 2) ⭐ OS containment in 3 layers

`sandboxing/` + `linux-sandbox/` + `windows-sandbox-rs/`: Seatbelt via `sandbox-exec` (anti-tamper hardcoded path), embedded bubblewrap + **seccomp** + `NO_NEW_PRIVS`, legacy Landlock; `AskForApproval` incl. `Granular`; per-command **execpolicy in Starlark**; `assess_patch_safety`; network-proxy.

Goose (round 2)

`permission/`: `GooseMode` modes (Auto/Approve/Chat); **`permission_judge` uses an LLM** to classify read-only; per-signature `ToolPermissionStore` with expiration; light execution isolation (direct shell; external Docker).

OpenClaw (round 2) ⭐ third-party pairing

`src/pairing/` + `docs/security/THREAT-MODEL-ATLAS.md`: **DMs as untrusted input**, `dmPolicy: "pairing"` default (pairing code, SQLite allowlist); multi-backend sandbox (Docker

`network:none`/`readOnlyRoot`/`capDrop:ALL`, SSH, OpenShell) with a **`non-main`** mode; `openclaw doctor`/`security audit`; caveat: `sandbox.mode` off by default in the main session.

Hermes (round 2)

`tools/approval.py` (detection + allowlist), per-thread callbacks; **six isolated terminal backends** (local, Docker, SSH, Singularity, Modal, Daytona); subagents with safe-by-default `_subagent_auto_deny`; anti-traversal `path_security.py`.

IronClaw (round 2) ⭐⭐ authority architecture

`crates/ironclaw_authorization` + `_approvals` + `_trust` + `_wasm` + `_process_sandbox` + `_secrets` + `_network` + `_safety`: exact-invocation authorization (fail-closed), approvals as **per-invocation leases with fingerprint**, **type-unforgeable trust class** (`#[serde(skip_deserializing)]`), WASM (fuel/memory/rate, egress denied), per-tenant Docker, zero-exposure secrets at the egress edge, anti-SSRF, bidirectional leak detector — the loop cannot reach the effects (verified by dependency tests).

ohmo (round 2.5) — the right half

`channels/impl/base.py`: **deny-by-default** allowlist + per-sender session isolation + blocking of remote admin commands + OpenHarness's sensitive paths. Gap: `permission_mode`/`sandbox_enabled` in `gateway.json` are **dead code** — no dial between deny-everything and full_auto.

software-agent-sdk (frameworks round)

`sdk/security/`: risk analysis (LLM analyzer + deterministic `defense_in_depth/` with an AST shell parser detecting **fetch-to-exec**) + confirmation policy (`AlwaysConfirm`/`ConfirmRisky` by threshold); the conversation **returns** in `WAITING_FOR_CONFIRMATION` (it does not block); secret masking.

n8n (round 2) — structural permission

Permission is **topological**: the author chooses which nodes sit on the `AiTool` port — allowlist by construction. Real HITL via `sendAndWait` (durable pause), forbidden in sub-agents; Guardrails node.

Frameworks (frameworks round) — left open

LangGraph and CrewAI have no native tool policy (you build it on `interrupt`/HITL); the Agents SDK (Software Development Kit) has guardrails at three levels (agent/run/tool) as a primitive, but containment is left to the adopter.