07 — Permissões e Sandboxing

🕒 estado da arte 2026-07 · revisão 2026-07-25 · 📖 ~9 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Distinguir** as duas camadas de defesa — política (o que o agente pode pedir) e contenção (o que o processo consegue fazer);
2. **Projetar** permissões em duas dimensões ortogonais (modo de sandbox × política de aprovação);
3. **Aplicar** a "trifecta letal" e a "regra de dois" como checklists de revisão de toolset e de arquitetura de sessão;
4. **Implementar** uma `PermissionPolicy` como domínio puro (testável sem LLM (Large Language Model)) + paths sensíveis indesligáveis (etapa 6);
5. **Avaliar** um harness real quanto ao seu *blast radius* — o que vaza se a injection vencer?

O problema

Um agente com shell é um usuário com shell: pode apagar arquivos, exfiltrar credenciais, fazer chamadas de rede. Os mecanismos de controle respondem a duas ameaças distintas: o **erro** (o modelo faz algo destrutivo por engano) e o **ataque** (prompt injection convence o modelo a agir contra o usuário). É a dimensão de maior divergência entre os harnesses — sinal de que a indústria ainda não convergiu, embora esteja convergindo rápido.

Dois níveis, frequentemente confundidos: **permissões** (política: aprovação, allowlists, modos) e **sandbox** (contenção: limites impostos pelo SO, mesmo que a política falhe).

Fundamentos científicos

- **A ameaça, definida** — *Not what you've signed up for* (Greshake et al., arXiv 2302.12173): a injection indireta — instruções plantadas em dados que o agente vai ler — é o vetor que nenhuma vulnerabilidade de código tradicional captura.
- **O mapa das defesas** — o survey de superfície de ataque em camadas (arXiv 2604.23338) e o de segurança agêntica (arXiv 2510.06445) organizam ameaças e defesas; o de computer-using agents (arXiv 2505.10924) foca em quem tem shell.

(Bibliografia completa: `livro/bibliografia.md`.)

Fontes da indústria

- **Making Claude Code more secure with sandboxing** (Anthropic): contenção sobre primitivas de SO (bubblewrap/Seatbelt), escrita no workspace, **rede negada por padrão** — e o egress passa por um proxy que roda *fora* do sandbox e faz allowlist por domínio. A fronteira de rede é um componente separado e privilegiado, não uma checagem in-process contornável.
- **How we contain Claude across products** (Anthropic): três regimes (gVisor efêmero, sandbox de SO + aprovação, VM selada com credenciais fora do guest) e a tese central — **fronteiras duras e determinísticas antes de defesas probabilísticas do modelo**. Detalhe honesto: o próprio proxy de egress quebrou duas vezes — trate seu proxy como o componente mais frágil, não o mais confiável.
- **Agent approvals & security** (OpenAI Codex): a matriz de **dois eixos ortogonais** — modo de sandbox (`read-only`/`workspace-write`/`danger-full-access`) × política de aprovação (`untrusted`/`on-request`/`on-failure`/`never`), com o `on-failure` disparando o prompt só *depois* do bloqueio do sandbox. O padrão de design mais copiável do mercado.
- **Agents Rule of Two** (Meta AI): um agente não deve satisfazer mais de dois dos três — processar input não confiável, acessar dados sensíveis, mudar estado/comunicar externamente — na mesma sessão. Critério de *arquitetura de sessão*, não substituto de defense-in-depth.
- **The lethal trifecta** (Simon Willison): dados privados + conteúdo não confiável + comunicação externa = exfiltração. Use como **checklist de toolset**: cada tool nova fecha qual vértice? O contraponto: defesas anunciadas caem quando "o atacante move por último".
- **Ataques ao OpenClaw** (The Hacker News): o caso real — RCE one-click (CVE-2026-25253), credenciais em texto plano, injection plantada em assinatura de e-mail/convite de calendário/issue. O vetor não foi o modelo, foi o **harness**: segredos no mesmo espaço das tools + entrada não confiável ilimitada.
- **Consulte também**: a coleção viva Awesome Harness Engineering — Permissions & Authorization e Awesome Harness Engineering — Security, Sandbox & Permissions reúnem mais recursos consultáveis desta dimensão (padrões, artigos e implementações), curados por problema.

O estado da arte

1. Duas dimensões ortogonais, não um slider

O modelo mental antigo ("YOLO ↔ pergunte tudo") morreu. O consenso é separar **capacidade física máxima** (sandbox) de **quando escalar ao humano** (política de aprovação) — configuráveis independentemente. O Codex é o exemplo canônico (modo × política, com `on-failure`). E há dois paradigmas de contenção que o benchmark separou:

- **contenção por SO** (o processo *não consegue*): Seatbelt + bubblewrap/seccomp + Landlock no Codex; 6 perfis Seatbelt + Docker no gemini-cli; WASM fail-closed + Docker per-tenant no IronClaw;
- **arquitetura de autoridade** (o loop *não alcança*): o IronClaw torna o loop estruturalmente incapaz de agir sem o kernel — trust class inforjável por tipos, aprovações como leases por invocação, verificado por testes de dependência. Nenhum harness combina os dois plenamente ainda — é a fronteira aberta.

2. Política sem contenção é aposta na obediência do modelo

A lição transversal do benchmark: harnesses com política elegante mas sem sandbox de SO (opencode, ohmo) apostam que o modelo obedece. Três defesas baratas e exportáveis que o estado da arte consolidou: **paths sensíveis indesligáveis** (`SENSITIVE_PATH_PATTERNS` do OpenHarness — nega `.ssh`, credenciais, `.kube/config` antes de qualquer regra de usuário, explicitamente contra injection); **parsing estrutural de shell** antes de julgar (o policy engine do gemini-cli entende redirecionamentos e wrappers; o `defense_in_depth` do software-agent-sdk detecta composições como fetch-to-exec via AST); e **credenciais fora do processo** (injetadas na borda de egress, nunca no espaço das tools — IronClaw, e a lição direta do caso OpenClaw).

3. Prompt injection é tratada como não-resolvível — o esforço migrou para o blast radius

O consenso de 2026, do modelo aos vendors: não se "detecta" injection de forma confiável. O trabalho migrou para **desenhar sessões que nunca acumulam a trifecta** (regra de dois como critério de quando quebrar contexto), **isolar credenciais** (keychain no host, VM selada) e **controlar egress** (allowlist por domínio via proxy externo). Na categoria de agentes pessoais, o vetor de terceiros ganhou defesa própria: **pairing/allowlist de contatos deny-by-default** (OpenClaw, ohmo) e **sandbox `non-main`** para toda sessão que não seja a do dono. E a norma emergente de honestidade: publicar **taxas de falso-negativo** do gate (o auto mode do Claude Code é discutido com números nos dois sentidos) em vez de afirmar segurança binária.

> **LEITURA EXECUTIVA**
>
> O que está mais moderno: as duas dimensões ortogonais; os dois paradigmas de contenção (SO × autoridade) e a constatação de que ninguém os combinou; e a migração de "detectar injection" para "reduzir blast radius" (trifecta/regra-de-dois como checklists, credenciais fora do processo, egress controlado). **O que roubar:** paths sensíveis indesligáveis; parsing estrutural de shell; `on-failure` (aprovar só após o bloqueio); pairing de contatos; publicar a taxa de falso-negativo do gate.

Mão na massa — harness-zero, etapa 6

A etapa 6 introduz a `PermissionPolicy` como **domínio puro**: uma função `(ação, contexto) → allow | ask | deny` que não conhece LLM nem chat — testável isoladamente (é o "domínio isolado" que o DDD nomeia, e o teste roda sem rede). Você implementa: os três veredictos (permitir/perguntar/negar), os paths sensíveis indesligáveis, e a **aprovação inline no chat** (o front pausa e pergunta — a manifestação visível da política). Exercício de completude: a avaliação de regras vem pronta; você adiciona o parsing mínimo de um comando shell antes de julgá-lo.

Verificação

1. Um harness só tem política de aprovação, sem sandbox de SO. Que classe de ataque ele não consegue conter, e por quê? (Política sem contenção; o modelo pode ser convencido.)

2. Você vai adicionar uma tool de envio de e-mail a um agente que já lê issues do GitHub e tem acesso ao repositório privado. Aplique a trifecta letal. (Fecha o terceiro vértice → exfiltração possível.)
3. Por que `on-failure` (aprovar só depois do bloqueio) pode ser melhor que `on-request` (aprovar antes de cada ação)? (Fricção × cobertura; o sandbox filtra o que nunca precisa de humano.)

Apêndice A — Como cada repositório trata permissões e sandboxing

> Evidência por harness, com paths — complementação online, expandida a cada rodada.

gemini-cli (rodada 1) ⭐ policy engine + sandbox de SO

`packages/core/src/policy/policy-engine.ts`: regras priorizadas com **parsing estrutural de shell** (`parseCommandDetails`, `stripShellWrapper`, detecção de redirecionamento), regras em TOML; 4 `ApprovalMode`; **6 perfis Seatbelt** (`sandbox-macos-*.sb`) + Docker/Podman com proxy; **trusted folders** gatekeepando hooks/agents.

OpenHarness (rodada 1) ⭐ paths sensíveis

`permissions/checker.py`: path rules, comandos negados, 3 modos; **`SENSITIVE_PATH_PATTERNS` hardcoded e indesligável** (`.ssh`, `.aws/credentials`, `.gnupg`, `.kube/config`) contra injection; sandbox via `sandbox-runtime`/Docker com allowlist de domínios; `trust_env=False` nas tools web (anti-SSRF).

opencode (rodada 1) — política sem contenção

`permission/`: rulesets com wildcards (`allow | ask | deny`, last-match-wins, default `ask`), aprovação via `Deferred` + evento; **subagentes derivam permissões restritas**; **sem sandbox de SO no core** (containers só no enterprise).

Codex CLI (rodada 2) ⭐ contenção por SO em 3 camadas

`sandboxing/` + `linux-sandbox/` + `windows-sandbox-rs/`: Seatbelt via `sandbox-exec` (path hardcoded anti-tamper), bubblewrap embutido + **seccomp** + `NO_NEW_PRIVS`, Landlock legado; `AskForApproval` incl. `Granular`; **execpolicy em Starlark** por comando; `assess_patch_safety`; network-proxy.

Goose (rodada 2)

`permission/`: modos `GooseMode` (Auto/Approve/Chat); **`permission_judge` usa um LLM** para classificar read-only; `ToolPermissionStore` por assinatura com expiração; isolamento de execução leve (shell direto; Docker externo).

OpenClaw (rodada 2) ⭐ pairing de terceiros

`src/pairing/` + `docs/security/THREAT-MODEL-ATLAS.md`: **DMs como input não confiável**, `dmPolicy: "pairing"` default (código de pareamento, allowlist SQLite); sandbox multi-backend (Docker `network:none`/`readOnlyRoot`/`capDrop:ALL`, SSH, OpenShell) com modo **`non-main`**; `openclaw doctor`/`security audit`; caveat: `sandbox.mode` off por default na sessão main.

Hermes (rodada 2)

`tools/approval.py` (detecção + allowlist), callbacks por-thread; **seis backends de terminal isolados** (local, Docker, SSH, Singularity, Modal, Daytona); subagentes com `_subagent_auto_deny` seguro por default; `path_security.py` anti-traversal.

IronClaw (rodada 2) ⭐⭐ arquitetura de autoridade

`crates/ironclaw_authorization` + `_approvals` + `_trust` + `_wasm` + `_process_sandbox` + `_secrets` + `_network` + `_safety`: autorização de invocação exata (fail-closed), aprovações como **leases por invocação com fingerprint**, **trust class inforjável por tipo** (`#[serde(skip_deserializing)]`), WASM (fuel/memória/rate, egress negado), Docker per-tenant, secrets zero-exposure na borda de egress, anti-SSRF, leak detector bidirecional — o loop não alcança os efeitos (verificado por testes de dependência).

ohmo (rodada 2.5) — a metade certa

`channels/impl/base.py`: allowlist **deny-by-default** + isolamento de sessão por remetente + bloqueio de comandos admin remotos + paths sensíveis do OpenHarness. Gap: `permission_mode`/`sandbox_enabled` do `gateway.json` são **código morto** — sem dial entre nega-tudo e full_auto.

software-agent-sdk (rodada frameworks)

`sdk/security/`: análise de risco (LLM analyzer + `defense_in_depth/` determinístico com parser AST de shell detectando **fetch-to-exec**) + política de confirmação (`AlwaysConfirm`/`ConfirmRisky` por limiar); a conversa **retorna** em `WAITING_FOR_CONFIRMATION` (não bloqueia); mascaramento de segredos.

n8n (rodada 2) — permissão estrutural

A permissão é **topológica**: o autor escolhe quais nós ficam na porta `AiTool` — allowlist por construção. HITL real via `sendAndWait` (pausa durável), proibido em sub-agentes; nó Guardrails.

Frameworks (rodada frameworks) — deixam aberto

LangGraph e CrewAI não têm política de tools nativa (constrói-se sobre `interrupt`/HITL); o Agents SDK (Software Development Kit) tem guardrails em três níveis (agente/run/tool) como primitiva, mas contenção fica por conta do adotante.