Why context engineering *isn't enough*

5 failure modes of coding agents — and how to build the harness that stops them.

Ends with three fixes for this week and a 5-minute self-check to find your biggest gap.

THE FIVE FAILURE MODES

1 · GREEN CI, BROKEN PROD
2 · COMPREHENSION DEBT
3 · DESTRUCTIVE AGENTS
4 · HIJACKED AGENTS
5 · REVIEW OVERLOAD

Renato Budinich · Harness engineer for coding agents · Version 3.0 · November 2026

WHY ALL FIVE HAPPEN

Your agents got better. Trust didn't.

You've engineered the context.

- ✓ AGENTS.md
- ✓ Skills
- ✓ MCP servers
- ✓ Prompts

Your process hasn't kept pace.

96%

of engineering leaders say theirs can't keep up with their agents.

Qodo, State of AI Code Quality, Sep 2026
300 US engineering leaders

Harness engineering restores trust.

Tests, sandboxes, CI gates and team workflows decide what ships, not the agent.

Context steers the agent; the harness keeps it on the rails.

1

✕ FAILURE MODE GREEN CI, BROKEN PROD

Green in CI, broken in production

Agents ship more code than ever, and passing checks no longer means it works.

✓ Tests pass → ✓ Review approves → ✕ Production breaks

82%

of tech leaders experienced at least one production failure tied to AI-generated code in the past six months

New Relic, 2026 State of AI Coding · 200 US tech leaders · self-reported

In a benchmark, 4 in 5 agent solutions that passed the functional tests had a security hole.

Zhao et al., ICML 2026 · SusVibes, SWE-agent with Claude 4 Sonnet, 79.3% · 186 features that human developers once shipped with a hole

✕ WHAT IT COSTS YOU

Your customers find the bugs your tests missed.

Let deterministic checks decide what ships.

✔ **After:** a green build means the code works, not just that an agent says it does.

NON-DETERMINISTIC
Can give a different answer on every run
Code generation · agent review

DETERMINISTIC
Same input, same answer, every run
Compiler and types · tests · reproducible builds · proof checkers · hooks

Reproducible builds

Pin every tool and dependency version, so the agent's sandbox, CI and production build the same thing.

- **Devcontainers**
  pin the environment in a container
- **Nix** with **devenv**
  pins down to system libraries
- **NixOS**
  builds production from the same pins
- **Terraform**
  declares the infrastructure as code

Tests that catch real bugs

Agents write tests that pass even when the code is wrong, or mock away every real failure.

- **Integration tests**
  run the real components together
- **Playwright**
  tests user journeys in a browser
- **Fuzz tests**
  throw generated inputs at the code
- **Mutation testing**
  plants bugs, flags tests that still pass
- **Semgrep**, **CodeQL**
  run ready-made security rules

Proofs for costly bugs

For money, permissions or concurrency, a checker can verify the design or prove the code against laws you write. The next slide compares them.

- **TLA+**
  checks the design
- **Dafny**, **Lean**, **Bend 2**
  prove the code

Formal methods for the code you can't afford to get wrong.

✔ **After:** your riskiest code has to satisfy laws you wrote, and a checker rejects anything that breaks them.

CHECK THE DESIGN

TLA+ / TLC

Keep your implementation language. Specify states, transitions and invariants, and TLC explores every reachable state of a finite model you configure.

Connect it to code with **conformance tests**
Specification → model-based tests → code
Code → execution traces → specification

Limit: TLC checks the design, not your code.
Conformance tests link the two, but only for the runs they cover.

PROVE THE CODE ITSELF

Dafny · Lean · Bend 2

Write only the critical component and its laws in a verification language. Agents can write the code and the proofs, and the checker accepts the code only if every proof holds.

Dafny: specs proved by a solver
You annotate the code with specs and a solver proves them, with less manual proof work. AWS rebuilt its authorization engine in Dafny and proved it against its spec; it handles about a billion requests a second.

Lean and Bend 2: dependent types
The type is the law, and the program is its proof.

Either way, **the guarantee is only as complete as the laws you write**: "returns a sorted list" also holds for code that returns an empty list.

TLC · Trace validation · Model-based tests · Lean · Dafny · AWS authorization engine in Dafny, 2025 · Hillel Wayne on agents weakening TLA+ properties

2

FAILURE MODE COMPREHENSION DEBT

Nobody understands the system anymore

Agents add parts faster than you can learn them, so your picture of the system goes stale.

45%

of developers would need AI's help to explain how a piece of code they wrote works

Sombra, AI at Work Survey, Sept 2026 · 1,000 US and Canadian developers · commissioned, self-reported

> *When something inevitably goes wrong, nobody has the knowledge to trace the failure back to its root cause.*
>
> Undo, Overcoming the Limitations of Coding Agents, 2026 · vendor report

WHAT IT COSTS YOU

When it breaks, nobody can fix it until someone learns what the agents wrote.

That learning time is comprehension debt · Jason Gorman, Codemanship, Sep 2025

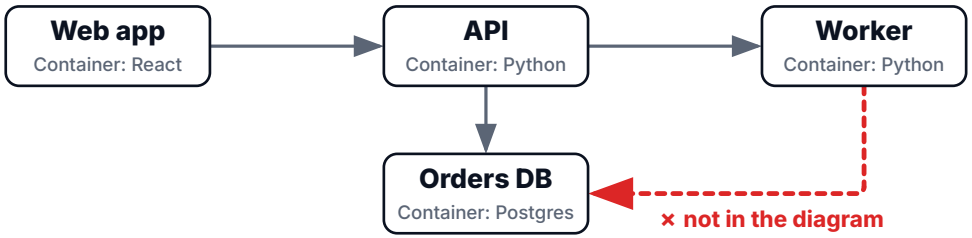
✓ THE HARNESS FAILURE MODE 2 · COMPREHENSION DEBT

Get flagged when your picture goes stale.

✔ **After:** every change matches your architecture diagram, or that part's owner approves the new diagram.

WHAT YOU KNOW

In **C4**, a **system** is the software your team ships, and its **containers** are the apps, services and data stores it's made of, as in this online shop.



DRAW IT WITH

- **Mermaid**: quick diagrams that render in Markdown
- **Structurizr**: one model, several views kept in sync
- **CodeBoarding**: drawn from the code, with a change diagram on every PR

WHAT TELLS YOU IT CHANGED

Export orders as CSV #142

architecture drift opened by your agent

✔ Tests
✖ **Architecture conformance:** Worker reads Orders DB, which isn't in the diagram

A deterministic **architecture conformance check** raises the flag: import rules (ArchUnit, dependency-cruiser, Import Linter), contract tests (Pact) or infrastructure policies (OPA).

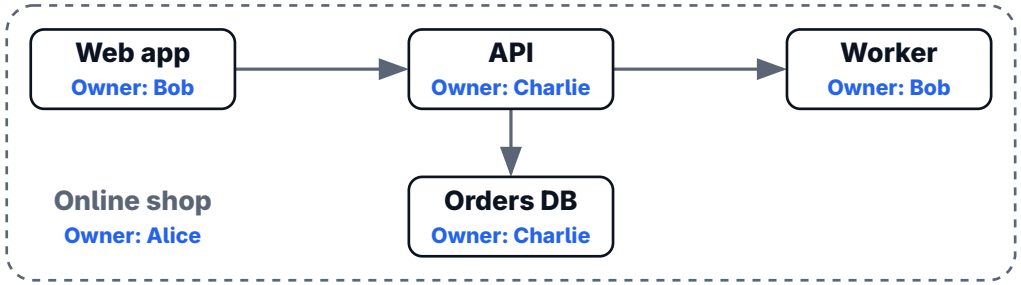
An agent review can raise it too, and can catch drift that no rule was written for.

Give every part of your architecture an owner.

✔ **After:** one person can explain each part, so you know who to call when it breaks.

WHO OWNS WHAT

Owners can sit at any level: Alice owns the whole online shop, Bob and Charlie the parts inside it.



Quiz yourself on your architecture
Two agent skills write the questions from your code and score you on the parts you own.
github.com/nareto/archquiz →

WHAT OWNERS USE

Map people to the architecture

A **CODEOWNERS** file lists each part's folders with its owner, so a change to the web app goes to Bob for review, and can be blocked until he approves.

Map traces to the architecture

Traces (OpenTelemetry, Jaeger) record every call between services: owners see who really uses their part, and a conformance check flags calls missing from the diagram.

Map comprehension debt to the architecture

Quiz each owner on their part: low scores show what they'd have to learn before they could fix it.

3

FAILURE MODE DESTRUCTIVE AGENTS

Agents delete what nobody asked them to

To go fast, agents get to run anything your terminal and tokens allow, and an LLM picks what to run.

JUL 2025 **Replit**	During a code freeze, it **deleted SaaStr's production database**.	Fortune
NOV 2025 **Google Antigravity**	Asked to clear a cache, it **wiped the user's whole D: drive**.	The Register
DEC 2025 **AWS Kiro**	It reportedly chose to **"delete and recreate the environment"**, a 13-hour interruption.	The Register (FT)
FEB 2026 **DataTalks.Club**	Asked to delete only resources created by mistake, Claude Code ran `terraform destroy` and **wiped production, snapshots included**.	Alexey Grigorev
APR 2026 **PocketOS**	A Cursor agent used a token it found to **delete the production database and its backups** in 9 seconds.	The Register

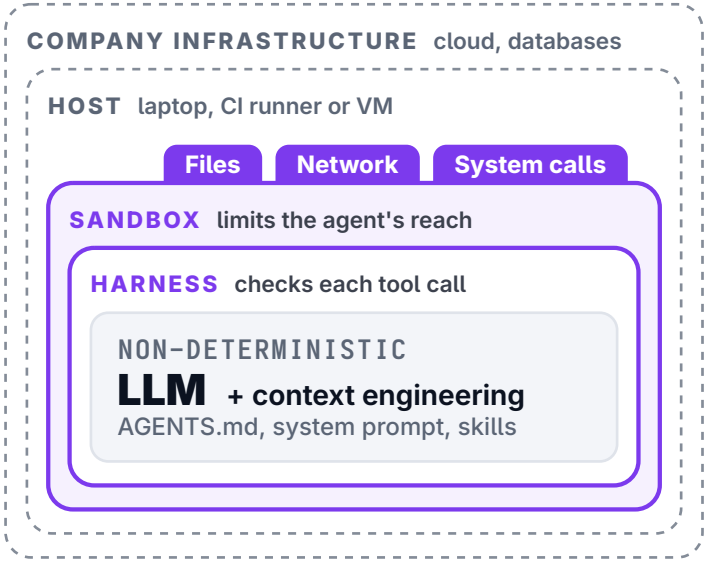
WHAT IT COSTS YOU

Your production data and its backups are one unlucky run away from gone.

✓ THE HARNESS FAILURE MODE 3 · DESTRUCTIVE AGENTS

Wrap the LLM in deterministic guardrails.

✔ **After:** your rules live in code, so `terraform destroy` is blocked whatever the agent's context says.



Every ring around the gray box is deterministic code.
This failure mode covers the two violet ones.

HARNESS Choose how strict to be.

Say it tries to run `terraform destroy`.

Nothing checks it	Another model reviews it	A rule blocks it	Don't give it the tool
It just runs.	It may stop it, but it can be wrong: not a guardrail.	A **deny rule** or **hook** rejects it, every time.	With no shell tool, it can't.

Riskier → Safer

SANDBOX Set what the agent can reach.

- **Files** — Let commands write only to the repo, and read nothing in `~/.ssh` or `~/.aws`.
- **Network** — Let commands call only the domains you list: your package registry, not your cloud API.
- **System calls** — Let as few system calls as possible reach the host's kernel, so a kernel bug can't undo the file and network limits.

TOOLS

- **Containers and OS sandboxes** use the host's kernel: **bubblewrap**, **Landlock**, **Seatbelt**, **Docker**
- **A user-space kernel** answers system calls itself: **gVisor**
- **MicroVMs** run their own kernel on virtual hardware: **Firecracker**, **Docker Sandboxes**, **E2B**

✓ THE HARNESS FAILURE MODE 3 · DESTRUCTIVE AGENTS

Guardrails you can set up today.

✔ **After:** every developer's agent runs with the same guardrails, and nobody can switch them off.

		CLAUDE CODE settings.json	CODEX config.toml	BUILD YOUR OWN extend pi, or fork OpenCode, Goose or Codex
HARNESS	**Don't give it the tool**	Any built-in tool can be removed, e.g. Bash: `"deny": ["Bash"]`. Extra tools come only via MCP.	Only some built-in tools can be removed, e.g. the shell: `shell_tool = false`. Extra tools come only via MCP.	Drop or **rewrite** any built-in tool, e.g. a `read` that skips `.env` files. Add tools in code.
HARNESS	**A rule blocks it**	**Deny rules** and **PreToolUse hooks**	`forbidden` rules and **PreToolUse hooks**	Write the check in code, e.g. a pi `tool_call` handler
HARNESS	**Another model reviews it**	**Auto mode**, the default	**Auto-review**, with your policy	Call any LLM, or a fast decision model: **Jev**, or **Laya** on your machine
SANDBOX	**Files**	`sandbox.enabled`: writes only to the project	`workspace-write`: writes only to the workspace	Run commands in a microVM that mounts only the repo
SANDBOX	**Network**	`allowedDomains`: only the domains you list	`network_proxy`: only the domains you list	Send traffic through an allowlist proxy: **Smokescreen**, **sandbox-runtime**
SANDBOX	**Kernel**	No setting: commands use the host's kernel	No setting: commands use the host's kernel	Run commands in a microVM, on their own kernel

Claude Code's sandbox covers shell commands only. To run either agent in a microVM, use Docker Sandboxes.

Enforce these settings for everyone through admin policy.

Claude Code: managed settings · Codex: managed configuration · turn off bypass permissions and --yolo

4

FAILURE MODE · HIJACKED AGENTS

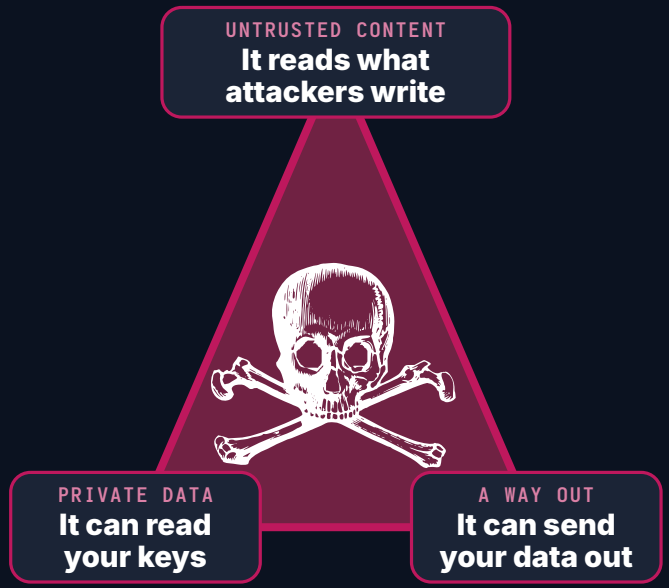
What your agent can reach, an attacker can too

Agents act on their own to move fast, and none can reliably tell your instructions from an attacker's.

149,329

open-source packages flagged as malicious in three months, 9 in 10 of them on npm

Sonatype, Q3 2026



SIMON WILLISON'S LETHAL TRIFECTA

UNTRUSTED CONTENT	**Packages it installs, issues, PRs, web pages.** Cline, Feb 2026: an issue title could steer its triage bot into exposing its npm token.	Cline
PRIVATE DATA	**Tokens, SSH keys, `.env` files, private repos.** Nx, Aug 2025: its hijacked package told victims' Claude and Gemini CLIs to find them, permissions skipped.	Wiz
A WAY OUT	**Web requests, git pushes, emails.** Mini-Shai-Hulud, May 2026: the worm sent stolen keys to a fake telemetry server and the victim's own GitHub. It reached two OpenAI laptops.	Sonar OpenAI

WHAT IT COSTS YOU

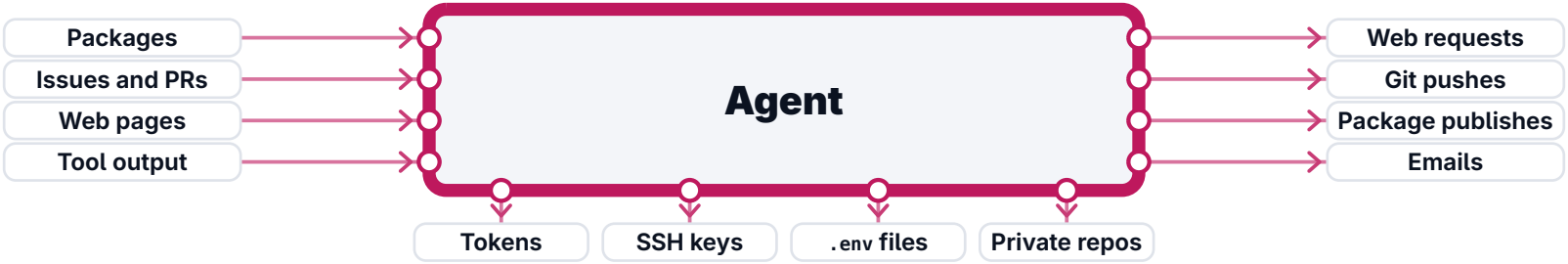
One poisoned package or issue, and an attacker holds every key you hold.

Mini Shai-Hulud took AWS, Kubernetes and Vault secrets, then published malware under its victims' own npm packages.

✓ THE HARNESS FAILURE MODE 4 · HIJACKED AGENTS

Limit what gets in, what's in reach and what leaves.

✔ **After:** no agent holds all three at once, so a poisoned issue gets the attacker nothing.



WHAT GETS IN untrusted content

- **A 7-day cooldown**: install only versions older than a week, e.g. with pnpm's `minimumReleaseAge`. axios was live for 3 hours, Nx for 4.
- **Install scripts off**, allowed per package: the default since **npm 12** and **pnpm 10**. Nx's and axios's malware ran as install scripts.
- **An injection classifier**: **Prompt Guard 2** or **Jev**. A tripwire, not a wall: a fake policy cut Jev's catch rate from 83% to 4% (Datadog).

WHAT THE AGENT CAN REACH private data

- **Only the secrets the task needs**: block `~/.ssh` and `~/.aws` with `denyRead` and `Read` **deny rules**. Claude Code can read them by default.
- **Narrow, short-lived keys**: one repo, read-only where possible, like **GitHub App tokens**. A stolen key can only do what it was made for.
- **CI**: no secrets or shell for agents that read outside issues or PRs (`permissions: {}`). That's the hole in Cline's triage bot.

WHAT CAN LEAVE a way out

- **A domain allowlist**: `allowedDomains` in the sandbox. An allowed host like GitHub can still carry data out.
- **An outbound classifier** on each request: **auto mode** or **jes**. Don't show it the untrusted text, or the attacker steers the judge too.
- **Leak scanning**: **push protection**, **gitleaks** or **TruffleHog** block pushes with keys. Run it in CI too: agents can skip git hooks.

Meta's Rule of Two: an agent with all three needs a person, or another reliable check, to approve its actions

5

FAILURE MODE · REVIEW OVERLOAD

Review is drowning your best engineers

Reading every line keeps you safe, but the queue eats the speed and judgment turns into a rubber stamp.

+98%
pull requests merged

+91%
time spent in review

Faros AI, 2025 · high vs low AI adoption · 10,000+ developers, 1,255 teams · observational comparison

WHAT IT COSTS YOU

Your best engineers become full-time reviewers, and the agent speed is gone.

✓ THE HARNESS FAILURE MODE 5 · REVIEW OVERLOAD

Review is step 4 of 5, so fix the steps around it.

✔ **After:** fewer, smaller PRs reach review, and a QA tester tries every change before your users do.

PUT LESS IN | CHECK WHAT COMES OUT

1 Decide what to build	2 Decide if it's worth it	3 Build the thinnest slice	4 Review	5 QA testing
Name the user need, stakeholder obligation or risk it serves. No outcome, no code.	Agents made code cheap to write, not to review, test and run. Drop what doesn't pay for that.	Elephant Carpaccio: "let users download this report as CSV", not "a reporting framework".	An agent loop reviews first. People review the change to the architecture diagram (slide 7), not every line.	A tester uses the change the way a customer would, before it ships.

4 · AUTOMATE REVIEW **Try it now: chief →**

Run a **Ralph Wiggum loop** on every PR, prompted for what you care about: bugs, bad code patterns, security holes. Each pass reviews with a fresh context, and the loop stops when one finds nothing new. **A person sees the PR only after the loop passes it.** It costs tokens, so cap the passes.

5 · QA TESTING

Green CI only proves what someone thought to check, so code can pass every test and still break in production (slide 3). **Pay QA testers, not your engineers, to try each change before it ships.** The less you put in at steps 1–3, the less there is to test.

✓ START THIS WEEK

Three fixes you can make this week.

1 Run an agent review loop on every PR

→ SLIDE 15

Install chief (needs Rust)

```
cargo install --locked chief --git \
  https://github.com/nareto/chief
```

On the PR branch

```
chief review --agent claude \
  --agent-sandbox native \
  --max-loop-iterations 8
```

Chief stops when a pass finds nothing new. Findings go to a YAML file. If the agent edits your code, the run fails.

2 Force the sandbox on for everyone

→ SLIDE 11

Claude Code managed settings

```
{"sandbox": {"enabled": true,
  "failIfUnavailable": true,
  "allowUnsandboxedCommands": false}}
```

Codex requirements.toml

```
default_permissions = ":workspace"
[allowed_permission_profiles]
":read-only" = true
":workspace" = true
```

3 Make new packages wait a week

→ SLIDE 13

pnpm pnpm-workspace.yaml

```
minimumReleaseAge: 10080  # 7 days
```

npm .npmrc (age in days)

```
min-release-age=7
```

Nothing to buy, nothing to host. Then find your biggest gap on the next slide.

YOUR 5-MINUTE SELF-CHECK

Tick what you can demonstrate today.

An empty box is a gap to investigate. Prioritize by consequence, not by the number of ticks.

1 · GREEN CI, BROKEN PROD →

- [ ] Agents, CI and prod use the same pinned versions
- [ ] Mutation testing shows our tests catch real bugs
- [ ] Security scanners check every PR for known holes
- [ ] Critical parts are model-checked or proved

2 · COMPREHENSION DEBT →

- [ ] A check flags drift from our architecture or contracts
- [ ] Every part has an owner who can explain it
- [ ] We quiz owners on the parts they own **Try it now: archquiz →**

3 · DESTRUCTIVE AGENTS →

- [ ] Code, not AGENTS.md, blocks destructive commands
- [ ] Agents run in a sandbox nobody can switch off
- [ ] Agents can’t reach production from their sandbox

4 · HIJACKED AGENTS →

- [ ] New package versions wait 7 days
- [ ] Agents and CI jobs hold only narrow, short-lived keys
- [ ] Agents reach only the domains we list
- [ ] Pushes with keys in them are blocked

5 · REVIEW OVERLOAD →

- [ ] Each request names the user need it serves
- [ ] Every PR delivers one thing a person can try
- [ ] Every PR passes an agent review loop first **Try it now: chief →**
- [ ] A QA tester tries every change before it ships

Not sure which empty box could hurt most? That’s what the free call is for.

FREE 20-MINUTE CALL · WITH RENATO

Your first fix, *in writing*

Tell me which boxes on slide 17 stayed empty and how your team ships agent-written code. Within two business days you get one written page with:

✖ Your top risk

Which empty box could hurt you most, and why

✔ Your first fix

A concrete step your team can take this week

→ Do you need help?

Which paid option fits, or why you don't need one

No repository access needed, just a call. If you want help with the fix: a one-week baseline on your repos, with a full refund if you're not happy, then only the fixes you need.

Not for you if your team is still choosing a coding agent or running a pilot: this is for teams already shipping with one.

Book your free call →

I work with 3 teams at a time.

So each gets my full attention. See if there's a spot on renatobudinich.com.