MCP Dev Summit Bengaluru

• Starting shortly

MCP DEV SUMMIT · BANGALORE · UP NEXT

When agents get SSH keys

Securing a distributed AI fleet with MCP

Mradul Dubey

Technical Lead · Apra Labs

apra-fleet · github.com/Apra-Labs/apra-fleet

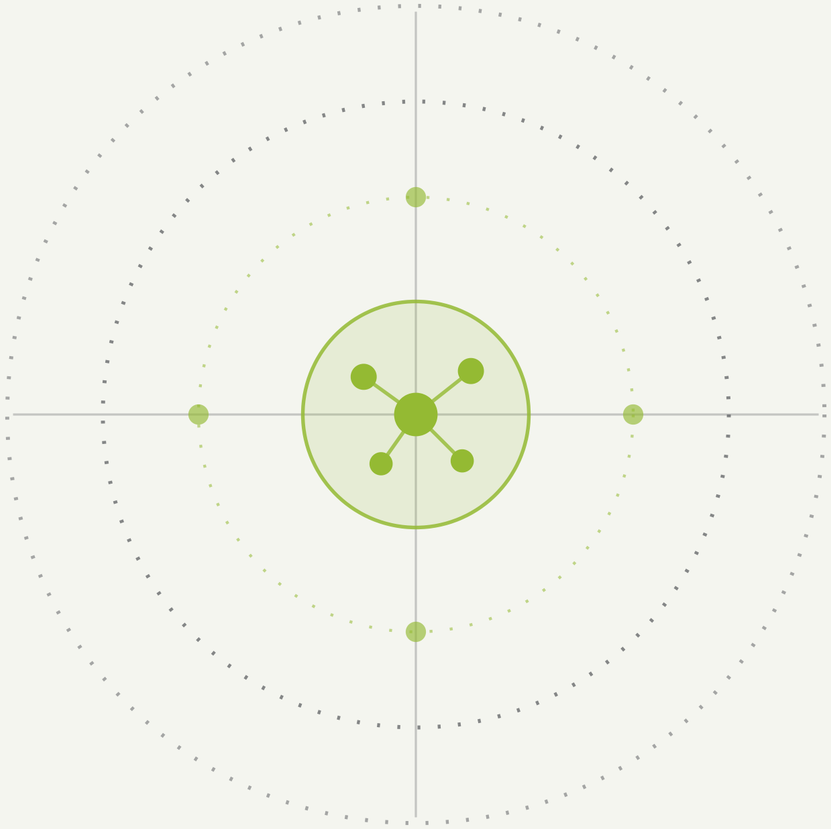
MCP DEV SUMMIT · BANGALORE

When agents get SSH keys

Securing a distributed AI fleet with MCP

What it takes when agents push to Git, run code over SSH, and start cloud VMs.

Mradul Dubey · Technical Lead, Apra Labs



apra-fleet · github.com/Apra-Labs/apra-fleet

WHICH SECURITY?

What this talk is — and isn't

What it isn't

Not comprehensive cybersecurity.

It does not claim a **determined, privileged attacker** can be stopped.

What it is

Scaling agentic development safely.

Removing the accidental leak, the default-on exposure, the secret that ends up somewhere no one meant to put it.

Every pattern moves a leak **up the cost curve** — from *"happens by default"* to *"requires intent, privilege & effort."* None of them claim to flatten it to zero.

MCP Dev Summit Bengaluru

The brakes make the car go faster.

A faster car needs better brakes.

Shrinking the breach possiblities isn't the tax you pay to ship a Agentic AI fleet. **It's what lets you ship one to production at all.**

No brakes
afraid to commit

Good brakes
commit with confidence

THE SETUP EVERYONE HAS SEEN

The Solo Agentic AI workflow

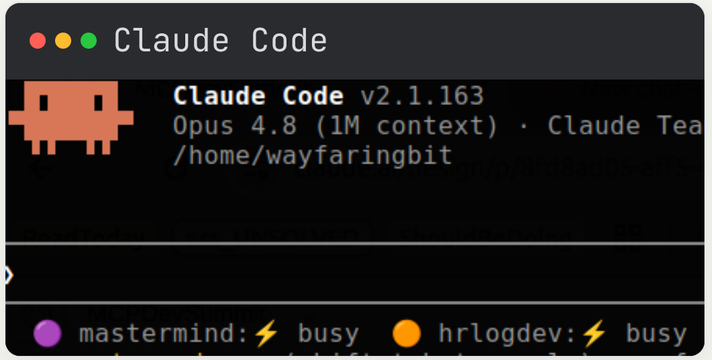
One agent, one machine, one repo. It works.

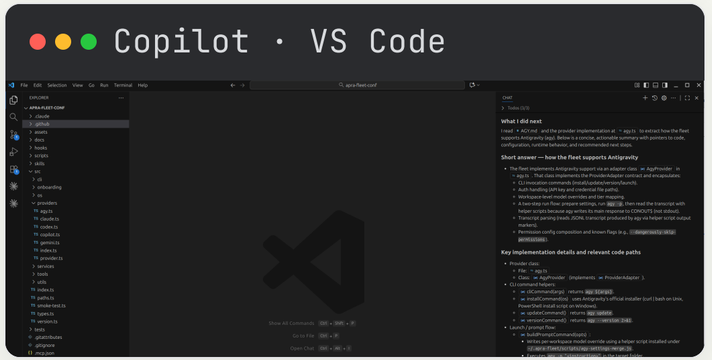
Agent
one machine

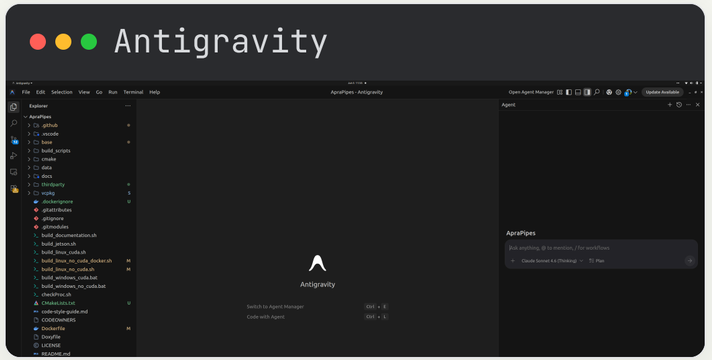
→

Repo
one target

Claude Code, Gemini CLI, Copilot, Antigravity — whichever you reach for, the solo loop just works.

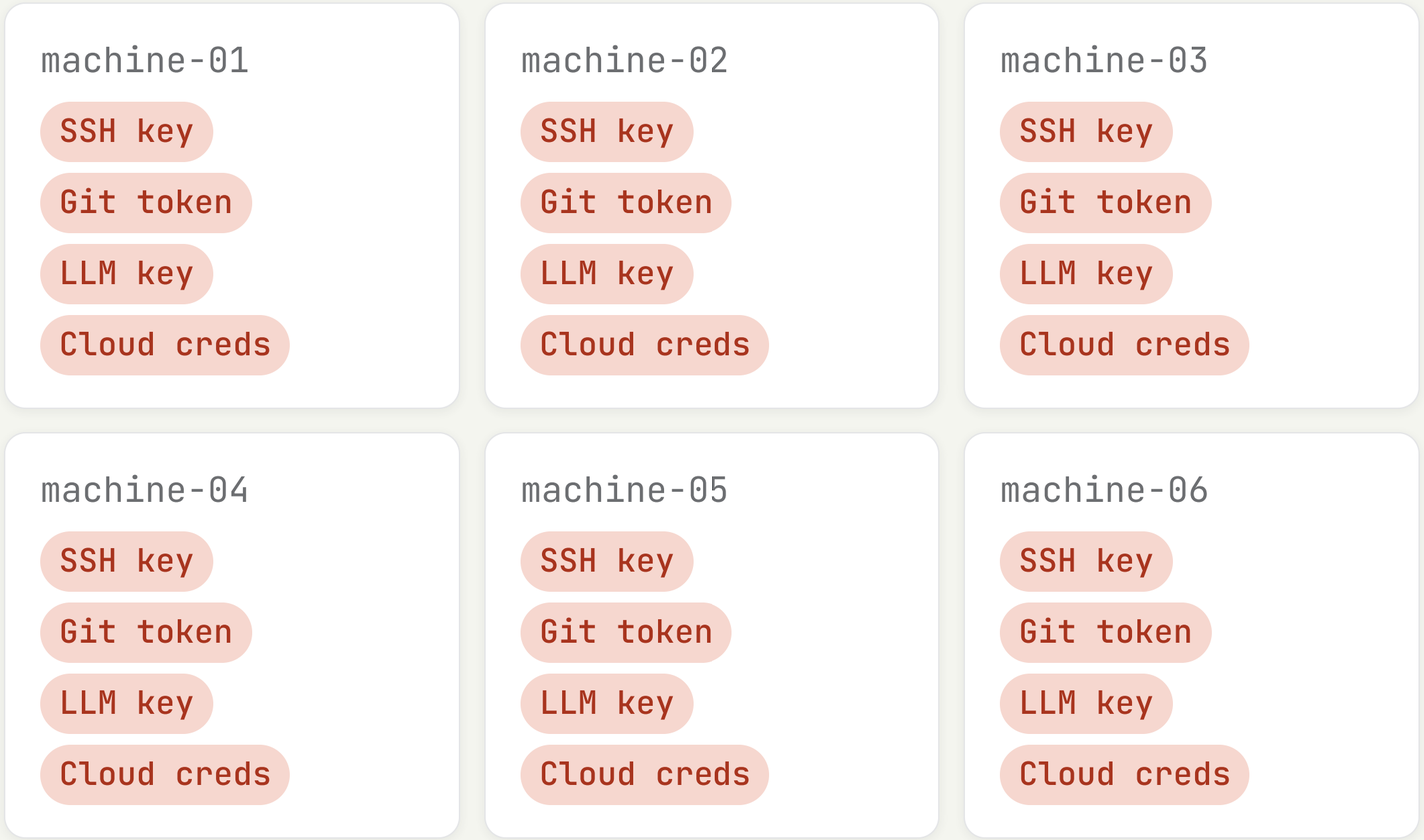






THE FLEET PROBLEM

Now make it production

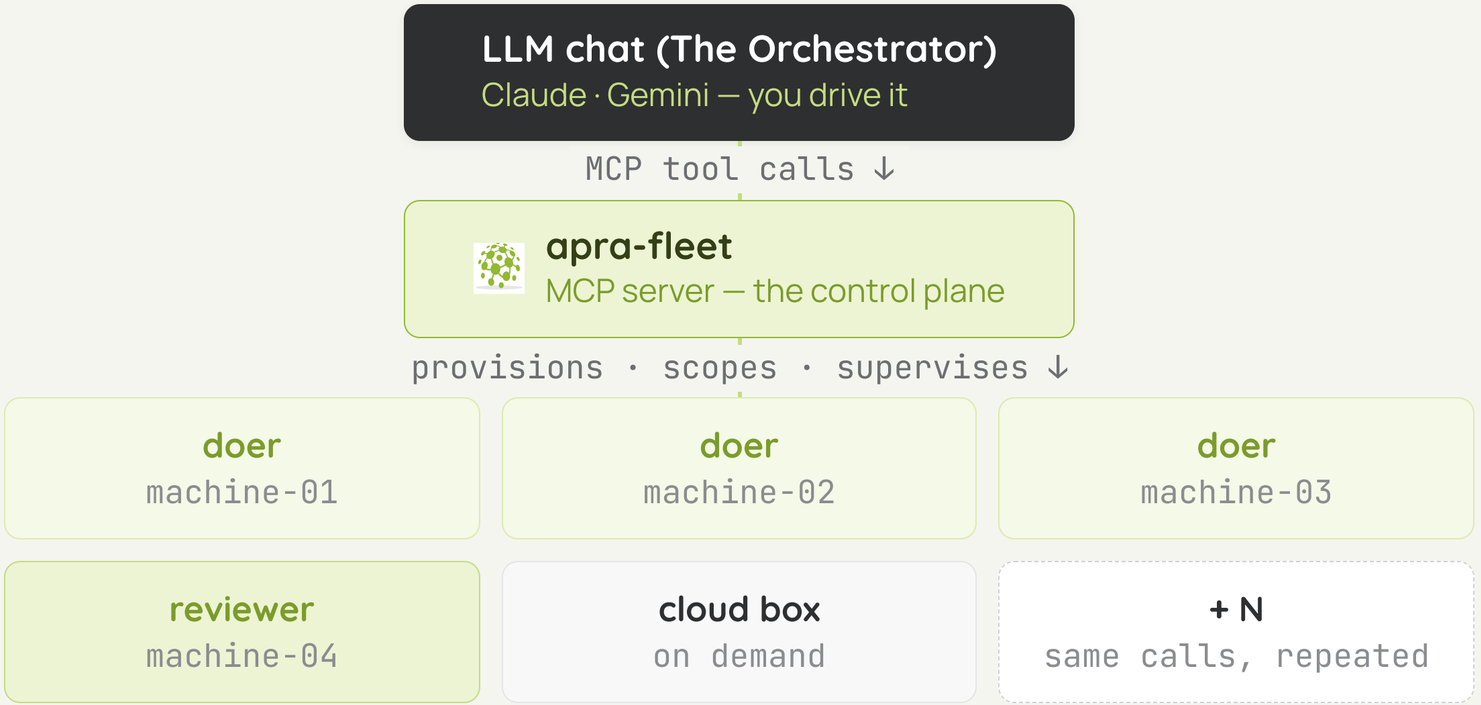


One control plane — but every worker still carries its own keys. The risk doesn't add up, it multiplies.

When — not if — a session is compromised, what can it actually reach? That surface is the **blast radius.**

FROM ONE MACHINE TO A FLEET

One orchestrator, many workers



ALL THROUGH A HANDFUL OF CALLS

Onboard
a machine over SSH, with a fresh RSA-4096 key.

Scope
give it a role — a doer that builds or a reviewer that reviews.

Provision
a short-lived, repo-scoped Git token.

Run
hand the worker its task, then supervise it.

Scale
spin up more VMs on demand — the fleet grows itself.

You drive one MCP server — it onboards, scopes, and supervises the whole fleet.

Assume the agent is already compromised.

Prompt injection will land. Design for after it does.

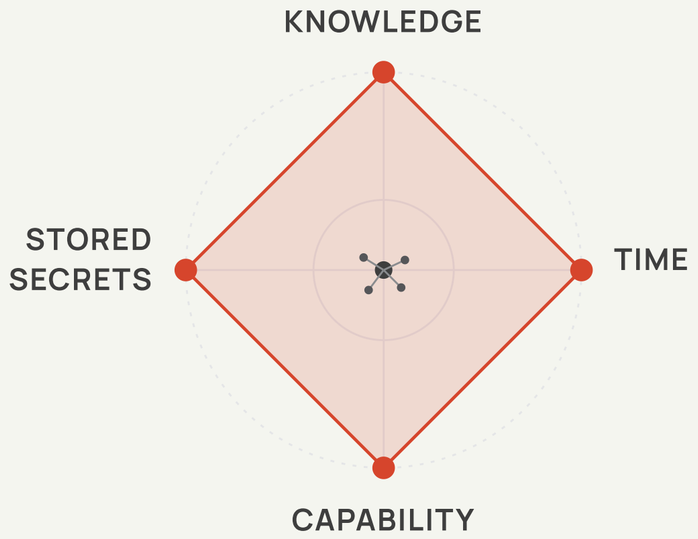
If your security only works while the model behaves, you don't have security. **You have hope.**

ROADMAP

Four ways to shrink the blast radius

KNOWLEDGE	A leaked transcript contains no secrets.
TIME	Yesterday's token is already dead.
CAPABILITY	The bad action is unrepresentable.
STORED SECRETS	Stolen files are ciphertext, not creds.

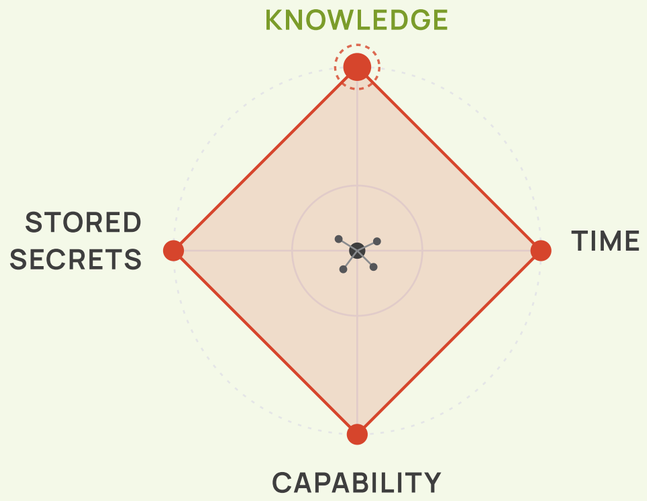
MCP server is the right place to shrink **all four.**



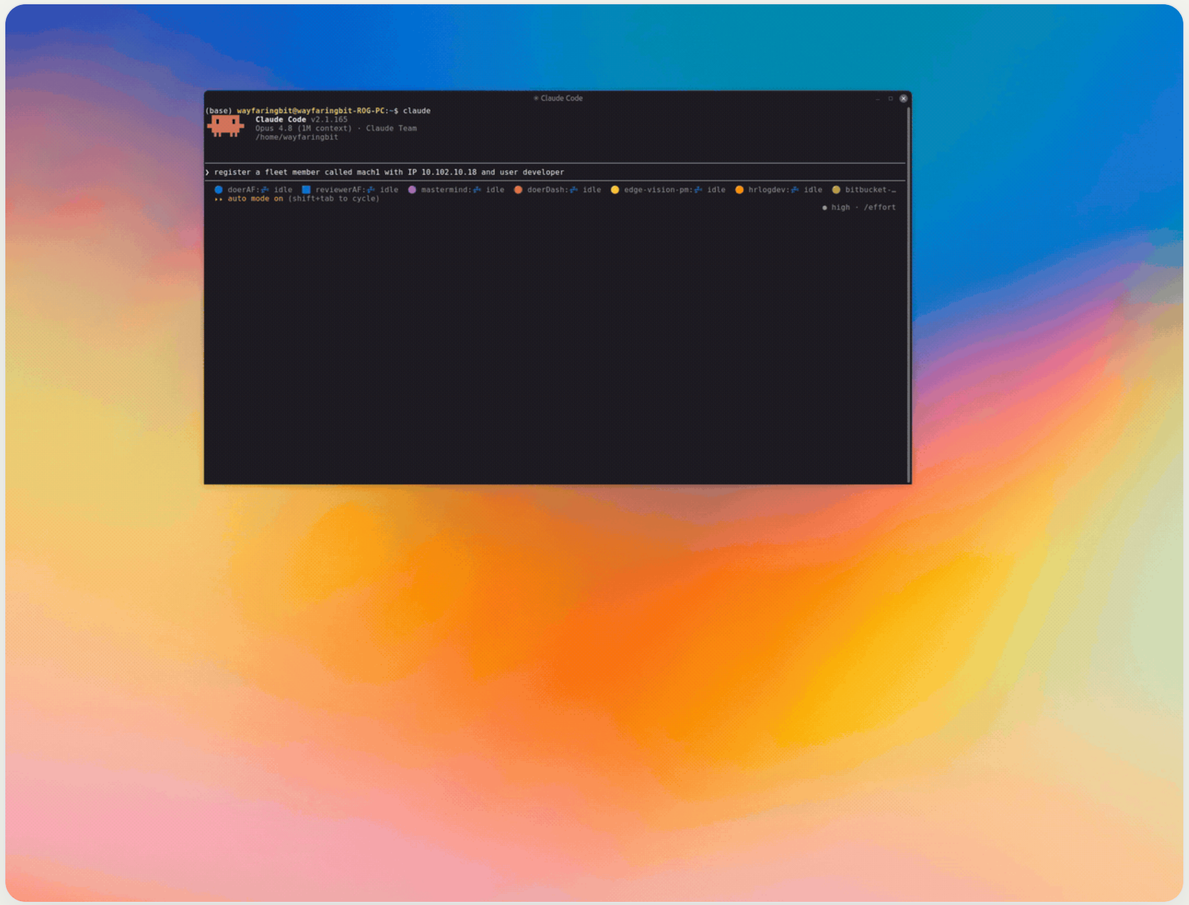
01

Secrets never touch the model

Start with the most obvious leak — and the one people get wrong constantly: secrets in the context window.



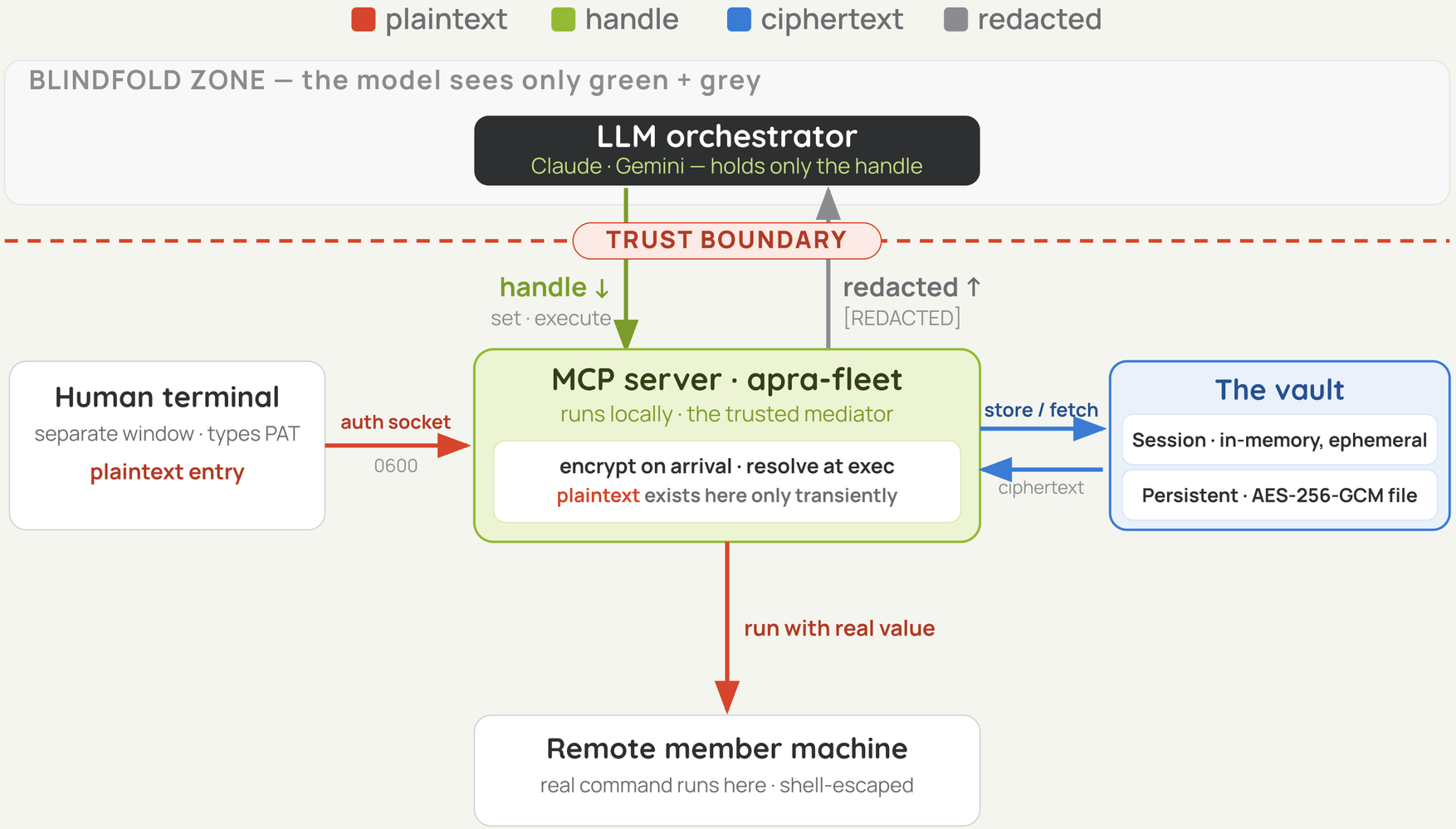
PATTERN 1 · HOW THE SECRET ACTUALLY ENTERS



1. The **orchestrator** calls `register_member` — and deliberately omits the password.
2. The **MCP server spawns a separate terminal** — `apra-fleet secret --set`.
3. The **human types the secret there.** It returns over a local socket, **encrypted the instant it arrives.**
4. The model is handed back only a **handle** — never the plaintext.

Two processes, one socket. The secret never crosses the transcript.

One trusted mediator, one blindfolded model



PATTERN 1 · OUT-OF-BAND ENTRY

The LLM model orchestrates a secret it never sees

1
Server spawns a **separate terminal**. The human types the secret there.

→

2
Travels a **local socket**, encrypted on arrival. Plaintext zeroed immediately.

→

3
Tool returns a **handle** — `{{secure.GITHUB_PAT}}`

→

4
At exec, server swaps the real value — shell-escaped — then **redacts** it from output.

If the whole transcript leaks tomorrow, the attacker gets a bag of handles and zero credentials.

Elicitation: The official MCP spec is essentially the same.

> Servers **MUST NOT** use form mode elicitation to request sensitive information such as passwords, API keys, access tokens, or payment credentials.

> Servers **MUST** use URL mode for interactions involving such sensitive information.

URL-mode elicitation: the credential never passes through the LLM context, the client, or any intermediate server. **Same blindfold the protocol is now standardizing.**

PATTERN 1 · THE TRAP, AND THE FIX

If the secret is a tool argument, it's in the transcript forever

✕ The naive design

```
set_secret({
  name: "GITHUB_PAT",
  password: "ghp_xxxxxxxxxxxx"
})
// the model generated it, saw it,
// and it's in the transcript. Forever.
```

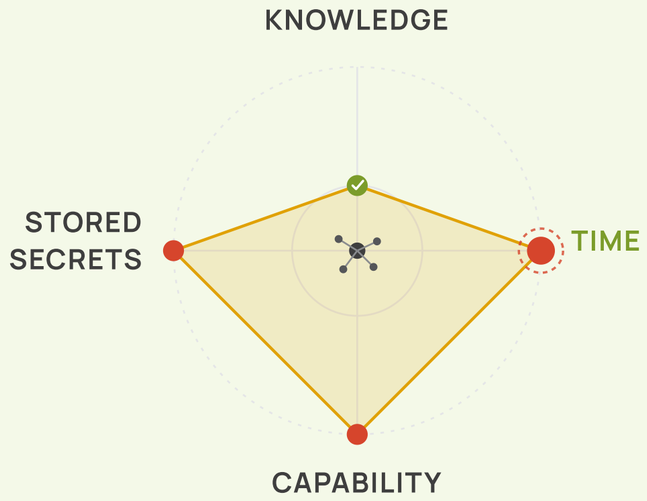
✓ The real schema: credential_store_set

```
credential_store_set({
  name, prompt, persist,
  network_policy, members, ttl_seconds
})
// no password field —
// the tool cannot receive the secret.
```

02

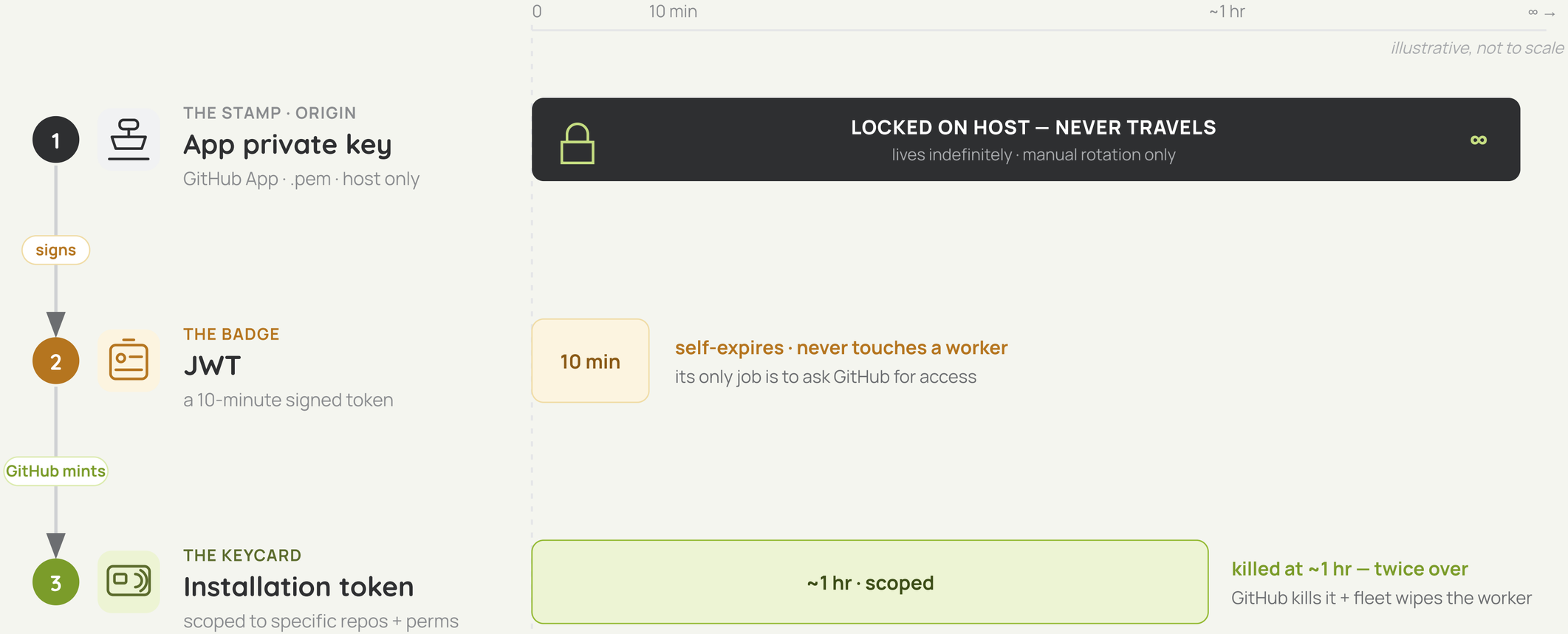
Credentials that are already expiring

How do we hand the agent Git access without handing it a key to the kingdom?



PATTERN 2 · THREE LIFETIMES

One key signs a badge; the badge earns a keycard



The biggest blast radius is the longest bar — so it's the one thing we never let off the host.

GitHub-App mode only. Bitbucket, Azure & plain PATs are bring-your-own-token.

PATTERN 2 · THE RIGHT SHAPE

Trade a master key for disposable ones

✕ Long-lived PAT

- One secret, distributed to every machine
- Broad scope — works everywhere
- Lives ~a year. Steal it once, own it for months

✓ Minted token

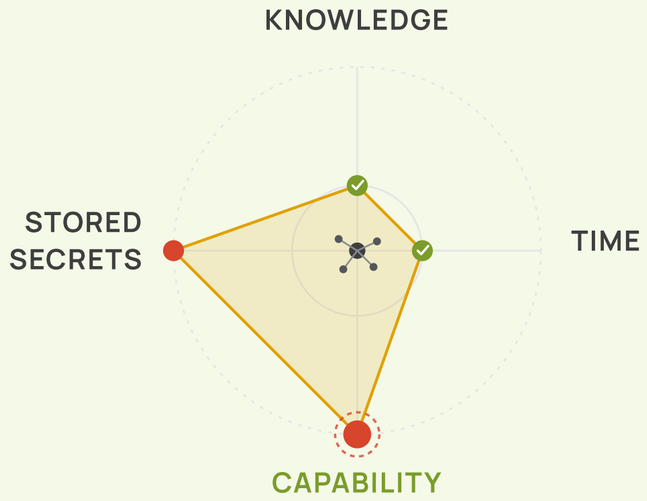
- Narrow — scoped to repos + permissions
- Minutes, not months — auto-dies
- Minted on demand, per job

The broker holds the master key; the workers only ever hold **expiring derivatives.**

03

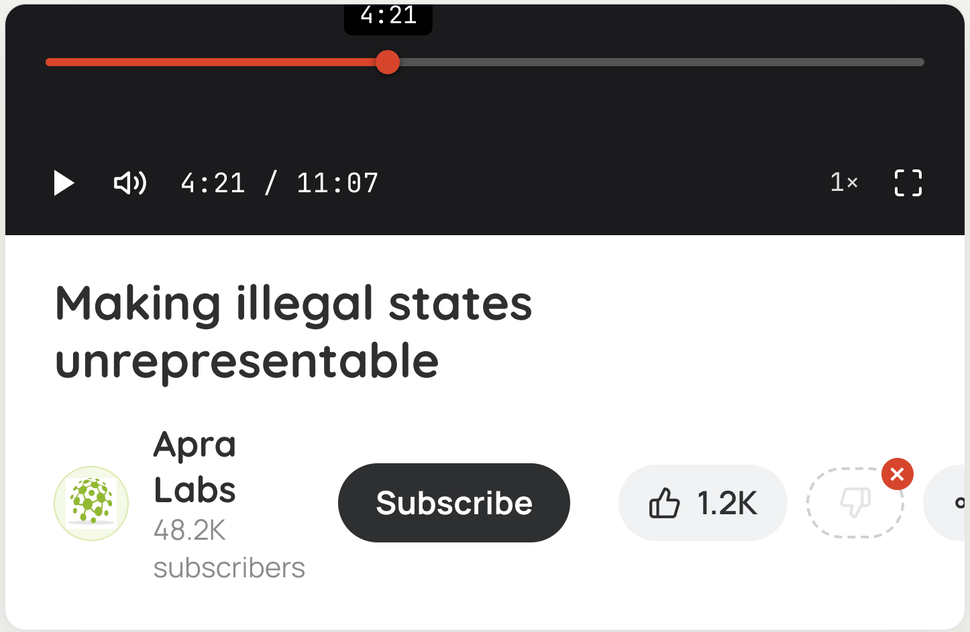
Make the bad action unrepresentable

Within that hour — what can the agent even do? Here MCP's tool model does what a policy doc never can.



PATTERN 3 · MAKE THE BAD STATE IMPOSSIBLE

If google made disliking videos "Unrepresentable".



The dislike isn't *disabled* or hidden behind a rule — it's **not in the interface at all.** You can't click what was never rendered.

PATTERN 3 · ROLE-SCOPED PERMISSIONS

Two roles, two tool surfaces

`doer`

Read · Write · Edit · Glob · Grep

Broad `Bash` — git, mkdir, cp, mv, rm, curl, tar, sed, awk, **sh**

Full build-and-test power.

`reviewer`

Read · Glob · Grep — and write to **only**

`docs/** · feedback*.md · progress.json`

No mkdir. No rm. No **sh**. It can't even make a directory.

On the member, the apra-fleet MCP server is **disabled** — the agent CLI enforces the allowlist at call time.

PATTERN 3 · THE DISTINCTION THAT MATTERS

Policy vs architecture

POLICY

"Reviewers should not edit code."

A sentence in a doc. Trusts behaviour. A prompt injection can scream "edit it" all it wants.

ARCHITECTURE

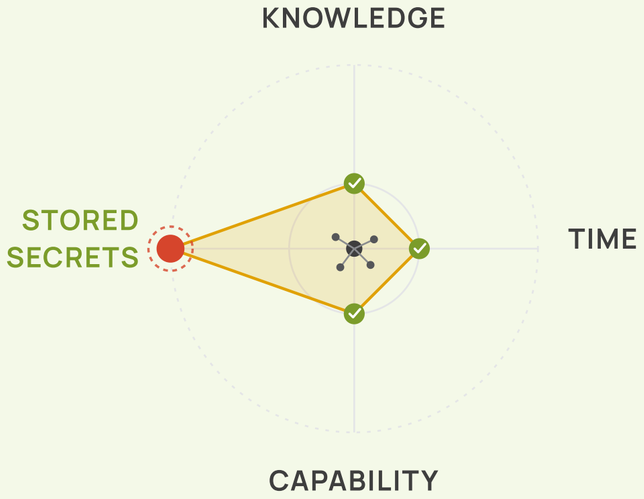
A `Write` to the source tree has nowhere to land.

The reviewer's write allowlist is docs + feedback files, nothing else. The violation isn't caught — **it's unrepresentable.**

HONEST BEAT Merge is still **policy** today — both roles have `Bash(gh:*)` and "don't merge" is a PM instruction. Making it architecture means scoping the minted token. Every real system is mid-migration; the discipline is knowing which boundaries are still just sentences.

04

Encryption at rest



PATTERN 4 · BUILD THE MENTAL MODEL FIRST

Where a secret lives in apra-fleet

A “secret” = a GitHub PAT, an LLM API key, the SSH password a member needs. It **enters out-of-band** (Pattern 1 — never through the model), then takes one of two homes:

PLAINTEXT

`"ghp_xxxxxxxx"`

the real secret — exists for milliseconds

encrypt

→

`AES-256-GCM · node:crypto`

ON DISK · persist: true

`credentials.json` → the locked `iv:tag:cipher` blobs · `salt` → the 32-byte key, a separate file

IN RAM · persist: false — the default

In-memory only · key is random per process · gone on restart. **Most secrets never touch disk.**

Plaintext
the real secret

Ciphertext
the locked, unreadable blob you store

Key
the 32 bytes that lock & unlock it (file: `salt`)

GCM
AES-256 plus a tamper seal — the “auth tag”

Two files, two homes — RAM and disk. Now an attacker shows up: which of these can they actually read?

PATTERN 4 · THE HONEST THREAT MODEL

How far does an attacker have to climb?

The hallway had a doormat at every door. **This is what's behind the lock now** — watch an attacker climb.

COST · PRIVILEGE

1	**The LLM transcript & logs** → a handle: {{secure.NAME}}	✓ nothing
2	**credentials.json — backup, sync, stolen drive** → iv:tag:cipher blobs · no key in this file	✓ nothing
3	**A co-user account on the box** → permission denied (chmod 0600)	✓ nothing
4	**Tampers with the ciphertext** → GCM auth tag · decrypt throws, never lies	✓ caught
5	**Steals a disk image / process dump (non-root)** → session keys live in RAM · never written	✓ nothing
6	**Local root — or anything as the fleet user** → reads key + ciphertext · full decrypt	✗ GAME OVER

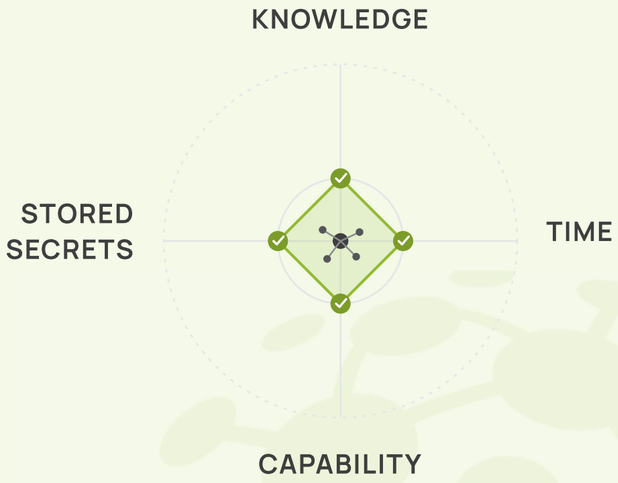
Every rung below root is cheap, common, accidental — and closed. The one that works beats all local encryption — ours included. That's the boundary of the category, not a gap in the design.

THE THESIS, RESTATED

Security as architecture, not policy

KNOWLEDGE	Secrets the model can’t see
TIME	Tokens that are already expiring
CAPABILITY	Tools that were never handed over
STORED SECRETS	Disk that’s ciphertext

None of them depends on **the agent behaving.**



TAKE THIS HOME

Four patterns to take home for your own MCP server

1. Never put a secret in a tool signature. Take a handle; resolve server-side; redact output.
2. Be the broker: hold the long-lived secret centrally, mint short scoped tokens to workers.
3. Compose a minimal tool surface per role. The safest tool is the one never exposed.
4. Make stored secrets ephemeral by default — encrypt regardless.

OPEN SOURCE · ON NPM

blindfold

The security layer behind everything you just saw — pulled out of apra-fleet so **any MCP server** can drop it in. Out-of-band capture, secret vault, handles that the model never sees — in one dependency.

```
$ npm i @apra-labs/blindfold
```

THANKS FOR LISTENING

Questions? Let's keep talking.

When Agents Get SSH Keys: Securing Distributed AI Fleet with MCP

Connect on LinkedIn
Mradul Dubey · /in/mraduldubey

Rate this session
Your feedback shapes the next one